



Verastream Host Integrator

Design Tool Guide

Table of contents

Welcome to Host Integrator	4
Terminal Sessions from the Start Menu	4
Referencing Web Service Files	5
Referencing API Documentation	5
Using the Design Tool	6
Implementation Options	6
Design Tool Features	8
Connecting to a host	8
Using models to encapsulate a host application	9
Abstracting a host application	9
Providing core runtime services	9
Recording command lists	9
Implementing preferences	10
Providing online and offline design modes	10
Working collaboratively	10
Adding event handlers	10
Planning the Host Integrator Project	12
Planning the Host Integrator Project	13
Host Integrator Components	15
Development Process	21
Learn to Use Host Integrator	24
Using the Design Tool	55
The Modeling Process: Getting Started	56
Work Collaboratively	58
Setting Up a Connection	61
Configuring Sessions	73
Importing Model Elements	138
Adding Entities to a Model	145
Adding Operations to an Entity	154
Tables and Procedures	157

Modeling Tips	189
Using Regular Expressions	205
Event Handlers	220
Debugging Models	290
Deploying Models	315
Using Web Services	336
Additional Resources	351
Connectors and APIs	748
Web Builder	1117
Using Web Builder	1118
Legal Notice	1162

1. Welcome to Host Integrator

Verastream integration encapsulates host functionality into services for rapid re-use in new applications. Verastream Host Integrator (VHI) contains these components:

- Design Tool

You use the Design Tool to build host application models quickly and easily by navigating through legacy host applications and selecting the appropriate fields.

- Web Builder

After you create a model, you use Web Builder to generate and deploy a variety of Web projects; including HTML 5 Web applications and objects for both Java and .NET environments. Web Builder generates all the necessary project files so that the source code can be quickly viewed, modified, and rebuilt using industry standard tools such as the latest versions of Visual Studio, Eclipse, and Visual Basic.

- Host Integrator Server

The Host Integrator server provides an enterprise production environment in which to deploy your models. The Host Integrator servers support failover and load balancing using the Administrative Console as a central management hub.

- Administrative Console

You use the Administrative Console to deploy, configure, monitor, and optimize your Host Integrator models and servers in a production environment

Take the tutorial

Spend 30 minutes and use Design Tool and Web Builder to build and deploy a simple Host Integrator application.

1.1 Terminal Sessions from the Start Menu

You can install VHI and have immediate access to your host. From the Start menu, choose either 3270 Terminal Session or 5250 Terminal Session, depending on your host type. You can also enter a known address such as `http://<server-name>:8081/Terminal3270` into your browser. An HTML 5 Web application screen displays an host connection dialog where you can enter the appropriate connection information, complete the host connection, and interact with your green screen host application. This feature gives you access to your host applications without having to open the Design Tool or Web Builder and without having to create and deploy a model.

Using Web Builder project properties for HTML 5 Web applications you can preconfigure a terminal session with a host name and port number. This means you will not be prompted for this information every time you launch the application. This configuration applies to the specific project you are creating and not to the Start menu terminal session links.

1.2 Referencing Web Service Files

Web services are available after deployment in both SOAP and REST formats. Developers can use these documents to identify inputs, outputs, and methods needed to consume the Web service.

You can access your Web service documents here:

List of WSDL documents for deployed models: `http://<session server>:9680/vhi-ws`

List of REST documents for deployed models: `http://<session server>:9680/vhi-rs`

Connect to model (non-pooled): Depending on whether you are accessing a SOAP or REST service - `http://<session server>:9680/vhi-ws/model/<model name>?wsdl` or `http://<session server>:9680/vhi-rs/model/<model name>?json`

Connect to session pool: Depending on whether you are accessing a SOAP or REST service - `http://<session server>:9680/vhi-ws/session/<pool name>?wsdl` or `http://<session server>:9680/vhi-rs/session/<pool name>?json`.

1.3 Referencing API Documentation

API documentation is provided for the Java, JDBC, C, and .NET connectors as well as two additional Host Integrator APIs. See [Connectors and APIs](#) for information.

2. Using the Design Tool

The Design Tool is a component of the Developer Kit for modeling and encapsulating an existing host application for integration into client/server and Web applications.

Host Integrator Web Builder automatically generates projects, including Web applications, .NET class libraries, and Java Beans.

- [Host Integrator Components](#)
- [Planning the Host Integrator Project](#)
- [Implementation Options](#)
- [Design Tool Features](#)
- [Development Process](#)

2.1 Implementation Options

- What is the host data you want to integrate into another environment or application?

If at all possible, plan on building a table that represents the host data in a way that can be queried through an SQL statement. Does the data presentation and logic of the host data as presented on the Web differ from the host application itself? If not, a simple rejuvenation may be sufficient.

- Will the host data be presented on a Web page for customers or users?

What is the Web presentation technology? Know the requirements and the environment where the host data will be used. If you're not integrating the data with other applications, a simple rejuvenation of the host application may be sufficient. Otherwise, you should plan on

setting up tables and procedures so that an external application, such as a Web service, has access to the host data. This decision has a direct impact on how you build your model.

- Know the development requirements for the external application.

Host Integrator provides APIs for integrating host data through SQL, ASP, ASP.NET, JSP, Web Services, COM, and .NET. For a quick implementation, many of these options can be generated using Host Integrator's Web Builder.

You can use Host Integrator with Microsoft's Active Server Pages (ASP), ASP.NET, or a Java Server Pages (JSP) environment. In this environment, you can use Host Integrator to give users access to host data from a Web browser.

You can use Host Integrator to make host data available to another client/server application, whether or not it is Web-based.

- Host Integrator support for object frameworks on application servers include Web Services, the Component Object Model (COM), and .NET. Application server deployments are especially well-suited for situations in which high volume host systems are extended to the

Web. Additional business logic can be developed to add more functionality and capabilities to the system without any work or development required in the host system environment.

2.2 Design Tool Features

- Connecting to a Host
- Using Models to Encapsulate a Host Application
- Abstracting a Host Application
- Providing Core Runtime Services
- Recording Command Lists
- Implementing Preferences
- Providing Online and Offline Design Modes
- Working Collaboratively
- Adding Event Handlers

You can use the following features to capture the functionality of your host application:

2.3 Connecting to a host

The Design Tool can connect to the following hosts:

Host	Models
IBM 3270	Models 2 (24x80) Normal and Extended, 3 (32x80) Normal and Extended, 4 (43x80) Normal and Extended, and 5 (27x132) Normal and Extended
IBM 5250	Models 3179-2, 3180-2 (132 Column Capable), 3196-A1, 3477-FC (132 Column Capable), 3477-FG (132 Column Capable), 3486-BA, 3487-HA (132 Column Capable), 3487-HC (132 Column Capable), 5251-11, and 5291-1
VT	VT102, VT400-7, VT400-8, and VT52 terminals

Host	Models
HP	HP2392A and HP 70092 terminals

You select the host connection type on the New Model or Session Setup dialog box, accessible from the File or Connection menu respectively.

2.4 Using models to encapsulate a host application

The main feature of the Design Tool is the modeling feature, which enables a host expert to create a model of a host application. First, you connect to a host via the Design Tool and then define entities for terminal screens, which may include patterns for identification, attributes to specify the location to input data, and one or more operations to allow programmed traversal of the host application, and variables which can be mapped to various attributes or various entities.

In most cases, you will use tables and procedures to create an abstraction of the host data so that it can be queried through a subset of the industry-standard Structure Query Language (SQL). See "Abstracting a Host Application" below.

The model file (`.modelx`) is saved in the Design Tool and then copied to a Host Integrator Server. For more information, see [The Modeling Process](#).

2.5 Abstracting a host application

You can create procedures (and underlying tables) to add a database abstraction layer on top of your host application model. Client application programmers can then access this database abstraction layer, either by a direct call to a procedure or through a subset of the industry-standard Structured Query Language (SQL). For an SQL query, the client application specifies a table, a set of input parameters, and a set of desired output parameters. Host Integrator then returns the desired data to the client application.

2.6 Providing core runtime services

In addition to the definition process, the Design Tool provides server-like services for the modeling and procedure definitions. This permits the user to test and debug models and database procedures prior to deployment. The model layer requires entity recognition, operation execution, and variable reads and writes. The debug layer takes arbitrary input and resolves a query (or returns an error) or executes a specified query.

2.7 Recording command lists

The command list recorder records host commands for operations required by the debug layer. On the Model menu, point to Record and then click Start Recording to begin. For information about creating login, logout, and move cursor command lists, see [Command List Edit](#).

2.8 Implementing preferences

There are several functional user preferences that can be implemented, including creating default names for attributes, automatic pattern generation, and proposing new operations when appropriate. On the Settings menu, click Preferences for more information.

2.9 Providing online and offline design modes

The Design Tool has the ability to display in online and offline design modes. As each entity is defined, two files are created to enable the design mode process.

- A "screen snapshot," or a snapshot file contains the contents of the display memory and which can then be available to the Design Tool for editing the corresponding entity in offline mode.
- A Host Emulator trace file is a Telnet trace for the terminal screen that the Host Emulator can send over a Telnet connection to simulate the screen being sent from the host.

Offline mode is available for all emulation options, while Host Emulator is available for IBM 3270 and 5250 emulations only.

2.10 Working collaboratively

Host Integrator provides the ability for multiple developers to work simultaneously on a project. Models are saved using the .modelx extension, which gives multiple developers the ability to work collaboratively and merge their changes using a standard version control package, such as Git. When you save a model as a modelx file, all entities, tables, and supporting files are converted into XML, validated by an .xsd file, and saved in the models directory, under the modelx folder. See [Working Collaboratively](#).

You can also import portions of models for use in other models. This makes model creation more efficient. Multiple developers can work on large models simultaneously and pull the various pieces together at a later time. See [Importing Model Elements](#) for more information.

2.11 Adding event handlers

An event handler gives you the ability to customize the behavior of a model.

The Design Tool offers a variety of features that assist you in creating event handlers. The result is a Java class that conforms to rules for event handling. This class is then mapped (attached) to specific objects of a model to customize its behavior.

You can attach event handlers to events associated with the entire model, a life cycle event, or to entities, attributes, operations, recordsets and recordset fields, and procedures. You can reuse a handler in multiple models or with multiple objects of the same type within the same model.

If you're ready to begin modeling, see [The Modeling Process](#).

More information

[The Modeling Process](#)

[Working Collaboratively](#)

[Importing Model Elements](#)

3. Planning the Host Integrator Project

3.1 Planning the Host Integrator Project

A model is the representation of a host application's connections, screens, navigation, and data flow that you build with the Verastream Host Integrator Design Tool. Once you have modeled your host application, you deploy the resulting model to a Host Integrator Server, where it can provide real-time access to host data through web-enabled services.

Building a good model of a host application requires a basic understanding of the business problem that will be solved using the Host Integrator model and associated components. Use the guidelines below to prepare for Verastream host application modeling.

- **Gather the basic information about the host and the host application.**

Know the host operating system and any relevant details about the host application software. The more you know about your host application, the greater the likelihood that you'll have an effective, robust model.

- **Determine the data you want to access in the host application.**

If you're not familiar with the host application yourself, a host application programmer or an experienced business user needs to identify logon procedures and the sequence of screens typically used to reach the data you want to access. As you build the host application model, each host screen used in the model is represented as an entity. An efficient host application model contains only as many entities as are required to perform the necessary transactions between the external application and the host application.

Take the time to create an easy-to-follow data map to simplify the process of building the model. This exercise may reveal multiple screens and operations that display the same data. Whenever possible, model the screen that shows the data in its updated form for the transaction in question.

- **Identify the environment where the host data should be available.**

Will the host data be presented on a web page for customers or users? Is the host data to be used by another application? Know the requirements and the environment where the host data will be used. If you are only planning to update the host application with a more modern web page look and feel, you can build a simple model. However, if you plan to alter the work flow or interoperate directly with other applications, your model will need to be more complex.

- What is the Web presentation technology? Know the requirements and the environment where the host data will be used. If you're not integrating the data with other applications, a simple rejuvenation of the host application may be sufficient. Otherwise, you should plan on setting up tables and procedures so that an external application, such as a Web service, has access to the host data. This decision has a direct impact on how you build your model.

- **Know the development requirements for the external application.**

Host Integrator provides APIs for integrating host data through SQL, ASP, ASP.NET, JSP, Web Services, COM, and .NET. For a quick implementation, many of these options can be generated using Host Integrator's Web Builder.

- You can use Host Integrator with Microsoft's Active Server Pages (ASP), ASP.NET, or a Java Server Pages (JSP) environment. In this environment, you can use Host Integrator to give users access to host data from a Web browser.
- You can use Host Integrator to make host data available to another client/server application, whether or not it is Web-based.
- Host Integrator support for object frameworks on application servers include Web Services, the Component Object Model (COM), and .NET. Application server deployments are especially well-suited for situations in which high volume host systems are extended to the

Web. Additional business logic can be developed to add more functionality and capabilities to the system without any work or development required in the host system environment.

- **Plan for error handling.**

Error handling is a very important part of the model-planning process. Learn where and how the model can fail or put the host application in an error condition. Then, build exception handling into your model for these cases to reduce the chance that your model will fail after it is deployed.

- **Review the modeling tips in the online help.**

The [modeling tips](#) cover basic model style guide recommendations, a comparison of table-based vs. non-table-based models, basics on screen recognition and pattern usage, and host synchronization.

More information

[Using the Design Tool](#)

[Learning to use Host Integrator](#)

[The Modeling Process](#)

3.2 Host Integrator Components

Host Integrator includes a variety of development and runtime components to integrate host data into Web applications or other client/server applications.

The Development Kit provides a full range of utilities with a limited license server, while the Server Kit includes a full license server without the development tools.

Component	Description	Development Kit	Server Kit
Design Tool	Create a model of the host application by navigating through host screens	Yes	No
Web Builder	Generate web-based applications from the model.	Yes	No
Connectors	Connect to the server to manage host connections and sessions.	Yes	Yes
Session Server	Access data on a variety of host systems.	Yes (limited)	Yes

Component	Description	Development Kit	Server Kit
Web Server	Run Java or HTML5 web application projects.	Yes	Yes (install option)
Host Emulator	Use to test an application without a host connection.	Yes	No

Component	Description	Development Kit	Server Kit
Administrative Console	Use to view and configure server and session information.	Yes	Yes

3.2.1 Development Kit

The Host Integrator Development Kit provides developers with the tools to model and build applications that integrate legacy application information. It includes the following components:

- Design Tool
- Web Builder
- Connectors
- Server (limited-license version)
- Administrative Console
- Log Utilities
- Host Emulator
- Verastream Help

3.2.2 Server Kit

The Host Integrator Server Kit is a suite of applications that allow you to deploy applications created with the Host Integrator Development Kit. It includes the following components:

- Server (full-license version)
- Administrative Console
- Connectors
- Log Utilities
- Verastream Help

3.2.3 Adding, Removing, and Repairing Components

You can add components to or remove components from your Host Integrator installation whenever it becomes necessary. You can also repair your existing installation using the same install program.

1. Navigate to `<install_directory>/setup` and run `setup.exe`. The install program will detect the existing installation and display the *Maintenance and Component Selection* panel. By default the Maintenance tab displays.
2. Repair - Reinstalls missing or corrupt files, registry keys, and shortcuts. Settings may be reset to their default values.

Reinstall - Completely removes from the system and begins the installation process anew. The Install Guide, in PDF format, is available from your installation directory.
4. Remove - Removes all Host Integrator files and directories as well as uninstalls all product components.
4. You cannot continue with your installation maintenance selection (repair, reinstall, remove) if you click Continue and navigate away from the Maintenance tab. Select the maintenance option and click Continue while on the Maintenance tab to perform the desired action.
5. To view a list of currently installed components, open the *Component Selection* tab. This tab displays the components that are currently installed. Select or clear components to create the desired installation.
6. Click *Continue* and setup will perform the necessary actions to install or remove your selections.

3.2.4 Starting and Stopping Services

There are several Host Integrator services. Different services are installed depending on your platform, type of product (Server or Developer Kit), and other options that you select during installation.

Host Integrator services include:

- Session Servers
Runtime engine that loads deployed models; connects to host systems, and services request from clients.
- Log Manager
Captures runtime messages in log files; and handles SNMP and email notification.
- Management Server
Supports the Administrative Console, session server load distribution domains, authentication and authorization security, and session pool scheduling.
- Web Server
Runs Java and HTML5 Web projects generated by Web Builder.
- Host Emulator

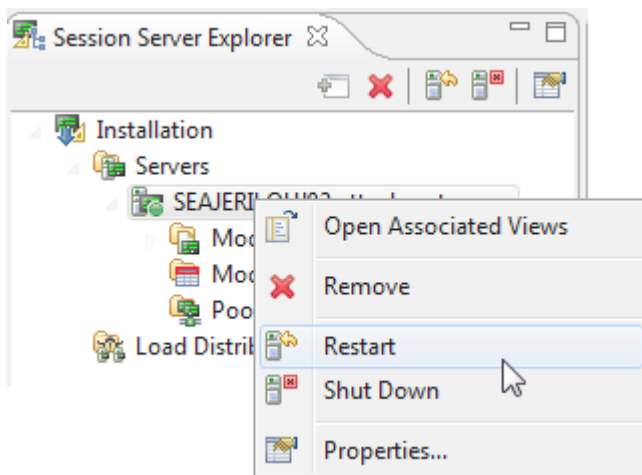
Simulates a host (3270 or 5250 only) for demonstration, training, or testing purposes.

On Windows, with the Developer Kit installed, the services start automatically on system startup. When you install the Server Kit, both the Host Emulator and Web Server are disabled at startup. For Linux or zLinux see the install guide for instructions on how to configure the system to start services automatically.

To start or stop services from the Administrative Console

The Session Server, Management Server, and Host Emulator services running on Windows or Linux servers can be restarted or stopped using the Administrative Console.

Open the Administrative Console, from each of the server Explorer panes, select the server you want to interact with, right-click and choose Restart or Shut Down.



To start or stop services from the command line

Commands or shell scripts can be run at a shell command prompt, called from other scripts, or run from a shortcut that you create.

- Windows

Navigate to `<install_directory>/HostIntegrator` or `/bin/services` directory. You can start, restart, or stop all services by running the appropriate batch file or by interacting with a specific service. To interact with a specific service, open the component directory, for example `HostEmulator`, and run the batch file for the action you want to perform; `Restart Host Emulator.bat`, `Start Host Emulator.bat`, or `Stop Host Emulator.bat`.

- zLinux

Navigate to `<install_directory>/opt/microfocus/verastream/HostIntegrator/bin` directory. You can start, restart, or stop all services by running the appropriate script file or by interacting with a specific service. For example, to restart all services, run `atstart -restart all` or to restart just the management server, run `atstart -restart mgmtserver`.

More information

[The Development Process](#)

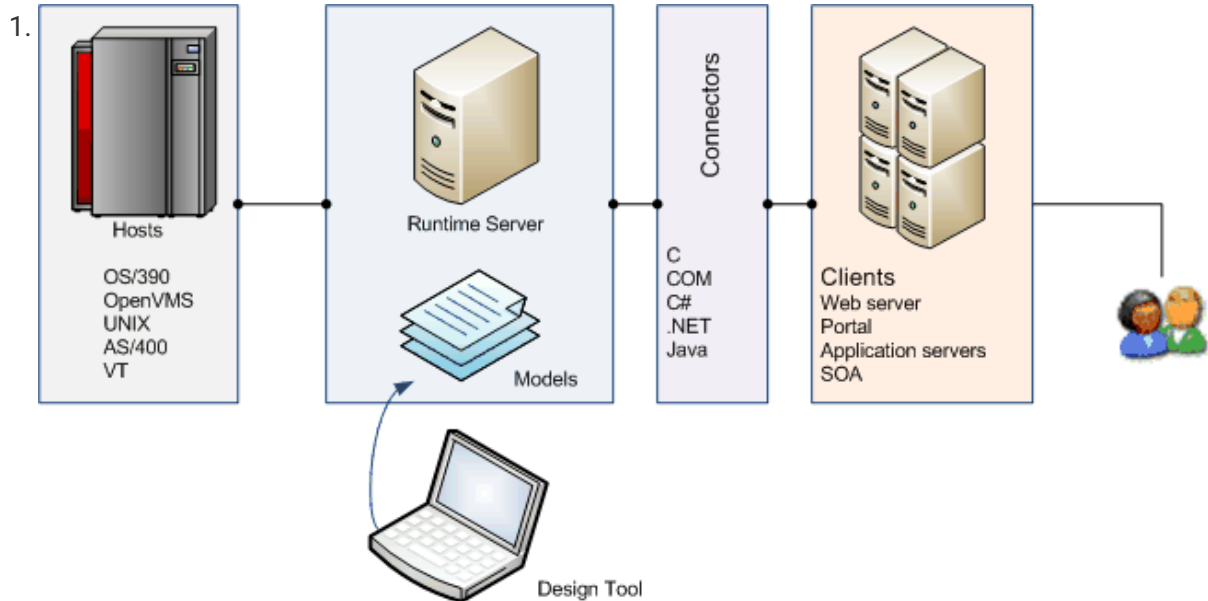
[Planning the Project](#)

3.3 Development Process

To build a Host Integrator application, follow these steps:

1. Plan your integration project

1. To create a good model of a host application you must have a basic understanding of the business problem that you want the model to solve. There are guidelines available to help you plan your project in [Planning the Project](#).



2. Create and deploy the host application model

2. There are a few steps you must follow to create and deploy a model.

Navigate through the host application, identifying the screens and fields that lead to the data that you want to integrate into the new Web or client/server application.

Expose the resulting components as database tables and procedures. A table is a database abstraction with attributes or recordset fields structured in table columns. You can create a procedure that defines how Host Integrator locates, retrieves, updates, inserts, or deletes the data when it fulfills a request through SQL or through another API. If you want to add functionality beyond what is provided in the Design Tool, you can add event handlers that extend or override default model behavior.

Test your model to confirm that it operates as you expect.

Deploy the model to a folder containing the model file and any supporting files on the Host Integrator server. Web services are automatically provided by the session server as an embedded SOAP stack or as a REST service after a model package is deployed using the Design Tool. The embedded Web service supports all model procedures and features, including `executeSQLStatement` and `ProcessString` event handlers. See [About Verastream Web Services](#) for detailed information on Host Integrator Web services.

3. Build a new Web or client/server application

3. You can build a new application using these options:

Use Web Builder to quickly build and deploy a Web application. You can build a Web application for a Java, .NET, or ASP interface, and then launch a standard editor for any of these interfaces to extend or modify the Web application.

Use Web Builder to generate a component interface that provides access to the procedures and screens in a host model. Component interfaces are available using Java Beans or .NET 2.0 class libraries.

Use the IDE of your choice to consume Verastream-generated components. Mix and match components or services to create composite applications. You can use the Host Integrator connectors to develop a new client/server or Web application. Using these APIs, a developer can write a client/server or Web application without extensive knowledge of how the host application works.

4. Deploy the application to the enterprise

4. In a production environment, the deployed Host Integrator servers provide dynamic load balancing and failover in transaction-intensive environments. Host Integrator supports all mainframe-based security packages and also has its own multi-level security, which can be tied to the security schema of the selected deployment platform (Windows or Linux).

5. Using the Administrative Console

5. The rich Administrative Console user interface keeps mass management in mind and provides central administration of the management server and Host Integrator servers. It is here you will be able to:

- Maintain control over sessions and pools

- Monitor logging of session information, activity, use and status

- Configure external reporting

- Configure SNMP and JMX support for third party consoles

- Manage models and edit model properties

3.4 Learn to Use Host Integrator

To get the most benefit from this tutorial, review the following topics before you begin:

[Using the Design Tool](#)

[The Development Process](#)

You will need to install the Developer Kit to run this tutorial.

3.4.1 Goal

The goal of this tutorial is to produce a Host Integrator application.

You will work with a simulation of a 3270 (CICSAccts) host application and use Verastream tools and utilities to build a simple Host Integrator application. This is a demonstration application, consisting of limited navigation and a database of generic customer data for eight fictitious customers. It is a useful tool for learning to create a model.

This tutorial walks you through these steps:

1. [Define the business need for the project](#)
2. [Develop the model requirements and map the data](#)
3. [Build, deploy, and test the model and web service](#)
4. [Creating a Web Builder project and generating a Web application](#)
5. [Deploy the project to the enterprise](#)

Let's get started.

3.4.2 Defining the Business Need

Often the business needs of a project have already been defined before the development process ever begins. Although you may not have been a part of the decision, understanding the business problem is important to building a successful application.

In this CICSAccts project, the business goal is to make specific customer information available to users over the Internet and to other business processes using Web services and other programmatic component interfaces. You will access the data through the CICSAccts application on a 3270 host.

To accomplish the goal you will: automate the login to the host, retrieve the host data, and display the data in a simple Web application.

3.4.3 Developing Model Requirements and Mapping the Data

Now that you understand the goal and the deliverables of the project, you are ready to use the Host Integrator Design Tool to build a model of the application. The model contains all the information required for host application navigation, screen recognition, data storing, and data retrieval.

During this phase, either working on your own or with someone familiar with the host application, you will navigate through the application and document each host screen's inputs, outputs, navigation path, and other quirks. This will result in a model requirements document that identifies the intricacies of each of the host screens that will be required to access the data.

The model requirements document contains two important features: the list of entities (unique host application screens) that need to be modeled, and a chart mapping the project data to the screen where the data will be retrieved.

In our application, in order to retrieve the necessary data, the main screen in the CICSAccts application needs to be treated as two distinct entities: One for input of information to do the search (Main) and one for output of the search (NameSearchResults). The output is a listing of information about each customer; this is the data that you need from the application.

This is a map of the data our model needs for inputs and outputs:

Attribute Name	Used for	Screen	Screen Field Name
LastName	input	Main	SURNAME
FirstName	input	Main	FIRST NAME
AcctNum	input	Main	ACCOUNT
RequestType	input	Main	REQUEST TYPE
LastName	output	NameSearchResults	SURNAME
FirstName	output	NameSearchResults	FIRST
MiddleInitial	output	NameSearchResults	MI
Title	output	NameSearchResults	TTL
Address1	output	NameSearchResults	ADDRESS
AcctNum	output	NameSearchResults	ACCT
Reason	output	NameSearchResults	ST

Attribute Name	Used for	Screen	Screen Field Name
ChargeLimit	output	NameSearchResults	LIMIT

3.4.4 Building and Deploying a Simple Model

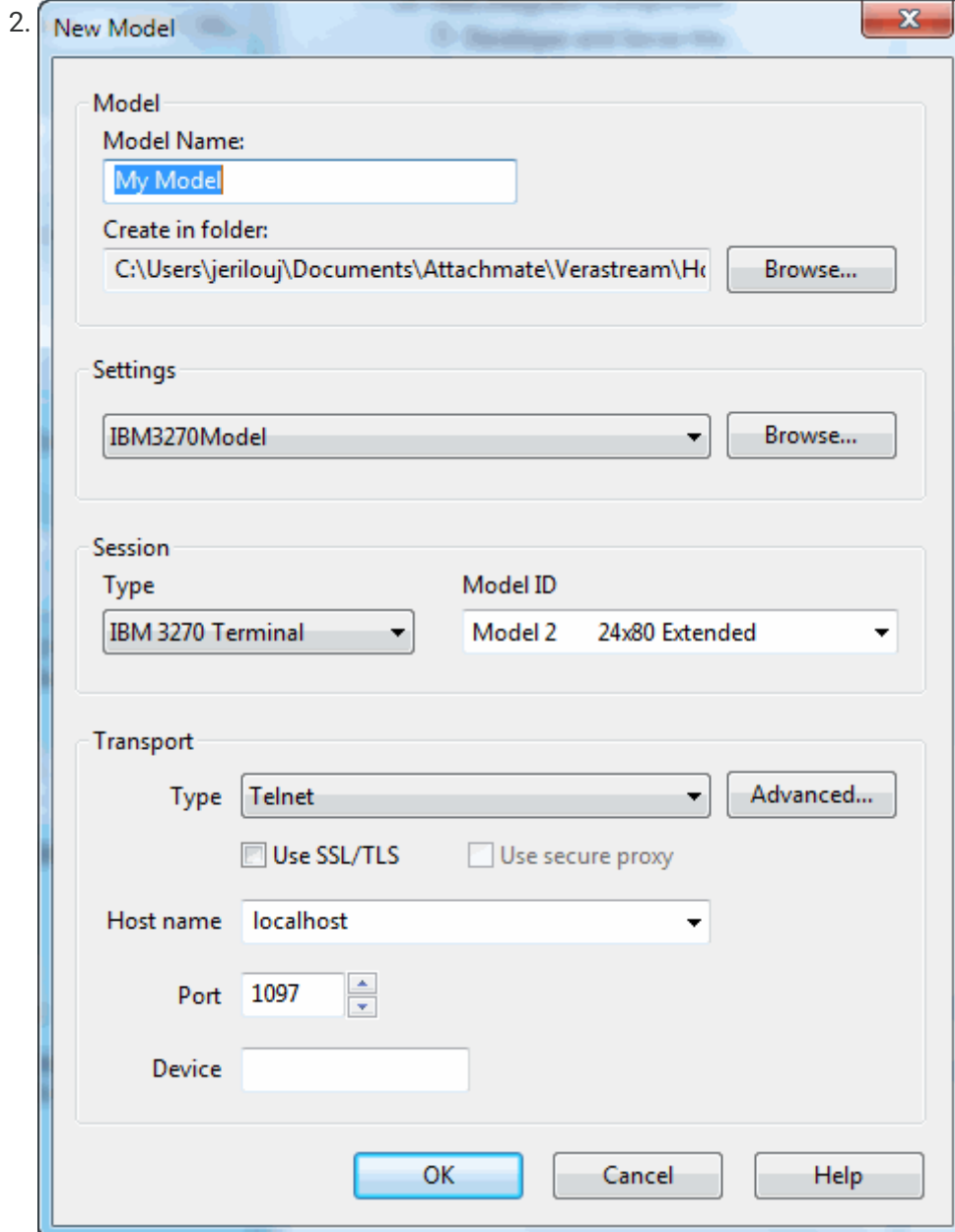
The completed model of the CICSAccts application lets you query the CICS database by a customer's last name. To build this model, you will:

- Connect to host using a login command list
- Create entities from host screens
- Create operations for host navigation
- Create recordsets for scrolling data
- Create tables and procedures for abstraction level queries
- Create procedures to retrieve data
- Test and deploy the model

Connect to Host Using a Login Command List

Open the Host Integrator Design Tool.

2. Click **File > New** to open the New Model dialog box.



3. Type `MyModel1` in the **Model Name** field. This creates a connection profile to the CICS host. Accept the default model location.

4. In the **Settings box**, click **Browse** and select `IBM3270Model`. This provides the appropriate default settings for the session type (IBM 3270 Terminal), terminal ID (Model 2 24X80 Extended), and transport type (Telnet). The CICS demonstration host uses port 1097.

4. Enter the following host connection information:

Field Name	Entry
Host name or IP address	localhost
Port	1097

Field Name	Entry
Device	(Leave blank)

5. From the **Settings** menu, select **Preferences**. In the Preferences Setup dialog box, select the option **When model file opened connect to host**. This configures the Host Integrator Design Tool to connect to the host automatically when the model is opened.

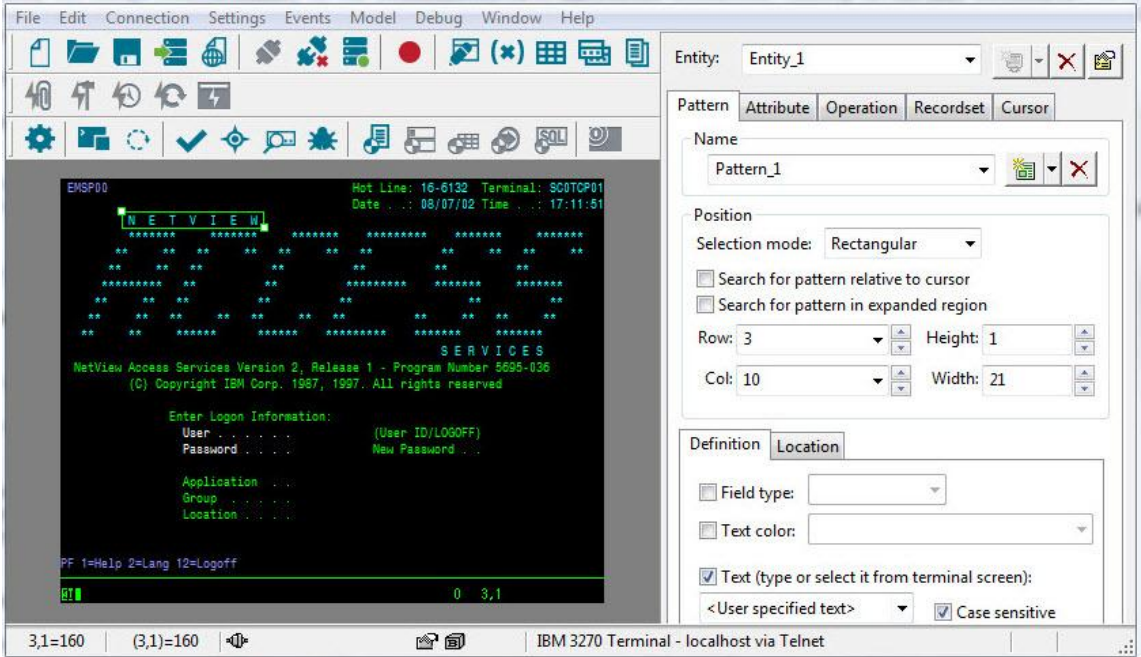
Click **File > Save MyModel.modelx**.

You have now finished configuring the Design Tool to connect to the demonstration 3270 application (CICSACcts) running in the Host Emulator on the local machine over port 1097. Every time a new entity is created (host screen is accessed), the Design Tool automatically notes all operations (navigation through the host application). You will learn more about these terms and processes as you proceed through this demonstration.

About the Design Tool

The Design Tool is divided vertically into two sections or panes:

- Terminal Window -- The left pane displays the actual host application screen.
- Entity Window -- The right pane contains the settings used to define the host screens that make up the model.



CICS APPLICATION NAVIGATION TIPS

Use these navigation tips to negotiate the CICSACcts application example:

The Page Down key on your keyboard is mapped to the host PA2 (page down) key.


The Scroll Lock key on your keyboard is mapped to the host CLR (clear) key.

Pressing **Enter** returns you to the main screen.

Creating a Login Command List

A login command list provides a quick and reliable login to a host. You can also use login commands to bypass a host login screen.

TO CREATE A COMMAND LIST TO LOG ON TO CICS:

If you're not already connected, click  to connect to the model.

Click **Model > Record > Start Recording**.


Click in the Terminal Window and then press the Scroll Lock key (this is mapped to the host CLR by default).

In the terminal window, type `accts`, and press **Enter**.

Click **Model > Record > Stop Recording**.

Select the **Save as login command list** option and click **Save**. The Command List Edit dialog box opens. You will use this dialog box to edit the login command list.

Tip

To open the Command List Edit dialog box, click **Model > Properties**, and then .


Editing the Login Command List

The CICSaccts demonstration application is run from the Host Emulator; to synchronize the Host Emulator login to the application, you need to add an additional wait command to the top of this recorded command list.

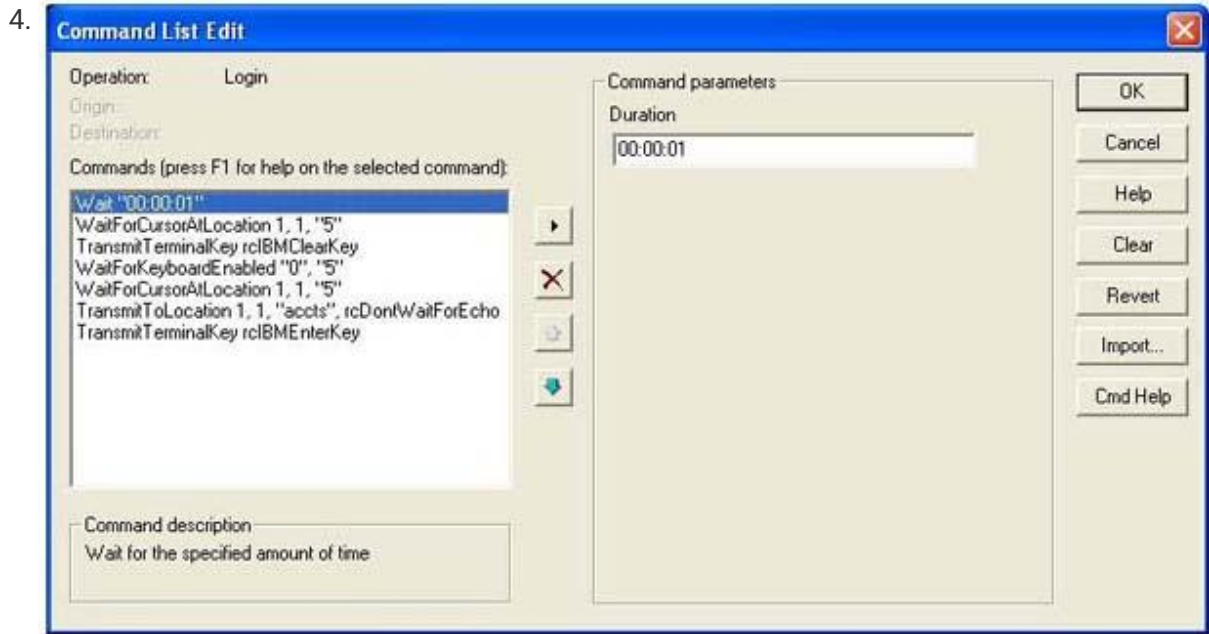
TO EDIT THE LOGIN COMMAND LIST:

Select the first command in the **Commands** field, `WaitForCursorAtLocation, 1, 1, "5"`. If the Row and Column values are not both 1, change the values to 1.

Click **Insert** , then point to **Host Events**, and click **Wait**.

With **Wait "00:00:01"** selected, click **Move Up**  to move this command to the top of the command list. Your commands should look like the command list in Figure 2.

4. Click **OK** to close both the Command List Edit and Model Properties dialog boxes.



Configuring the Login Command List to Load Automatically

The Design Tool automatically executes this login command list whenever you connect to the model.


TO CONFIGURE THE LOGIN COMMAND LIST TO LOAD ON CONNECTION:


Click **Settings > Preferences**.

In the Preferences Setup dialog box, on the **General** tab, select **Execute login commands**, and click **OK**.

Save your model.

TO TEST THE LOGIN COMMAND LIST:

On the tool bar, click  to disconnect your current session.

Click  to reconnect and run the login command list. Each time you open the model, the Design Tool automatically connects and logs you onto the host and displays the Main screen of the host application.

Create Entities From Host Screens

An entity is usually a single host application screen, such as the login screen or the main menu screen of an application. A model is made up of entities (host screens) and the attributes, recordsets, and operations associated with those screens.

Up to this point, you have recorded all navigation through the host application in the login command list. Since the end application does not access data from these initial screens, you do not have to configure each screen in the login sequence as an entity. The first screen you need to define is the Account File Menu screen.

TO ADD A SCREEN AS AN ENTITY:

When the first screen, Account File: Menu, appears after connecting to the host, click **New Entity**



in the Entity Window. By default, the entity is named `Entity_1`.

Change the entity name to `Main`.

Creating a Pattern for Entity Recognition

A pattern is a model element that is used in the screen signature to identify a particular host application screen. Host Integrator uses patterns to distinguish individual entities. Select any static text on the screen, such as field labels, or screen headers and footers, to be designated as a pattern. Each pattern can include up to 259 characters.


Tip

To help ensure that each entity has a unique identifier, it is a good idea to configure at least two patterns to define each entity, one at the top of the screen and one at the bottom.

TO CREATE PATTERNS FOR THE MAIN ENTITY:

1. In the Terminal Window, use the mouse to select the text: `ACCOUNT FILE: MENU`



2. On the Pattern tab, click **New Pattern** .

By default, the pattern is named `Pattern_1`. Accept this default name.

Click **Apply**.

The Design Tool has now recorded the position, properties, and signature parameters of the pattern. You can access and edit these settings on the Pattern tab.

Typically, it is a good practice to define at least two patterns per entity, and on most host screens you would select a second string of unique text to use as your second pattern. However, when you defined the requirements for the model, you realized that for this application you must take a different approach because the same host screen is used to both request and display customer information.

When you enter customer information on the screen (Figure 4), the data is returned in the blank area at the bottom of the screen (Figure 5). Compare Figure 4 to Figure 5; all of the text on the top half of the screen remains the same. It is difficult to assign a unique pattern on the first screen because all the static text on the first screen also appears on the second screen.

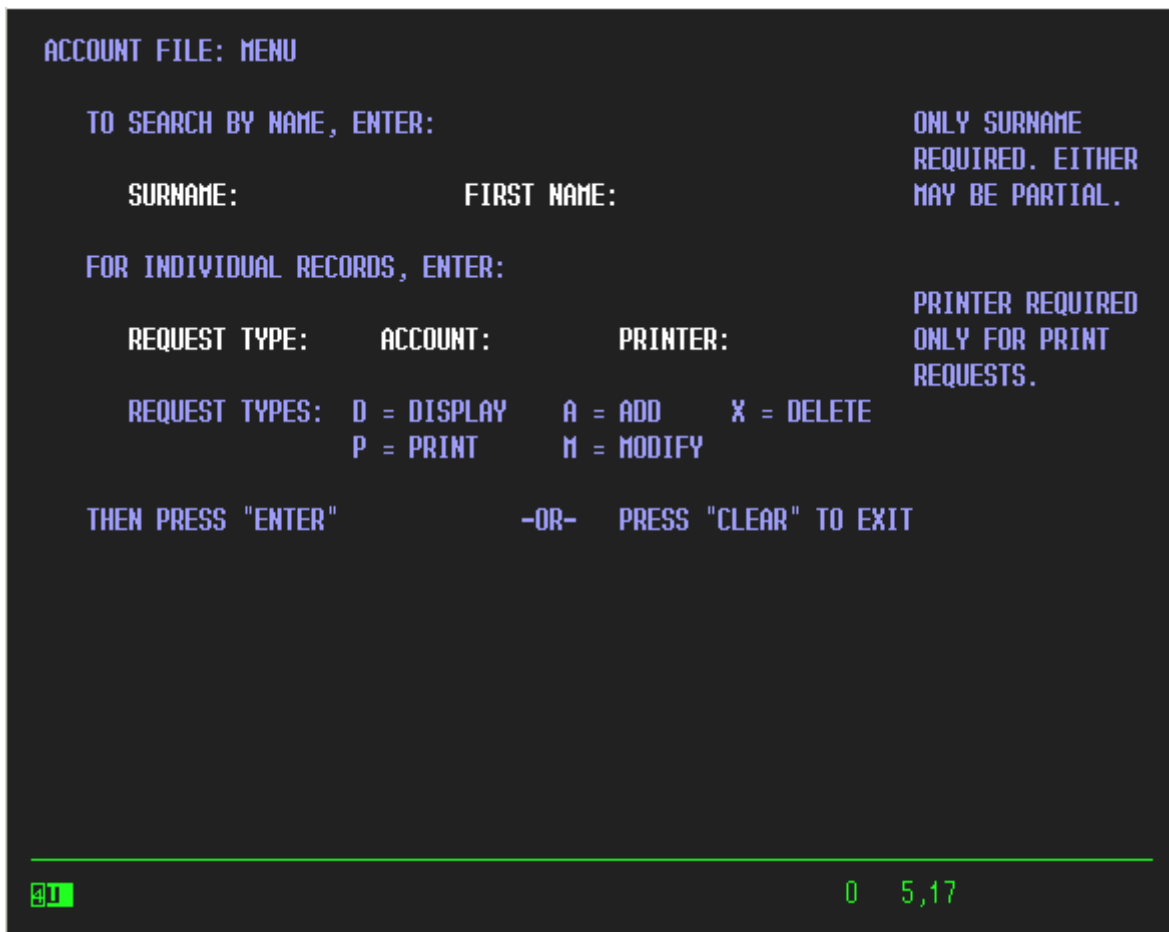


Figure 4 -- Entity 2 \"main\"

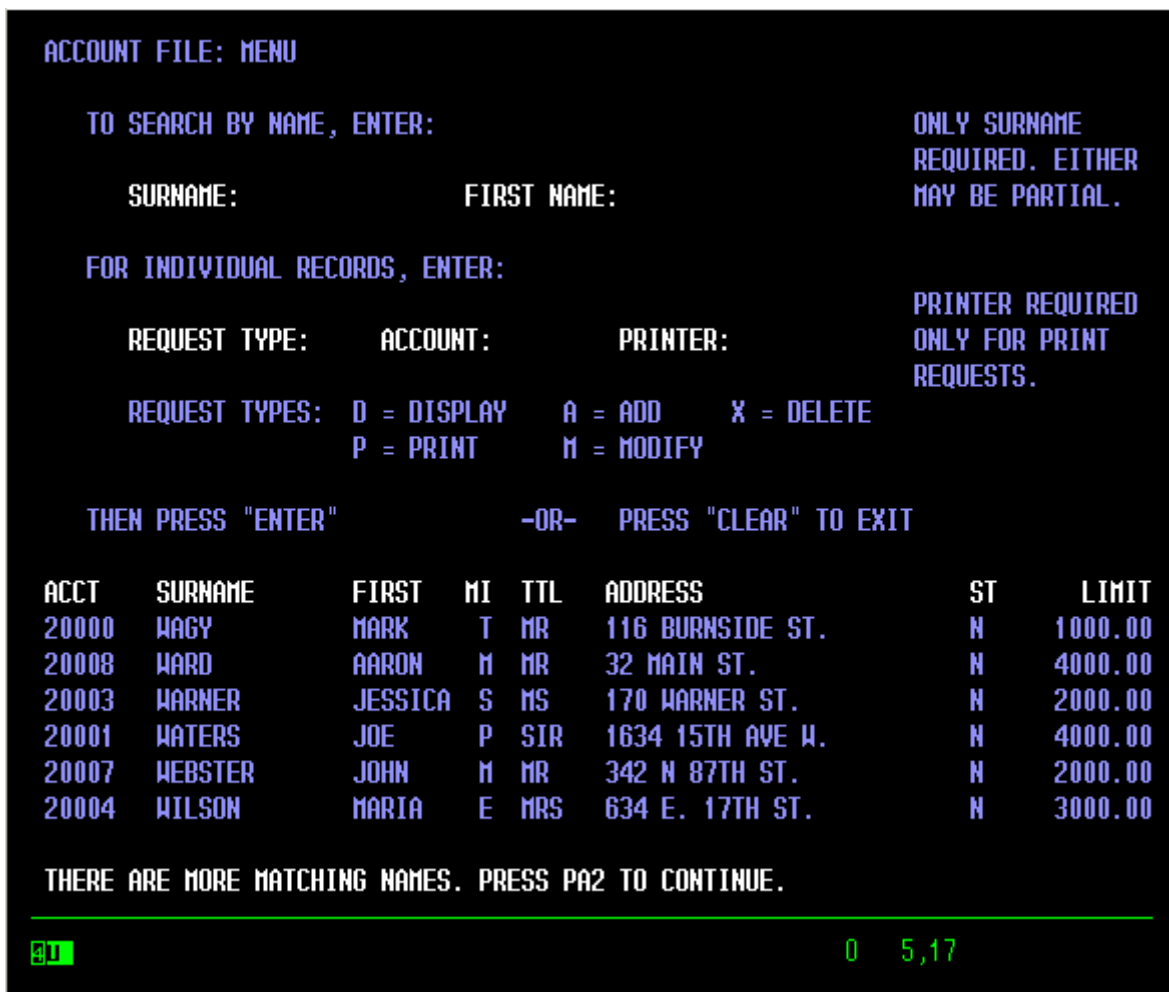


Figure 5 -- Entity 2 \ "NameSearchResults\ "

Can you treat this as one entity? You could, but that would add complications when creating operations and recordsets.

The simplest solution is to create two entities and use an exclude; Host Integrator "recognizes" the first entity by the presence of specific text, and the second by the absence of that same text.

To use an exclude, select a pattern that exists only on the second screen, and then configure Design Tool to recognize the first screen by the absence of that pattern. In other words, when the pattern is not there, then by default, the first screen is being accessed.

TO CREATE A PATTERN USING AN EXCLUDE:

In the Terminal Window, click in the **SURNAME** input area, type **w** (uppercase W), and then Press **Enter**. You are now on the second entity (the same screen as the first entity, but with data).

2. Use the mouse to select the entire account information heading:



Figure 6 -- Select this text to create a second pattern

3. On the **Pattern** tab, click the **New Pattern** button. Accept the default pattern name, `Pattern_2`.

At the bottom of the **Pattern** tab, select **Screen properties not present** as part of the Definition and click **Apply**.

Click **Yes** to dismiss the warning message box.

When you use the *Screen properties not present* setting, Design Tool automatically initiates the creation of a new entity. Before continuing, you must complete the steps to add this second entity and identify it with patterns.

TO ADD THIS SCREEN (WITH DATA) AS A NEW ENTITY:

Click the **New Entity** button in the Entity Window.

Select the default entity name (`Entity_2`) and replace it with `NameSearchResults`.

TO ADD PATTERNS FOR THE SECOND ENTITY, NAMESEARCHRESULTS:

1. In the Terminal Window, use the mouse to select the text: `ACCOUNT FILE: MENU`



Figure 7 - Select this text in the Terminal Window to create a pattern in the second entity

2. On the **Pattern** tab, click the **New Pattern** button. Accept the default name.
3. Use the mouse to select the entire account information heading:



3.  The image shows a terminal window with a black background. A table of account information is displayed in blue text and is highlighted with a green rectangular selection box. The table has the following structure:

ACCT	SURNAME	FIRST	MI	TTL	ADDRESS	ST	LIMIT
20000	WAGY	MARK	T	MR	116 BURNSIDE ST.	N	1000.00
20008	WARD	AARON	H	MR	32 MAIN ST.	N	4000.00

Figure 8 - Select this text to create a second pattern in the second entity

3. This is the same text you used when you identified the Main entity; however, in that instance you configured the Design Tool to make sure that the pattern was not present when identifying



the Main entity. In identifying the NameSearchResults entity, you are configuring the Design Tool to verify the pattern is present.

4. On the Pattern tab, click the **New Pattern** button. Accept the default name and click **Apply**.

On the Confirm Operation dialog box, click **Discard**. Then click **Yes** to confirm the discard and close the dialog box. (Design Tool automatically creates navigational operations; however, you need to first identify attributes and fields.)

You have now created two entities, **Main** and **NameSearchResults**.

The next step is to identify the fields or attributes that you need on each entity. To return to the

Main entity, click **Disconnect**  on the toolbar, and then **Connect**  to reconnect to the host, run the login script, and open the Main entity.

Configure Attributes for Data Entry and Retrieval

An attribute is a selected area on an entity that contains data that is accessible through the model. This area might be a data entry field or a text field that changes depending on the choices you make on prior screens. You use attributes to get and set data on entities. Create attributes only for those host fields that need to be accessed; give attributes meaningful names that identify their purpose.

TO CREATE ATTRIBUTES:

1. On the **Main** entity, select the **SURNAME** field input area. (On block mode applications such as this 3270 application, you can double-click the field to select it.)

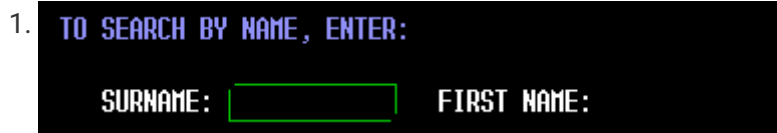


Figure 9 – The defined area of Attribute_1

2. On the **Attribute** tab, click **New Attribute**  to add this as `Attribute_1`.

3. Rename `Attribute_1` to `LastName` and if necessary:

Adjust the start position to row 5, column 17.

Adjust the end position to height 1, width 12.

4. Repeat this process to include these attributes:

Attribute	Attribute Name	Start (row,column)	End (height, width)
SURNAME	LastName	5,17	1,12
FIRST NAME	FirstName	5,44	1,7

Attribute	Attribute Name	Start (row,column)	End (height, width)
REQUEST TYPE	RequestType	9,22	1,1
ACCOUNT	AcctNum	9,35	1,5

5. Click **Apply**.

Next, add these same attributes to the **NameSearchResults** entity:

TO ADD THE ATTRIBUTES:

1. To navigate to the **NameSearchResults** entity, on the **Main** entity, click the **SURNAME** input field and type **w** (uppercase). Then press **Enter**.
1. A Confirm Operation dialog box displays listing the commands needed to navigate to the **NameSearchResults** entity.
2. Click **Approve**.


Repeat the steps to add the same four attributes (**LastName**, **FirstName**, **RequestType**, **AcctNum**) on the **NameSearchResults** entity.

Click **Apply** and save your model.

Create Operations for Host Navigation

Operations are typically used to navigate between entities, so they have an origin entity and a destination entity. You've already configured the Design Tool to auto-generate operations as you move through the host application. When you navigated from the **Main** entity to **NameSearchResults**, you approved the operation `[ToNameSearchResults]{.uiElements}`.

So far, you have only navigated forward through the application. To move backwards by selecting the prior entity you need to create a return operation. This ensures entities can navigate both forwards and backwards in a model.

Open the drop-down list of available entities. The icon preceding the **Main** entity selection  **Main** indicates the home entity. A red house indicates that you have not defined operations to access this entity from your current location.

TO CREATE A RETURN OPERATION:

Click in the Terminal Window and press **Enter**.

In the Confirm Operation dialog box, review the new operation, **ToMain**, and then click **Approve**.

To review the navigation operations, open the drop-down list in the Entity box. Both entity icons are now available. Select each entity and view the entity's Operation tab.

Tip

If you click random keys while navigating, each key press is recorded in the operation. If you see unnecessary operations, use the Delete button to remove them from your operation command list.

Create Recordsets for Scrolling Data

Recordsets handle scrolling or tabular data. You create recordsets to manage dynamically changing data that spans several host screens. In the Design Tool, you create a recordset by defining the:

- Position and layout of the recordset on the screen
- Methods for scrolling through the recordset data
- Fields of data in the recordset

DEFINING THE POSITION AND LAYOUT OF THE RECORDSET

You are going to create a recordset from the data displayed in the list of records on the NameSearchResults entity.


Select the **NameSearchResults** entity and click the **Recordset** tab.

2. With the mouse, select the full scrollable area on the screen. Do not include the column titles when you select the scrollable region.

2.

ACCT	SURNAME	FIRST	MI	TTL	ADDRESS	ST	LIMIT
20000	WAGY	MARK	T	MR	116 BURNSIDE ST.	N	1000.00
20008	WARD	AARON	M	MR	32 MAIN ST.	N	4000.00
20003	WARNER	JESSICA	S	MS	170 WARNER ST.	N	2000.00
20001	WATERS	JOE	P	SIR	1634 15TH AVE W.	N	4000.00
20007	WEBSTER	JOHN	M	MR	342 N 87TH ST.	N	2000.00
20004	WILSON	MARIA	E	MRS	634 E. 17TH ST.	N	3000.00

Figure 10 -- The scrolling region of this screen is selected. When selected, the lines around this area of your screen are green

3. Click **New Recordset** . The **Position** tab should show the **Top** of the selection at Row 17, Column 2, and the **Bottom** at Fixed row, Row 22.

By default, the recordset is named `Recordset_1`. Change the name to `AccountList` in the **Name** box and then click **Apply**.

DEFINING THE SCROLLING OPERATION THROUGH THE RECORDSET DATA

You view the recordset data by using terminal keys to page down and scroll through the data. Navigational operations are defined within each recordset.

ADDING A PAGE-DOWN OPERATION

You'll need to define a page-down operation for the **AccountList** recordset in the **NameSearchResults** entity.

TO DEFINE A PAGE-DOWN OPERATION FOR THE ACCOUNTLIST RECORDSET:

Select the **Operation** tab of the **NameSearchResults** entity.

To record your operation, from the **Model** menu, choose **Record > Start Recording**.

Click in the Terminal Window and press the Page Down key, which is mapped to host key PA2.

Click **Model > Record > Stop Recording**.

In the **Stop Recording** dialog box, rename the operation from `ToNameSearchResults` to `PageDown` and click **Save**.

6. In Step 3, several additional commands may have been recorded. This operation only requires these two commands:

```
CheckOperationConditions
TransmitTerminalKey rcIBMPA2Key
```

6. If other commands are listed, delete them.
7. Click **OK** to save and close the Operation Edit dialog box.

USING THE PAGE-DOWN OPERATION

After you create the recordset operation, you must tell Host Integrator when to use it. The Recordset Operations dialog box contains a predefined set of procedures that the Design Tool uses for specific operations. A page-down procedure displays an entire page of new data. You need to associate a specific operation with this page-down procedure.

TO ASSOCIATE A SPECIFIC OPERATION WITH A PROCEDURE

From the **Recordset** tab, click the **Operations** button at the bottom left of the tab.

On the Recordset Operations dialog box, open the **PageDown** drop-down list.

Select your PageDown operation from the list of options.

Click **Close**, and then click **Apply**.

TERMINATING THE PAGE-DOWN OPERATION

Next, you need to communicate to Host Integrator when to stop scrolling through the NameSearchResults data. To do this, you first create a pattern so Host Integrator recognizes that the end of the data has been reached. You can often use a comment such as *END OF DATA* for this.

Unfortunately, the CICS application does not display an end-of-data comment. Instead, you can configure Host Integrator to recognize the absence of the text, "THERE ARE MORE MATCHING NAMES," as an indication that it has reached the end.

TO CREATE THE PATTERN

On the **NameSearchResults** entity, open the **Pattern** tab.

Click in the Terminal Window and press **Enter**.

In the **Surname** field, type **W** (uppercase W), and press **Enter**.

Select the text: **THERE ARE MORE MATCHING NAMES** and then click the **New Pattern** button.

Rename the pattern to **MoreNames**.

Clear the **Use in entity signature** check box.

Select **Screen properties not present**.

Click **Apply**.

Now that you have created the pattern, you must tell Host Integrator to use the MoreNames pattern to signal the end of the PageDown operation.

TO SIGNAL THE END OF THE PAGEDOWN OPERATION

Open the **Recordset** tab and click the **Termination** button at the bottom of the tab.

Open the **Scroll Down** tab, clear the **Scroll operation results in same recordset data** check box in the **Scroll termination criteria** pane.

Select **Screen contains pattern**, and then from the drop-down list, select the **MoreNames** pattern.

To configure the **Exclude** records, on the last screen option select **Read until __ blank records are found**. Then accept the default setting of **1**.

Click **Close**, and then **Apply**.

You can now successfully traverse and terminate the AccountList recordset.

Creating Recordset Fields


A field is a single piece of data in a recordset. Defining a field on a recordset is similar to defining an attribute on an entity. For the CICSAccts application, you need the customer's first and last name, middle initial, title, and address, as well as the reason and charge limit on their account.

TO ADD FIELDS TO THE ACCOUNTLIST RECORDSET

1. From the **Recordset** tab, open the **Fields** tab.
2. In the Terminal Window, select **account number 20000** on the first row of data (the first row of data is used by default to define what data is contained in this column):

2. 

Figure 11 -- Defining the AcctNum field

3. On the **Fields** tab, click **New Field** .
3. A new field called `Rfield_1` is created.
4. Rename the new field to `AcctNum`.
5. Select the entire surname field on the first row of the recordset. Then click the **New Field** button and rename it to `LastName`.

5. 

Figure 12 -- Defining the Surname field

6. Use the data below to add and define the remaining fields.

Select	Default Name	Rename to	Start	End
WAGY	RField_2	LastName	9	20
MARK	RField_3	FirstName	23	29
T	RField_4	MiddleInitial	32	32
MR	RField_5	Title	35	38
116 BURNSIDE ST	RField_6	Address1	67	67
N	RField_7	Reason	67	67

Select	Default Name	Rename to	Start	End
1000.00	RField_8	ChargeLimit	72	79

7. Click **Apply** and save your model.

Create Tables and Procedures for Abstraction Level Queries


You have now created a model of your host application that defines host screens as entities, screen data as attributes, and tabular data as recordset fields. You have also created operations that describe how to navigate from one entity to another. At this stage of the process, the model fully describes the host application layout.

Tables and procedures are functionally interlocked; tables are populated with data by a procedure, and you can access the unstructured data in a way that resembles a structured database table.

Because Host Integrator tables do not contain data, but rather "organize" host data into a database-like view, the only way to access abstracted table data is through a procedure. Procedures define how Host Integrator locates, retrieves, updates, inserts, or deletes data when it fulfills a request from the client application.

The Web application developer needs to know only which procedure to run, along with the appropriate inputs and outputs; there's no need to know how the host application works. For more information, see [Tables Overview](#).

TO CREATE A TABLE

1. From the **Model** menu, click **Tables**.
2. On the Introduction dialog box, select **Don't show me this dialog again** and click **Finish**. You will not be using the Table Wizard to create this table.
3. On the **Tables** dialog box, click **New**. On the **Create a New Table or Procedure** dialog box, select **Table** and click **OK**.
4. In the **Name** field, change Table_1 to `Accounts`.
5. In the **Description** field, type `Table of account information`.
6. Click **Insert Column**  six times, to create Column 1 through Column 6. Columns identify the attributes and recordset fields that make up the table.
7. Using the following chart, rename Column_1 through Column_6. For example, rename Column_1 to `LastName` and so forth. You do not need to enter data in any of the other fields.

Default Name	Rename to
Column_1	LastName
Column_2	FirstName
Column_3	MiddleInitial
Column_4	Title
Column_5	Address1

Default Name	Rename to
Column_6	AcctNum

8. Click **Apply** to save the **Accounts** table.

Create Procedures to Retrieve Data

Procedures retrieve data and populate tables.

TO CREATE A PROCEDURE THAT RETRIEVES THE ACCOUNT INFORMATION

1. In the Tables dialog box, click **New** and select **Procedure** (using the Procedure Wizard). Then click **OK**.
2. In the **Name** field of the Procedure Wizard, type `SearchByName`.
3. In the **Description** field, type `Search by name`. Then verify that the **Procedure type** option is `Select` and click **Next**.
4. Identify the procedure parameter needed to manipulate and return host data and specify which attribute or field to use to filter the records:
 4. a. In the Filter Parameters dialog box, go to the `LastName_ row` and click in the empty **Write parameter to** cell.
 4. b. Open the drop-down list and select **Main.LastName** to write the filter parameter to the last name attribute on the **Main** entity.
 4. c. In the **Required** column, confirm that **Required for LastName** is selected. You need to enter a last name to proceed.
 4. d. Click **Next**.
5. Choose the parameters to be returned as output. The data to be returned is in the **AccountList** recordset on the **NameSearchResults** entity.
5. In the Output Parameters dialog box, click in each **Read parameter from** cell and select the following data sources from the drop-down menu:

Parameter	Data Source
LastName	NameSearchResults.AccountList.LastName
MiddleInitial	NameSearchResults.AccountList.MiddleInitial
Address1	NameSearchResults.AccountList.Address1

Parameter	Data Source
AcctNum	NameSearchResults.AccountList.AcctNum

5. This maps the procedure fields with the data in the **AccountList** recordset on the **NameSearchResults** entity.

6. Click **Next**.

7. View the Summary, and click **Finish** to return to the Tables dialog box.

Review the information in the Tables dialog box for the `SearchByName` procedure. You can expand **Accounts** to view the procedure.

Verify that the **Home** entity is **Main**. This table begins and ends at the **Home** entity. Each procedure must have a home entity to ensure that queries and procedures begin from a known point in the host application.

10. Click **OK** and save your model.

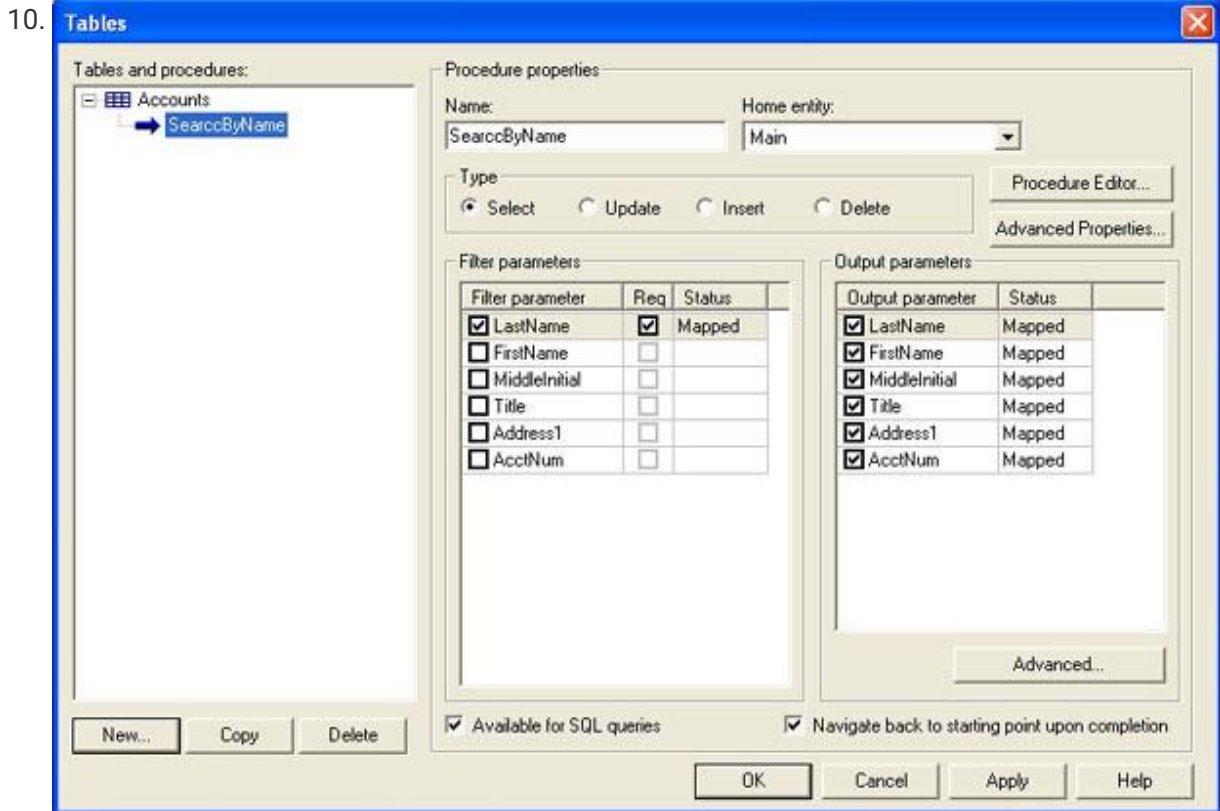


Figure 13--SearchByName procedure details

Test and Deploy the Model

There are multiple utilities you can use to test your model before you deploy it. In this tutorial we are going to use two debug utilities, Procedure Test and SQL Test.

Because the model you created is a simple, two-entity model, comparing the entities and identifying the traversal path is easy. But consider a model that contains hundreds of entities with a multitude of attributes and complicated navigation, and you can understand how all of these testing utilities become important debugging tools:

[Validator](#) checks a model's completeness.

[Signature Analyzer](#) ensures each screen is uniquely identified.

[Navigator](#) reviews the model's traversal path.

[Procedure Test](#) confirms procedures run correctly.

[Model Debug Messages](#) diagnoses problems such as synchronization with the host by showing the model's behavior while interacting with a terminal datastream.

[SQL Test](#) verifies that SQL results are correct.

TO RUN A PROCEDURE TEST

From the **Debug** menu, click **Procedure Test**. Since you have only one table and one procedure, they are selected by default.

In the **Procedure Filters** box, type **W** (uppercase W) in the **Value** field (associated with the **LastName** filter.)

Click **Execute**. Seven entries should be returned and displayed in the **Procedure Outputs** box.

Click **Clear** and enter **K** (uppercase K) to test your procedure again.

Click **Execute**. You should see one entry under Procedure outputs.

Click **Close**.

TO RUN AN SQL TEST

From the **Debug** menu, click **SQL Test**.

2. In the SQL statement box type the following SQL command:
2. `SELECT * FROM accounts WHERE lastname LIKE W`
3. Click **Resolve** to resolve the SQL statement to a procedure.
4. Click **Execute**.
4. Seven records are returned and displayed in the **Output recordset** box.
5. Click **Close** and save the model.

The Host Integrator supports a subset of the SQL 92 standard for SELECT, UPDATE, INSERT, and DELETE statements. Some of these commands are used in a slightly different manner. For more details, see [SQLSyntax](#).

TESTING WEB SERVICES

In Host Integrator, Web services are automatically provided by the session server as an embedded SOAP stack after a model package is deployed using the Design Tool. As the provider of a Web service, you publish the WSDL document to give developers access to your Web service. With the WSDL-generation URL, developers can use utilities to generate specific files to locate and consume the Web service.

This location lists the available Web services WSDL documents:

```
http://<session server>:9680/vhi-ws
```

Optionally, you can access the WSDL for a specific model by using the model name. For example:

```
http://<session server>:9680/vhi-ws/model/<model name>?wsdl.
```

Testing your service is an important step before you run your process in a production environment. Testing is an easy two-step procedure:

After you deploy the model to the Host Integrator Server, a message box informs you that the model is successfully deployed.

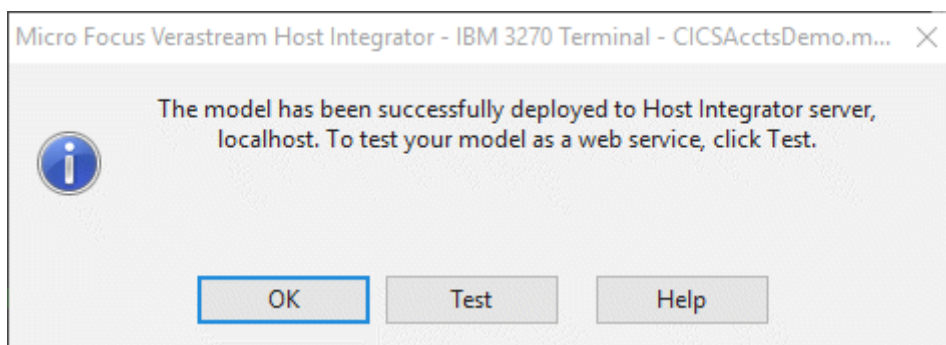


Figure 14 -- Deployment successful message

1. Click **Test** to launch the Web Services Explorer, a browser-based test tool, where you can test the Web service.
2. In the **Actions** view of Web Services Explorer, enter the parameters of the **WSDL** in order to test the service and click **Go**.

Tip

To test Web services that are protected by WS-Security, Basic Authentication, or HTTPS, use a third-party development environment, such as Microsoft Visual Studio.

Although the way that Web services are implemented in Host Integrator is beyond the scope of this tutorial, you can read more about them in [About Verastream Web Services](#).

Extending Your Model with Event Handlers

Event handling extends the capabilities of Host Integrator models by interrupting the interpretation of a model and turning control over to user-supplied procedural code. For example, you can use event handlers to:

- Convert cryptic host application codes to user-friendly descriptions
- Convert formats for currencies or dates
- Secure access to sensitive data
- Create other functions that might otherwise require custom client-side programming

You can also use event handlers to increase Host Integrator error-handling capabilities. For example, you can add an event handler at the point an error occurs and then implement event-handler code that intercepts the error, takes control, and corrects the error.

Although event handlers are beyond the scope of this tutorial, you can read more about this powerful feature in [About Event Handlers](#).

About Deploying Host Application Models

To move into production testing, you need to deploy your model to the Host Integrator server that handles the communication between your custom application and the model, and between the model and the host application. When deploying models, use either:

Design Tool Deployment - If you are deploying one model to one server (local or remote), with one model configuration, use the deployment options on the Design Tool's File menu.

Command Line or Script Deployment - If you are deploying a model to one or more production servers, use deployment commands from a command line or in a script. This enables you to automate the model's deployment to each Host Integrator server.

For more information about using these different deployment methods, see [Deploying Models](#).

Deploying MyModel.modelx

You are using your workstation as the development, testing, and production environments; therefore, you will deploy your model to the local session server, localhost:

Open your model in the Design Tool.

To deploy your model, click **File > Deploy to Local Server**.

When the dialog box informs you that you have successfully deployed your model, click **OK**.

Close the Design Tool.

3.4.5 Creating a Web Builder Project and Generating a Web application

After you have completed, tested, and deployed the model, you are ready to create a project from the Host Integrator model. For the CICSAccts example, you will use Web Builder to generate Web pages from the model.

Web Builder generates projects ranging from simple screen-based rejuvenation to full procedure-based integration. A rejuvenation Web application uses the model's entities, attributes, and recordsets as the basis for creating the Web front end. An integration project uses procedures to create a Web application or component interfaces, JavaBeans, and .NET class libraries.

You can use Web Builder to create an HTML 5 Web application that can be used either *out of the box* or customized by your Web application developer. Screen-based rejuvenation may be adequate for some projects, but, in this tutorial, you will use the model's procedures to integrate the CICSAccts application into a more sophisticated application.

Note

The steps you follow to generate a project are the same on either a .NET or Java platform. Select the type of project you want to build based on your platform and whether you want to generate a Web application or component interface.

To build an HTML 5 Web application For CICSAccts


Open the Design tool and then choose **File > Web Builder**.

In Web Builder, click **New** to create a new project.

Select **HTML 5 Web Application** as the type of project to create.

Select **MyModel** as the Host Integrator model name.

Enter `MyModel_Application` as the project name.

6. Click **Properties** and then **Build**.
6. When the Build Project dialog box displays a **BUILD COMPLETED** message, your Web application automatically runs. (You can run the project manually in the Web Builder by selecting your project and clicking **Run**.)
7. When the Web application displays, click  to open the available Procedures.
8. Click **Search by name**, enter a **W** in the **LastName** text box, and then click **Execute**.
8. The resulting Web page looks like the one below. Close the application and Web Builder.

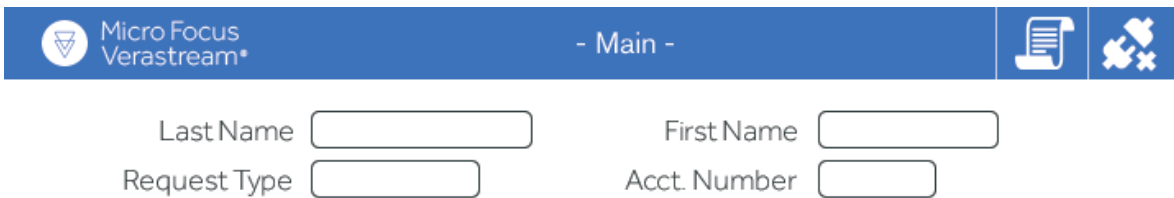
8. 

Figure 15--HTML 5 Web client generated by Web Builder

Deploying the Project to the Enterprise

In a work environment, after you have developed the new Web or client/server application and the application is tested, you can deploy it to the enterprise using the customer's preferred hardware platform and network configuration.

For specific details on deploying a project, review:

[Deploying Models](#)

[Deploying Web Services](#)

3.4.6 Congratulations!

Using Verastream Host Integrator, you have created an easy-to-use Web application that accesses the CICSAccts host application. You have:

- **Defined the business needs for the project.**

You need a clear understanding of the business problem to be solved by your solution.

- **Developed the model requirements and mapped the data.**

In the "real" world, you would work with system administrators to understand the user's computing environment, programmers to learn what programming languages are preferred, and host application users to understand how the host application functions. This would lead to setting the model requirements and mapping the host application data. During this exercise, you were given the model requirements and mapping information.

- **Built and deployed the model.**

Using the Design Tool, you created a model of the host application, which was automatically deployed to your local Host Integrator Server along with the automatically generated Web service. You have:

- Created patterns to uniquely identify each entity

- Identified operations to navigate through the host application

- Configured attributes for data entry and retrieval

- Used recordsets to handle scrolling data

- Created tables of host data

- Developed procedures to query and retrieve data from those tables

- **Created a Web project using Web Builder.**

You generated a Web application to test the procedures in your model, and delivered this Web application as a simple front-end to the host application. You also have access to a Web service to deliver to Web programmers for a more complex application that integrates host data through the model.

- **Deployed the project to the enterprise.**

Because you did not move the CICSAccts application into production, you didn't have to perform this step.

Although your project will be more complex than this straightforward example, the steps you perform and the utilities you use are the same each time you create, deploy, and implement a Verastream Host Integrator model.

4. Using the Design Tool

4.1 The Modeling Process: Getting Started

To begin the modeling process:

Open the Design Tool.

On the File menu, click New to specify all the necessary settings in the New Model dialog box. When you have finished supplying the necessary settings, provide a model name, and click OK.

You should now be connected to your specified host.

Now, you're ready to add an entity to your model.

4.1.1 Creating a Model

With the Design Tool, you create a model of the host application. The model consists of a main model file and several supporting files that are located in the `<documents>\Micro Focus\Verastream\HostIntegrator\models\<your model folder>`. The supporting files saved in your models directory vary depending on whether you are saving your model as a `.model` file or as a `.modelx` file.

For more information:

[Adding entities](#)

[Adding patterns](#)

[Adding attributes](#)

[Adding recordsets](#)

[Adding operations](#)

[Working collaboratively](#)

[Creating tables](#)

[Creating procedures](#)

[Using event handlers](#)

[Deploying models](#)

You can use Web Builder to quickly and easily generate a web application or component interface such as a web service or JavaBeans, based on the table of a host application model, or based on a host application screen layout.

More information

[Learning to Use Host Integrator](#)

[Host Integrator development process](#)

4.1.2 Configuring a Model

The Design Tool provides several ways to configure unique properties and settings that enhance the capabilities of the host application model. You can use the pre-configured settings files provided with Host Integrator to start with the defaults appropriate for the terminal session you are modeling. Settings include:

[Variables](#) - which allow attributes to be accessible from all parts of the Host Integrator

[Model preferences](#) - to customize default settings for the Design Tool

[Model properties](#) - to configure settings for the current model

[Advanced entity properties](#) - to configure settings for the current entity

[Character mode](#) - to configure settings for character mode terminal types (for example, VT)

[Events](#) - to configure synchronization for character mode terminal types

[Command lists](#) - to create a login, a logout, or a move cursor command list

[Descriptions](#) - to add descriptions in exported documentation created for the model.

More information

[Model examples](#)

4.2 Work Collaboratively

When you are building your host application project files it is often important for multiple people to contribute to a project. Configuration management software, such as Git, provides your team with the ability to manage and merge your project files efficiently. Because Host Integrator saves your project files in an XML format, making them accessible to multiple developers, it is easy to work collaboratively.

Each host application project contains a number of files that you need to keep together in order to ensure the project's uniqueness. Files are generated and saved in a directory that uses the same name as the model you are using. The main model file, `.modelx`, uses that same name. It is important to keep all the project files together and preserve their folder structure.

The `<Model name>` directory, which, by default, is created in `my documents\Micro Focus\Verastream\HostIntegrator\model`. Each Model name directory contains the following files and directories:

- The `<model name>` `.modelx` and `modelx_1.xsd` files.

- The `<entity name>` `.entityx` and `entityx_1.xsd` files, in the entities subfolder.

- The `<table name>` `.tablex` and the `tablex_1.xsd` files, in the tables subfolder.

- The Scripts directory.

In particular, `build.xml`, the `\src` subfolder, and the `\lib` subfolder are required for event handler maintenance.

In previous versions of Host Integrator, project files were saved with a `.model` extension. By default your project is saved using its existing extension. Using the Save As option you can choose which format to use. If you choose to save your file as a `.model` file, the directory contains these files:

- The `<model name>` `.model` file.

- The `.snapshot` file.

- The Scripts directory.

In particular, `build.xml`, the `\src` subfolder, and the `\lib` subfolder are required for event handler maintenance.

4.2.1 Using source control to manage and merge changes

Using the multiple plain text files generated in the Design Tool and a tool, such as Git, different developers can work on the different files needed to build your host application project.

4.2.2 Things to remember

It is always better to make changes to the model using the Design Tool. In the Design Tool all objects are available to copy from the user interface and every change is validated to make sure that the model stays consistent.

Because the objects are available in XML format, you can open the modelx, entityx and tablex files in an editor. Make sure that the editor you choose is capable of using the associated .xsd files. The .xsd file checks the syntax and possible element values, but it is up to the developer to check the valid syntax before attempting to run the project.

If your projects are referencing the same mainframe application, you can copy and rename the entire .entityx or .tablex file into the destination model. You can copy entityx or tablex files from one model directory into another model directory. If the name in the destination model already exists, you can simply give the new entityx or tablex files a new name. The new names will be available as entities or tables in the Design Tool.

- One advantage of an editor is the ability to globally copy and rename an object. You can copy information within the same modelx file to create a number of variables that use the same properties or within a file (modelx, entityx, or tablex) of one project, you can copy a complete definition of an object, such as a variable, or copy a part of an object such as a

pattern or a recordset, belonging to an entity, or copy one or more columns that are part of a table.

For example, it is easy to copy the information contained within the variable tags, and then add this variable to another project by pasting it after an existing variable in your XML file.

You must rename the variable to a unique value:

```
<Variable>
  <Name>newUserID</Name>
  <InitOption>VarInitUninitialized</InitOption>
  <VarKind>VarKindSetting</VarKind>
  <Attributes>
    <VarAttrsRead/>
    <VarAttrsWrite/>
  </Attributes>
  <Setting>HostUserName</Setting>
</Variable>
```

4.2.3 Importing model elements

With this option, developers can work separately on different sections of the same model, import sections into the destination model, and reuse model elements as often as needed. You can import elements from older formatted models into the current format and vice versa. You are not restricted by the different versions of existing project files. There is complete documentation on this option available. See [Importing Model Elements](#).

4.2.4 Optional Files

The `Recordings` directory, which is where the Host Emulator trace files are stored is optional. If you do not intend to load and run a model recording in the Host Emulator, this directory is not needed.

Note

If you want to save the settings for this model so that it will be the basis for creating other models, select Save Settings As. The `.dtool` file you create can include Design Tool-specific configuration information such as window size, colors, keymapping, and preferences, as well as information about the host application and connection settings.

More information

[Validating host application models](#)

[Troubleshooting tips and techniques](#)

4.3 Setting Up a Connection

4.3.1 Design Tool Connection Settings

Entity Window Options

Entity Window Options

The Entity window contains all of the settings used to define the host screens that make up the model. The Entity box contains all of the currently defined entity names with visual indicators to indicate whether each entity listed is "reachable" from the current location via dynamic traversal or navigation.

When you are in offline mode, all screens are considered reachable and display a green icon since the screen shots are loaded from a .snapshot file which has also recorded the navigation of the model. Next to the Entity box, there are three buttons that allow a user to add a new entity, delete the current one, or view the Advanced Entity Properties dialog box.

There are five tabs available from the Entity panel:

- Pattern tab - contains all of the settings used to define patterns on a selected entity.

- Attribute tab – contains all of the settings used to define attributes on a selected entity.

- Operation tab - use this tab to edit or define operations on a selected entity.

- Recordset tab - contains all of the settings used to define recordsets on a selected entity.

- Cursor tab - contains cursor movement settings to be used with character-mode hosts. (If you are not working with a character-mode host, Cursor tab controls are unavailable.)

See [Entity Settings](#) for detailed information.

Terminal Window Menus

The Terminal window contains all of the menu items that can be used to configure host screens that make up the model.

The following menus are available on the Terminal window:

- File

- Edit

- Connection

- Settings

- Events

- Model

- Debug

- Window

- Help

See [Terminal Settings](#) for detailed information.

4.3.2 Entity Settings

The Entity window contains all of the settings used to define the host screens that make up the model. The Entity box contains all of the currently defined entity names with visual indicators to indicate whether each entity listed is "reachable" from the current location via dynamic traversal or navigation.

When you are in offline mode, all screens are considered reachable and display a green icon since the screen shots are loaded from a .snapshot file which has also recorded the navigation of the model. Next to the Entity box, there are three buttons that allow a user to add a new entity, delete the current one, or view the Advanced Entity Properties dialog box.

There are five tabs available from the Entity panel:

[Pattern](#)

[Attribute](#)

[Operation](#)

[Recordset](#)

[Cursor](#)

See [Adding Entities to a Model](#) for information on how to add and define your entity.

4.3.3 Using SSH: Overview

You can configure SSH connections when you need secure, encrypted communications between a trusted host and your PC over an insecure network. SSH connections ensure that both the client user and the host computer are authenticated; and that all data is encrypted. Passwords are never sent over the network in a clear text format as they are when you use other protocols, such as Telnet.

[Data Encryption Standards](#)

[Data Integrity](#)

[Digital Signatures](#)

[How does SSH Work?](#)

[SSH Authentication Options](#)

[Using Model Variables for SSH Authentication](#)

[Public Key Authentication](#)

[Enter Username and Password](#)

Data Encryption Standards

Encryption protects the confidentiality of data in transit. This protection is accomplished by encrypting the data before it is sent using a secret key and cipher. The received data must be decrypted using the same key and cipher. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. You can use the cipher list on the Advanced VT SSH dialog box to specify which ciphers the SSH connection should use.

Verastream Host Integrator supports the following data encryption standards:

AES (also known as Rijndael) (128-, 192-, or 256-bit) CBC mode and CTR mode

TripleDES (168-bit) CBC mode

Blowfish CBC

Cast 128

arcfour 128

arcfour

Data Integrity

Data integrity ensures that data is not altered in transit. SSH connections use MACs (message authentication codes) to ensure data integrity. The client and server independently compute a hash for each packet of transferred data. If the message has changed in transit, the hash values are different and the packet is rejected. The MAC used for a given session is the MAC highest in the client's order of preference that is also supported by the server.

Verastream Host Integrator supports the following MAC standards:

SHA256

SHA1

SHA512

MD5

RIPMD160

RIPMD160 openssh.com

SHA1-96

MD5-96

If your SSH server on the host supports it, you always have the option of selecting **None** when choosing a MAC or data encryption standard.

Note

The values for both the MAC and data encryption standards that you can select are dependent on whether the **Only show FIPS validated values** is enabled. This option filters the values available.

Digital Signatures

Digital signatures (user key and host key algorithms) are used for public key authentication. The authenticating party uses the digital signature to confirm that the party being authenticated holds the correct private key. The SSH client uses a digital signature to authenticate the host. The SSH server uses a digital signature to authenticate the client when public key authentication is configured.

Verastream Host Integrator supports the following digital signature algorithms:

ECDSA

RSA

DSS

EdDSA

The known_hosts file

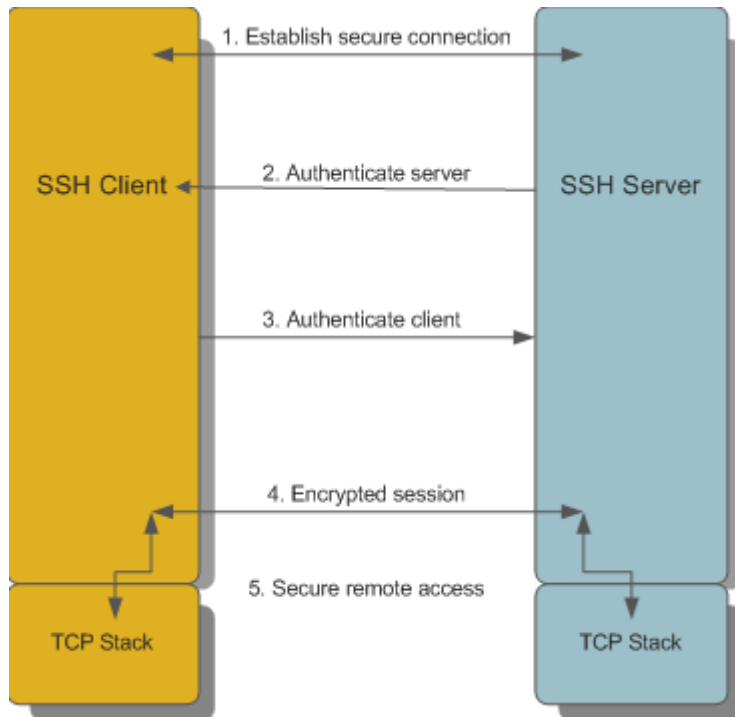
Every time an SSH client connects to a host, it stores a host key for that host. These stored host keys are referred to as known host keys or just known hosts. In OpenSSH, these known hosts are stored in `/etc/ssh/known_hosts` and in `~/.ssh/known_hosts` in each user's home directory. VHI uses the known_hosts file to verify the identity of the server the model is going to connect to.

You can add the remote host's public key to the user's known_hosts file using either SSH or VHI.

By default, VHI uses the known_hosts file, located in directory `~/ .ssh` . You can specify a different file by using predefined model variable `KnownHosts` . See [Using Model Variables for SSH Authentication](#).

How does SSH work?

These are the basic steps involved in creating a SSH channel to transmit data securely. It is assumed that the SSH Server is trusted and present in the known_hosts file.



1. Establish a secure connection

1. The client and server negotiate to establish a shared key and cipher to use for session encryption, and a hash to use for data integrity checking.

2. Authenticate the server

2. Server authentication enables the client to confirm the identity of the server. The server has only one chance to authenticate to the client during the authentication process. If this authentication fails, the connection fails.

3. Authenticate the client

3. Client authentication enables the server to confirm the identity of the client user. By default, the client is allowed multiple authentication attempts. The server and client negotiate to agree on one or more authentication methods.

4. Send data through encryption session

4. Once the encrypted session is established, all data exchanged between the SSH server and client is encrypted.

5. A channel is created and a terminal emulation using the terminal type specified in the configuration dialog box is started

5. Users now have secure remote access to the server and can execute commands through the secure channel.

More information

[Configuring a VT session](#)

[Using Model Variables for SSH Authentication](#)

[Advanced VT SSH Options](#)

SSH Authentication Options

You can enable authentication in two ways; interactively using the Design Tool or using model variables in the Session Server and Design Tool.

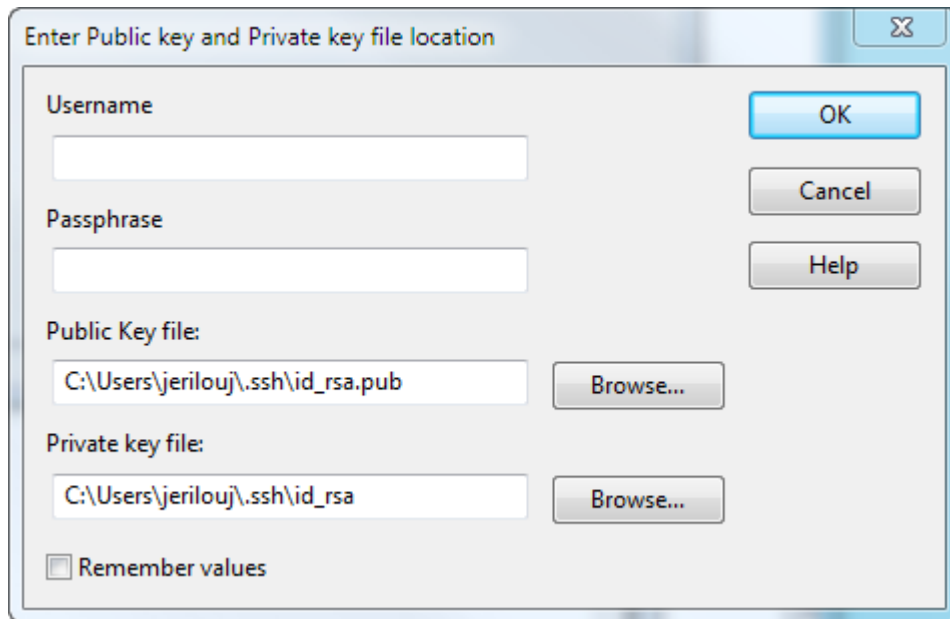
To authenticate using model variables, see [Using Model Variables for SSH Authentication](#). This is the preferred option.

If you have not configured authentication using model variables in the Design Tool, you are prompted to specify the authentication values:

- Password

Specify the login username and password for that user on the SSH server host. The password is sent to the host through the encrypted channel.

- Public key and Private key



Specify the username, passphrase, and location of the public and private key files.

Relies upon public/private key pairs. Public keys and private keys are pairs of cryptographic keys that are used to encrypt or decrypt data. Data encrypted with the public key can only be decrypted with the private key; and data encrypted with the private key can only be decrypted with the public key.

To configure public key authentication, each client user needs to create a key pair and upload the public key to the server. If the key is protected by a passphrase, the client user is prompted to enter that passphrase to complete the connection using public key authentication. Public keys are not sensitive information and may be known to anybody, whereas the private key is protected very carefully by a strong passphrase.

Caution

If you are using utility `ssh-keygen` to create the private and public key for SSH, be aware that the newer `ssh-keygen` versions default to an OpenSSH format to generate private keys. This is not supported by VHI. The public/private key pair must be in PEM format. Verify that the header of the private key contains the text, `RSA PRIVATE KEY`. You can convert keys with OpenSSH private key format using `ssh-keygen` to the old PEM format. Use the command `ssh-keygen -m PEM -t rsa` to generate the files `id_rsa` and `id_rsa.pub` in the correct format.

- SSH Agent

Specify the username.

SSH agent is a program to hold private keys used for public key authentication (ECDSA, RSA, DSA). Host Integrator connects to the agent for authentication.

The SSH agent:

Stores keys securely in encrypted form

Enables you to only specify a username when connecting using SSH. Host Integrator connects to the SSH agent, and the agent takes care of the needed authentication. You do not have to specify a password, key, or passphrase.

If you plan to authenticate using public keys, before you configure Host Integrator:

Verify that the SSH agent you are using is available and configured on either your Windows or Linux system

Start the SSH agent and specify the location of the private key file. When the keys require a passphrase that should be entered as well. The agent is now running as a daemon at the background.

Note

The key data must be in OpenSSH format. Remove any new lines, comments, or other data. Whatever tool you use to create the private key, must be used to export the key to OpenSSH format. If the public key is in SSH2 (SECSH) format, run the following OpenSSH command to convert the certificate from SSH2 to OpenSSH:

```
ssh-keygen -i -f ~/.ssh/id_dsa.ssh2.pub > ~/.ssh/id_dsa.pub
```

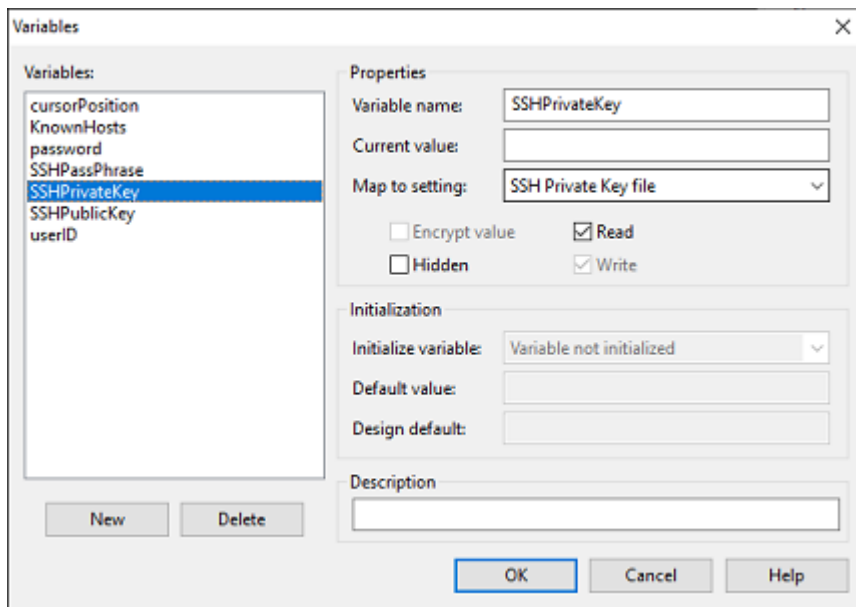
On each of the authentication dialog boxes you can enable the option Remember values. When selected, the values you entered are assigned to the associated model variables.

Using Model Variables for SSH Authentication

You can specify authentication credentials using model variables. Model variables are placeholders for data and are very useful when specifying fixed values, such as host user IDs and passwords.

A list of default variables are listed when you create a new model. If you selected **Remember values** when setting values using the Design Tool, the values are assigned to the associated model variable.

Special character `~` can be used to specify the home directory, independent of the platform. For example, `~/.ssh/id_rsa.pub` refers to the public key in the users `.ssh` home subdirectory.



Variable name	Map to setting
userID	Host User Name
password	Host password
SSHPublicKey	SSH Public key file
SSHPrivateKey	SSH Private key file
SSHPassPhrase	SSH passphrase for the private key

Variable name	Map to setting
SSHKnownHosts	File with public keys of SSH servers to connect to

SETTING MODEL VARIABLE VALUES IN THE ADMINISTRATIVE CONSOLE

Model variable values that are set using the Design Tool are stored in the model file. To remove this sensitive data from the model file and store it on the Session server:

After setting authentication variable values in the Variables dialog box, test the model.

If successful, remove the values for the password, username, and passphrase variables.

Deploy the model to the Session server.

In the Administrative Console, open the Properties page for the session pool associated with the model and re-enter the model variable values. If you need to provide a unique model variable value for each session in the pool, create a model variable list that contains a set of values for the model variables in the model on which your session pool is based.

If the Session server is not able to connect because the values are not set, an error message is written to the Session server log file: `An error occurred in communications - SSH password authorization failed (username/password).`

Public Key Authentication

Enter your VT host credentials to log on to the host. This dialog box appears if you are using SSH public key authentication to connect.

- Username - Enter your VT host username to log on to the host.
- Passphrase - Enter the passphrase to decrypt your private key. If your private key is not encrypted, leave this field blank.
- Public Key file - Enter or browse to the location of your public key file.
- Private Key file - Enter or browse to the location of your private key file.

Enter Username and Password

You must supply authentication credentials to establish a connection to the host.

Supply a username to log on to the host.

For the connection to be established, supply the needed password.

This dialog box appears if you have not specified values for the model variables.

4.3.4 Using SSL/TLS

Telnet Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols are available for 3270 and 5250 session types, and Telnet Extended SSL/TLS support is available for 3270 session types. These Telnet options help you implement a connection between a host requiring this form of security and the Host Integrator session server. To implement a secure connection between the client and the Host Integrator session server, use the security options in the Administrative Console.

If SSL is implemented on the host for encryption purposes only, select the SSL/TLS checkbox.

SSL 3.0, TLS 1.0, and TLS 1.1 are no longer supported by Host Integrator, but can be accessed for legacy host connections, if needed. See the Installation Guide topic, [Encrypting Host Connections](#) for more information.

Client Authentication

If the host requires client authentication from Host Integrator, your private key and client certificate must be stored in a file named `certificate.pem`. The file must be in PEM format with the private key first, followed by the certificate chain in chain order.

You must create and store this file in a subdirectory named `securehost`. For example:

- On Windows `\Program Files\Micro Focus\Verastream\HostIntegrator\securehost`
- On Linux `/opt/microfocus/verastream/HostIntegrator/securehost`

If your certificate and private key are in PFX format, you can use the OpenSSL command line utility or other conversion tool to convert it to standard PEM format. For example, [this conversion tool](#).

It is good practice to open the resulting file in a text viewer to verify it is in PEM format with the private key first. PEM certificates are text files containing base64-encoded data and lines such as `"-----BEGIN CERTIFICATE-----"` and `"-----END CERTIFICATE-----"`.

FIPS Validation

To use FIPS 140-2 validated TLS version 1 encryption for SSL support, you must first define an environment variable, `VHI_FIPS = 1`. After this variable is set all SSL/TLS connections will use the FIPS 140-2 Crypto Libraries.

4.4 Configuring Sessions

4.4.1 Configuring a Host Session

Use the Session Setup or the New Model dialog box to connect to a host and specify emulation options. To open the Session Setup dialog box, click Session Setup on the Connection menu.

Note

You can open the Session Setup dialog box if you have not connected it to a host session. Otherwise, this option is unavailable. Use the Connection Properties dialog box to review settings after you are connected.

The options in the Session Setup dialog box vary according to the values you select for session type and transport type. Click one of the following for more information:

[Configuring a 3270 terminal session](#)

[Configuring a 5250 terminal session](#)

[Configuring a VT terminal session](#)

[Configuring an HP terminal session](#)

Once you have set up the options in this dialog box, you can click Connect to connect immediately, or click OK to close the dialog box. If you click OK, you can always connect later. Connection settings can be saved to a settings file (.dtool) with other configuration information.

Configuring a 3270 Terminal Session

The following options are available in the Session Setup or New Model dialog box.

After you connect, you can review these settings in the Connection Properties dialog box.

Options	Description
Session Type	Specifies the type of session to configure.
Model ID	Specifies the terminal (also known as a display station) you want the Design Tool to emulate.

Options	Description
Transport Type	This setting specifies the transport type being used to connect to the host. You can use Telnet, Telnet Extended, SSH, or NS/VT transport types as well as Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols. Telnet is available for 3270,5250, VT, and HP session types, while Telnet Extended is only available for IBM 3270 session types. Choose SSH to connect to a VT host using SSH security protocols. NS/VT is available for HP session types.
Use SSL/TLS	See Using SSL/TLS
Host Name or IP Address	Use this box to identify the host to which you want to connect. You can either select a host from the drop down list or type one into the host name field. When you start the Design Tool without a value for this setting, the Design Tool looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location. When it finds a Hosts file, the Design Tool changes the value of this setting.
Port Number	Specifies the host port or socket number that the Telnet session should use. You can enter any number between 0 and 65,535 in this field. You can configure Port in either the Session Setup or View Settings dialog boxes. The default port number for Telnet is 23 .

Options	Description
Device Name	Specifies the Device name (also known as LU name) to which to connect. Supports up to 32 characters. You can map the Device Name setting to a model variable in the Model menu > Variables. This allows a different setting for each server runtime session. If a model variable is mapped to the Device Name setting, the Device Name configured in Session Setup will be used as a default unless changed by a session pool model variable list or a Host Integrator API SetModelVariable method.

More information

[Using SSL/TLS](#)

[Using SSH](#)

ADVANCED 3270 TELNET AND TELNET EXTENDED DIALOG BOXES

You can configure the following options in these dialog boxes:

Option	Description
Terminal ID	In a 3270 session with a Telnet transport type, this setting overrides the Model ID selected in Session Setup. It allows you to specify a Terminal Model that Host Integrator does not implicitly support (but might work with), such as Fujitsu. For best results, specify one of the Terminal Models available in Session Setup and leave the Terminal ID box empty. When the Terminal ID box is empty, the Design Tool uses the Model ID defined in Session Setup by default. Only specify a Telnet Terminal ID, if after experimenting with other available Terminal Models IDs in Session Setup, you still cannot connect to the host. If you must define a Telnet Terminal ID, specify a string you know is acceptable to the host. Otherwise, you may experience problems connecting to the host and emulation problems after connecting to the host. Typically, these types of problems occur if the host is not configured to recognize the terminal specified in the Telnet Terminal ID string.
Telnet Location	Specifies where the connection originated. Can also be used to provide informational messages to the host from the computer. You are not required to enter anything in this box. Supports up to 260 characters. You cannot change this value while you're connected to a gateway.

Option	Description
Send Keep Alive packets	To become aware of connection problems as they occur, you can configure your model to Send Keep Alive packets.
Keep Alive Timeout	The interval between the keep alive requests sent by the Design Tool. The range of values is 1-9999 seconds, and the default is 600 seconds.

Configuring a 5250 session

The following options are available in the Session Setup or New Model dialog box.

After you connect, you can review these settings in the Connection Properties dialog box.

Options	Description
Session Type	Specifies the type of session to configure.
Model ID	Specifies the terminal (also known as a display station) you want the Design Tool to emulate.
Transport Type	This setting specifies the transport type being used to connect to the host. You can use Telnet, Telnet Extended, SSH, or NS/VT transport types as well as Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols. Telnet is available for 3270,5250, VT, and HP session types, while Telnet Extended is only available for IBM 3270 session types. Choose SSH to connect to a VT host using SSH security protocols. NS/VT is available for HP session types.
Use SSL/TLS	See Using SSL/TLS
Host Name or IP Address	Use this box to identify the host to which you want to connect. You can either select a host from the drop down list or type one into the host name field. When you start the Design Tool without a value for this setting, the Design Tool looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location. When it finds a Hosts file, the Design Tool changes the value of this setting.

Options	Description
Port Number	Specifies the host port or socket number that the Telnet session should use. You can enter any number between 0 and 65,535 in this field. You can configure Port in either the Session Setup or View Settings dialog boxes. The default port number for Telnet is 23 .
Device Name	Specifies the Device name (also known as LU name) to which to connect. Supports up to 32 characters. You can map the Device Name setting to a model variable in the Model menu > Variables. This allows a different setting for each server runtime session. If a model variable is mapped to the Device Name setting, the Device Name configured in Session Setup will be used as a default unless changed by a session pool model variable list or a Host Integrator API SetModelVariable method.

More information

[Using SSL/TLS](#)

[Using SSH](#)

ADVANCED 5250 TELNET DIALOG BOX

You can configure the following options in this dialog box:

Option	Description
Telnet location	Specifies to the host where the connection originated. Can also be used to provide informational messages to the host from the computer. You are not required to enter anything in this box. Supports 260 characters. You cannot change this value while you're connected to a gateway.
Username	Enter your username for the AS/400 Sign On screen if you want to bypass being prompted for it at connection time. You'll also need to enter your password.
Password	Enter your password for the AS/400 if you want to bypass being prompted for it at connection time. You'll also need to enter your username.
Auto SignOn	Select this option to have the transport protocol automatically log you on to the host as soon as you establish a connection in the Design Tool. By default, this check box is not selected.
Send Keep Alive packets	To become aware of connection problems as they occur, you can configure your model to Send Keep Alive packets.

Option	Description
Keep Alive Timeout	The interval between the keep alive requests sent by the Design Tool. The range of values is 1-9999 seconds, and the default is 600 seconds.

Configuring a VT Session

The following options are available in the Session Setup dialog box.

Some of these options are dependent on the transport type you are using, either Telnet or SSH.

Options	Description
Session Type	Specifies the type of session to configure.
Terminal Type	Specifies the terminal (also known as a display station) you want the Design Tool to emulate.
Transport Type	This setting specifies the transport type being used to connect to the host. You can use Telnet, Telnet Extended, SSH, or NS/VT transport types as well as Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols. Telnet is available for 3270,5250, VT, and HP session types, while Telnet Extended is only available for IBM 3270 session types. Choose SSH to connect to a VT host using SSH security protocols. NS/VT is available for HP session types.
Host Name or IP Address	Use this box to identify the host to which you want to connect. You can either select a host from the drop down list or type one into the host name field. When you start the Design Tool without a value for this setting, the Design Tool looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location. When it finds a Hosts file, the Design Tool changes the value of this setting.
Port Number	Specifies the host port or socket number that the SSH session should use. You can enter any number between 0 and 65,535 in this field. You can configure Port in either the Session Setup or View Settings dialog boxes. The default port number for SSH is 22 .

Options	Description
Device Name	Specifies the Device name (also known as LU name) to which to connect. Supports up to 32 characters. You can map the Device Name setting to a model variable in the Model menu > Variables. This allows a different setting for each server runtime session. If a model variable is mapped to the Device Name setting, the Device Name configured in Session Setup will be used as a default unless changed by a session pool model variable list or a Host Integrator API SetModelVariable method.

See [Customizing the VT Terminal](#) for Advanced Telnet and SSH Configurations.

After you connect, you can review these settings in the Connection Properties dialog box.

Configuring an HP Session

On an HP terminal, with the Transport Type set to **Telnet** or **NS/VT**, the following options are available in the Session Setup and New Model dialog box.

After you connect, you can review these settings in the Connection Properties dialog box.

Options	Description
Session Type	Specifies the type of session to configure.
Model ID	Specifies the terminal (also known as a display station) you want the Design Tool to emulate.
Transport Type	This setting specifies the transport type being used to connect to the host. You can use Telnet, Telnet Extended, SSH, or NS/VT transport types as well as Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols. Telnet is available for 3270,5250, VT, and HP session types, while Telnet Extended is only available for IBM 3270 session types. Choose SSH to connect to a VT host using SSH security protocols. NS/VT is available for HP session types.
Host Name or IP Address	Use this box to identify the host to which you want to connect. You can either select a host from the drop down list or type one into the host name field. When you start the Design Tool without a value for this setting, the Design Tool looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location. When it finds a Hosts file, the Design Tool changes the value of this setting.

Options	Description
Port Number	Select one of the following depending on which transport type you selected in the Session Setup or New Model dialog box; Telnet or NS/VT. The default port number for Telnet is 23 . The default port number for NS/VT is 1570 .

See [Customizing HP](#) for advanced HP Telnet configuration information.

More informationi

[Customizing the Terminal](#)

[Setting Up a Connection](#)

4.4.2 Customizing the Terminal

A quick way to customize the terminal is to create a model using a pre-configured settings file provided with Host Integrator. The settings file has defaults appropriate for the terminal session you are modeling.

[Customizing the 3270 Terminal](#)

[Customizing the 5250 Terminal](#)

[Customizing the VT Terminal](#)

[Customizing the HP Terminal](#)

Customizing the 3270 Terminal

You can control various aspects of the Design Tool's behavior as a 3270 terminal. First, [configure a 3270 session](#) and then configure any of the options available from the Settings menu to customize your 3270 terminal.

3270 Terminal Setup

- National character set

The values identify the various host character sets that Host Integrator supports. The Design Tool uses this setting to choose a conversion table that it uses to convert host characters (EBCDIC) into PC characters (ANSI). This setting should match the national character set used by your host system. If it doesn't match, then some characters, such as accents, may not display correctly. See your host documentation for definitions of the characters in each set. The default value is US English.

You can also define a custom code page. After creating the custom code page file, select Custom in the national character set list. A set of examples is included with Host Integrator; see [Creating a Custom Code Page](#) for details.

- Country Extended Graphics Code

When this check box is selected, additional characters are available in the configured National character set. See your host documentation for details. By default this box is selected.

You can change the settings in the Terminal Setup dialog box if you have opened a model but you are not connected to it.

- If you have no model open, this dialog box is unavailable.
- If you are connected to a model, the settings in this dialog box can be viewed but cannot be changed.

Default 3270 keyboard mapping and functions

Function	Keystroke	Environment	Function
Alt Cursor	Alt+6	host	
Attention	Ctrl+F1	host	Interrupts the host application program. Not all host Telnet programs support the Attention key.
Backspace	Backspace	host	
Backtab	Shift+Tab	host	
Clear	Ctrl+F2 or Scroll Lock	host	When you first log on, your host connection is in implicit state, meaning that there is only a single partition. In implicit state, Clear erases all data in the partition. When there are multiple partitions, Clear erases all data in all partitions, and returns the emulator to implicit state. When there are multiple partitions, use the Clear Partition function to clear only the current partition. Most host applications use only a single partition.
ClearPartition	Ctrl+F4	host	Erases all data in the current partition.
Connect	Alt+C	Design Tool	
Copy	Ctrl+C	Design Tool	
Cursor Blink	Alt+8	host	Changes the blink rate of the cursor, cycling through two options (Blinking Disabled and Windows System).

Function	Keystroke	Environment	Function
Cursor Select	Ctrl+F3	host	Simulates a light pen select in the field containing the cursor. Sets or clears the attribute indicating that a field has been modified. You can use the right mouse button in selectable terminal fields to perform a cursor select; in this way, the mouse can be used to simulate a light pen.
Delete Character	Delete	host	Deletes the character the cursor is on.
Delete Word	Alt+Delete	host	Deletes one word of text from an input field.
Disconnect	Alt+D	Design Tool	
Down	Down Arrow	host	
Duplicate	Ctrl+Page Up	host	Inserts a dup character into the buffer. The cursor moves to the first position of the next unprotected field if the screen is formatted, or to row 1, column 1, if the screen is unformatted.
Enter	Enter	host	
EraseEOF	End	host	Erases all data from the cursor location to the end of the current field.
Erase Input	Alt+F5	host	Erases all unprotected fields in the current partition.
Field Delimit	Ctrl+Alt+F	host	Toggles IBM standard and extended field delimiters on or off during a host session. Toggling can be useful for troubleshooting.
Field Mark	Ctrl+Page Down	host	Inserts a field mark character.

Function	Keystroke	Environment	Function
Home	Home	host	
Insert	Insert	host	
Keyclick	Alt+7	host	Toggles the keyclick on or off. When this function is on, the PC emits a low-pitched sound in numeric fields, a high-pitched sound in alphabetic fields, and no sound in protected fields.
Left	Left Arrow	host	
Left, double speed	Alt+Left Arrow	host	
New Line	Shift+Enter	host	
Next Window	Alt+N	Design Tool	
NumLock	Num Lock	host	
PA1	Page Up	host	
PA2	Page Down	host	
PA3	Ctrl+3	host	
Pan Left	Ctrl+Left Arrow	host	Moves the portion of the current partition visible on the screen so that a different part of the partition is visible. The distance panned is determined by the host upon establishment of the partition. If Host Integrator can display the entire partition on the terminal screen, this function has no effect.

Function	Keystroke	Environment	Function
Pan Right	Ctrl+Right Arrow	host	Moves the visible portion of the current partition so that a different part of the partition is visible. The distance panned is determined by the host upon establishment of the partition. If Host Integrator can display the entire partition on the terminal screen, this function has no effect.
Partition Jump	Ctrl+Tab	host	Moves the cursor to the next partition. Continuing to use this function cycles you through all available partitions.
Paste	Ctrl+V	Design Tool	
PF1-PF12	F1-F12	host	
PF13-PF24	Shift+F1+Shift+F12	host	
Reset	Esc	host	
Right	Right Arrow	host	
Right, double speed	Alt+Right Arrow	host	
Scroll Down	Ctrl+Down Arrow	host	Moves the visible portion of the current partition so that a different part of the partition is visible. The distance scrolled is determined by the host upon establishment of the partition.
Scroll Up	Ctrl+Up Arrow	host	Moves the visible portion of the current partition so that a different part of the partition is visible. The distance scrolled is determined by the host upon establishment of the partition.

Function	Keystroke	Environment	Function
Show Command Line	Alt+L	Design Tool	
SysReq	Alt+Print Screen	host	The definition of this key and its values vary by host application.
Tab	tab	host	The definition of this key and values vary by host application.
Toggle Terminal Keyboard	Alt+K	Design Tool	
Toggle Toolbar	Alt+B	Design Tool	

Function	Keystroke	Environment	Function
Up	Up	host	

Customizing the 5250 Terminal

You can control various aspects of the Design Tool's behavior as a 5250 terminal. First, [configure a 5250 session](#) and then configure any of the options available from the Settings menu to customize your 5250 terminal.

5250 Terminal Setup

- National character set

The values identify the various host character sets that Host Integrator supports. The Design Tool uses this setting to choose a conversion table that it uses to convert host characters (EBCDIC) into PC characters (ANSI). This setting should match the national character set used by your host system. If it doesn't match, then some characters, such as accents, may not display correctly. See your host documentation for definitions of the characters in each set. The default value is US English.

You can also define a custom code page. After creating the custom code page file, select Custom in the national character set list. A set of examples is included with Host Integrator; see [Creating a Custom Code Page](#) for details.

You can change the settings in the Terminal Setup dialog box if you have opened a model but you are not connected to it.


If you have no model open, this dialog box is unavailable.

If you are connected to a model, the settings in this dialog box can be viewed but cannot be changed.

Default 5250 keyboard mapping

Function	Keystroke	Environment	Description
Alt Cursor	Alt+6	host	
Attention	Esc	host	Interrupts the host application program. Workstation uses the Attn key to alert the AS/400 system that a request (such as Enter) is not being honored. The Attn key is used to lock or unlock the keyboard whether the keyboard is locked or unlocked.
Backspace	Backspace	host	Moves the cursor back to the previous position in which a character was entered. If the cursor is already in the first position of a line, Backspace moves back to the last position in the previous line.

Function	Keystroke	Environment	Description
Backtab	Shift+Tab	host	Moves the cursor back to the first position in the current line. When the cursor isn't on an input field, Backtab moves the cursor to the first position in the previous input field. If the screen does not have multiple input fields, Backtab moves the cursor to row 1, column 1.
Begin Bold	Ctrl+B	host (word processing)	
Begin Underline	Ctrl+U	host (word processing)	
Bottom of Page	Ctrl+Down Arrow	host (word processing)	
Carrier Return	Left Ctrl, Right Ctrl	host (word processing)	
Center	Ctrl+C host (word processing)		
Clear	Pause	host	Signals the host to erase all user-entered text from the screen.
Connect	Alt+C	Design Tool	
Delete	Delete	host	Deletes the character at the cursor position. All characters to the right of the cursor (in the same input field) shift one position to the left. Null characters are inserted at the right end of the line. If you invoke this function when the cursor is not in an input field, the input inhibit symbol appears on the line. (In the 5250 status line, the input inhibit indicator is a vertical bar in inverse video.) Press Reset (Left Ctrl) to clear the input and enable input.
Disconnect	Alt+D	Design Tool	
Down	Down Arrow	host	
Down Double	Alt+Down Arrow	host	Moves the cursor down one or more lines. The number of lines the cursor moves is determined by the value of Vertical Scroll in the Terminal Setup dialog box. The default is two lines.

Function	Keystroke	Environment	Description
Duplicate	Shift+Insert	host	Uses the data from the equivalent field in the previous contents of this field. If a field is programmed to allow you'll see a symbol at the cursor position when you invoke this function. Invoking this function when the cursor is not in a field or not in a field that supports this function causes the inhibit symbol to appear in the status line. (In the 525 the input inhibit indicator is the letters II in inverse video. Press Reset (Left Ctrl) to clear the symbol and enable input.
End Bold	Ctrl+J	host (word processing)	
End of Line	Ctrl+Right Arrow	host (word processing)	
End Underline	Ctrl+J	host (word processing)	
Enter	Enter	host	Transmits any data you have typed in the Terminal window to the host application.
Erase EOF	End	host	Erases all data from the cursor position to the end of the field, without moving the cursor.
Erase Input	Alt+End	host	Erases the contents of all input fields on the screen and moves the cursor to the beginning of the first input field. If the screen has multiple input fields, Erase Input moves the cursor to row 1, column 1 of the screen, and no data is erased.
Extended Graphics	Alt+Right Shift	host	<p>The following special characters are available in extended graphics mode:</p>  <p>■ Blue indicates what character is inserted if you press this key with Shift.</p> <p>■ Purple indicates what character is inserted if you press this key with Alt.</p> <p>Host Integrator remains in extended graphics mode until you press Alt+Right Shift again or Reset (Left Ctrl).</p>
F1-F12	F1-F12	host	

Function	Keystroke	Environment	Description
F12-F24	Shift+F1- Shift+F12	host	
Field Exit	Right Ctrl	host	Moves the cursor out of an input field, inserting null characters at the current cursor location to the end of the field. The cursor then moves to the first position of the next input field. If you use Field Exit on a right-adjust field, the data to the left of the cursor shifts one position to the right. The vacated positions are filled with zeros or blanks as appropriate for the program, and the cursor advances to the next input field.
Field Minus	Numeric keypad –	host	Moves the cursor out of a signed-numeric or numeric-only field, inserting a minus sign in the last position of a signed-numeric field or changing the last position in a numeric-only field to a minus sign character that tells the system that this field has a negative value.
Field Plus	Numeric keypad +	host	Except in a signed-numeric or numeric-only field, this function is identical to Field Exit. In a signed-numeric field, this function moves the cursor to the next field, removing a minus sign if there is one in the last position. In a numeric-only field, this function moves the cursor to the next field, changing the last position to a plus sign character that tells the system that this field has a positive value.
Half Index Down	Ctrl+H	host (word processing)	
Half Index Up	Ctrl+Y	host (word processing)	
Help	Scroll Lock	host	Provides help from the system or, when an error condition exists, an explanation of the error condition. See your host documentation for more information on error codes.

Function	Keystroke	Environment	Description
Hex Mode	Alt+F7	host	Use this function before entering a hexadecimal value. To enter a hexadecimal value, type two characters. The characters can be 0-9 or a-f. No other input is allowed. When you press Alt+F7 to put Host Integrator in hex mode, the letter H appears on the 3488 status line (default), a little to the left of center. Reset (Left Ctrl) causes Host Integrator to exit hex mode. When you enter the first character of your hex value, the letter H on the status line becomes a capital H. If Preserve entry mode is on, pressing Left Ctrl at this point takes the character out of the buffer but doesn't exit hex mode. (This is a way of retracting the character you typed. To show this, the capital H becomes a small h again. Host Integrator now expects you to enter two more characters of your hex value.) Press Left Ctrl again to back complete the hex mode.
Home	Home	host	Moves the cursor to the first input position on the screen. If the screen contains no input fields, Home moves the cursor to the first column. If you use this function when the cursor is already at the first input position on the screen, Host Integrator treats it as a backspace aid key instead.
Insert	Insert	host	The insert symbol appears in the 3488 status line (default) and the 5250 status line, the letters IM are displayed in inverse video. (Host Integrator is in insert mode.) Characters you type in insert mode are inserted at the cursor position. Existing characters at and to the right of the cursor shift one position to the right for each character you type. There must be a null character at the right end of the insert buffer. If you type a character you type in insert mode. If you attempt to insert more characters than there are nulls, an X appears in the status line. Input is inhibited. Press Reset Left Ctrl to remove the insert mode and enable input.
Insert Symbols	Ctrl+A	host (word processing)	
Left	Left Arrow	host	
Left Double	Alt+Left Arrow	host	Moves the cursor one or more columns to the left.

Function	Keystroke	Environment	Description
New Line	Shift+Enter	host	Moves the cursor to the first input position on the next line. If there are no input positions on the next line, the cursor moves to the first input position on the next line with an input field. If you press Newline when there are no lines between the current line and the end of the screen that contains input fields, it wraps to the first line in the screen. Newline differs from Tab in that it does not drop to the second line of a multi-line field. Tab always moves to the next field.
Next Stop	Ctrl+N	host (word processing)	
Next Text Column	Ctrl+D	host (word processing)	
Next Window	Alt+N	Design Tool	
PA1-PA3	Alt+F1-Alt+F3	host	
Page Down	Page Down	host	Scrolls down one page in the current host screen (equivalent to Page Up).
Page End	Ctrl+P	host (word processing)	
Page Up	Page Up	host	Scrolls up one page in the current host screen (equivalent to Page Down).
Plus CR Mode	Alt+F12	host	Execute this function once and Host Integrator shows hexadecimal codes in front of each field in the Terminal window. To indicate the field and display attributes for each field. If you configured Host Integrator to use the 3488 status line, then the debug status line, then d appears in the status line. Execute this function again and Host Integrator shows different two hexadecimal codes indicating the extended character attributes for each field. A c appears in the 3488 or debug status line. Execute this function a third time and Host Integrator shows three hexadecimal codes indicating the character attributes for each character in the Terminal window. An a appears in the debug status line. Press Reset Left Ctrl twice to exit Plus CR Mode.
Print	no mapping	host	Sends an image of the screen to the host. How and when the image is printed depends on the configuration of the host.

Function	Keystroke	Environment	Description
Print Screen	Print Screen	Design Tool	Prints the contents of the Terminal window.
Required Carriage Return	Right Ctrl	host (word processing)	
Required Space	Ctrl+Spacebar	host (word processing)	
Required Tab	Ctrl+Tab	host (word processing)	
Reset	Left Ctrl	host	Press Reset once to exit from various modes and clear errors. Press Reset twice to exit Plus CR mode.
Right	Right Arrow	host	
Right Double	Alt+Right Arrow	host	Moves the cursor one or more characters to the right.
Roll Down	Shift+Down Arrow	host	Scrolls up one page in the current host screen (equivalent to Page Up).
Roll Up	Shift+Up	host	Scrolls down one page in the current host screen (equivalent to Page Down).
Rule	Alt+Page Down	host	Toggles the rule line into or out of view. The rule line consists of a horizontal line, a vertical line, or both, indicating the row and column containing the cursor.
Show Command Line	Alt+L	Design Tool	
SLP Auto Enter	not mapped	host	The full name of this key is Selector Light Pen Auto Enter. It simulates a light pen select at the cursor location.
Stop	Ctrl+S	host (word processing)	
SysReq	Alt+Print Screen	host	The SysReq key is allowed whether the keyboard is locked or unlocked. Use this function to: select and start an alternate program, notify the host system that the terminal is ready to select another program, or request that the keyboard be unlocked.

Function	Keystroke	Environment	Description
Tab	Tab	host	Moves the cursor from the current position to the first Tab stop in the next input field. If there are no input positions available in the next input field, Tab moves the cursor to the first Tab stop at the current cursor position. If there are no Tab stops at the current cursor position, Tab moves the cursor to the first Tab stop in the first input field on the screen. If the screen contains input fields, Tab moves the cursor to row 1, column 1 on the screen.
Tab Advance	Ctrl+T	host (word processing)	Moves the cursor from the current position to the next Tab stop. If there are no Tab stops in the current field, Tab Advance moves the cursor to the first Tab stop in the next input field on the screen. Tab Advance is defined for Text Assist mode.
Toggle Toolbar	Alt+B	Design Tool	
Toggle Terminal Keyboard	Alt+K	Design Tool	
Top of Page	Ctrl+Up	host (word processing)	
Test Request	Alt+Scroll Lock	host	Signals the host to enter test request mode, a set of test requests created by IBM and provided by the AS/400. This function is not available in the sign-on screen.
Up	Up	host	
Up Double	Alt+Up	host	Moves the cursor up one or more lines.

Function	Keystroke	Environment	Description
Word Underline	Ctrl+W	host (word processing)	

Customizing the VT Terminal

You can control various aspects of the Design Tool's behavior as a VT terminal. First, configure a VT session and then configure any of the options available from the Settings menu to customize your VT terminal.

[Configuring VT Emulation Settings](#)

[Advanced VT SSH Options](#)

[Advanced VT or HP Telnet](#)

[VT Control Functions](#)

[Default VT Keyboard Mapping](#)

[VT Screen Setup](#)

CONFIGURING VT EMULATION OPTIONS

From the Settings menu, click Terminal. The Emulation tab includes the following:

- Host Character Set

The Host Integrator VT terminal supports the following character sets:

DEC Supplemental

ISO Latin-1 (ISO-8859-1) This is the ISO-8859-1 character set

ISO Latin-9 (ISO-8859-15)

PC English (437)

PC Multilingual (850)

Windows Latin (1252)

The default value for the Host character set depends on the type of terminal you are emulating. This setting reflects the current terminal state of VT Host Character Set, which can be changed by the host. The associated default setting, saved with the model is VT Host Character Set Default.

When a soft reset is performed, or when initially connecting, the default host character set is used.

The current terminal state for the host character set can be altered by invoking the DECSTR sequence. The Host character set may also be specified by the Select Character Set (SCS) sequence.

- Terminal ID

The setting in this list determines the response that Host Integrator sends to the host after a primary device attributes (DA) request. This response lets the host know what terminal functions it can perform. Host Integrator's response for each Terminal ID is exactly the same as the VT terminal's response; some applications may require a specific DA response. This terminal ID setting is independent of the Terminal type setting.

The options are VT100, VT220, VT320, VT420, and VT52.

This setting reflects the current terminal state of VT Terminal ID, which can be changed by the host. The associated default setting, saved with the model, is VT Terminal ID Default.

- Online

Host Integrator is always in either remote or local mode. Keeping this check box selected means that Host Integrator can communicate with the host as a terminal; this is known as remote mode. Host Integrator transmits each character typed from the keyboard to the host. As Host Integrator receives characters from the host, it displays them on the screen.

When Host Integrator is in remote mode, but is not connected to a host computer, characters typed at the keyboard do not appear on the screen.

When you clear this check box, Host Integrator can no longer communicate with the host; this is known as local mode. In local mode, Host Integrator does not attempt to communicate with a host computer. Characters entered from the keyboard appear on the

screen, but are not transmitted to the host; nor is any data from the host (for example, notification of a mail message) received by Host Integrator.

- Newline

Selecting this check box causes Host Integrator to send both a carriage return and linefeed when you press Enter (known as new line mode). When Host Integrator receives a linefeed, form feed, or vertical tab, it moves the cursor to the first column of the next line. When this check box is left cleared (linefeed mode), the Return key sends only a carriage return. A linefeed, form feed, or vertical tab received from the host moves the cursor down one line in the current column.

If lines on the display is being overwritten (that is, the host is not sending a linefeed along with a carriage return), select this check box. If the New line check box is selected but the host does not expect to receive a linefeed with each carriage return, lines will be double-spaced on the display.

This setting reflects the current terminal state of VT New Line, which can be changed by the host. The associated default setting, saved with the model, is VT New Line Default.

- Autowrap

Selecting this check box causes characters to wrap to the next line automatically when the cursor reaches the right margin of the display. This setting is different from the VAX host's terminal wrap characteristic, which is set with this DCL command: SET TERMINAL/[NO]WRAP

The host command determines whether characters wrap automatically when they reach the maximum terminal width set by the host's SET TERMINAL/WIDTH command. Host Integrator's Autowrap option, in contrast, determines whether characters wrap when they reach the right margin of the display. When Host Integrator is communicating with the host:

If...	Then...
If terminal wrap is set on the host	Characters wrap when they reach the maximum terminal width, regardless of the setting of the Autowrap check box.
If terminal wrap is not set and the Autowrap check box is selected	Characters wrap at the right margin of the display

If...	Then...
If terminal wrap is not set and the Autowrap check box is cleared	Characters never wrap when they reach the right margin of the display. New characters overwrite the character at the right margin until a carriage return is entered

When Host Integrator is not communicating with the host (that is, the Online check box is not selected), the Autowrap check box works as described in the last two items above, as if the host's terminal wrap characteristic is not set.

- National Replacement Character Set

This setting specifies a set of character translations that occur between the local computer and the host in 7-bit mode. This setting affects how characters entered from the keyboard or from a local file are transmitted to the host, and how characters sent from the host are written to local files, to the screen, or both.

Translations are not performed unless the Use NRC setting is set to Yes.

This setting reflects the current terminal state of the National Replacement Set, which can be changed by the host. The associated default setting, saved with the model, is VT National Replacement Set Default.

- Use NRC (7-bit) Set

This setting specifies whether the translations specified by the National Replacement Set setting should be performed.

This setting reflects the current terminal state of VT Use NRC, which can be changed by the host. The associated default setting, saved with the model, is VT Use NRC Default.

- Answerback message

This setting allows you to enter an answerback message if the host expects an answer in response to an ENQ character.

This setting reflects the current terminal state of VT Answerback Message, which can be changed by the host. The associated default setting, saved with the model, is VT Answerback Message Default.

Select the *Insert special characters* check box to include escape sequences and ASCII control codes in the message.

Select *Auto answerback* to specify whether the answerback message is automatically sent to the host after a communications line connection.

- User Features Locked

This setting specifies whether certain features can be changed by the host. When this setting is set to Yes, the following properties cannot be changed by the host: - Tab stops - The Keyboard Locked setting which specifies whether or not the keyboard is locked (that is,

it cannot be used). - The Inverse Video setting which specifies whether the foreground and background colors for screen attributes are reversed.

- User-defined keys locked

When this setting is set to Yes, the Host Integrator locks the user keys to prevent the host from clearing or redefining them. When user-defined keys are locked, the only way they can be redefined is to first unlock them, and the only way to unlock them is by selecting No. This setting is not saved to a Host Integrator settings file.

If you want to define new user keys or allow the host to define them, the keys must be unlocked. If user keys are locked and an application tries to redefine a key using a DECUDK sequence, Host Integrator ignores the sequence.

This setting reflects the current terminal state of VT User-defined Keys Locked, which can be changed by the host. The associated default setting, saved with the model, is VT User-defined Keys Locked Default.

ADVANCED VT SSH OPTIONS

- User authentication

Click in the box next to any authentication method to clear or enable that method. You must select at least one authentication method. You can use the arrows to specify your order of preference. The first method you select that is supported by the server is used.

- Cipher list

Use this list to specify the ciphers you want to allow for connections to the current host. When more than one cipher is selected, the SSH client attempts to use ciphers in the order you specify, starting from the top. To change the order, select a cipher from the list, then click the up or down arrow. The cipher used for a given session is the first item in this list that is also supported by the server.

- HMAC list

Specifies the HMAC (hashed message authentication code) methods you want to allow. This hash is used to verify the integrity of all data packets exchanged with the server. When more than one HMAC is selected, the SSH client attempts to negotiate an MAC with the server in the order you specify, starting from the top. To change the order, select an HMAC from the list, then click the up or down arrow.

- Key exchange algorithms

Specifies which key exchange algorithms the client supports, and the order of preference. In some cases, you may need to change the order of the key exchange algorithms to put DH Group14 SHA1 ahead of the other values. This is required if you want use the hmac-sha512

HMAC, or if you see the following error during key exchange: "Unable to exchange encryption keys."

- Host key algorithms

Specifies the host key type the client will accept from the server. You must select at least one host key type. The SSH client will use a key type from the server in the order you specify, starting from the top. To change the order, select an algorithm from the list, then click the up or down arrow.

- Keep alive

When **Keep Alive** is selected, Host Integrator sends NOOP messages to the server through the secure tunnel at the specified interval. Use this setting to maintain the connection to the server. Use **Interval in seconds** to specify how frequently server alive messages are sent. If this setting is not enabled, the SSH connection will not terminate if the server dies or the network connection is lost.

The SSH **Keep Alive** setting is not related to the TCP keep alive setting that can be set in the Windows registry to keep all TCP/IP connections from being timed out by a firewall. To change the TCP/IP keep alive behavior, you need to edit the Windows registry.

- Enable compression

When Enable compression is selected, the client requests compression of all data. Compression is desirable on modem lines and other slow connections, but will only slow down response rate on fast networks.

- Only show FIPS validated values

Select this to filter the Cipher and HMAC lists to show only FIPS validated values. SSH only uses the values you configure. If you choose a non-FIPS value while running in FIPS mode an error message asking to you to specify a valid cipher displays.

Federal Information Processing Standards (FIPS) are guidelines established by the United States government to standardize computer systems. To use FIPS 140-2 validated

encryption, in a Windows environment, you must first define an environment variable, VHI_FIPS = 1.

Advanced VT or HP Telnet

You can configure the following options in this dialog box:

- Terminal ID

When Host Integrator connects to a Telnet host, the Telnet protocol negotiates a Telnet terminal ID with the host. In general, this negotiation results in the selection of the correct terminal ID. However, if you are having trouble running a host application, the negotiation between Telnet and the host could be the issue.

This setting determines:

- which screen control sequences the host sends to Host Integrator to format the screen.
- the position of the cursor
- what characters to display in a host application

To override Host Integrator's election of a terminal, select a terminal ID from the list (because this list box is editable, you can just type in any value). If you enter a terminal ID that the host does not recognize, Host Integrator reverts to a list of default values until one is found that the host supports. The default terminal ID values are VT100, VT 220, VT 320, VT 420 and VT52 and HP and HP2392A. The terminal type and the settings are independent of each other. If you enter a terminal ID string, it may be up to 40 characters long taken from a set of uppercase letters, digits, and the two punctuation characters, hyphen and slash. It must start with a letter and end with a letter or digit.

- Location (Optional)

Specifies to the host where the connection originated. Can also be used to provide informational messages to the host from the PC. Usage conventions vary by site. Supports up to 260 characters. You cannot change this value while you're connected to a gateway.

- Send LF after CR

A "true" Telnet host expects to see a CrNu (carriage return/null) character sequence to indicate the end of a line sent from the Design Tool. There are some hosts on the Internet that are not true Telnet hosts, and they expect to see a Lf (linefeed) character following the Cr at the end of a line. If you're connecting to this type of Telnet host, select this check box.

- Initiate Option Negotiation

Specifies whether certain connection options, including whether to always request a binary mode connection, should be negotiated when the Telnet connection is established. Connections to some hosts on the Internet are expedited if this check box is cleared so that the Design Tool does not attempt to initiate negotiations for Telnet options.

- Request Binary

Telnet defines a 7-bit data path between the host and the terminal (Host Integrator). This type of data path is not compatible with certain national character sets. Fortunately, many

hosts allow for 8-bit data without zeroing the 8th bit, which resolves this problem. Select this check box to force the host to use an 8-bit data path.

- Set Host Window Size

Sends the number of rows and columns to the Telnet host whenever they change. This enables the Telnet host to properly control the cursor if the window size is changed.

- Ctrl-break Character

Specifies what happens when you press the following keys:

ATTN key (Ctrl-F1 by default) in a 3270 session

Ctrl-break in a VT or HP session

Options to send to the host are:

Telnet Abort Output

Telnet Break

Telnet Interrupt Process

- Send Keep Alive Packets

In some cases, Host Integrator may become aware of Telnet communication problems only after a significant delay or when it attempts to send data to the host. This can cause

problems if you enter a large amount of data on one screen or if you keep your connection open during periods of inactivity.

To become aware of connection problems as they occur, you can configure your model to send Keep Alive packets. Four methods are available:

None - No keep alive packets are sent (default)

Send NOP Packets - Periodically a No Operation (NOP) command is sent to the host. The gateway and host are not required to respond to these commands, but the TCP/IP stack can detect if there was a problem delivering the packet.

System - The TCP/IP stack keeps track of the host connection. This method requires less system resources than Send Timing Mark Packets or Send NOP Packets, but most TCP/IP stacks send Keep Alive packets infrequently.

Send Timing Mark Packets - Periodically a Timing Mark Command is sent to the host to determine if the connection is still active. The gateway or host should respond to these commands. If a response is not received or there is an error sending the packet, it will shut down the connection. To view the average amount of time has waited for a response to a Timing Mark Command in the Design Tool, open the View Settings dialog box and select the Telnet Average Keep Alive Roundtrip setting.

- Keep Alive Timeout

The Keep Alive Timeout is the interval between the keep alive requests sent by the Design Tool. The range of values is 1-9999 seconds, and the default is 600 seconds.

- Local Echo

Controls how the Design Tool responds to remote echo from a Telnet host: - Automatic - (Default) Attempts to negotiate remote echo, but does what the host commands. - Yes - Negotiates local echo with the host, but always echoes. - No - Negotiates remote echo with the host, but does not echo.

VT CONTROL FUNCTIONS

Control functions cause Verastream to perform certain actions, such as move the text cursor, add a line of text, assign character attributes, and change character sets. Typically, the host application sends control functions to Verastream to perform the desired actions. There are three symbols used in this section that describe a sequence:

Symbol	Description
ESC	Stands for the Escape character, and begins an escape sequence
CSI	Stands for the Control Sequence Introduction character. When Verastream receives this character, it recognizes the string that follows as a control sequence

Symbol	Description
DCS	Stands for the Device Control String character, and begins a device control sequence

Control Function Notation

The following notation is used throughout this section:

Most control functions have a mnemonic identifier. You will never need to enter the mnemonic; it's simply a convenient word to help you remember the name of the control function. For example, DECCOLM is the Digital mnemonic for setting the number of display columns.

Control functions are case sensitive, and must be typed exactly as shown.

Where appropriate, a slash is used through a zero (Ø) to distinguish it from the uppercase letter O.

Note the difference between a lowercase L (l) and the number 1 (1). References are occasionally given to a character's decimal value in the character set charts. For example, the space character is ASCII decimal 32. Control characters are always in the same positions in the charts, and can be located uniquely by their decimal value; the word "ASCII" is omitted.

This topic shows control functions in their 8-bit format. You can always use the 7-bit equivalent escape sequence to represent the 8-bit control, as shown in the Recognized C0 Control Characters Table.

Parameters you supply for a sequence are enclosed in angle brackets. For example, when using the `CSI<n>C` control function, replace `<n>` with the number of spaces you want the cursor to move. If `<n>` is omitted, the default is used. For most sequences, the default is 1 (when there is not a corresponding Ø sequence).

Numeric parameters are represented by ASCII strings. For example, in the sequence `CSI10;13H` the numbers are the strings 10 and 13, not the decimal values 10 (LF character) and 13 (CR character). Numeric parameters are constrained to the range 0–9999, inclusive. Any numeric parameter with a value greater than 9999 is interpreted as 9999. The plus sign (ASCII decimal 43) and the minus sign (ASCII decimal 45) are not within the range of legal control characters. Therefore, signed numbers, for example "+10" or "-12," cause a control sequence or a device control string to be rejected. Do not include signs with numeric parameters.

Single-Character Control Functions

A single-character control function is made up of one control character (in contrast to multiple-character control functions). There are two sets of single-character control functions available on VT200, VT300, and VT400 terminals:

- C0 control characters have decimal values 0–31. Verastream supports only the C0 characters shown in the **Recognized C0 (7-Bit) Control Characters Table**. Each C0 character is encoded in 7 data bits, with the high-order bit always 0. C0 codes, therefore, can be used in both 7-bit and 8-bit operating environments. LF and CR, for example, are single-character C0 control functions.
- C1 control characters have decimal values 128–159. Verastream supports only the C1 characters shown in the Recognized C1 (8-Bit) Control Characters Table. Each C1 character is encoded in 8 data bits, with the high-order bit always 1. C1 characters provide a few more functions than C0 characters, but can only be used directly in an 8-bit operating environment. DCS, for example, is a single-character C1 control function.

In 7-bit operating environments, C1 characters must be converted to a two-character escape sequence equivalent.

Recognized C0 (7-Bit) Control Characters Table

The following table lists the C0 control characters that Verastream recognizes. C0 controls can be used in both 7-bit and 8-bit environments. To represent C0 controls in a Verastream macro, use the Chr\$(<n>) syntax, where <n> is the decimal value of the C0 control. See the “Decimal” column in the following table:

Name	Character	Octal	Keystroke	Action
Null	NUL	0	^@	Ignored when received
Enquiry	ENQ	5	^E	Transmits the answerback message
Bell	BEL	7	^G	Sounds a bell
Backspace	BS	10	^H	Moves cursor left one position on the current line
Horizontal tab	HT	11	^I	Moves cursor to the next tab stop, or to the right margin.
Linefeed	LF	12	^J	Causes a linefeed or new line operation.
Vertical tab	VT	13	^K	Same as linefeed
Form feed	FF	14	^L	Same as linefeed
Carriage return	CR	15	^M	Moves the cursor to the left margin of the current line

Name	Character	Octal	Keystroke	Action
Shift out (locking shift 1)	SO	16	^N	Maps the G1 character set into GL. G1 is designated by using a Select Character Set (SCS) sequence.
Shift in (locking shift 0)	SI	17	^O	Maps the G0 character set into GL. G0 is designated by using a Select Character Set (SCS) sequence.
Device control 1 (XON)	DC1	21	^Q	Continues sending characters when transmit is set to Xon/Xoff.
Device control 3 (XOFF)	DC3	23	^S	Stops sending characters when transmit is set to Xon/Xoff.
Cancel	CAN	30	^X	Cancels the sequence when received during an escape or control sequence.
Substitute	SUB	32	^Z	Same as cancel; displays a backwards question mark.
Escape	ESC	33	^[Introduces an escape sequence and cancels any escape sequence or control sequence in progress.

Name	Character	Octal	Keystroke	Action
Delete	DEL	177	(none)	Ignored when received.

Recognized C1 (8-Bit) Control Characters

In 8-bit environments, C1 controls can be sent directly. In a 7-bit environment, you can send an 8-bit C1 control character by converting it to an equivalent 7-bit escape sequence. The 8-bit controls are single character codes (such as CSI), whereas their 7-bit equivalents are two-character sequences (such as ESC).

To form an equivalent 7-bit escape sequence from an 8-bit control character:

Subtract the hexadecimal value 40 from the C1 control code's value.

Precede the result with the ESC character.

For example, the IND character (decimal 132) has a hexadecimal value of 84. To convert IND to a 7-bit equivalent, first subtract hexadecimal 40: 84 hex - 40 hex = 44 hex. Hexadecimal 44 is the letter D. Therefore, to represent the IND character in a 7-bit environment, you would use ESCD.

These are the C1 control characters that Verastream recognizes:

Name	Character	Hex	7-Bit Equiv.	Action
Index	IND	84	ESC D	Moves cursor down one line in same column, and scrolls at bottom margin.
Next line	NEL	85	ESC E	Moves cursor to first position of next line, and scrolls at bottom margin.
Horizontal tab set	HS	88	ESC H	Sets a tab stop at the cursor position.
Reverse index	RI	8D	ESC M	Moves cursor up one line in the same column, and scrolls at top margin.
Single shift 2	SS ₂	8E	ESC N	Maps G2 into GL for the next character only.
Single shift 3	SS ₃	8F	ESC O	Maps G3 into GL for the next character only.
Device control string	DCS	90	ESC P	Introduces a device control string.
Control sequence	CR	9B	ESC [Introduces a control sequence.
String	ST	9C	ESC \	Ends a device control string.

Multiple-Character Control Functions

The C0 and C1 codes listed in the Recognized C0 (7-Bit) Control Characters and Recognized C1 (8-Bit) Control Characters tables are single-character control functions, performing simple functions.

Multiple-character control functions, in contrast, can perform many more functions than the C0 and C1 controls, and are formed by a sequence of characters. There are three types of multiple-character control functions, introduced by the following single-character C0 and C1 controls:

- Escape sequences - introduced by the ESC character

- Control sequences - introduced by the CSI character

- Device control strings - introduced by the DCS character

Multiple-character control functions include both control characters and normal ASCII text, such as letters, numbers, and punctuation. For example, the multiple character device control string `DCSØ!u%5ST` assigns a user-preferred supplemental character set.

- Escape Sequences

An escape sequence begins with the C0 character ESC (decimal 27). To enter the ESC character in the Design Tool, press the ESC key. After receiving an ESC character, Verastream interprets the next characters as part of the sequence.

- Control Sequences

A control sequence begins with the C1 control character CSI (decimal 155). To enter the CSI character in the Design Tool, press the ESC key plus the [key.

Control sequences usually include variable parameters. The format is:

```
CSI P...P I...I F
```

P...P - Zero or more parameters. You can have up to 16 parameters per sequence, using a semicolon (;) to separate each one.

I...I - Zero or more intermediate characters.

F - The final character indicating the end of the control sequence.

For example, the following control sequence sets the scrolling region, where the top margin is at line 7 and the bottom margin is at line 18:

```
CSI7;18r
```

In this example, 7 and 18 are the parameters and r is the final character. This sequence does not have any intermediate characters.

- Device Control Strings

A device control string begins with the C1 control character DCS (decimal 144). To enter the CSI character in the Design Tool, press the ESC key plus the P key. Device control strings always include a data string. The format is:

```
DCS P...P I...I F <data string> ST
```

P...P - Zero or more parameters. You can have up to 16 parameters per sequence, using a semicolon (;) to separate each one.

I...I - Zero or more intermediate characters.

F - The final character indicating the end of the device control string.

<data string> - A data string of zero or more characters. Use a semicolon to separate individual strings. The particular range of characters included in a data string is determined by the individual device control string. Any character except ST (decimal 156) can be included in a data string.

ST - The string terminator. Type `Chr$(27) & "E` for 7-bit mode, or `Chr$(156)` for the 8-bit equivalent. (You can also press `e-\` in the terminal window.)

- APC, OSC, and PM

The application program command (APC), operating system command (OSC), and privacy message (PM) are also C1 controls. However, VT terminals—and Verastream—ignore them. The controls have the same format as device control strings and end with an ST. When

Verastream receives an APC, OSC, or PM introducer, it discards all following characters until receiving a string terminator (so the whole sequence is discarded).

- Interrupting Control Sequences

You can use the following C0 control characters to interrupt a control function or recover from an error:

ESC Cancels a sequence in progress, and begins a new sequence.

CAN Cancels a sequence in progress. Verastream interprets the characters that follow the CAN character as usual.

SUB Same as CAN, except displays a backwards question mark.

Using Nonprintable Characters

Some host application models require you to use nonprintable characters within strings. For example, they may be required when waiting for cursor positioning sequences sent from a VT or HP host using the Host Communication string event [WaitForCommString](#), or sending control sequences via the [TransmitANSI](#) command. To represent nonprintable control characters when defining your model in Host Integrator dialog boxes, use the standard C programming convention of encoding each character as a `\nnn` sequence, where `nnn` is the octal numeric value of the ASCII character, or use one of the `\x` sequences that have been defined for the more common control characters. For example, the escape character (0x1B), which prefixes VT cursor positioning sequences, is encoded as `\033`; carriage returns can be encoded using `\r`.

When a model file is read by Host Integrator, the `\nnn` sequences are parsed into their literal nonprintable character equivalents. Because backslashes (`\`) are interpreted as the beginning of a nonprintable character sequence, Host Integrator represents literal backslashes in strings with two backslashes. For example, if you want to wait for the display string "Enter a backslash (`\`) to continue" in an operation, what you will see in the Operation Edit dialog box for `WaitForDisplayString` is "Enter a backslash (`\`) to continue". If you modify a model file in XML format or the model file itself, you must follow this syntax to embed literal backslashes within string arguments.

Alphabetic Sequence	Equivalent Octal	Also Known As
<code>\a</code>	<code>\007</code>	Control-G (BEL) Bell
<code>\b</code>	<code>\010</code>	Control-H (BS) Backspace
<code>\t</code>	<code>\011</code>	Control-I (TAB) Horizontal tab
<code>\n</code>	<code>\012</code>	Control-J (LF) Linefeed
<code>\v</code>	<code>\013</code>	Control-K (VT) Vertical tab
<code>\f</code>	<code>\014</code>	Control-L (FF) Formfeed

Alphabetic Sequence	Equivalent Octal	Also Known As
\r	\015	Control-M (CR) Carriage return

DEFAULT VT KEYBOARD MAPPING

To find a VT keystroke's default mapping, click the key and click the Default button and note the corresponding Action in the lower half of the dialog box. Click Cancel if you do not want to revert the key's mapping to its default value.

VT Screen Setup

These settings reflect the current terminal state, which can be changed by the host.

- Display columns

This setting specifies the number of columns displayed in the Terminal window.

- Display columns

This box sets the number of lines on the display, not including the status line. The maximum number of rows you can enter depends on the display resolution.

When you change the number of rows, characters are scaled vertically to fit the desired number of rows in the Terminal window. The display is erased before the new setting takes effect.

- Status Line Type

The VT status line at the bottom of the Terminal window is a one-line display that shows information about the current session. When the Status line is set to Indicator it shows the current row and column of the text cursor.

When the Status line is set to Host Writable, host applications can display messages on it. The control function DECSASD selects whether the Terminal window or the status line is the active area for displaying text.

- Interpret controls

The VT terminal character set includes 65 control characters with decimal values 0-31 and 127-159. This option determines whether Host Integrator should interpret or display control characters.

Customizing the HP Terminal

You can control various aspects of the Design Tool's behavior as an HP terminal. First, configure an HP session and then configure any of the options available from the Settings menu to customize your HP terminal.

See [Configuring Sessions](#) for information on how to configure the HP terminal.

Configuring HP Emulation Options

Configure HP Keyboard Options

A quick way to customize the terminal is to create a model using a pre-configured settings file provided with Host Integrator. The settings file has defaults appropriate for the terminal session you are modeling.

CONFIGURING HP EMULATION OPTIONS

These settings reflect the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Field Separator Default.

Emulation Options	Description
Field Separator	When the Host Integrator is transmitting in block, page, and format modes, it sends a field separator character after each field of the formatted screen except the last one. The value selected here specifies which ASCII character is sent. The values are ASCII decimal 0-127. The default is US (ASCII decimal 31).
Block Terminator	Under certain conditions, the Host Integrator transmits a block terminator character at the end of each block of data transmitted. The value selected here specifies which ASCII character is sent to indicate that the end of the block has been reached. The values are ASCII decimal 0-127. The default is RS (ASCII decimal 30).
Return Definition	Type a character in these boxes to be generated whenever Return is pressed. If the second character is a space, only the first character is generated. The values are ASCII decimal 0-127. The default is CR (ASCII decimal 13).
Host Prompt	An HP 3000 sends a DC1 character to indicate that it is ready to accept a line or block of characters. This character is sent immediately after the MPE colon prompt is sent. This list allows you to change which character is expected. Most hosts either use the DC1 (^Q) character or no prompt (which shows up simply as a space). Select the appropriate host prompt from this list (turn on DISPLAY FUNCTIONS to see the control codes sent by the host before changing this value). The values are ASCII decimal 0-127. The default is D1 (ASCII decimal 17)

Emulation Options	Description
Start Column	<p>For every line in display memory, the Host Integrator attempts to remember the leftmost column that was entered from the keyboard, as opposed to that received from datacomm. This way, the Host Integrator can distinguish the host prompt portion of each line from the user-entered portion. This information is used when you enable LINE MODIFY or MODIFY ALL to determine the leftmost column that should be transmitted to the host when you press Enter or Return. Under some circumstances, it is impossible for the Host Integrator to tell which column was the first user-keyed column; when that happens, it uses the value you enter in this box to determine the leftmost column to be transmitted. When Display Columns are set to 80, enter a value from 0 to 79. When you're in 132-column mode, enter a value from 0 to 131. The values are 0-131. The default is 0.</p>
Host Character Set	<p>The available host character sets for HP terminals are HP Roman 8 and HP Roman 9. The default is HP Roman 8.</p>
Online (Remote Mode)	<p>Specifies whether Host Integrator transmits each number, letter, or special character typed at the keyboard to the host. As characters are received from the host, Host Integrator displays them on your screen.</p>
Inhibit EOL Wrap	<p>When cleared, the Host Integrator automatically returns the cursor to the left margin in the next line when the cursor reaches either the right margin or the right screen edge. When selected, the cursor is not automatically advanced when you reach the right margin. As you type additional characters, each one overwrites the character at the right margin until you explicitly move the cursor by pressing Return or using an arrow key.</p>
Enq/Ack	<p>Some HP 3000 and HP 1000 computers use a form of handshaking called Enq/Ack (Enquire/Acknowledge) to prevent the terminal (or in this case, the Host Integrator) from falling too far behind the host system and losing data. With Enq/Ack pacing, the host system sends 80 characters followed by an ASCII Enq character and stops transmitting. When the Host Integrator has processed all of the characters preceding the Enq, it sends an ASCII Ack character, which tells the host it is ready for more data. "Classic" HP 3000 architecture uses this form of pacing. MPE<i>i</i>X systems do not; the Enq/Ack setting is disregarded in this environment.</p>

Emulation Options	Description
Use Host Prompt	Clear this check box if you want the Host Integrator to ignore the host prompt. Clearing the Use host prompt check box has the same effect as selecting the Inhibit Handshake and Inhibit DC2 check boxes. Ignoring the host prompt forces the Design Tool to behave as though both inhibits are on, thus preventing handshaking. Over an X.25 network, this prevents communications problems caused by applications that use handshaking. When the Use host prompt check box is cleared, the Design Tool always responds to a primary status request from the host that both Inhibit handshake and Inhibit DC2 are enabled. This can affect a host application that explicitly changes one of these inhibits.
Inhibit Handshake	This check box, along with Inhibit DC2 and some other factors, determines the type of handshaking that precedes each block transfer of data from the Host Integrator to the host system. When selected, the DC1 handshake for block transfers is inhibited.
Transmit Functions	Most keyboard keys have an associated ASCII character. Several keys perform functions for which there is no character defined; for example, Home and PgUp. Certain host software programs, such as HP Slate, need to be informed when you press one of these non-ASCII keys. Selecting this option signals the Host Integrator to inform the host system whenever you press one of these keys. When this check box is selected and the Design Tool is operating in character/remote mode, each time you press one of those keys the associated escape sequence is transmitted to the host. Most applications that require this feature automatically send the escape sequences to enable and disable the feature, so you probably will never need to enable it manually.
SPOW	Ordinarily, the Spacebar overwrites and erases existing characters. When the SPOW (SPace OverWrite) check box is selected, spaces entered from the keyboard (not spaces echoed from the host), move the cursor over existing characters, but do not overwrite them with spaces: the SPOW latch is turned on by a carriage return and off by a linefeed, tab, or home up.

Emulation Options	Description
Block Transfer Unit	When operating in block mode, a block of one or more characters is transmitted when you press Enter or when the host requests a block transfer from terminal memory. This option determines how much data Host Integrator transmits on each block transfer. When set to Line, data is transmitted one line at a time, or one field at a time in format mode. When set to Page, data is transmitted one page at a time.

CONFIGURE HP KEYBOARD OPTIONS

To find a HP keystroke's default mapping, click the key and click the Default button and note the corresponding Action in the lower half of the dialog box. Click Cancel if you do not want to revert the key's mapping to its default value.

HP Terminal Function Key Configuration

The Function Keys tab lets you select which set of key labels are displayed along the bottom of your screen, and lets you customize eight user keys. A user key definition consists of the following:

- A 16-character label (eight characters per line)

- A string of up to 80 characters that is produced when the key is pressed

- An attribute determining how the Design Tool processes the string when the key is pressed

To reconfigure a function key:

1. Select the key to configure by clicking it with the mouse or by pressing the key on the keyboard.
2. Enter a key string of up to 80 characters to be processed when you press the function key in the Key string box.
2. These characters may be handled as if they were entered into the keyboard, or to specify Host Integrator commands. Use the Home, End, arrow keys, or the mouse, to quickly edit a long string. To delete characters, use Backspace or Delete.
2. To include escape sequences and ASCII control codes in the user key string, select **Insert special characters**. If you're using the Tab key to tab through the dialog box fields, you must clear the Insert special characters check box; otherwise you'll insert the ASCII tab character each time you press the Tab key.
2. The following table shows some examples of keys and key combinations that create certain escape sequences (shown by the two-letter mnemonic that appears on your screen):

Press this...	To include this sequence
Enter	CR
Tab	HT
Backspace	BS
Esc	EC
Ctrl+Q	D1
Ctrl+S	D3
Ctrl+E	EQ
Ctrl+X	CN

 **Note**

To remove a special character, you must first clear the **Insert special characters box**.

1. In the two lines provided in the **Label** box, type up to eight characters per line for the user key label. Use spaces to position and center text in the label. You can also use the editing keys (for example, Ins and Del).
2. Assign an attribute to the user key in the **Attribute** box
2. **Normal** – The key string is treated exactly as if it had been typed from the keyboard; a carriage return is not automatically transmitted.
2. If the Host Integrator is in local mode, the string displays on the screen and any embedded escape sequences are executed locally. In remote mode with local echo off, the string is transmitted to the host. It executes and displays only if the host system echoes. For example, use this attribute to store commands that have changing parameters (like the phone number needed to dial different modems).
2. **Local** – The key string is executed locally, and not transmitted to the host. For example, you could assign a user key to home the cursor and clear the display.
2. **Transmit** – In remote mode, the Host Integrator sends the key string to the host after completing a block transfer handshake, automatically transmitting a carriage return. For example, use this attribute to store commonly used commands (such as program run commands) in a user key. In local mode, pressing a user key with this attribute has no effect.
3. If you don't want the Host Integrator to display the function keys and labels along the bottom of the screen, clear the **Show function keys** check box.

Advanced HP Telnet

See [Advanced VT or HP Telnet](#) in the Customizing VT section.

Port Number

Select one of the following depending on which transport type you selected in the Session Setup or New Model dialog box:

- Telnet port

Specifies the host port or socket number that the Telnet session should use. You can enter any number between 0 and 65,535 in this field. You can configure Port in either the Session Setup or View Settings dialog boxes. The default port number for Telnet is 23.

- NS/VT portion

Specifies the host port or socket number that the NS/VT session should use. You can enter any number between 0 and 65,535 in this field. You can configure Port in either the Session Setup or View Settings dialog boxes. The default port number for NS/VT is 1570.

!!! note NS/VT is a proprietary HP protocol that connects to HP3000 hosts only.

Identifying Commands

A command can be a host keystroke, a Design Tool command, or any combination of these.

Creating a Custom Code Page

You can create a custom code page for 3270 and 5250 models by modifying any of the sample code page files provided with Host Integrator. By default, these examples are located in `<VHI folder>\codepage` folder for the Design Tool.

The basic steps for creating and implementing a custom code page are:

Create a text file that includes the customization. The easiest way to create this file is to open an appropriate code page sample and make a modification.

Save the modified file as `custom.codepage` in the same folder as the code page samples for the Design Tool. For a server version, add `custom.codepage` in a `codepage` directory beneath the server directory.

After saving `custom.codepage`, restart the Host Integrator server or Design Tool.

Editing a Code Page

A code page is a text file that includes character mappings or property values that differ from the default set (US English).

The property values at the top include the character set ID, the IBM code page, and, in the case of a 5250 model, the keyboard type. Additional information for specifying the Windows font when displaying international characters in the Design Tool can be included. The `CanadianFrench.codepage.sample` is used in the example below.

Example

```
//=====
// VHI Custom 3270/5250 character translation table sample file.
// IBM EBCDIC codepage (CanadianFrench)
//
// Note: To enable this national character set for 3270 and 5250 sessions,
// copy and rename this file to "custom.codepage"
//
// Copyright..... Copyright (c) 2019 Micro Focus
// All Rights Reserved.
//=====
IBMCCSID=697 // IBM Character set ID
BMCodePage=37 // IBM codepage
BMAS400Keyboard=CAI // AS/400 Keyboard type
WindowsFontCharSet=0 // (ANSI)
```

If, for example, you want to create a custom code page that has Canadian French settings except for the keyboard type, you could change the keyboard type from CAI to USB and then save it as `custom.codepage`.

```
BMAS400Keyboard=USB // AS/400 Keyboard type (custom)
```

The lower portion of the code page sample is a series of character mappings. A character mapping consists of 3 values on a single line, separated by spaces or commas:

Host (EBCDIC) value, prefixed with H. They can be specified as decimal or hexadecimal numbers.

Windows single-byte value used to display a given character in the Design Tool, prefixed with W. They can be specified as character literals, decimal values, or hexadecimal values.

6 bit Unicode character, prefixed with U. They can be specified as character literals when they are 8 bits or less, decimal, or hexadecimal values.

In the samples, mappings that are the same as the US English default are commented out with // at the beginning of the line. For example, the French.codepage.sample has H44 mapped to W '@' and U '@' for the 'commercial at' sign and is not commented out. (In US English, H44 is mapped to lowercase a with a grave accent sign.) You could create a custom code page based on the French code page, with a modification that returns H44 to the US English default.

Using the Custom Code Page Log File

After creating and saving the code page and then restarting the Host Integrator server or Design Tool, look at the log file for the code page: `<Host Integrator install folder>\codepage\codepage.log`.

When the code page is deployed successfully, the log looks like this:

```
-----  
Custom codepage logfile opened: Thu Jun 22 08:17:18 2006  
-----  
  
Info Duplicate character mapping: We8 ==> U0e48  
Info Duplicate character mapping: We9 ==> U0e49  
Info Duplicate character mapping: Wea ==> U0e4a  
Info Duplicate character mapping: Web ==> U0e4b  
Info Duplicate character mapping: Wec ==> U0e4c  
Info === Read 221 Lines of text  
Info === Found 99 host character mappings  
Info +Added Mapping: H41 <==> Wa0  
--  
--  
Info === Found 94 unicode character mappings  
Info +Added Mapping: : W1d <==> U00a2  
Info +Added Mapping: : W1e <==> U00a6  
  
-----  
Custom codepage logfile closed: Thu Jun 22 08:17:18 2006  
-----
```

When a code page has errors, the log file includes warnings such as the ones shown below:

```
Warn Line 11 Unrecognized property "MyProperty"  
Warn Line 20 !Host value invalid or out-of-range.  
Warn Line 20 !Windows value invalid or out-of-range  
Warn Line 20 !Unicode value invalid or out-of-range.  
Warn Line 20 !Character mapping ignored "H3F U10000"  
Warn Line 22 !Host value invalid or out-of-range.  
Warn Line 22 !Windows value invalid or out-of-range  
Warn Line 22 !Unicode value invalid or out-of-range.  
Warn Line 22 !Character mapping ignored "HH WW UU"
```


4.4.3 Mapping the Keyboard

Mapping the Keyboard

You can customize your keyboard to associate keystrokes with Design Tool commands or terminal keystrokes. This process is known as keyboard mapping.

A keystroke can be:

- A single key, such as K, F1, or Num Lock

- A combination of keys that you press at the same time, such as Ctrl+F2 or Alt+Shift+M

When you use more than one key in a keystroke, all keys preceding the final key must be modifier keys—Alt, Ctrl, or Shift. You can create keystrokes with a single modifier key (Ctrl+F7) or with multiple modifier keys (Shift+Ctrl+F7).

The command that you map to a keystroke can be:

- A terminal key, such as Attention or PF1

- A Design Tool command, such as .aboutDlg or .Connect

- Any combination of the above—you can build the Command string to include multiple terminal keys, and commands.

Drag-and-Drop Keymap Options

The following are some shortcuts you can use in the Keyboard Setup dialog box:

- Click a PC keystroke and then, holding down the left mouse button, drag it to the terminal keyboard. Already mapped keys become small cyan rectangles; unmapped keys become a rectangle, but they don't change color. When you move this rectangle over a key on the terminal keyboard, a black outline appears around this key (indicating correct positioning over the key). The rectangular mouse cursor either retains its form (indicating that you can map to this key) or turns into a red circle with a line through it (indicating that you cannot map to this key).
- The same process works in reverse. Select a terminal keystroke, and then drag it to the PC keyboard.
- To remove mapping from a key, click the key, and then drag it off the Keyboard Setup dialog box. When you drag the key off the edge of the dialog box, the image of the key turns into a trash can; you can then release the left mouse button and the mapping is cleared.

CHANGING THE PC OR TERMINAL KEYBOARD

The Keyboard Type dialog box lets you specify which PC and Terminal keyboard are shown in the Keyboard Setup dialog box. On the Settings menu, click Keyboard to open the Keyboard Setup dialog box, and then click Keyboards to open the Keyboard Type dialog box. The Keyboard Type dialog box contains two list boxes:

PC - Click the down arrow to see the available PC keyboards.

Terminal - Click the down arrow to see the available terminal keyboards.

Other options that affect keyboard mapping are available in the View Settings dialog box (available from the Settings menu).

Identifying a Keystroke

To identify a keystroke:

1. Click any modifier keys (Alt, Ctrl, or Shift) that you want to use in the keystroke. This step is optional. You can create keystrokes with no modifier keys (such as F7), with a single modifier key (Ctrl+F7), or with multiple modifier keys (Shift+Ctrl+F7).
2. Select a primary key.

This can be any key on the PC keyboard—except an unavailable key. Cyan keys are already mapped. To remap a key, click Remove, and then continue.

If your keyboard keys on your keyboard do not exactly match the PC keyboard keys shown in the Keyboard Setup dialog box, use your keyboard, instead of the mouse, to select a PC keystroke. That way, you'll be able to see how your keyboard keys correspond to the keys on the Design Tool's graphical PC keyboard. Select a key, click an empty spot anywhere under **PC Keyboard**, and use your keyboard to press the keys you want. Do not use your mouse to click any keys.

Note

You cannot remap the Windows logo function keys.

Removing a Keystroke's Mapping

Select the keystroke on the PC keyboard in the Keyboard Setup dialog box

Click the Remove button.

The key's mapping is cleared (so that pressing the keystroke has no effect in the Design Tool).

Resetting a Keystroke's Mapping

To reset a keystroke to its default setting...

Identify a PC keystroke in the Keyboard Setup dialog box.

Click the Default button.

Don't confuse the Default button with the Defaults button (which deletes all keyboard customizations).

This procedure only works for PC keystrokes mapped to an action other than the default action for the keystroke.

Restoring the Default Keyboard Mapping

To restore the default keyboard mapping, click Defaults in the Keyboard Setup dialog box.

All keyboard customizations are removed. For information on the default keyboard mapping for VT and HP terminals, click a mapped cyan/teal PC key and note the corresponding Action in the lower half of the Keyboard Mapping dialog box.

Determining Keyboard Mapping

Shading, color, and key appearance are used in the Keyboard Setup dialog box to show how your keyboard is mapped. Cyan keys are mapped, by default, but their mappings can be changed.

Keys not appearing in cyan are not mapped. (Letter and number keys are not considered mapped because it is unlikely you would want to redefine the mappings for these keys.)

If you want to see how a key on the PC is mapped, click the key to select it; the key is selected. If this PC key is mapped to a terminal key, the corresponding key on the terminal keyboard, below, is also selected. For example, click the F1 key on the PC keyboard. It changes color, as does the F1 key (in 5250 sessions) or the PF1 key (in 3270 sessions) on the terminal keyboard. Click F1 on the PC keyboard again to clear the key.

Note

Num Lock key status affects the mapping: If you click the Num Lock key while the Keyboard Setup dialog box is open, you must exit and return to the dialog box before that change is reflected in the displayed keyboard mapping.

Some keys are mapped to Design Tool commands. For example, click the Caps Lock key on the PC keyboard. The terminal keyboard is replaced by a set of fields.

Caps Lock is mapped to the Design Tool command `.Toggle(rc_CapsLockState)`, which turns capital letters on or off.

Click Caps Lock again to clear it.

If you click a modifier key (Alt, Ctrl, or Shift) on the PC keyboard, the color pattern on the remaining keys changes. For example, try clicking Alt. The keys that turn cyan are mapped to Alt. For example, in 5250 sessions, the F1 and D keys become cyan when you click Alt. This means that Alt+F1 and Alt+D are mapped in the Design Tool's default keyboard mapping.

Keys that appear dimmed when you click Alt, Ctrl, or Shift are currently mapped (in combination with the selected modifier key) and cannot be changed. Typically, these are keystrokes that are defined by Windows. For example:

Keystroke	Map to setting
Alt+F4	Close the current window (Windows standard)

Keystroke	Map to setting
Ctrl+Esc	Display the Windows task list (Windows standard)

Linking a Keystroke to a Command

Once you have identified a PC keystroke and one or more commands to map it to, click the Map button.

MAPPING A KEYSTROKE

There are three things you need to do in the Keyboard Setup dialog box to map a keystroke:

- Identify the keystroke
- Identify one or more commands
- Link the keystroke to the commands

With the Design Tool's Keyboard Setup dialog box, you can identify either the keystroke or the action first, but the linking process must be the final step.

Keyboard settings can be saved to a settings file (`.dtool`) with other configuration information.

Mapping a Keystroke

There are three things you need to do in the Keyboard Setup dialog box to map a keystroke:

- Identify a keystroke
- Identify one or more commands
- Link the keystroke to the command

With the Design Tool's Keyboard Setup dialog box, you can identify either the keystroke or the action first, but the linking process must be the final step. Keyboard settings can be saved to a settings file (`.dtool`) with other configuration information.

TO IDENTIFY A KEYSTROKE

1. Click any modifier keys (Alt, Ctrl, or Shift) that you want to use in the keystroke.
 1. This step is optional. You can create keystrokes with no modifier keys (such as F7), with a single modifier key (Ctrl+F7), or with multiple modifier keys (Shift+Ctrl+F7).
2. Select a primary key.
 2. This can be any key on the PC keyboard—except an unavailable key. Cyan keys are already mapped. To remap a key, click Remove, and then continue.
2. If your keyboard keys on your keyboard do not exactly match the PC keyboard keys shown in the Keyboard Setup dialog box, use your keyboard, instead of the mouse, to select a PC

keystroke. That way, you'll be able to see how your keyboard keys correspond to the keys on the Design Tool's graphical PC keyboard. Select a key, click an empty spot anywhere under **PC Keyboard**, and use your keyboard to press the keys you want. Do not use your mouse to click any keys.

 **Note**

You cannot remap the Windows logo function keys.

REMOVING A KEYSTROKE'S MAPPING

Select the keystroke on the PC keyboard in the Keyboard Setup dialog box

Click the Remove button.

The key's mapping is cleared (so that pressing the keystroke has no effect in the Design Tool).

RESETTING A KEYSTROKE'S MAPPING

To reset a keystroke to its default setting:

Identify the keystroke in the Keyboard Setup dialog box.

Click the Default button.

Tip

Don't confuse the Default button with the Defaults button (which deletes all keyboard customizations).

This procedure only works for PC keystrokes mapped to an action other than the default action for the keystroke.

RESTORING THE DEFAULT KEYBOARD MAPPING

To restore the default keyboard mapping, click Defaults in the Keyboard Setup dialog box.

All keyboard customizations are removed. For information on the default keyboard mapping for VT and HP terminals, click a mapped cyan/teal PC key and note the corresponding Action in the lower half of the Keyboard Mapping dialog box.

DETERMINING KEYBOARD MAPPING

Shading, color, and key appearance are used in the Keyboard Setup dialog box to show how your keyboard is mapped. Cyan keys are mapped, by default, but their mappings can be changed.

Keys not appearing in cyan are not mapped. (Letter and number keys are not considered mapped because it is unlikely you would want to redefine the mappings for these keys.)

If you want to see how a key on the PC is mapped, click the key to select it; the key is selected. If this PC key is mapped to a terminal key, the corresponding key on the terminal keyboard, below, is also selected. For example, click the F1 key on the PC keyboard. It changes color, as does the F1 key (in 5250 sessions) or the PF1 key (in 3270 sessions) on the terminal keyboard. Click F1 on the PC keyboard again to clear the key.

Note

Num Lock key status affects the mapping: If you click the Num Lock key while the Keyboard Setup dialog box is open, you must exit and return to the dialog box before that change is reflected in the displayed keyboard mapping.

Some keys are mapped to Design Tool commands. For example, click the Caps Lock key on the PC keyboard. The terminal keyboard is replaced by a set of fields.

Caps Lock is mapped to the Design Tool command `.Toggle(rc_CapsLockState)`, which turns capital letters on or off.

Click Caps Lock again to clear it.

If you click a modifier key (Alt, Ctrl, or Shift) on the PC keyboard, the color pattern on the remaining keys changes. For example, try clicking Alt. The keys that turn cyan are mapped to Alt. For example, in 5250 sessions, the F1 and D keys become cyan when you click Alt. This means that Alt+F1 and Alt+D are mapped in the Design Tool's default keyboard mapping.

Keys that appear dimmed when you click Alt, Ctrl, or Shift are currently mapped (in combination with the selected modifier key) and cannot be changed. Typically, these are keystrokes that are defined by Windows. For example:

Keystroke	Map to setting
Alt+F4	Close the current window (Windows standard)
Ctrl+Esc	Display the Windows task list (Windows standard)

Keystroke	Map to setting
Ctrl+Alt+Del	Restart system (DOS standard)

LINKING A KEYSTROKE TO A COMMAND

Once you have identified a PC keystroke and one or more commands to map it to, click the Map button.

4.4.4 Setting Colors and Fonts

Configuring Colors

The Colors tab in the Display Setup dialog box lets you associate colors with screen attributes in the Terminal window.

Color settings can be saved to a settings file (`.dtool`) with other configuration information.

SETTING COLORS WITH A MOUSE

While the Colors tab is displayed, move your cursor outside of it. The cursor now looks like a pointing hand.

Place the cursor over text in the Terminal window and click the left button. In the Colors tab, the **Item** box shows the attribute of the text you clicked.

Move the cursor back into the dialog box and click a color in the **Foreground (text)** box. All text with the attribute shown in the **Item** box now appears in the new color.

Alternatively, you can click a color first and then, holding down the left mouse button, drag the cursor off the Colors tab. In this case, the cursor becomes a paint brush.

As you move the paint brush cursor across the Terminal window, the value in the **Item** box changes to show the attribute of the text beneath the paint brush.

When the cursor is over an area whose color the Design Tool cannot change, the paint brush cursor looks like a paint brush with a slashed circle covering it.

When you release the mouse button on text with an appropriate attribute, the text--and any additional text that shares this attribute--are set to the specified color.

Click Defaults if you want to reset all colors to their default values. Click Save on the File menu to retain the new color settings after you exit the Design Tool.

SETTING COLORS FOR AN ATTRIBUTE

From the Settings menu, select Display... to open the Display Setup dialog box.

In the Item box, select a value. The Sample Text window shows the current colors for the selected item—whether or not any part of your current Terminal window actually has the same attribute.

In the Foreground (text) box, select a color. Any text matching the selected attribute immediately changes color as you move from color to color.

You can return to the Item box and set a color for a different attribute, or click OK to exit the Display Setup dialog box.

Click Defaults to reset all colors to their default values.

Configuring Display Fonts

Use this dialog box to specify the display font and style in the Design Tool.

Changing the Design Tool's font has no effect on the font used by your printer, the status line, menu commands, or dialog boxes.

Font

Specifies the display font in the Design Tool session window. You can use any font, including TrueType fonts. Arial is an example of a TrueType font that is not monospaced, and therefore is not supported by the Design Tool.

By default, the Design Tool uses the `r_ansi` font. This provides a 24 x 80 display that accurately emulates the terminal.

When you resize the Terminal window, the Design Tool chooses a new font size so the correct number of rows are displayed on the screen. If the Design Tool cannot display the font you have chosen, the default (`r_ansi`) is displayed instead.

The available fonts can change as you change the model ID in the Session Setup dialog box, because not all model IDs support all fonts.

Note

When you print all or part of a host screen from a terminal session, the font used is the currently configured display font.

Font Style

Specifies a font style, regular or bold. The Design Tool does not support italicized fonts. The default is Regular.

Display variable width fonts

Specifies whether proportionally-spaced font types appear in the Fonts box. The default is Yes.

Auto sizing

Specifies that for whichever font or style you select, the Design Tool automatically adjusts the font size to fit all text in the Terminal window. To change the font size, set this setting to No. By default, Auto Font Size is enabled.

FONTS INSTALLED BY THE HOST INTEGRATOR

The following files are installed automatically by the Host Integrator installation program:

Font Type	Description
Dialog.fon	The font used in the Design Tool's dialog boxes.
_ansi.ttf	The file for the TrueType versions of the r_ansi fonts.
R_ansi.fon	The file for the bitmap versions of the r_ansi fonts.
R_apl.ttf	The file for the TrueType versions of the r_apl fonts. You can't select this font from the Fonts tab—it's used to display characters in the Operator Information area (OIA) only.
R_contr.fon	You can't select this font from the Fonts tab—it's used to display control characters in VT and HP terminal emulation sessions.
R_contr.ttf	You can't select this font from the Fonts tab—it's used to display control characters in VT and HP terminal emulation sessions.
R_ibmasc.fon	The r_ibmasc font.
R_ibmhex.fon	You can't select this font from the Fonts tab—it's used only when you're in hexadecimal mode.
R_ibmsl.fon	The font used to display host symbols and status lines.
R_symbo_.fon	The file for the symbol fonts.

Font Type	Description
R_symbol.ttf	The file for the TrueType versions of the symbol fonts.

To use a monospaced (fixed-pitch) font other than Host Integrator's default display font, `r_ansi`, that font must already be installed under Windows. If the font you want to use doesn't appear in the Font box in the Display Setup dialog box, you need to install it using Windows Control Panel. Refer to your Windows documentation for information on installing new fonts.

4.4.5 Command List Edit

Use the Command List Edit dialog box to create a login, a logout, or a move cursor command list to be saved in the model file. To view a detailed example of a login and logout command list, see the Pine model example.

Command lists will be generated if operation generation is on (this is the default).

Configure your login or logout command lists to be automatically executed on a host connect or disconnect by selecting the Execute login command list and Execute logout command list check boxes on the General tab of the Preferences Setup dialog box.

Select a command from the Command list and press the F1 key for online help on that command.

Use the Copy button to grab an image of the whole command list, which you can then paste into a text editor making the entire list visible.

To create a login command list

On the Connection menu, select Session Setup to open the Session Setup dialog box.

In the Session box, select the terminal type from the Type box.

Type the host name or IP address in the Host name or IP address box and click Connect.

On the Model menu, point to Record and click Start Recording. Type your username and password and advance to the next screen.

On the Model menu, point to Record and click Stop Recording.

On the Stop Recording dialog box, select Save as login command list and click Save to open the Command List Edit dialog box.

View and edit the commands associated with the login command list that you've just created in the Commands box and click OK to save it.

To view and edit the login command list, use the buttons next to the Login (after host connect) box in the Model Properties dialog box.

To execute your login command list, disconnect and then reconnect to the host, and click the Execute login command list button on the standard toolbar. Alternatively, you can open the Model Properties dialog box and click the Execute button.

To create a logout command list

Move to the home entity of the model.

When you're ready to logout of the host, click Start Recording.

Type the command used to logout of the host on the terminal screen.

On the Stop Recording dialog box, select Save as logout command list and click Save to open the Command List Edit dialog box. By default, the Stop Recording dialog box opens if you are disconnecting from a host while recording.

View and edit the commands associated with the logout command list that you've just created in the Commands box and click OK to save it. Delete the `.MoveCursor` command.

To view or edit the logout command list, use the buttons next to the Logout (before host disconnect) box in the Model Properties dialog box. Make sure to save your model before exiting.

To execute your logout command list, disconnect and then reconnect to the host, and click the Execute logout command list button on the standard toolbar. Alternatively, you can open the Model Properties dialog box and click the Execute button.

Notes:

- It is recommended that you create a logout command list on your home entity. If you record a logout command list from an entity other than the home entity, it may not return the expected results when run in either the Design Tool or on the server. This happens because the default behavior of the logout command is to navigate to the home entity and then execute the logout command list. If you are not on the home entity when recording the logout command list, the following message will appear: "Execution of the logout command list will always start at the home entity. Since recording started at a different entity, we will add a 'Navigate to <name> entity' command to the start of your command list. At run-time, this may cause unnecessary host navigation. Do you want to continue?"
- If you create a model that uses a login command list that goes past at least one entity to reach the home screen entity, it is not necessary to exit the model using the same sequence. For example, a login command list may include a sequence to reach the home entity using signon entity "A," main menu "B," and home entity "C." The appropriate logout command list should, however, should proceed directly from home entity "C" to signon entity "A."

To create a move forward or move backward command list

1. Click the right arrow button and select the terminal key you want to use to move the cursor forward.
1. Select Tab from the Key list and then configure your tab position settings using the Cursor tab on the Entity window.
2. Click OK to save your move forward command list and return to the Advanced Model Properties dialog box.

Follow the same directions to create a move backward command list.

Build Command

The Build Command dialog box provides the following information about the selected command:

Description - displays a read-only description

Syntax - provides a read-only text description

Parameters - contains the values that determine the characteristics or the behavior of the command

4.5 Importing Model Elements

Elements of an existing model can be imported into other models. Developers can work simultaneously on the same model, importing the different elements to create a new model. You can also reuse parts of a model, streamlining your work and letting multiple developers work efficiently.

Settings and properties are not inherited from the source model. You can reconfigure them in the destination model if you want to keep the existing settings and properties.


[Terminology and Best Practices for Importing Models](#)

[Copy Objects](#)

[Copy Entity](#)

4.5.1 Workflow

1. Open the destination model in the Design Tool.
1. The destination model is the model that you are importing to. It is a good idea to always backup your model before importing model elements.
2. From the File menu, choose **Import Model Elements** to open the Import Model Elements dialog box. You can open either a `.modelx` file or a `.model` file.
3. Browse to the import model. You import from the source model (in the dialog box) to the destination model, which is open in the Design Tool.
3. The left pane displays a tree representation of the source model, including all entities, tables, and variables.
4. Choose the elements of the model you want to import into the destination model.

 **Note**

Model elements are those items that can be imported and comprise entities, attributes, operations, recordsets, recordset fields, tables, table columns, procedures, compound procedures, and model variables. When an operation containing a `WaitForMultipleEvents` command is imported, the referenced Host Events are also imported.

Patterns and events cannot be imported separately from their entity.

5. Preview your selected elements in the Destination pane. This pane is divided into the following elements.
5. **Insert elements** are new elements that are added to the model when the import is complete.
5. **Update elements** are elements that are updated after the import is complete.
5. **Referenced elements** are elements that are referenced by other elements. If these are not resolved, either manually or automatically, a temporary replacement version of the element is created. These replacement elements are marked with a `_notImported` suffix.
6. Click OK.

 **Note**

If an element already exists in the destination model, importing it overwrites the current element.

By default, the destination model with all imported model elements is validated using the Host Integrator Validator.

7. Resolve any validation errors.
8. If you are importing a model that contains an event handler, copy the Java code from the model source Scripts sub-directory to the corresponding sub-directory in the destination model.

A reference to the event handler is imported along with the parent element.

4.5.2 Terminology and Best Practices for Importing Models

Elements of an existing model can be imported into other models. Developers can work simultaneously on the same model, importing the different elements to create a new model. You can also reuse parts of a model, streamlining your work and letting multiple developers work efficiently.

Terminology

- Destination model

The destination model is the model that you are importing to and the model which is open in the Design Tool.

- Master model

After the import is complete, the new model that was created by importing elements from the source model into the destination model is called the master model.

- Source model

You import from the source model to the destination model. The source model is selected in the Model Import dialog box.

- Model elements

Model elements are those items that may be imported and comprise entities, attributes, operations, recordsets, recordset fields, tables, table columns, procedures, compound procedures, and model variables.

- Referenced elements

A referenced element is referenced by another element, but not selected for import. All referenced elements must be resolved, either manually or automatically, before the new model can be deployed.

Note

If the option **Automatically select referenced elements** is selected on the Preferences dialog box, then `_notImported` elements do not occur. If this option is cleared, then these replacement elements must be resolved manually.

Best Practices

- Always backup your model before you begin the import process.
- Understand your model.
- Elements that are identified as `_notImported` in the preview pane of the Model Import dialog box are not problems. It is not unusual to have elements that you do not want to import.

An element, for example an attribute, might reference another element, such as a variable. When the attribute is imported, but the variable isn't, then the referenced element is missing. The import function creates a temporary replacement version of the referenced element. These replacements are marked with the `_notImported` suffix and inserted into the destination model.

Since all referenced elements must eventually be resolved before a model can be deployed, after finishing the import, use the Design Tool to modify the attribute you have just imported. Find the reference to the placeholder variable and change it to refer to a "real" variable in your model.

- Create a base model with all the entities up to the home entity. This model should include a model of lifecycle event handlers.

If all the screens you want to model are known in advance, it is beneficial to include the entities that will be duplicated across different models in the base model. This prevents a particular entity from having different names across different models and reduces the size of the model.

- Developers should use the base model and extend it to include other model elements. The model should be given a unique name. For example, `<DeveloperInitials>`
`_<BaseModelName>`.
- Add a prefix to the names of event handlers to make them unique to each model. This ensures that there are no clashes during the import process.
- Code freeze both models during the import merge process.

Test the destination (master) model when the import is complete.

Provide the master model to the developers to continue the task of extending the model.

Frequently Asked Questions

- Can I import models with event handlers?

Yes. If you import a model that contains an event handler, a reference to it is imported along with the parent element. However, you must manually copy the Java code from the model

source Scripts sub-directory to the corresponding sub-directory in the destination model files.

- How do I make sure I don't overwrite existing elements during import?

When you select an element that is already in the destination model the element is replaced. However, you must confirm your decision before the element is overwritten. This option is set on the **Import** tab of the Preferences Setup dialog box and is selected by default.


- Do I have to validate my new master model?


By default, all import models are validated in the Host Integrator Validator. A model must successfully validate before it can be deployed.

- Can I change how sub-items and referenced items are selected in the Model Import pane?

Options that determine how to work with the Import Model Elements dialog box are set on **Settings > Preferences > Import**. See [Import Preferences](#) for a description of each option, as well as the default settings.

- What do the  green and  yellow equal signs mean?

All of the element's properties in the source model are the same as those in the destination model. For example, if an entity exists in both models and all of the entity's sub-items are also marked with a  green equal sign, you do not have to import this element.

However, a  yellow equal sign indicates that the element properties are the same, but there are differences between one or more sub-items of the element.

More information





[Creating a model](#)

4.5.3 Copy Objects

You can copy objects you have already created for reuse. Copying patterns, attributes, or recordsets allows you to quickly create new model objects based on objects that you have already defined within a given entity.

You can copy patterns, attributes, and recordsets as described below. You can also copy tables and procedures, and complete entities can be replicated as shown in [Copy Entity](#).

1. From the drop down list attached to the following buttons, click Create Copy:

-  Attribute
-  Recordset
-  Pattern
-  Operation

8. The copied object has a similar name to the original object with a number appended (for example, Password_2).

2. Make the necessary modifications to the new copy.

Avoiding Problems

Use these best practices to avoid common problems:

If you copy a pattern that is configured to be a signature pattern for an entity, edit the pattern characteristics to avoid invalid signatures.

When you copy an entity, you must identify a signature that uniquely identifies the entity. It cannot have the same signature as the source entity you copied from.

Avoid copying an object that is relative to a pattern, since the relative relationship is usually not the same for the new object.

Copying or importing operations that act on entity-specific objects may not function as originally designed, even if each command in the operation is copied successfully.

4.5.4 Copy Entity

You can copy entities you have already created. This is a fast way to build models that contain multiple occurrences of similar screens.

To Copy an Entity

1. Click **Copy From** on the options list next to the **Entity** button. This option is available when you are on an undefined screen.
2. Select the entity to copy.

 **Note**

On the left side of the Copy Entity object dialog box is a list of entities within this model. Entities that cannot be copied are marked with a red X.

3. Confirm that this item can be copied as you expect.
3. The right pane's Snapshot tab includes a snapshot of the selected entity. The Entity tab shows the name of the source entity and the objects within the entity.
3. The **Copy status** column indicates whether the entity can be copied. If an object cannot be copied (for example, you are trying to copy an attribute from a 132 column entity to an 80 column entity), an error icon is displayed.
4. Click **Copy**.
5. Edit the resulting entity.

At a minimum, you must identify a signature that uniquely identifies the entity. It cannot have the same signature as the source entity you copied from.

Modify any patterns that are defined for the source entity and are not present in the target entity. They are listed with a red X to the left of the pattern name.

More information

[Adding Entities to a Model](#)

4.6 Adding Entities to a Model

An entity is a unique host application screen. After you label your host application screen as an entity, you must define it using entity definitions and entity properties.

Note

Object names for entities, attributes, operations, recordsets, recordset fields, and, variables are not case sensitive.

TO ADD AN ENTITY TO A MODEL:

1. When the first screen appears after connecting to the host, click the **New Entity** button in the Entity window. (All of the other options are unavailable at this point.)
2. By default, the entity is named **Entity_1**. To change the name, select it and type a new name in the Entity box.
3. Click **Apply** to save your changes.
4. Now, add a pattern to the entity. See [Adding Patterns](#).

You can also copy entities as shown in [Copy Entity](#).

4.6.1 Adding Patterns

A pattern is a selected area on an entity that does not contain data that changes from session to session.

TO ADD A PATTERN TO AN ENTITY:

1. After you've created an entity, use the cursor to select a static section of the host screen to create a pattern.

1. Each pattern can have up to 259 characters.

Tip

You can choose to auto-generate patterns by clicking the **Auto-generate** button, but you will still have to assign properties and error conditions to these patterns on the Pattern tab.

2. On the Pattern tab, click the **New Pattern** button. All of the other options on this tab are unavailable at this point.

3. By default, the pattern is named **Pattern_1**. To change the name, select it and type a new name in the **Name** box. The Design Tool has now recorded the position, properties, and the signature parameters of the pattern.

4. Click **Apply** to save your changes.

5. Now, add attributes to identify text fields on the entity as shown in [Adding Attributes](#).

TO COPY A PATTERN:

After you have created a pattern, you may need to create another that is similar. You can copy the pattern and then make the necessary modifications to the copy.

Select the pattern you want to copy.

Click the list next to the **New Pattern** button and then select **Create Copy**. A new operation with a similar name and a number appended to the end is created. For example: Pattern_2_2.

Modify the new pattern so that it is distinguishable from the pattern on which it is based.

Tip

If you copy a pattern that is configured to be a signature pattern for an entity, be sure to edit the pattern characteristics to avoid invalid signatures.

4.6.2 Adding Attributes

An attribute is a selected area on an entity containing data that needs to be accessible via the model file.

After you've created an entity and added pattern to that entity, you're ready to add an attribute or attributes.

Tip

You can choose to auto-generate attributes by clicking the Auto-generate button, but you will still have to assign variables, properties, and error conditions to these attributes on the Attribute tab.

TO ADD AN ATTRIBUTE TO AN ENTITY:

1. Use the cursor to select a text field on the host screen.
2. On the Attribute tab, click the **New Attribute** button. All of the other options on this tab will be unavailable.
3. By default, the attribute is named **Attribute_1**. To change the name, select it and type a new name in the **Name** box.
3. The Design Tool records the position and other defining characteristics of the attribute.
3. You can accept the Design Tool's default properties, or change them on the Attribute tab.
4. Click **Apply** to save your changes.

Some entities require that you add recordset information. After adding these elements to an entity, you can define operations to navigate between entities.

TO COPY AN ATTRIBUTE:

After you create an attribute, you may need to create another that is similar. You can copy the attribute and then make the necessary modifications to the copy.

Select the attribute you want to copy.

Click the list next to the **New Attribute** button and select **Create Copy**. A new attribute with a similar name and a number appended to the end is created (for example, `Password_2`).

Modify the new attribute so that it is distinguishable from the attribute on which it is based.

TO ADD COLORS TO AN ATTRIBUTE:

1. From the **Settings** menu, select **Display...** to open the **Display Setup** dialog box.
2. In the **Item** box, select a value.
2. The Sample Text window shows the current colors for the selected item and whether any part of your current Terminal window actually has the same attribute.
3. In the **Foreground (text)** box, select a color.
3. Any text matching the selected attribute immediately changes color as you move from color to color.
4. To set a color for a different attribute, return to the **Item** box.
5. Click **OK** to exit the Display Setup dialog box.
6. Click **Defaults** to reset all colors to their default values.

More information

[Customizing the Terminal](#)

4.6.3 Adding Recordsets to an Entity

TO ADD A RECORDSET TO AN ENTITY:

1. On the Recordset tab, select a scrollable area on the terminal screen and click the **New Recordset** button in the **Name** box.
2. By default, the recordset is named **Recordset_1**. To change the name, select it and type a new name in the **Name** box.
3. Configure your recordset using the **Recordset Position**, **Layout**, and **Fields** tabs.
4. Click **Apply** to save your changes.

To copy a recordset:

After you have created a recordset, you may need to create another that is similar. You can copy the recordset and then make the necessary modifications to the copy.

Select the recordset you want to copy.

Click the list next to the **New Recordset** button and select **Create Copy**. A new recordset with a similar name and a number appended to the end is created (for example, `AcctTransData_2`).

Modify the new recordset so that it is distinguishable from the recordset on which it is based.

Testing Recordsets

This is an example of testing a fetch record in the CICSAccts model.

On the File menu, choose **Open** and select `CICSAccts.modelx`.

Click **Return** to connect.

From the **Entity** list on the Entity window, select **NameSearchResults**.

Select the **Recordset** tab and click the **Test** button in the **Name** box.

In the Test Recordset dialog box, select **Fetch Records** from the **Action** box.

Click the **Execute** button.

View the results of the data fetch test in the **Fetch returned** box.

Fetch Records

TO TEST A RECORD FETCH:

Open `CCSDemo.modelx` in the Design Tool.

On the Connection menu, choose **Connect** to localhost via Telnet.

From the **Entity** list on the Entity window, select **NameSearchScreen**.

Select the **Recordset** tab and click the **Test** button in the **Name** box.

In the Test Recordset dialog box, select **Fetch Records** from the **Action** box.

To filter out a record or records from within a recordset, click the **Edit** button to open the Filter String Edit dialog box.

Click the **Execute** button.

View the results of the data fetch test in the **Fetch returned** box.

Note

After executing the first **Fetch Records**, you must reset the recordset by executing the **Set Current Record Index** action.

Inserting a Record

TO INSERT A SPECIFIC RECORD OR MULTIPLE RECORDS INTO A RECORDSET:

Confirm that SIDemo is running in the Host Emulator, then open `SIDemo.modelx` in the Design Tool.

On the **Connection** menu, click connect to localhost via Telnet.

From the **Entity** list on the Entity window, select **CustomerPurchases**.

Click the Recordset tab and select **CustomerList** from the **Name** box.

Click the Test button to open the Test Recordset dialog box.

Select **Insert Record** from the **Action** box.

In the **Insert record** box, notice that the **Select** and **Customer** fields have been created for this recordset and are listed in rows.

To insert a record, under the **Value** column, type `B. JONES` into the **Customer** text box.

Click **Execute**. If the insert is successful, the record contents should appear in the Terminal window.

Note

After you execute an insert on a recordset, you need to reset the recordset by either navigating away from it and returning, or executing the home operation on the recordset before executing any other method on it.

Selecting a Record

There are two possible ways to select a record: by condition or by index. The following procedures use `SIDemo.modelx` as an example.

TO SELECT A SPECIFIC RECORD BY CONDITION IN A RECORDSET:

Confirm that `SIDemo` is running in the Host Emulator, then open `SIDemo.modelx` in the Design Tool.

On the Connection menu, click **Connect to localhost via Telnet**. Confirm that `SIDemo` is running in the Host Emulator, then open `SIDemo.modelx` in the Design Tool.

In the **Entity** box, select **CustomerPurchases** and click the Recordset tab.

Click the **Test** button to open the **Test Recordset** dialog box.

From the **Action** box, select **Select Record**.

Click the **Edit** button to open the Filter String Edit dialog box.

In the Filter String Edit dialog box, select **Customer** from the **Recordset fields** box, click the **=** button, and type the following in the **Filter string** box: `Q. ARMSTRONG` The filter string should appear as follows: `CustomerList.Customer = "Q. ARMSTRONG"`

Click **OK**.

In the **Test Recordset** dialog box, click **Execute**.

If the selection is successful, the record contents for `Q. ARMSTRONG` are displayed on the terminal screen along with the following message:

```
The selection operation reached an expected destination.
```

Note

This action is the analog of the `SelectRecordByFilter` method.

If the record exists on an entity that does not contain any defined recordsets, the Test Recordset dialog box closes.

TO SELECT A SPECIFIC RECORD BY INDEX IN A RECORDSET:

This action is the analog of the `SelectRecordByIndex` method. If the record exists on an entity that does not contain any defined recordsets, the Test Recordset dialog box closes.

Follow the procedures described in Step 1 through Step 4 above.

From the **Action** box, select **Set Current Record Index**.

In the **Set the current record index to** box, type `3`, and click **Execute**. To check that your current record index is set to the line number you want, view the indicator (for example, **Current record index: <line number>**) at the bottom of this dialog box.

Select **Select Record** from the **Action** box and click **Execute**.

If the select is successful, the record contents for `Q. ARMSTRONG` is displayed on the terminal screen along with the following message:

```
The selection operation reached an expected destination.
```

Update Current Record

TO UPDATE THE CURRENT RECORD IN A RECORDSET:

Open `Purchases.modelx` in the Design Tool.

On the Connection menu, choose **Connect to localhost via Telnet**.

In the **Entity** box, select **CustomerPurchases** and click the Recordset tab.

Click the **Test** button to open the Test Recordset dialog box.

From the **Action** box, select **Set Current Record Index**.

In the **Set the current record index to** box, type `2`.

Click **Execute**. To check that your current record index is set to the line number you want, view the indicator (for example, **Current record index:** `<line number>`) at the bottom of this dialog box.

From the **Action** box, select **Update Current Record** and click **Execute**. Look in the **Update current record** box and notice that **Select** appears in the **Field** column and the **Value** column is blank.

9. In the **Value** column, Type `1` and then click **Execute**.

9. The value you entered (`1`), is displayed on line 3 of the **Select** column in the Terminal window.

Tip

To test any scrolling operations that have been defined, click any of the available buttons in the **Perform Scrolling Operation** box. If there are no scrolling operations in the recordset, the buttons are unavailable.

Set Current Record Index

TO SET THE CURRENT RECORD INDEX OF A RECORDSET:

Confirm that SIDemo is running in the Host Emulator.

In the Design Tool, open `SIDemo.modelx`.

On the Connection menu, click **Connect to localhost via Telnet**.

In the **Entity** box, select **CustomerPurchases** and click the Recordset tab.

Click the **Test** button to open the Test Recordset dialog box.

From the **Action** box, select **Set Current Record Index**.

In the **Set the current record index to** box, type `2`.

Click **Execute**.

To check that your current record index is set to the line number you want, view the indicator (for example, **Current record index:** `<line number>`) at the bottom of this dialog box.

From the **Action** box, select **Fetch Records** and click **Execute**. Look in the **Fetch returned** box and notice that records were fetched beginning with line 3: Q. ARMSTRONG appears in the **Customer** column.

Note

To test any scrolling operations that have been defined, click any of the available buttons in the **Perform Scrolling Operation** box.

4.6.4 Defining Entities and Recordsets for Procedures

Host Integrator fulfills SQL queries from client applications by navigating to the pertinent entities in a host application that contain the table data and either reading, modifying, or deleting the data. A procedure's definition must specify all entities and recordsets that contain table data.

In addition to defining the entities and recordsets that contain table data, you can also define branching entities. These entities provide some flexibility for a procedure when it is traversing through a host application, as well as error entities that you can use to trap errors in a procedure.

- Adding Entities to a Procedure

You must add every entity to a procedure that contains the attributes and/or recordset field that the procedure is able to query.

- Adding Recordsets to a Procedure

Recordsets are areas on an entity that contain dynamically changing information, usually scrolling sets of data that are a result of a data fetch.

- Adding Branch Entities to a Procedure

Use branch entities in procedures when an operation has alternate entity destinations. There are some cases where traversal is not deterministic, for example, when an operation has alternate destinations defined. These alternate destinations can be added as branch entities in a procedure. When the operation is executed at runtime, the path the procedure takes is determined by which branch entity is recognized after the operation completes. If none of the branch entities is recognized, the procedure fails.

- Adding Error Entities to a Procedure

Error Entities are screens containing patterns that indicate an error has occurred in the procedure. You can define error entities by purposely entering bad data in a host application and capturing the resulting screen as an entity. Adding one or more error entities to a procedure is a good way to build error checking into your model.

More information


[Inserting a Recordset](#)

[Inserting an Error Entity](#)

[Inserting a Branch Entity](#)

4.6.5 Creating Descriptions for Entity Definitions

The following options allow you to include descriptions in any exported documentation. This data is included in Web Builder projects or any documentation generated using [Export](#) options. This applies to Advanced Attribute Properties, Advanced Operation Properties, Advanced Recordset Properties, and Advanced Recordset Field Properties.

Click Advanced Properties  next to the **Name** box on the corresponding Attribute, Operation, Recordset, and Recordset Field tabs to open these dialog boxes.

4.7 Adding Operations to an Entity

After you've created an entity and have added patterns and attributes to that entity, you're ready to add or edit an operation.

Tip

Object names for entities, attributes, operations, recordsets, recordset fields, and, variables are not case sensitive.

TO ADD OR EDIT AN OPERATION:

On the Operation tab, click the **New Operation** button in the **Name** box. By default, the operation is named `Operation_1`.

To change the name, select it and type a new name in the **Name** box.

Configure your new operation in the **Destination** box, **Command list** box, and **Properties** boxes.

After configuring your operation, click **Apply** to save your changes.

TO COPY AN OPERATION:

You can copy the operation to create another that is similar.

Select the operation you want to copy.

Click the list next to the **New Operation** button and select **Create Copy**. A new operation with a similar name and a number appended to the end is created (for example, `ToMainMenu_2`).

Modify the new operation so that it is distinguishable from the operation on which it is based.

More information

[Adding entities to a model](#)

4.7.1 Mapping Operations

The Design Tool provides several commands that enable advanced configuration of entities in the host application model. To edit an existing operation, click the **Operation** tab and then click the **Edit** button to open the **Operation Edit** dialog box.

See [Operation command summary](#) for a complete list of available commands.

4.7.2 Reset Current Recordset

This operation can be used when multiple database records have the same entity definition but scrolling changes entity attributes and entity recordset contents. To ensure that new records are fetched, add the **Reset Recordset** command in a scrolling operation before the actual keystroke that performs the scroll.

TO USE RESET CURRENT RECORDSET:

1. On the Operation tab, create a Down operation and rename it `DownReset` . If you already have a Down operation on the entity, you can click **Copy Operation** and rename it `DownReset` . For an IBM host, the operation looks like this.

1. `CheckOperationConditions`
`TransmitTerminalKey rcIBMpf2Key`

 **Note**

For VT, the terminal key is `VTNextPage` ; for HP, the terminal key is `HPPageDown` .

2. Add the `ResetRecordset` command as the second line of the `DownReset` operation. The operation for an IBM recordset reset looks similar to the following:

2. `CheckOperationConditions`
2. `ResetRecordset "<recordsetname>"`
2. `TransmitTerminalKey rcIBMpf2Key`

More information

[Adding Recordsets to an Entity](#)

4.8 Tables and Procedures

4.8.1 Overview

Using tables and procedures in your host application model enables you to create a database abstraction of the host data. Client applications can then query this data using a Verastream connector to interact with the model.

A table is a structure that contains columns that are used as input and output parameters for procedures. Typically, table columns are named to match their corresponding model attributes and fields.

A procedure defines how Host Integrator locates, retrieves, updates, inserts, and/or deletes data when it fulfills a request submitted by a client application using a Host Integrator API.

Tables and procedures are interlocked functionally. Tables are a way of "organizing" host data into a database-like view of the data, and procedures manipulate that data. The only way to access the abstracted table data is through a procedure.

Client applications interact with tables and procedures using either a subset of the industry standard Structure Query Language (SQL), or by interacting directly with the procedures. You can access the data in your host applications via SQL queries even if your host applications are not designed to respond to SQL queries.

[SQL Overview](#)

[Tables Overview](#)

[Procedures Overview](#)

SQL Overview

With the Table/Procedure feature, you can abstract your host application so that client applications can perform queries using a subset of the industry standard Structure Query Language (SQL). This makes it possible for you to access the data in your host applications via SQL queries, even if your host application is not designed to respond to SQL queries.

In a query statement, client applications can select, update, insert, and delete data in the host application by specifying a context, a set of input parameters, and a set of desired output parameters in the form of an SQL statement. Using this statement Host Integrator determines the proper query to run and returns the desired results.

Use the table option to set up a table definition consisting of a set of columns. Use procedures to tell Host Integrator how to navigate to the host application screens where the data resides and pass any commands to the host application necessary to select, update, insert, or delete the host data. Once abstracted, the host application model responds to SQL queries from client applications in exactly the same way that a true SQL database does.

Note

Procedure execution does not require SQL—see [Executing Procedures Using Connector APIs](#).

HOW HOST INTEGRATOR FULFILLS SQL QUERIES

When the Host Integrator receives an SQL query from a client application, it determines which procedure or set of procedures it must use to satisfy the query, and then executes those procedures.

You can use any VHI procedure type (SELECT, UPDATE, DELETE, or INSERT) to modify host data, but only a SELECT procedure type can return data.

For SELECT statements, Host Integrator will use the necessary procedures to return set of data that exactly matches the WHERE clause in the query. Any data that does not exactly match the WHERE clause is thrown out during a process known as post-fetch filtering. This filtering is not used for LIKE expressions, thus all data found by the procedure concerning a LIKE expression is returned. This implementation of LIKE diverges from the SQL-92 standard.

SQL SYNTAX

Host Integrator supports a subset of the SQL 92 standard for SELECT, UPDATE, INSERT, and DELETE statements. This section describes the syntax convention that Host Integrator supports.

CASE SENSITIVITY

The SQL-92 language standard requires that the names of objects be compared without regard to letter case. Comparisons between column values, however, are case sensitive by default.

If you do not see the results you expect because of case sensitivity, you can add COLLATE CASE_INSENSITIVE to explicitly specify a case-insensitive comparison of text column values. In this example, the state value ('ri') will be compared without case:

```
'SELECT Name, ContractDate, AcctNumber FROM Accounts WHERE MiddleInitial = 'c' AND  
State = 'ri' COLLATE CASE_INSENSITIVE AND LastName = 'smith'
```

Note

SQL keywords such as JOIN or ORDERBY are recognized by Host Integrator, but not supported. You can [extend Host Integrator's native SQL support using an event handler](#). For example, you could do the following in an event handler:

Pass in an SQL string.

Remove the portions not supported by Verastream.

Pass the remaining SQL to the model for processing.

Modify the results based on the custom extensions from the original client string.

TABLE AND COLUMN NAMES

Table and column names in Host Integrator are identified using the following convention:

```
TableName=Identifier
```

```
ColumnName=Identifier
```

```
Identifier=RegularIdentifier | DelimitedIdentifier
```

A regular identifier is a string of not more than 128 characters; the first character must be a letter (upper or lower case), while the rest can be any combination of upper or lower case letters, digits, and the underscore character. No SQL reserved words can be used.

A delimited identifier is any string of not more than 128 characters enclosed in double quotes. The double quote character is represented by two immediately adjacent double quotes.

Table and column names are not case sensitive.

LITERALS

```
Literal={CharacterString | Number }
```

Character strings are written as a sequence of characters enclosed in single quotes. A single quote character is represented by two immediately adjacent single quotes. Any comparisons between literals and columns must be between the same type: strings can only be compared to strings and numbers can only be compared to numbers.

EXPRESSIONS

```
Expression = { Expression + Expression | Expression - Expression | Expression *  
Expression | Expression / Expression | - Expression | ( Expression ) | Literal |  
ColumnName }
```

CONDITIONS

```
Condition = { Condition OR Condition | Condition AND Condition | NOT Condition |  
(Condition) | Comparison }
```

```
Comparison = Expression { = | <> | < | <= | > | >= | LIKE } Expression
```



```
SimpleCondition = { SimpleCondition OR SimpleCondition | SimpleCondition AND  
SimpleCondition | (SimpleCondition) | SimpleComparison)
```

```
SimpleComparison = ColumnName { = | LIKE } Literal
```

A distinction is made between Conditions and SimpleConditions because only SimpleConditions can be used as inputs to a procedure. SimpleConditions can be used in any WHERE clause, but Conditions may be used only in a SELECT statement's WHERE clause because the results can be filtered. Both Conditions and SimpleConditions can refer only to columns from one table. Joins and subqueries (SELECT statements inside another SQL statement) are not supported.

Note

In the examples for the SELECT, UPDATE, INSERT, and DELETE statements assume that the SQL statements used with a model can resolve to a procedure in the model.

SELECT STATEMENT SYNTAX

Use the following syntax for SELECT statements (Arguments between ([]) are optional and those between ({}) are required:

```
SELECT [DISTINCT] {column-list} FROM {table} [WHERE {condition}] [ORDER BY {column-  
list}]
```

Arguments

- [DISTINCT] -- Specifies that all rows returned be unique. If there are two identical rows, one is removed from the output.
- {column-list} -- Any valid table column name(s).
- {table} -- The name of the table.
- {condition} -- Any condition or simple condition.

SELECT statements return only those rows exactly matching the WHERE clause. The results are sorted in the order specified in the ORDER BY clause. For procedure resolution, the table name is taken from the FROM clause, the filter parameters are taken from the WHERE clause, and the outputs are taken from the SELECT and WHERE clauses.

Examples

```
SELECT * FROM Patients WHERE AdmitNum = 20000
```

```
SELECT AdmitNum, SSN FROM Patients WHERE LastName LIKE 'W'
```

```
SELECT DISTINCT AdmitNum, SSN FROM Patients WHERE LastName LIKE 'W'
```

```
SELECT * FROM Patients WHERE LastName LIKE 'W' ORDER BY AdmitNum
```

```
SELECT * FROM Patients WHERE LastName LIKE 'W' AND FirstName = 'JOHN' ORDER BY
AdmitNum
```

UPDATE STATEMENT SYNTAX

Use the following syntax for UPDATE statements:

```
UPDATE {table} SET {{column} = {value} [, ...]} [WHERE {simple-condition}]
```

ARGUMENTS

- {table} -- The name of the table.
- {column} -- Any valid table column.
- {value} -- Valid values are characters and numbers.
- {Simple-condition} -- Simple conditions can contain characters and numbers and comparisons (=, <>, <, <=, >, and =).

UPDATE statements update all records matching the WHERE clause with the values in the SET clause. For procedure resolution, the table name is taken from the UPDATE clause, the filter parameters are taken from the WHERE clause, and the data parameters are taken from the SET clause.

Arguments between ([]) are optional and those between ({ }) are required.

EXAMPLE

```
UPDATE Patients SET FirstName = 'Colin', LastName = 'Moulding', AdmitYear = 1999
WHERE AdmitNum = 56564
```

INSERT STATEMENT SYNTAX

Use the following syntax for INSERT statements:

```
INSERT INTO {table} [( {column-list} )] VALUES {value-list}
```

Arguments

- {table} -- The name of the table.
- {Column-list} -- Any valid table column name(s). If no column list is defined, the order of the columns is taken from the table definition.
- {values-list} -- Valid values are characters and numbers.

INSERT statements add a record to the specified table. For procedure resolution, the table name is taken from the INSERT INTO clause. Data parameters are taken from the VALUES clause.

Arguments between ([]) are optional and those between ({ }) are required.

Examples

```
INSERT INTO Patients (AdmitNum, FirstName, LastName, Room) VALUES (31415, 'Doe', 'John', '123')
```

```
INSERT INTO Patients (AdmitNum, FirstName, LastName, Room) VALUES (31415, 'Doe', 'John', '123'), (31416, 'Doe', 'Jane', '131')
```

DELETE STATEMENT SYNTAX

Use the following syntax for DELETE statements:

```
DELETE FROM {table} [WHERE {simple-condition}]
```

Arguments

- {table} -- The name of the table.
- {Simple-condition} -- Simple conditions can contain characters, numbers, and comparisons (=, <>, <, <=, >, and >=).

The delete statement deletes a record from the specified table. For procedure resolution, the table name is taken from the DELETE FROM clause and the filters are taken from the WHERE clause.

Arguments between ([]) are optional and those between ({ }) are required.

Example

```
DELETE FROM Patients WHERE AdmitNum = 31415
```

SQL SYNTAX RESTRICTIONS

Host Integrator supports the SQL-92 arguments and features, with the following exceptions:

SELECT statements containing GROUP BY or HAVING clauses.

DEFAULT and NULL may not be used for column values in UPDATE or INSERT statements.

INSERT statements that specify inserting DEFAULT VALUES.

Nested queries are not supported.

Parameter references in expressions are not supported.

The AVG, BIT_LENGTH, CASE, CAST, CHAR_LENGTH, CHARACTER_LENGTH, COALESCE, CONVERT, COUNT, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, EXTRACT, LOWER, MAX, MIN, NULLIF, OCTET_LENGTH, POSITION, SESSION_USER, SUBSTRING, SUM, SYSTEM_USER, TRANSLATE, TRIM, and UPPER functions are not supported in expressions.

JOIN operations are not supported.

The SET clause can only accept literal values.

For UPDATE and DELETE statements, all comparisons in the WHERE clause must be between columns and literals.

For SELECT statements, you can use any valid expression in the WHERE clause. However, only comparisons between columns and literals will be used in a procedure. The other expression will be used to filter the output of the procedure(s). Any data that does not exactly match the WHERE clause is thrown out during a process known as post-fetch filtering.

Host Integrator supports a usage of the LIKE operator that turns off post-fetch filtering. In this case, LIKE is equivalent to =, but post-fetch filtering is not performed on the column. If you expect to get everything the host sends, use LIKE. If you want to filter the results (including filtering for case sensitivity), use =. For example:

```
SELECT Name, ContractDate, AcctNumber FROM Accounts WHERE MiddleInitial = 'c'  
AND State LIKE RI AND LastName = 'smith'
```

SQL PREDICATE (LOGICAL TEST) RESTRICTIONS

Only TRUE predicates are allowed. The forms IS NOT * TRUE and IS FALSE are not supported.

The BETWEEN and NOT BETWEEN predicates are not supported.

The IN and NOT IN predicates are not supported.

The LIKE operator is used to provide an input to a procedure without post-fetch filtering. The NOT LIKE and LIKE ... ESCAPE predicates are not supported.

Predicates involving quantifiers (ALL, SOME, ANY, EXISTS, UNIQUE) are not supported.

Predicates involving MATCH are not supported.

Tables Overview

Using Host Integrator tables enables you to create a database abstraction of your host application so that client applications can query it using a subset of the industry standard Structure Query Language (SQL). Table columns are usually associated with attributes and fields in the Host Integrator model. This makes it possible for you to access the data in your host applications via SQL queries even if your host application is not designed to respond to SQL queries. Host Integrator tables themselves don't contain data; rather they provide a database-like view of the underlying host data.

In a query statement, the client application specifies a table, a set of input parameters, and a set of desired output parameters in the form of an SQL statement. From this statement Host Integrator determines the appropriate procedure or procedures to run in order to return the desired results.

Host Integrator accomplishes this through the use of tables and procedures. Using the Tables dialog box, you can create one or more tables for a host application. Each table can have one or more associated procedures. The procedures are comprised of entity navigation paths and [parameter mappings](#) that tell Host Integrator how to read, write, and modify host data represented by the table.

Use the Table Wizard or the Tables dialog box to create a database abstraction of the Host Integrator model.

When you export the documentation for your host application model, a list of all table and procedure names is generated. Use this list to access the tables and procedures from one of the Host Integrator APIs

More information

[Creating tables](#)

Procedures Overview

Procedures tell Host Integrator how to fulfill the queries it receives from client applications. The procedures you create for your table determine what host data can be read, inserted, updated, or deleted. Each procedure has a unique signature that describes what it does. The signature includes a procedure type (SELECT, UPDATE, INSERT, and DELETE) and a set of parameters. Host Integrator uses these signatures to translate SQL statements into a set of procedures.

You can use any VHI procedure type (SELECT, UPDATE, DELETE, or INSERT) to modify host data, but only a SELECT procedure type can return data.

Procedures use one or two of the three types of parameters:

Filter parameters— Specify which records will be acted upon

Data parameters— Specify new values for the records

Output parameters— Specify what values to return

The key component of a procedure's definition is the parameter mapping. Each parameter in a procedure corresponds to a column in the table and is mapped to an attribute, a recordset field, or another parameter. Each procedure has a predefined traversal path through the host application; during the traversal operations, data is exchanged between parameters and attributes and recordset fields. The following chart shows which parameters are used in which procedures:

Procedures	Filter Parameters	Data Parameters	Output Parameters
SELECT	X		X
UPDATE	X	X	
INSERT		X	

Procedures	Filter Parameters	Data Parameters	Output Parameters
DELETE	X		

Procedures should be as complete as possible: if you do not provide a procedure for a particular operation, it is not possible for a client application to access or modify that table data. Procedures should also contain robust error handling to recover from unexpected or incomplete queries. Using the Procedure Editor, you can include error entities that define errors returned from a procedure.

Use the [Procedure Wizard](#) to quickly create a basic procedure. For more complicated procedures, create the procedure using the Tables dialog box and the Procedure Editor.

After adding procedures to your model, you can use [Web Builder](#) to quickly and easily generate a web application or a component interface, such as a web service or JavaBeans, based on the procedures of a host application model.

Note

When creating procedures to be used for generating a web application with Web Builder, you must have unique procedure names throughout the model. Do not create a procedure with the same name for two different tables.

More information

[Creating procedures](#)

[Parameter mapping](#)

[Creating tables](#)

4.8.2 Creating Tables

Using Host Integrator tables enables you to create a database abstraction of your host application so that client applications can query it using a subset of the industry standard Structure Query Language (SQL). Table columns are usually associated with attributes and fields in the Host Integrator model. This makes it possible for you to access the data in your host applications via SQL queries even if your host application is not designed to respond to SQL queries. Host Integrator tables themselves don't contain data; rather they provide a database-like view of the underlying host data.

In a query statement, the client application specifies a table, a set of input parameters, and a set of desired output parameters in the form of an SQL statement. From this statement Host Integrator determines the appropriate procedure or procedures to run in order to return the desired results.

Host Integrator accomplishes this through the use of tables and procedures. Using the Tables dialog box, you can create one or more tables for a host application. Each table can have one or more associated procedures. The procedures are comprised of entity navigation paths and parameter mappings that tell Host Integrator how to read, write, and modify host data represented by the table.

Use the Table Wizard or the Tables dialog box to create a database abstraction of the Host Integrator model.

When you export the documentation for your host application model, a list of all table and procedure names is generated. Use this list to access the tables and procedures from one of the Host Integrator APIs.

Creating a table using the Table dialog box

In the Design Tool, click Tables on the Model menu to open the Tables dialog box.

Then, click New in the Tables dialog box and select Table in the Create a new table or procedure dialog box that appears.

Type a name for the new table in the Name box. This is the name Host Integrator will use to identify the table in the Table Editor and that client applications use to query the host application model.

4. Type a description of the table in the Description box. To add columns to the table:

- Click the Add Column (+) button to the right of the Columns box to add one or more new columns.
- In the Name field, replace the auto-generated column name (Column1, Column2, etc) with a meaningful name, typically the name of the attribute or recordset field that this column will be mapped to in a procedure.
- Select the column's data type by clicking the down arrow in the Data Type entry area and selecting one of the options: Float, Integer, or Text.
- Click the Key box to identify this column as the key.
- If required, specify the column's minimum and maximum values in the Column properties minimum/maximum boxes.

Enter a description of the column in the Description box.

5. If you want the Host Integrator to return a partial set of data to a querying application using SQL, select the Allow SQL SELECT statements to return a subset of columns when all columns

are requested box. If you clear this check box, Host Integrator returns an error to a querying application if it cannot return all columns when a wildcard * is specified in SQL.

Note

You can create tables using the Table Wizard, which prompts you for the necessary information.

DEFINING TABLE COLUMNS

1. Click the Insert Column button to the right of the **Columns** box.
2. Type the name of the column in the **Name** column. Use a name that reflects the name of the attribute or recordset field to keep the table column and the attribute or field closely linked.
3. Select the data type for the column by clicking the down arrow at the far right side of the Data Type box to display the data type list. The options are Integer, String, or Float.
4. If required, specify the column's minimum and maximum properties in the Column properties **minimum/maximum** boxes.
5. To identify a column as the key, select the **Key** box.
6. Enter a description of the column in the **Description** box.

4.8.3 Procedures

Creating Procedures

Procedures tell Host Integrator how to fulfill the queries it receives from client applications. The procedures you create for your table determine what host data can be read, inserted, updated, or deleted. Each procedure has a unique signature that describes what it does. The signature includes a procedure type (SELECT, UPDATE, INSERT, and DELETE) and a set of parameters. Host Integrator uses these signatures to translate SQL statements into a set of procedures.

Procedures define how Host Integrator locates, retrieves, updates, inserts, and/or deletes data in attributes and/or recordset fields. Procedures make it possible for Host Integrator to fulfill requests through a connector API. The client application can access these procedures using either an ExecuteSQLStatement method or a PerformTableProcedure method.

You can use any VHI procedure type (SELECT, UPDATE, DELETE, or INSERT) to modify host data, but only a SELECT procedure type can return data.

Procedures use one or two of the three types of parameters:

Filter parameters— Specify which records will be acted upon

Data parameters— Specify new values for the records

Output parameters— Specify what values to return

The key component of a procedure's definition is the parameter mapping. Each parameter in a procedure corresponds to a column in the table and is mapped to an attribute, a recordset field, or another parameter. Each procedure has a predefined traversal path through the host application; during the traversal operations, data is exchanged between parameters and attributes and recordset fields. The following chart shows which parameters are used in which procedures:

Procedures	Filter Parameters	Data Parameters	Output Parameters
SELECT	X		X
UPDATE	X	X	
INSERT		X	

Procedures	Filter Parameters	Data Parameters	Output Parameters
DELETE	X		

Procedures should be as complete as possible: if you do not provide a procedure for a particular operation, it is not possible for a client application to access or modify that table data. Procedures should also contain robust error handling to recover from unexpected or incomplete queries. Using the Procedure Editor, you can include [error entities](#) that define errors returned from a procedure.

Use the Procedure Wizard to quickly create a basic procedure. For more complicated procedures, create the procedure using the Tables dialog box and the Procedure Editor.

After adding procedures to your model, you can use Web Builder to quickly and easily generate a web application or a component interface, such as a web service or JavaBeans, based on the procedures of a host application model.

Note

When creating procedures to be used for generating a web application with Web Builder, you must have unique procedure names throughout the model. Do not create a procedure with the same name for two different tables.

You can create procedures using the Procedure Editor or by using the table-first approach.

More information

[Creating Procedures Using the Table-first Approach](#)

CREATING A PROCEDURE WITHOUT SQL

The procedure feature in Host Integrator is often associated with SQL, but you can also explicitly execute a procedure directly through the Host Integrator connector APIs using the `PerformTableProcedure` method. This enables you to bypass procedure resolution and post-fetch filtering. Depending on the structure of your host application and whether you benefit from the additional capabilities provided by SQL, executing procedures directly might improve the performance of your deployed models.

The process for creating procedures is the same whether you execute the procedure using SQL or the `PerformTableProcedure` method.

See [Using Host Integrator Connector APIs](#) for information on using the `PerformTableProcedure` method.

More information

INSERT PROCEDURE EXAMPLE

Load the example model CICSAccts.modelx into the Design Tool. The Main entity appears in the model window.

Select Tables from the Model menu.

Expand the Accounts table by clicking the plus sign (+) next to it. The procedures for the Accounts table appear below.

Click the InsertAccount procedure.

Evaluate the structure of this procedure in the Procedure Editor and in the Tables dialog box.

SELECT PROCEDURE EXAMPLE

Load the [example model](#) CICSAccts.modelx into the Design Tool. The Main entity appears in the model window.

Select Tables from the Model menu.

Expand the Accounts table by clicking the plus sign (+) next to it. The procedures for the Accounts table appear below.

Click the GetAccount procedure. Evaluate the structure of this procedure in the Procedure Editor and in the Tables dialog box.

CREATING COMPOUND PROCEDURES

A compound procedure is a procedure that consists of two or more subprocedures. Compound procedures must contain at least one SELECT subprocedure combined with a SELECT, UPDATE or DELETE subprocedure. By combining one or more subprocedure into a compound procedure, you can perform more than one query level task at the same time, like selecting several records and updating or deleting them. Compound procedures cannot include an INSERT subprocedure.

When you create a compound procedure, the order the subprocedures are listed is the order they will be invoked. The first subprocedure in a compound must be a SELECT procedure. Output parameters from a prior procedure supply the filter parameters for the following procedure.

Before you can create a compound procedure, you must already have created the subprocedures that make up this compound procedure.

To create a compound procedure:

Click the table in the **Tables and procedures** box for which you want to create the new compound procedure.

Click **New** in the Tables dialog box and select **Compound procedure** from the list in the Create a new table or procedure dialog box.

A new compound procedure appears beneath the current table; enter a name for the new compound procedure in the **Name** box.

Enter a description of the compound procedure in the **Description** box.

Select the compound procedure's type: Compound procedures can have a type of either SELECT, UPDATE, or DELETE. A compound procedure cannot include an INSERT procedure. If a compound procedure is built from subprocedures of differing types, the compound procedure's type is considered to be the last subprocedure's type.

6. Click the **Insert** button next to the **Select procedures** box and select the first SELECT subprocedure to add to this procedure.

 **Note**

The first subprocedure in a compound must be a SELECT procedure. The SELECT procedures available in the **Select procedures** box are all the SELECT procedures in the table. Next to each SELECT procedure is a green dot, a yellow dot, or a red X. If the SELECT procedure is marked with a green dot, it can be used as a valid subprocedure. If it is marked with a yellow dot, the subprocedure doesn't provide any additional parameters that aren't already present. If it is marked with a red X, the inputs for this subprocedure are not available as outputs from a previous SELECT subprocedure.

7. Repeat step 6 for each SELECT subprocedure you want to add to the compound procedure.

To change the order that the SELECT subprocedures are invoked, highlight a subprocedure and click the up or down arrow to change its place in the execution order.

Optionally, select the UPDATE or DELETE subprocedure to add to the compound procedure by clicking the down arrow next to the **Update/Delete procedure** list.

The UPDATE or DELETE procedures in the list are those available in the current table. A compound procedure can contain only one UPDATE or DELETE subprocedure, and it is always the last subprocedure in the compound procedure.

Mark Procedures as Hidden

Select **Hide in web services and Web Builder** if you do not want the procedure to be visible in either Web Builder or in the WSDL available from the Web Services Explorer. This option marks a procedure as hidden. Selecting this option does not prevent someone from invoking the procedure, it merely treats it as internal or private.

Compound Procedures that Use PerformTableProcedure

You can also create procedures that use the Host Integrator connectors' [PerformTableProcedure](#) rather than the SQL API. In this case, the Available for SQL queries check box at the bottom of the Tables dialogue box should be cleared. Review the information on [Executing Procedures Using Connector APIs](#).

Compound Procedure Example

Load the example model CICSAccts.modelx into the Design Tool. The Main entity appears in the model window.

Select Tables from the Model menu.

Expand the Accounts table by clicking the plus sign (+) next to it. The procedures for the Accounts table appear below.

Click the CompoundNameSearch procedure.

Evaluate the structure of this procedure in the Procedure Editor and in the Tables dialog box.

EXECUTING PROCEDURES USING CONNECTOR APIS

The procedure feature in Host Integrator is often associated with SQL, but you can also explicitly execute a procedure directly through the Host Integrator connector APIs using the `PerformTableProcedure` method. This enables you to bypass procedure resolution and post-fetch filtering. Depending on the structure of your host application and whether you benefit from the additional capabilities provided by SQL, executing procedures directly might improve the performance of your deployed models.

The process for creating procedures is the same whether you execute the procedure using SQL or the `PerformTableProcedure` method.

See [Using Host Integrator Connector APIs](#) for information on using the `PerformTableProcedure` method.

More information

[SQL syntax](#)

[Creating tables](#)

DELETING A PROCEDURE

Load the [example model](#) CICSAccts.modelx into the Design Tool. The Main entity appears in the model window.

Select Tables from the Model menu.

Expand the Accounts table by clicking the plus sign (+) next to it. The procedures for the Accounts table appear below.

Click the DeleteAccount procedure.

Evaluate the structure of this procedure in the Procedure Editor and in the Tables dialog box.

DEBUG PROCEDURE

Use the Debug Procedure dialog box to test and step through the procedure logic that Host Integrator uses to fulfill SQL requests. This allows you to debug your procedure definitions before deploying your model.

You can display the Debug Procedure dialog box in two ways, by either clicking the Debug button in the Test Procedure dialog box, or by clicking the Debug button in the SQL Test dialog box.

To perform procedure debugging, you must either be connected to the host and have access to the application the model is based on, or you must load model in the Host Emulator and connect to it.

To debug a procedure, follow these steps:

1. Click Debug in either the Test Procedure or Test SQL dialog box.
1. The Debug Procedure dialog box appears. In the Stack context box, Host Integrator displays the context of the currently executing procedure or SQL query. If you are testing an SQL statement in the Test SQL dialog box, the statement you are testing displays and the filters from the WHERE clause of the SQL statement display in the Filters box. If you are testing a procedure in the Test Procedure dialog box, the procedure you are testing displays in the format TableName.ProcedureName.
2. Click Run to test the procedure. Host Integrator will test the procedure and display the results of the query in the **Outputs** box (if any).

Stepping through a Procedure's Logic

To step through a procedure's logic one step at a time, follow these steps:

1. Click Debug in either the Test Procedure or Test SQL dialog box.
1. The Debug Procedure dialog box appears. In the **Stack context** box, Host Integrator displays the context of the currently executing procedure or SQL query. If you are testing an SQL statement in the Test SQL dialog box, the statement you are testing displays. Any data or filter parameters that are compared to recordset fields or output parameters (in the case of an SQL query requesting a subset of what the procedure provides) appear in the **Filters** box. If you are testing a procedure in the Test Procedure dialog box, the procedure you are testing displays in the format `TableName.ProcedureName`.
2. Click Step Into. In the **Commands** box, the complete logic for the current procedure displays. The yellow arrow points to the line being evaluated.

 **Note**

The syntax of the command language is generated by Host Integrator. You cannot modify it.

3. To step forward through the procedure logic one line at a time, click Step.
3. Each time you click Step, Host Integrator proceeds to the next line; As it finds the data that fulfills the procedure, the data displays in the Output box. If the Terminal window is visible, you will see the Design Tool navigate to the appropriate host screen while it fulfills the SQL query.
4. Click Step Out to end the step through and return to the previous level.

Setting a Break Point

The Debug Procedure feature includes the ability to set a break point in a procedure which pauses the operation at a chosen point. This aids in debugging procedure logic.

In the Commands box, click the line of code where you want to set the break point.

Click Set Break. A red circle appears to the left of the line, indicating the break point that you set. The next time you run a procedure test, the Design Tool will stop the operation at this line.

To remove the break point, click the line containing the break point and click Remove Break.

Creating Procedures Using the Table-First Approach

Procedures define how Host Integrator locates, retrieves, updates, inserts, and/or deletes the data contained in table columns when it fulfills a request through SQL or through an API.

Using the table-first approach means that you have already created a table and populated it with columns. There are three steps to creating a procedure using the table-first approach:

Set up the procedure information in the Tables Dialog Box.

Defining the traversal path through the host application.

Map the procedure parameters to attributes or fields.

Note

You can also use the Procedure Wizard set up the table information and map the parameters; the wizard creates the traversal paths based on the parameter mapping. To add branching and error entities, you need to use the [Procedure Editor](#).

SET UP THE PROCEDURE INFORMATION IN THE TABLES DIALOG BOX

Click the table that contains the data you need in the Tables and **procedures** box.

Click New in the Tables dialog box.

Select **Procedure** from the list, and then click OK

A new procedure appears beneath the current table, rename the procedure by entering a new name in the **Name** box.

Select a **Home Entity** from the list.

To add a description for the procedure, click Advanced Properties and enter a description in the Description text box.

Select the **Type** of procedure you are generating: Select, Upgrade, Input, or Delete.

8. Select the parameters by checking the appropriate table columns. Depending on the type of procedure being created, you will define one or more of these parameters:

Filter parameters**— Identify the records you want to select, update, or delete

Output parameters**— Contain data that is to be returned from the select procedure

Data parameters**— Fields or attributes to update or insert

9. In the **Req** column, select all table columns that are required in a query. Host Integrator returns an error to the client application if it doesn't include all required columns in its query statement.

Select **Hide in web services and Web Builder** if you do not want the procedure to be visible in either Web Builder or in the WSDL available from the Web Services Explorer. This option marks a

procedure as hidden. Selecting this option does not prevent someone from invoking the procedure, it merely treats it as internal or private.

To make this procedure only available using the PerformTableProcedure method, clear the **Available for SQL queries** check box.

If this procedure is part of a compound procedure, you may want to clear the **Navigate back to starting point upon completion** check box. This allows one procedure to begin where another procedure ends. Although this may provide better performance, it can create navigation errors.

DEFINING THE TRAVERSAL PATH THROUGH THE HOST APPLICATION

Each procedure has a predefined traversal path through the host application. During the traversal operations, data is exchanged between parameters and entity attributes or recordset fields.

In the Tables dialog box, click Procedure Editor to open the Procedure Editor.

2. To insert an entity:

2. a. Use the diagram to click the icon or path prior to the insertion point.

2. b. Then, click the Insert Entity button on the left button bar to open the Insert Entity dialog box.

3. To insert a recordset:

3. a. Use the diagram to click the entity containing the recordset.

3. b. Then click the Insert Recordset button on the left button bar to open the Insert Recordset dialog box.

MAP THE PROCEDURE PARAMETERS IN THE PROCEDURE EDITOR

Each parameter in a procedure represents a column name in the table, which is then mapped to an attribute or field from one of the entities or recordsets in your procedure. In the Tables dialog box, click Procedure Editor to open the Procedure Editor. Use the Data Exchange tab to map table column names to attributes and fields used in the procedure.

To map table columns to entity attributes

Click the entity in the diagram that contains the attributes to be mapped.

2. The Data Exchange tab lists the entity's attributes. Map these to procedure parameters by clicking the down arrow in the appropriate Data Exchange column:

Update attribute from parameter - Select the procedure parameter whose value will be written to this attribute when the procedure is executed.

Write attribute to parameter - Select the procedure parameter to receive this attribute's value when the procedure is executed.

To map table columns to recordset fields

Click the recordset in the diagram that contains the fields to be mapped.

2. The Data Exchange tab lists the recordset's fields. Map these to procedure parameters by clicking the down arrow in the appropriate Data Exchange column:

Update field from parameter - Select the procedure parameter whose value will be written to this field when the procedure is executed.

Write field to parameter - Select the procedure parameter to receive this field's value when the procedure is executed.

Note

As a mapping shortcut, select the attribute or field on the Data Exchange tab and click the Use as Filter, Use as Data, or Use as Output button. This has the additional benefit of automatically creating a column in the procedure's table and designating it as a parameter of this procedure, if it does not yet exist.

Mapping Procedure Parameters

Parameter mapping is the key component of a procedure's definition. Parameter mapping links the attributes or fields in your host application model with parameters in your procedure, which are then used by the procedure to select, update, insert, or delete host data.

The type of parameter you choose to map is dependent on the type of procedure you are creating. The Tables dialog box and Procedure Editor provide different options.

There are three types of parameters:

Parameters	Description
Filter parameter	Identify the records you want to select, update, or delete.
Output parameter	Contain data that is to be returned from the select procedure.
Data parameter	Fields or attributes to update or insert.

This Procedure	Requires...
Select	A filter parameter to identify the records to select and output parameters that return the requested data.
Update	A filter parameter to identify the records to update and data parameters that identify the attributes and/or fields to be updated.
Insert	Data parameters that identify the attributes and/or fields that are inserted.

This Procedure	Requires...
Delete	A filter parameter to identify the records to delete.

SPECIAL MAPPING OPTIONS

There are two circumstances where you may want to use special mapping options:

To refine which records are returned

The filter parameter you use to retrieve host data is not itself available as an attribute or recordset field that can be returned to a querying application.

To solve these situations:

Refine Which Records are Returned from a Recordset

Comparing parameters to recordset fields enables you to perform finely detailed filtering of data that a procedure retrieves from a recordset. This makes it possible to create complex procedure logic capable of sophisticated data manipulation. Only the records that satisfy all comparison mappings are retrieved.

These instructions assume that you have already inserted the recordset. If you have not done so, follow the instructions for inserting a recordset into a procedure.

To compare a filter parameter to a recordset field for a SELECT procedure:

Click the needed procedure in the **Tables and procedures** box in the Tables dialog box.

Click Procedure Editor to open the Procedure Editor.

Click the recordset containing the fields you want to compare to filter parameters.

Click the **Recordset** tab; on the Records tab, select **Fetch/Update records** as the Action to perform.

5. In the **Records** section, specify which matching records to retrieve from the recordset:

Select **First matching record only** to return the first record that matches the filter. If no matching records are found to satisfy the query, the procedure fails. If defined as **First matching record only**, the operation that leads away from the parent entity may reference fields in the recordset.

Select **All matching records** to return all the records that match the filter. Only the operation that leads away from the recordset may reference fields in the recordset.

6. In the **Record filter** section, click the **Compare field to parameter** column next to each recordset field that you want to use to filter the data that will be retrieved when this procedure runs; select a filter parameter from the list.

When the procedure runs, the parameter is compared to the data retrieved from the recordset; only the data matching all the compare filters is returned to the querying application.

Return Filter Parameters as Output Parameters

In some cases, a filter parameter you use to retrieve host data is not itself available as an attribute or recordset field that can be returned to a querying application. To solve this problem, you can map a filter parameter to an output parameter.

To map a filter parameter to an output parameter:

In the Tables dialog box, click the needed procedure in the **Tables and procedures** box.

Click the Advanced button beneath the list of output parameters to open the Parameter to Parameter Mapping dialog box. The output parameters for the current procedure are listed in the **Output parameter** column.

Click the **Filter parameter** column next to the output parameter you want to map. Then, click the list to select the filter parameter you want to map to the output parameter.

Repeat step 3 for each filter parameter you want to map to an output parameter.

More information

[Inserting entities into a procedure](#)

[Procedures overview](#)

[Creating procedures](#)

ADVANCED ATTRIBUTE MAPPING

Use the Advanced Attribute Mapping dialog box to map an attribute or recordset field to multiple output parameters.

To map an attribute or recordset field to more than one output parameter:

In the Data Exchange tab, click **Write attribute to parameter** (for attributes) or **Write field to parameter** (for recordset fields) next to the attribute or recordset field you want to map to multiple output parameters.

2. Select Advanced from the list.

2. The Advanced Attribute Mapping dialog box appears. The attribute or recordset field you are mapping appears in the **Source attribute** box. All the available output parameters appear in the **Available items** box. The list of available output parameters is derived from the output parameters you selected in the Tables dialog box. 3. Click each output parameter you want to map this attribute or recordset field to and click the right arrow. Do this for each output parameter.

2. Each output parameter you map to this attribute or recordset field appears in the **Destination items** box.

3. To remove a output parameter from the mapping, click it in the **Destination items** box, click it and then click the left arrow.

4. Click OK.

4. The Advanced Attribute Mapping dialog box closes. All the output parameters you mapped to this attribute or recordset field appear in the **Write attribute to parameter** box separated by commas.

Advanced Parameter to Parameter Mapping

In some cases, a filter parameter you use to retrieve host data is not itself available as an attribute or recordset field that can be returned to a querying application. To solve this problem, you can map a filter parameter to an output parameter in a SELECT procedure.

To map a filter parameter to an output parameter in a SELECT procedure:

1. In the Tables dialog box, click the procedure for which you want to map parameters in the **Tables and procedures** box.

2. Click the Advanced button beneath the list of output parameters to open the Parameter to Parameter Mapping dialog box. The **Output parameter** column lists the output parameters for the current procedure.

3. Click the **Filter parameter** column next to the output parameter you want to map. Then, click the list to select the filter parameter you want to map to the output parameter.

3. The list of available filter parameters is derived from the **Filter parameters** box in the Tables dialog box. If the filter parameter you are mapping here is available in the form of an attribute or recordset field on an entity, we recommend that you map the attribute or recordset field to an output parameter instead of mapping the two parameters together.

4. Repeat step 3 for each filter parameter you want to map to an output parameter.

5. Click Close to return to the Tables dialog box.

Inserting a Recordset

Use the Insert Recordset dialog box to insert a recordset into the current procedure. Before inserting a recordset, the entity containing the recordset must be in the procedure.

TO INSERT A RECORDSET INTO A PROCEDURE

1. In the Procedure Diagram Pane of the Procedure Editor, click the entity that contains the recordset you want to insert into this procedure.
2. Click the Insert Recordset button on the Icon Button Bar to open the Insert Recordset dialog box. Select the recordset you want to insert from the Entity box.
3. Click OK to confirm your changes or Cancel. When the recordset is added to the procedure, notice that a looping operation is automatically added as well. Looping operations move into and out of a recordset until the operation acts upon the all the recordset fields.
4. Click the Recordset tab to define the properties of this recordset within the procedure.

USING THE INSERT RECORDSET DIALOG BOX

The Insert Recordset dialog box consists of the following fields:

- Entity - The **Entity** box lists all the recordsets available in the entity currently selected in the Procedure Editor. Click a recordset to add it to the current procedure.
- Preview - The Preview pane displays a preview of the new recordset's place in the procedure.

Inserting an Entity into a Procedure

Use the Insert Entity dialog box to insert an entity into the current procedure.

Inserting a branch entity

- Inserting an error entity
- In the Procedure Diagram Pane of the Procedure Editor, click the entity that precedes the insertion point for the new entity.

Click the Insert Entity button on the Icon Button Bar to open the Insert Entity dialog box.

Choose the entity to insert from the Entity list.

USING THE ENTITY DIALOG BOX

The Insert Entity dialog box provides the fields listed below. Take the default values of these fields, or modify the values to work correctly with your procedure.

- Entity List — The Entity box lists all the entities in the current table. Click an entity name to insert it at the current location.

A green ball identifies those entities that can be successfully traversed from the insertion point. A red X indicates Host Integrator cannot navigate to that entity; there is a problem with either the navigation path or input parameters. You can add an entity with a red X, but you need to correct its navigation or parameters to provide a way to traverse to the added entity.

- New Entity or a Sandbox Entity Radio Buttons — These radio buttons affect the entity list by listing all the entities available or a sandbox entity. A New Entity is a modeled host screen containing attributes or recordsets that you want to make part of the procedure.
- Sandbox Entity — This entity is one that has been removed from the procedure, but retains all the customization you have done on that entity. For example, you may have added an entry and mapped its attributes, but later you need to move it to a different location in the procedure. You can delete the entity, which moves it to the sandbox, and then insert the entity without losing the mapping. The Sandbox Entity radio button is disabled if there are no sandbox entities.
- Insert after entities — The new entity is inserted after the selected entity with a path between the two. If there is a branching operation prior to the insertion point of a new entity, and you wish to have multiple paths to the new entity, select additional originations here.
- Path to entity — Specify a dynamic or static path to the entity. Select a dynamic path to allow Host Integrator to determine the best path to this entity based on the defined entities and operations. If a path cannot be found, the Host Integrator returns an error message. A static path requires you to select an operation from the list. Note: Only static paths can branch.

It is important to select **Path to entity** whenever you have "loopback" operations. When a default "Use for navigation commands to this destination" is not in place and multiple loopback operations exist, from the **Path to entity** drop down list, select the explicit operation to ensure it is inserted as the correct operation and that the **Path to entity** is read.

- Preview Pane — As you build the procedure, the Preview pane displays a visual layout of the relationship of the entities, operations, and recordsets in the procedure.

INSERTING A BRANCH ENTITY

Use the Insert Branch Entity dialog box to provide more than one path through the procedure. The paths can be based on a user's input or the outcome of an entity. For example, the user may make a selection on a host application screen that would bring up either the Add Customer screen or the Update Customer screen. Or, the entity may display the next entity based on the customer's credit balance.

To insert a branch entity in a procedure

1. In the Procedure Diagram Pane of the Procedure Editor, click the entity that precedes the insertion point for the new branch entity
2. Click the Insert Branch button on the Icon Button Bar to open the Insert Branch Entity dialog box.
3. Click the entity to insert from the Entity box.
4. If there is more than one option in the Branch until box, select the duration of the branch (either until the procedure moves to another entity or until the end of the procedure).

The Preview Window displays a preview of the new entity's place in the procedure, along with the operations that are able to navigate to it.

Using the Insert Entity Dialog Box

The **Entity** box lists all the entities in the current table. Click an entity name to insert it at the current location.

A green ball identifies those entities that can be successfully traversed from the insertion point. A red X indicates a problem with either the navigation path or input parameters that mean Host Integrator cannot navigate to that entity. You can add an entity with a red X, but you need to correct its navigation or parameters to provide a way to traverse to the added entity.

- **New or Sandbox Entity** - These radio buttons affect the entity list by listing all the entities available or a sandbox entity. A New Entity is a modeled host screen containing attributes or recordsets that you want to make part of the procedure.

A **Sandbox Entity** is an entity that has been removed from the procedure, but retains all the customization you have done on that entity. For example, you may have added an entity and mapped its attributes, but later you need to move it to a different location in the procedure. You can delete the entity, which moves it to the sandbox, and then insert the entity without losing the mapping. The Sandbox Entity radio button is disabled if there are no sandbox entities. | - **Branch until** - If there is more than one option in the Branch until box, select the duration of the branch. This is either until the end of a procedure, or until the procedure moves to the entity that you identify in this box. - **Preview Pane** - As you build the procedure, the Preview pane displays a visual layout of the relationship of the entities, operations, and recordsets in the procedure.

INSERTING AN ERROR ENTITY

Error Entities are screens containing patterns that indicate an error has occurred in the procedure. You can define error entities by purposely entering bad data in a host application and capturing the resulting screen as an entity. Adding one or more error entities to a procedure is a way to build error checking into your model.

Note

If the procedure is using an operation that includes the error entity, you do not need to insert it into the procedure.

To insert an error entity in a procedure:

In the Procedure Diagram Pane of the Procedure Editor, click the entity that preceded the insertion point for the error entity.

Click the Insert Error Entity button on the Icon Button Bar to open the Insert Error Entity dialog box.

Select the entity you want to insert as an error entity from the Entity box.

Using the Insert Error Entity Dialog Box

The Insert Error Entity dialog box enables you to select the error entity to insert into the procedure.

- Entity - The Entity box lists all the entities in the current table. Click an entity name to insert it at the current location.

A green ball identifies those entities that can be successfully traversed from the insertion point. A red X indicates Host Integrator cannot navigate to that entity; there is a problem with either the navigation path or input parameters. You can add an entity with a red X, but you need to correct its navigation or parameters to provide a way to traverse to the added entity.

- Preview Pane - As you build the procedure, the Preview pane displays a visual layout of the relationship of the entities, operations, and recordsets in the procedure.

4.9 Modeling Tips

[Pre-Modeling Activities](#)

[Modeling Practices](#)

[Optimizing the Model for Deployment](#)

[Troubleshooting Models](#)

[Working with Character Mode Host Applications](#)

[Synchronization Techniques](#)

4.9.1 Pre-Modeling Activities

Review the host application and take advantage of host system experts to determine:

What business function are you encapsulating?

How many screens are required to complete the business function?

What are the inputs and the outputs?

Does the application ever change? How significant are these changes?

What errors can occur? It's important to document all paths and error conditions the model might encounter. Even if some paths are of no immediate business concern, they must be recognized and handled in the model.

If possible, consult with more than one expert user of the host application. Users of complex host applications often have different approaches and areas of expertise. Before you begin modeling, talking to diverse experts makes it likely you will end up knowing more about the application than any one user.

- Determine the number of models needed

You cannot access more than one host application with one model. If the application you want to develop needs to access data from more than one host application, you must create two models and develop an application that can share variables, which will allow you to integrate host data from two or more Host Integrator models.

4.9.2 Modeling Practices

As a common practice, you should avoid using both schemes together in one model because entity navigation paths and cursor positions can become out of synch.

If the host supports command driven navigation, use it. This can greatly reduce navigation.

- Do not model a blank entity.

Modeling blank entities can cause undesired side effects. Some command-driven hosts use blank entities to accept commands. In these cases, you can write your commands to a blank entity as part of a command sequence without actually having it exist as an entity.

- Use tables to manipulate host application data
- There is less processing overhead using tables, because there is only one "round trip" needed to the server.
- Using SQL reduces the amount of coding necessary.
- Set naming conventions
- Determine a standard notation for entities and attributes.

Decide how to handle duplicate attribute names; there may be several fields called `account_number` among the host screens.

If using COM objects, do not use a Visual Basic reserved word as an attribute name.

- Do not use reserved SQL words to name tables.
- Determine global settings

Set traversal operation preferences; traversal operations are usually automatically generated.

- Set entity and attribute generation preferences; these are usually created manually so you can generate only the entities and attributes that are necessary.
- Create concise operations

Break large operations into a sequence of smaller operations, using alternate and intermediate destinations. Modeling is very like programming. Do not be in a hurry. Be careful and meticulous. It is often advantageous to work on a pair of screens at a time, the second screen being the destination of the first, and get each pair working error-free before moving to the next set. Sometimes you may have to capture a lot of screens quickly; if this is the case, import them one by one into what will become your carefully crafted model.

- Minimize patterns used to identify an entity
- Create patterns at the top and bottom of each entity to ensure the entire screen is displayed.
- Do not add more patterns than necessary as this adds to the processing time to recognize an entity.
- Parameters and Constants
- DefaultValue commands should have the Application parameter set to "Default value for server and design tool."

When an attribute is available, all constant values should use `TransmitToAttr`.

- Correctly spell and set all constant values.
- Timing for 5250 Models

When creating 5250 based models, define patterns for each of the input fields on the sign-on screen and on any screen where the timing of screen recognition is difficult because the AS/400 sends multiple writes to create the screen.

To set up input field patterns:

On the Pattern tab, select the Definition subtab.

The default definition is Text. Clear the Text check box and select the Field type check box.

Select Unprotected as the Field type.

By defining these patterns for the input fields, the screen recognition process waits until these fields are ready for input. This ensures proper screen recognition and enhances model performance.

4.9.3 Optimizing the Model for Deployment

After you have completed your model, take these steps to prepare the model for optimum performance before deploying it to Host Integrator Server.

- Delete unnecessary entities
- Include in your model only entities that contain necessary data or those that are intermediaries between entities containing this data.
- Do not create an entity for a host blank screen unless you have exhausted all other options. Blank screens have the potential of producing run-time entity signature ambiguity that could crash a session.

- Delete unnecessary or unused attributes

To improve performance and reduce memory consumption, make sure that the attributes of each entity contain necessary data or input fields.

- Delete all unnecessary operations

Validate the paths between entities to ensure they are traversable by the shortest route.

- Reduce the screen signature

Review the screen signature of every entity in the model and reduce it to the minimum number of patterns required to uniquely identify the entity.

- Optimize traversal operations

Before deploying the model, make sure the traversal operations are as efficient as possible. Open the model in off-line mode and follow these steps to review the operations for each entity:

Each traversal entity is named To `<entity>` and the destination entity matches this name.

Edit operations to remove the `checkOperationConditions` command if there are no pre or post conditions on this operation.

4.9.4 Troubleshooting Models

- Validate the model using Validator
- Use Web Builder to create a web client

Use Web Builder to quickly and easily generate a Web application of the host application model. Use this application to test your model functionality.

- Test possible error conditions for each operation

You can test possible error conditions for each operation by inputting bad data or by reproducing mainframe error conditions.

- Test the model parameters

Test the write attributes, recordsets, procedures, and SQL statements.

Every entity should be able to navigate to the home entity.

Make sure that all screens, including error screens, can recover themselves (that is, they can navigate to the home entity).

- Recognizing screens as entities

Invariably, your application will contain one or two screens that prove difficult to model. Problems range from unpredictable host data transmission timings to the host redrawing seemingly static portions of the screen. With these screens, it will take a little bit of extra observation and experimentation to model accurately. Evaluate the order in which the host writes the screen. Watch the behavior of the cursor. Use [model debug messages](#) to look for events that could be causing the problem.

- Synchronizing with the host
- Getting out of sync with your host is one of the most common problems encountered when modeling host applications. If your model works well in the Design Tool, but seems to malfunction in the server environment, the lack of synchronization is the most likely reason. Host sessions running on a Host Integrator server run faster than those in the Design Tool because there is no display to update. Therefore, the server can expose synchronization problems that don't appear in the Design Tool.

This advice applies particularly to operations that scroll recordsets or perform other tasks within a single screen. Even on block mode host applications, Host Integrator will sometimes try to perform operations too quickly, producing incorrect results. See [Synchronization Techniques](#) for more information.

- Never use WaitFor (some time period) to synchronize unless there is absolutely no other alternative; 1 millisecond over and you're out of synch.

4.9.5 Working with Character Mode Host Applications

In character mode applications, the host is constantly moving the cursor to write data, so just waiting for the cursor to end up in the correct position after a host update may be sufficient. If the cursor is placed in varying positions but in the same place relative to some text, then a WaitFor command is recommended.

To ensure synchronization, end all operations with an appropriate WaitFor command.

4.9.6 Synchronization Techniques

Your host application may be unpredictable when slower execution masks missed synchronization issues. Synchronization in a Host Integrator model ensures that the entire host screen has been received before reading or transmitting data. If you don't develop methods to ensure synchronization, the model can read incorrect values from attributes and fields, write to the wrong locations, or fail to position the cursor at a desired location.

Synchronization is important in operations, when writing attributes, and when using cursor movement commands.

Implementing synchronization techniques is critical for any VT or HP character mode model, and can be important for 3270 and 5250 block mode models where the screen is received in multiple command chains. To view a model example of a VT host application, see the Pine model located in your `<VHI install directory>\examples\ModelSamples` folder.

There are two basic techniques for ensuring synchronization.

- You can issue Wait commands such as WaitForDisplayString, WaitForCursorAtAttribute, WaitForCursorAtField, WaitForCursorAtLocation, WaitForCommString, WaitForUpdate, WaitForKeyboardEnabled, WaitForMultipleEvents
- You can validate an entity based on patterns or cursor position.

Strategies for Synchronizing Character Mode (VT and HP)

Character mode applications require extra effort to ensure that the model is in synch with the host screen.

In general, using an appropriate Wait command to end an operation is the most reliable technique for ensuring synchronization. The wait commands below are listed with the most highly recommended strategies listed first.

If possible, use `WaitForDisplayString`. This command applies to cases where the command is prompting for input.

If not, use `WaitForCommString`. This command applies when there are escape sequences in the host data and the last set of escape sequences is unique.

`WaitForCursorAtLocation`. This command can be used if the cursor always arrives at the same location. This command may not be dependable if the cursor passes through a location in a transition before arriving at the location a second time.

`WaitForUpdate`. This command is not recommended unless you're dealing with a small update area. The first character changed in the specified region will satisfy this wait condition. If you're updating a model created with an earlier version of Host Integrator, be sure to disable the compatibility switches associated with `WaitForUpdate`.

When the host response could differ from one time to the next, use a `WaitForMultipleEvents` command. This command allows several globally defined events to be encapsulated into one command within an operation. This enables Host Integrator to rely on an order of events rather than the number of data packets that are sent from a character mode host.

WAITFORCOMMSTRING COMMAND AND NONPRINTABLE CHARACTERS

For VT and HP character mode hosts, you can wait for datastream sequences that include nonprintable characters such as carriage return, linefeed, and the escape key. These sequences can be captured by copying and pasting text from the Model Debug Messages dialog box into the Operation Edit dialog box when the `WaitForCommString` command is selected. An example of this syntax is `"\033EnterData"`, which represents the escape key followed by the string data. This syntax represents the C programming language conventions that use a backslash () followed by three octal numbers to represent any character.

WAITFORMULTIPLEEVENTS COMMAND

Many operations have alternate destinations; the best way to synchronize at a destination is to use the `WaitForMultipleEvents` command. In the example below, the first event is satisfied when reaching a destination, but the second event is satisfied with a second destination.

Tips:

Add a `WaitForUpdate` command to a page down operation to prevent the operation from completing until a particular location on the screen has been updated by the host.

See the AddressBook entity in the Pine model example to view a recordset with several types of scrolling operations, including a `PageDown`, `PageUp`, `LineUp`, and `LineDown` operation.

Example:

Create global events in the Event Edit dialog box using host event commands at the end of the operation to verify that the cursor position has returned to a particular location.

1. Click the right arrow button and select New Host Screen and Cursor Enter Position. Make sure to specify the row and column coordinates.

1. The first command waits for the cursor to return to a specific position, and the second command specifies a second cursor position. The following events now appear in the Events box:

1. Event 1 (WaitForCursorEnterPosition Row, Col)

1. Event 2 (WaitForCursorEnterPosition Row, Col)

2. Click OK.

Open the Operation Edit dialog box, click the right arrow button, point to Events, and select the WaitForMultipleEvents command.

Under Command parameters, click the Edit button to open the Event Expression Editor dialog box.

Select Event 1, click the And button, and select Event 2.

Click OK. The Operation Edit dialog box opens and displays your new WaitForMultipleEvents command in the Commands box.

VT EPHEMERAL SCREENS

Some VT applications produce transient, or *ephemeral* screens. These are screens that appear between a *departure entity* and an *arrival entity*. These screens do not persist long enough to identify them as an entity to be modeled, and they transition without any user input. Ephemeral screens can appear in [Model Debug Messages](#), but since the screens do not wait for user input, it is impossible to model them in the normal fashion. However, it is possible to use [Event Handlers](#) to capture the screen data and use Java or .NET code to process or dispatch the screen data.

To do this:

Attach an `EntityEventHandler` object to the *departure screen* and implement the `EntityDeparture` event handler method.

When the `NextEntity` property is empty (signifying that the next entity is unknown) call the `OverrideDeparture` function, and use the `TerminalScreen` to access the ephemeral screen data. `OverrideDeparture` serves to keep the `EntityDeparture` event handler active until a recognized screen is encountered.

Process the screen data in the event handler as needed, such as writing to a file, sending to a database, or returning as a recordset.

Here is a Java example. The .NET code would be very similar:

```
public void entityDeparture(EntityDepartureEvent event) throws ApptrieveException {
    if (event.getNextEntity().length() == 0) {
        event.overrideDeparture();
        myProcessScreenFunction(event, event.getTerminalScreen());
    }
}
```

Note

Verify that the model setting *Verastream 7.8 Ephemeral Screens VT Compatibility* is not enabled. See [Compatibility](#) for more information.

VT SCROLLING

Some VT applications scroll data off the screen without stopping to wait for another user command. VHI screen recognition typically picks up the *settled* screen using a *departure entity* just prior to the scrolling operation and an *arrival entity* that the Design Tool recognizes at the end of the scrolling operation. However, the data that has already scrolled off the screen may be lost. To avoid this, you can use multi-page recordsets.

To do this:

On the *departure entity*, modify the `ToArrivalEntity` operation `WaitForCommString "\033[J", "5", 5` to `WaitForCommString "\033[J", "5", 1` so screen recognition stops before scrolling occurs. This is a manual modification of the recognition pattern (the numbers and type of wait may vary by host application and might need to be adjusted).

On the *arrival entity*, complete the following:

define a recordset that contains the rectangular scrolling area (which may or may not comprise the entire screen).

create a `PageDown` operation with only `CheckOperationConditions` in the command list, and add the page down operation to your recordset.

if there is a unique pattern that occurs at the end of scrolling, add this pattern to the current entity. Verify that *Use in entity signature* is not checked.

set the Recordset Termination rule to use *Screen contains pattern*: using the pattern you just added. In the absence of a unique pattern, your recordset termination rule might need to use other criteria; for example, blank records, or duplicate records.



Note

The names *departure entity*, *arrival entity* and `ToArrivalEntity` are placeholders. Please substitute your actual entity and operation names.

Verify that the model setting *Verastream 7.8 VT Scrolling Compatibility* is not enabled. See [Compatibility](#) for more information.

Writing Data to the Screen

When writing data to the screen, notice the mode of the terminal. If the mode is insert, for example, you may have to issue a special command to clear a field before transmitting the data (see the Pine model for an example). If the mode is overwrite, then you may need to utilize the Erase to end of attribute setting in your attributes. Failing to do so could result in garbled data when a shorter piece of data is written over a longer one, leaving the end of the original data in place to be sent to the host.

If you are working with a character mode host, take special note of the Attribute Echo tab options related to waiting for character echo. Failing to wait for character echo can often cause subsequent operations or cursor movement to fail. The same issues apply to static transmission commands (`TransmitANSI`, for example) that you can utilize in operations.

Strategies for Synchronizing 3270 and 5250 Applications

Add synchronization if a block mode screen is received in multiple command chains. Your options include:

Add a validation pattern to the entity that is recognized only when the last command chain is received

Add a WaitForDisplayString command to the operation

Add a WaitForUpdate command to the operation. If you're modifying a model that was created in an earlier version, be sure to disable the compatibility switches for VHI 4.5

WaitForUpdateCompatibility and VHI 5.5 WaitForUpdate 3270 and 5250.

Add WaitForKeyboardEnabled to the operation.

Note

Entity validation and synchronization based on cursor position are also available.

Entity Validation (Validation tab of Advanced Entity Properties)

Configuring entity validation is helpful when modeling character mode applications since there is nothing in the character mode datastream to denote what constitutes a screen. In general, using the Wait options described above is recommended over using to entity validation.

On the Validation tab of the Advanced Entity Properties dialog box, use the arrival validation options to configure how to validate a certain entity. This validation enables Host Integrator to delay its entity arrival notification until certain conditions are met. Select one of the following options to configure entity validation:

- Option 1: Select the Wait for cursor check box to validate the entity once the cursor arrives at a certain position. Specify the position in the Row and Col boxes.
- Option 2: If Option 1 fails, select the Wait for patterns check box and move one or more configured patterns from the Unused list to the Check list. Once a validation pattern is moved to the Check list, Host Integrator will not validate the entity until this pattern is identified.
 - Make sure to clear the Use in entity signature check box on the Pattern tab before using a pattern for entity validation.
 - Using a pattern in entity validation isn't necessarily the best solution for modeling character mode applications because the pattern may appear while the cursor is moving around the screen. Try configuring Option 1 before trying this method.
- Option 3: Select the Wait for condition check box to create a condition in the Condition Edit dialog box. This option often handles situations where the final cursor position may be at any of several locations.

Synchronizing for Cursor Movement and Writing Attributes

When you need to synchronize cursor movement for writing attributes, use a post-write operation to position the cursor at the next attribute. The Wait commands described above work for these synchronization tasks as well.

If post-write operations don't cover cursor movement requirements, you may need to define cursor movement command lists for VT and character mode HP hosts. Cursor movement definitions are especially important when the cursor cannot be moved to a location on the screen with the arrow keys. Without a cursor movement command list, Host Integrator's default is to use the arrow keys. For example, if the cursor is at (2,2) and the attribute is at (2,10), Host Integrator sends eight right arrow keys to the host attempting to end up at the correct location. Sometimes, these host applications can be unpredictable and the cursor may not end up in the same position each time that screen is used. If Host Integrator does not reach the expected location, you will receive the error that the cursor could not be moved.

- If the host uses the Tab key to move between fields, define a global move forward cursor movement command list in the Advanced Model Properties dialog box. To be more precise about cursor movement between entities, configure the options on the Cursor tab.
- One synchronization technique when working with block mode terminals is to move the cursor before scrolling and then wait for the host to restore it to the normal initial position when it is finished.

Example using the Tab key with a VT host: `TransmitTerminalKey rcVtTabKey`

Example using the Return key with an HP host: `TransmitTerminalKey rcHPReturnKey`

In each case, the command sends the tab key or return key to move the cursor. If the host does more than move the cursor to the next location (for example, it updates a status line or redraws other sections of the screen), then you should be using tabstops. Use the Model Debug Messages option to determine if the host is doing more than just moving the cursor.

Note

On HP host applications, the Enter key and the Return key are quite different. On HP host applications, the Enter key refers to the keypad Enter key.

Once you've defined a move forward command list, Host Integrator will repeatedly execute this command list until it arrives at the starting location for the attribute or attributes you are writing to. If the host never positions the cursor at the attribute, you will receive an error that the cursor could not be moved.

Tabstops (Cursor Tab)

If the host sends more than one packet in response to the tab key, define tabstops on any entity that has multiple attributes. When tabstops have been defined on an entity, Host Integrator will wait for the cursor to arrive at a defined tabstop after each execution of a move forward or move backward command list.

If moving through the fields of your host application requires application interaction, you may have to define the tabstops manually. Using other techniques, such as implementing Wait commands in post-write operations, is recommended over the use of tabstops.

You can define tabstops for an entity on the Cursor tab.

Debugging Synchronization Problems

Use the [Model Debug Messages](#) debug option to review the real-time data sent and received from a host.

4.9.7 Supporting Files

Project directories can contain the following:

- Host information, including name and port number for a Telnet connection
- Login, logout, or move cursor command lists
- Entity definitions, including patterns, attributes, operations, recordsets, and fields
- Variables with default values
- Tables, including column and query definitions
- Model properties

In addition to creating a model or modelx file, the Design Tool also generates the following project files:

Supported files for .model file types

Screen snapshots file (.snapshot file) contains a captured image of a defined entity, which can be viewed during offline design of the model with the Host Emulator.

The Scripts directory Source files, JAR/assembly files, and class files associated with event handlers are stored within this folder. In particular, `build.xml`, the `\src` subfolder, and the `\lib` subfolder are required for event handler maintenance.

Deployment files, located in the `\deploy` folder, are generated after you deploy the model to the local server.

Supported files for .modelx file types

Modelx and modelx_1.xsd files located in the `\models` folder


Entityx and entityx_1.xsd files located in the `\entities` folder within the modelx folder.

Tablex and tablex_1.xsd files located in the `\tables` folder.

The Scripts directory Source files, JAR/assembly files, and class files associated with event handlers are stored within this folder. In particular, `build.xml`, the `\src` subfolder, and the `\lib` subfolder are required for event handler maintenance.

Deployment files, located in the `\deploy` folder, are generated after you deploy the model to the local server.

The `.xsd` files define the valid XML that can be used for `modelx`, `entityx`, and `tablex` files.

 **Note**

If you want to save the settings for this model so that it will be the basis for creating other models, select *Save Settings As*. The `.dtool` file you create can include Design Tool-specific configuration information such as window size, colors, keymapping, and preferences, as well as information about the host application and connection settings.

When you decide to deploy your model to the Host Integrator Server, use *Deploy to local server* or *Deploy to remote server*.

4.10 Using Regular Expressions

A regular expression (regex or regexp) is a special text string used to describe a search pattern, according to certain syntax rules. For example, `1[0-9]+` matches "1" followed by one or more digits.

Use regular expressions judiciously and only when necessary. If you require even greater complexity than regular expressions can support, consider using event handlers instead. Using regular expressions, or event handlers, indiscriminately can result in significant performance overhead.

Host Integrator regular expressions are based on Perl syntax. Host Integrator supports regular expressions for:

- Signature patterns for entity recognition—match a regular expression over a region of the terminal screen
- Patterns for errors—you can use regular expressions to support "or" conditions
- Recordset termination—terminate when a Host Integrator condition is satisfied

Host Integrator conditions can contain regular expressions. However, while conditions can use values of model variables, entity attributes, and recordset fields, regular expressions cannot. Regular expressions only support the use of literal values.

- Recordset filtering—exclude records that match condition
- Attributes—read and write substitutions in the attribute value
- Selection of data—parsing out a particular piece of data from a string, for example everything after the third space up to the last comma (think of parsing a first name or last name from a string or grabbing the state from a street address)
- Recordset Fields—read and write substitutions

Note

Perl programming documentation is available at [Perl regular expressions quick start](#). This is a good introduction to regular expressions.

4.10.1 Special characters

Certain characters are reserved for special use. If you want to use any of these characters as a literal in a regular expression, you need to escape them with a backslash. If you want to match `1+1=2`, the correct regex is `1\+1=2`. Otherwise, the plus sign will have a special meaning.

`\/ literal /`

`\\ literal \`

`\. literal .`

`* literal *`

`\+ literal +`

`\? literal ?`

`\\ literal |`

`\\(literal (`

`\\) literal)`

`\\[literal [`

`\\] literal \`

`\\- The - must be escaped inside brackets: [a-z0-9_\\.\\-\\?!]`

Character	Description	Example
<code>//</code>	Used to search a string for a match.	"Hello World" =~ /World/; In this statement, World is a regex and the // enclosing /World/ tells Perl to search a string for a match. The operator =~ associates the string with the regex match and produces a true value if the regex matched, or false if the regex did not match. In this case, World matches the second word in "Hello World", so the expression is true.
<code>\\</code> (backslash)	Escape character used to represent characters that would otherwise be a part of a regular expression.	"\\." = the period character.
<code>[abc]</code>	Match any character listed within the square brackets.	<code>[abc]</code> matches a, b or c

Character	Description	Example
<code>\d, \w, and \s</code>	Shorthand character classes matching digits 0-9, word characters (letters and digits) and white space respectively. Can be used inside and outside character classes	<code>[\d\s]</code> matches a character that is a digit or whitespace
<code>\D, \W, and \S</code>	Negated versions of the above. Should be used only outside character classes.	<code>\D</code> matches a character that is not a digit
<code>\b</code>	Word boundary. Matches at the position between a word character (anything matched by <code>\w</code>) and a non-word character (anything matched by <code>[^\w]</code> or <code>\W</code>) as well as at the start or end of the string if the first or last characters in the string are word characters or an alphanumeric sequence. Use to perform a "whole words only" search using a regular expression in the form of <code>\bword\b</code> . <code>\b</code> also matches at the start or end of the string if the first or last characters in the string are word characters.	<code>\b4\b</code> matches 4 that is not part of a larger number.
<code>\B</code>	Non-word boundary. <code>\B</code> is the negated version of <code>\b</code> . <code>\B</code> matches at every position where <code>\b</code> does not. Effectively, <code>\B</code> matches at any position between two word characters as well as at any position between two non-word characters.	<code>\B.\B</code> matches b in abc
<code>.</code> (period)	Match any single character	<code>"."</code> matches x or any other character.

Character	Description	Example
x (reg character)	Match an instance of character "x".	x matches x
^x	Match any character except for character "x".	[^a-d] matches any character except a, b, c, or d
^ (caret)	Match the beginning of a string. Matches a position rather than a character.	^ matches a in abc\ndef . Also matches d in "multi-line" mode.
\$ (dollar)	Match the end of a string. Matches a position rather than a character. Also matches before the very last line break if the string ends with a line break.	.\$ matches f in abc\ndef . Also matches c in "multi-line" mode.
(pipe symbol)	Or. Match either the part on the left side, or the part on the right side. Can be strung together into a series of options. The pipe has the lowest precedence of all operators. Use grouping to alternate only part of the regular expression.	abc (pipe symbol)def (pipe symbol)xyz matches abc, def or xyz abc(def (pipe symbol)xyz) matches abcdef or abcxyz
(abc) (parentheses)	Used to group sequences of characters or expressions.	(Larry(pipe symbol)Moe(pipe symbol)Curly) Howard matches Larry Howard, Moe Howard, or Curly Howard "

Character	Description	Example
<code>\1</code> , <code>\$1</code> , <code>\$\$</code>	<p><code>\1</code> Refers to first grouping, used in the expression <code>\$1</code></p> <p>Refers to first grouping, used in the replacement string <code>\$\$</code></p> <p>Literal “\$” used in the replacement string.</p>	<p><code>/(.+)((\r?\n(pipe symbol)\r)\1)+\b/ig, “\$1”</code> Removes duplicate lines from a list. The <code>(.+)</code> grabs a line of text and the parenthesis save it for a reference. The <code>(\r?\n(pipe symbol)\r)</code> grabs the line separator, either <code>\r\n</code>, <code>\n</code>, or <code>\r</code>. Next, <code>\1</code> references the first line and so <code>((\r?\n(pipe symbol)\r)\1)+</code> matches 1 or more subsequent lines that match the first line. Notice that in Javascript, a reference within the expression is <code>\1</code> while a reference in the replacement string is <code>\$1</code>. The <code>\b</code> prevents “street” and “streets” from being seen as the same word.</p>
<code>{}</code> (curly braces)	Used to define numeric qualifiers	<code>a{3}</code> matches <code>aaa</code>
<code>{N,}</code>	Match must occur at least “N” times	<code>Z{1,}</code> matches when “Z” occurs at least once
<code>{N,M}</code>	Match must occur at least “N” times, but no more than “M” times	<code>a{2,4}</code> matches <code>aa</code> , <code>aaa</code> or <code>aaaa</code>
<code>?</code> (question mark)	Makes the preceding item optional or once only. The optional item is included in the match if possible.	<code>abc?</code> matches <code>ab</code> or <code>abc</code>

Character	Description	Example
* (star)	Match on zero or more of the preceding match. Repeats the previous item zero or more times. As many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.	"go*gle" matches ggle, gogle, google, gooogle, and so on.
+ (plus)	Match on 1 or more of the preceding match. Repeats the previous item once or more. As many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.	"go+gle" matches gogle, google, gooogle, and so on (but not ggle.)
i	Case insensitive search	/expression/i
g (plus)	Global replacement. Replaces all matches.	/expression/g
q(?=u)	Matches q only before u. Does not match the u. This is positive lookahead. The u is not part of the overall regex match. The lookahead matches at each position in the string before a u.	q(?=u) matches the "q" in question, but not in Iraq.

Character	Description	Example
q(?!u)	Matches q except before u.	q(?!u)) matches "q" in Iraq but not in question.

For information about additional pattern matching operators for conditions and filters, see [Condition Edit Filter](#).

4.10.2 Examples of Regular Expressions

- Matches when an error message is displayed on the status line.

```
ERROR [0-9]{1,4}: .*
```

- Match 3 instances of a string.

```
"/(John){3}/" (Matches John John John)
```

- Match any of several first names, followed by a common last name.

```
"(Homer|Marge|Bart|Lisa|Maggie) Simpson" (Matches any member of the Simpson family)
```

- Condition matching "Page N of M" when N = M.

```
PageStatus =~ s/Page ([0-9]+) of [0-9]+/$1/ = PageStatus =~ s/Page [0-9]+ of ([0-9]+)/$1/
```

- Recordset condition to match records where myfield starts with "P".

```
myrecordset.myfield =~ m/P.*$/
```

- Recordset condition where the field is not numeric.

```
myrecordset.myrecordsetfield =~ /[0-9]+/
```

ADDITIONAL RESOURCES

Regular expressions can be complex. A number of resources are available on the Internet to help you understand regular expressions.

[Test tool for regular expressions](#) including examples and syntax.

[A detailed description of regular expression syntax in Perl](#)

4.10.3 Details of a Substitution Regular Expression Example

The regular expression described here is used in [Substituting a Regular Expression for a Recordset Field](#) under [Substitutions](#). This example changes a value usually displayed with a trailing minus sign to a value with a leading minus sign.

Search for: `(^[^-.\\d])(\\d+(?:\\.\\d+)?)-(?=[^-.\\d]|$)`

Replace with : `$1-$2`

A "capture group" is a regular expression surrounded by parentheses that is remembered as a numbered variable for use in the replacement.

Detail of Expression	Description
<code>(^ "pipe symbol"[^-.\\d])</code> A numbered capture group (Becomes \$1 in the replacement)	Select from 2 alternatives: Beginning of line or string Any character that is not in this class: <code>[^-.\\d]</code> (minus, decimal, or digit)
<code>(\\d+(?:\\.\\d+)?)</code> A numbered capture group (Becomes \$2 in the replacement)	<code>\\d+(?:\\.\\d+)?</code> Any digit, one or more repetitions Optional (zero or one repetitions) of dot followed by one or more digits
<code>-</code>	(Match on trailing minus)

Detail of Expression	Description
([^\d] (pipe symbol)\$) Match a suffix but exclude it from the capture.	Select from 2 alternatives: Any character that is not in this class: [-.\d] (minus, decimal, or digit) End of line or string

This translates a string like "123.45-" to "-123.45". Any leading non-numeric characters are retained and trailing non-numeric characters are removed so that "ABC123.45-XYZ" would be changed to "ABC-123.45".

4.10.4 Substitutions

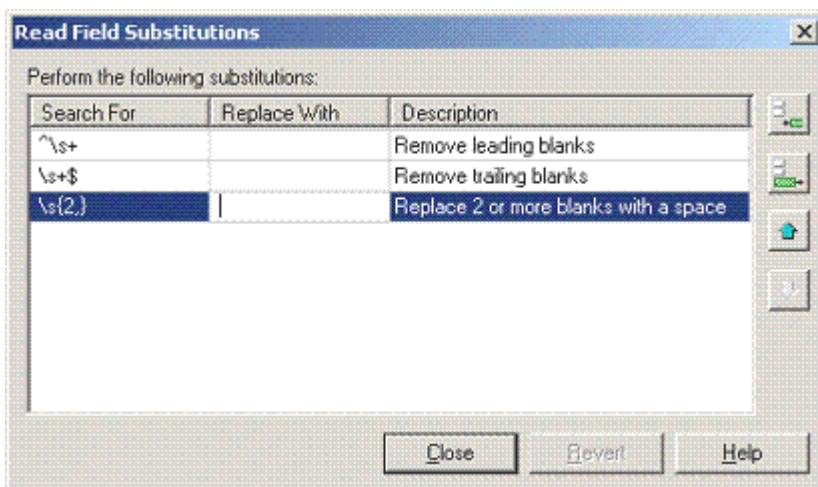
Read or Write Substitutions (Attribute or Field)

Use this dialog box to configure substitutions in attributes or recordset field strings. Select either the Attribute Properties tab or the Recordset Fields tab and click Advanced in the **Read** or **Write** box.

Define the substitution you would like to perform by specifying the string you want to search for and the string you want to use as a replacement. You can include regular expressions to specify your substitutions. For example, to remove trailing blanks from an attribute or recordset, type `\s+$` for the string to search for and leave the replacement specification blank.

The list of string substitutions are applied in the order listed in the user interface. By performing multiple replacements, it is possible to isolate words or substrings. Each substitution is equivalent to the Perl regular expression syntax `s/<Search for>/<Replace with>/g`. The "g" stands for "global", which replaces all matches (not just the first one).

For example, to replace 2 or more blanks with a single space:



Regular expressions in Host Integrator uses the syntax described in How to Use Regular Expressions.

SUBSTITUTION EXAMPLES

Purpose	Search For	Replace With
Remove leading blanks (white space at beginning is replaced with an empty string)	<code>^\s+</code>	
Remove trailing blanks	<code>\s+\$</code>	
Remove all spaces	<code>\s</code>	
Remove first word	<code>^[A-Za-z]+</code> <code>\s</code>	
Remove last word	<code>[A-Za-z]+\$</code>	
Change negative numbers represented with parentheses (123) to be a negative sign -123.	<code>\(([0-9]+)</code> <code>\)</code>	<code>-\$1</code>
Prepend value (at beginning)	<code>(^.*\$)</code>	<code><value>\$1</code>

Purpose	Search For	Replace With
Append value (at end)	(^.*\$)	\$1<value



Substituting a Regular Expression for a Recordset Field

Use the Substitution dialog box to configure advanced read and write options for both attributes and recordset fields.

Note

Use regular expressions judiciously and only when necessary. Added performance overhead can occur when using regular expressions.

In this example, use the sample model CCSDemo to substitute a recordset field usually displayed with a trailing minus sign, with one which will be displayed with a leading minus sign. There are also steps to test the output.

1. Open CCSDemo in the Design Tool.
2.  Click  to connect to the host.
3. Select AcctTransactions from the Entity list to navigate to the AcctTransactions entity.
4. Click the Recordset tab, then select the Fields tab on the lower portion of the Recordset tab.
5. Select the row named Amount.
6. In the Read group box, click Advanced to open the Read Field Substitutions dialog box.
7. Click Insert to add a row for a new substitution.
8. Create an expression that matches a money field with a trailing minus sign. In the Search For column of the newly created row, enter the expression:
`(^[^-\.\d])(\d+(?:\.\d+)?)-(?=[^-\.\d]|\$)` See this expression in detail
9. To transpose the minus sign to the front of the field, enter the following in the Replace With column: \$1-\$2
10. In the Description column, type: `Move minus sign to front`
11. Close the Read Field Substitutions dialog box and click Apply to update your model.
12. To test the results, select Procedure Test from the Debug menu.
13. From the Table drop down list, select Transactions and confirm that the GetTransactions procedure is selected.
14. In the Value field of the Procedure Filters box, type `167439459`
15. Click Execute.
16. Scroll down the procedure output and note that several amounts have a leading minus sign.

4.10.5 Condition Edit Filter

Use this dialog box to create conditions or filters for terminating the fetching of data from recordsets and for executing operations. Select entries from the **Variables**, **Entity attributes**, or **Recordset fields** list to build string expressions in the **Condition/Filter String** box. Use the condition operand buttons to create your string or type the string directly into the box.

To reference attributes, fields, and variables, use the following syntax:

Attributes: `<AttributeName>`

Fields: `<RecordsetName>.<FieldName>`

Variables: `Variables.<VariableName>`

For example, create the following condition that addresses cursor syncing when the cursor is expected to arrive at one of two possible locations:


```
(Variables.CursorRow = 5 And Variables.CursorColumn =27)
```

-Or-

```
(Variables.CursorRow = 24 And Variables.CursorColumn =1)
```

The following condition operands are supported by the Host Integrator:

Condition operands	Description
=	Equal to
=*	Equal to (without being case-sensitive)
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=~	The regular expression matches the value expression. For example, object =~/[0-9]+/ means that object is numeric.
!~	The regular expression does not match the value expression.
And	And
Or	Or
Not	Not
(Left parenthesis
)	Right parenthesis
+	Add
-	Subtract
*	Multiply

Condition operands	Description
/	Divided by

Certain Host Integrator API methods use filter expressions as arguments (for example, see the `FetchRecords` and `SelectRecordByFilter` methods in the [Visual Basic Methods Reference](#)). See the [Host Integrator API Reference](#) for more information about the connectors. To view an example of creating a filter string, see the documentation for using the `Select Record` action in the `Test Recordset` dialog box.

Tip

You can use expressions on both the left and right sides of the condition statement. Make sure to enclose data in quotation marks.

More About Regular Expressions

A regular expression is a pattern that can match various text strings; for example, `I[0-9]+` matches `I` followed by one or more digits.

SYNTAX FOR PATTERN MATCHING

```
<ValueExp> =~ [m]/<RegEx>/[i][m] <ValueExp> !~ [m]/<RegEx>/[i][m] <ValueExp> =~ s/
<RegEx>/<ReplaceWith>/[g][i][m]
```

where

m This is the "Match" operator. It means read the input string expression (on the left of the `=~` or `!~` operator), and see if any part of it matches the expression within the delimiters following the `m`. The `=~` operator means return boolean true if there is a match; the `!~` operator means the opposite

s/ "Search and replace" or "substitution" operator. In the input string (to the left of the `=~` operator), the regular expression match is replaced. When the regular expression uses parentheses, backreferences (such as `$1`) can be used in the `ReplaceWith` string. The output is the result after replacements.

/i Ignore case option for case-insensitive pattern matching.

/g Global option. Replaces all regex matches in the string, not just the first one.

/m Multi-line option. Within the regular expression, caret (^) and dollar (\$) match the beginning and end of lines respectively within the string.

If your regular expression or replacement value contains the forward slash character (/), you can use a different delimiter symbol (such as !).

EXAMPLE

You can set up an expression to terminate a recordset where the last screen is identified by text on the screen stating the current page and the total number of pages, such as "Page 1 of 12" or "Page 12 of 12".

To terminate the recordset, define an attribute called PageStatus at the screen location where the "Page x of y" text occurs. In the Recordset Termination dialog, define the following termination condition:

```
PageStatus =~ s/Page ([0-9]+) of [0-9]+/$1/ = PageStatus =~ s/Page [0-9]+ of ([0-9]+)/$1/(
```

4.11 Event Handlers

4.11.1 Using Event Handlers

You can use an event handler to customize the behavior of a model.

The Design Tool offers a variety of features that assist you in creating event handlers. The result is a Java class that conforms to rules for event handling. This class is then mapped (attached) to specific objects of a model to customize its behavior.

You can attach event handlers to events associated with the entire model, a life cycle event, or to entities, attributes, operations, recordsets and recordset fields, and procedures. You can reuse a handler in multiple models or with multiple objects of the same type within the same model.

To create an event handler:

Navigate to the object in the model that relates to the behavior in the model that you want to extend or override.

Use the New Event Handler dialog box to generate a basic template that includes the supported events for that handler type.

Edit the template in your default editor to add the event functionality. Methods for all supported events are included in the template, but they are commented out (disabled). Enable the event by defining the corresponding method.

Rebuild the model JAR or .NET application extension file. Once the class is in the model JAR or .NET application extension file, it can be mapped to a model object. You can repeat the rebuild at any time, even if the event handler is already attached to an object.

Attach the handler to the selected object in the model. You can attach the same handler to multiple objects. No rebuild is required.

The procedure for testing the event handler will vary according to the type of event you are creating.

Build Event Handlers For Check-in To Source Control

A commandline utility is now available to build event handlers in a Windows Development Kit installation. Close the Design Tool before running the following batch files.

In the `VHI\bin` directory, there are two files:

`buildeventhandlers.bat` - builds event handlers for a specified model

`cleaneventhandlers.bat` - cleans executables from the event handler, leaving behind source code. Used to prepare the model for check in to source control.

The batch files each take a single argument. The argument is a path to the directory containing the model with event handlers.

Working With Existing Models

To use the `cleaneventhandlers.bat` file with existing models from VHI 6.6 or earlier, you can either copy the new `build.xml` from `<VHI install folder>\lib\scripts\java\templates`, or edit the custom `build.xml` by adding the following commands to the clean model target:

```
<delete file="{vhi_model.jar}" >  
<delete file="tmp/vhi_model.jar" >
```

EXAMPLE

Review a [sample event handler](#) that reformats currency amounts in recordset fields.

About Event Handlers

Event handling is a feature in Verastream Host Integrator that extends the capabilities of Host Integrator models by allowing you to define specific events that suspend the interpretation of a model and turn control over to user-supplied procedural code.

One use of event handlers is to improve the presentation of host data in new client applications. For example, you can use event handlers to:

- Convert cryptic host application codes to user-friendly descriptions
- Convert formats for currencies or dates
- Secure access to sensitive data
- Create other functions that might otherwise require custom client-side programming

You can also use event handlers to extend the Host Integrator error-handling capabilities. For example, you can add an event handler at the point the error occurs and then implement event-handler code that intercepts the error, takes control, and corrects the error.

Implementing Event Handlers

You can attach an event handler to any of the following model objects. For each of these eight varieties of event handler, there are one or more defined events. Click each model object type to see details on the events and possible uses for event handlers associated with each.

The [model](#)

The "life cycle" (events such as Model Load, Model Unload, Authenticate User) that occur as the server manages sessions with the model.

An [entity](#)

An [attribute](#)

An [operation](#)

A [recordset](#)

A [recordset field](#)

A [procedure](#)

Creating, Building, and Deploying Event Handlers

Use the Design Tool to review and select event handler attachment points, configure your event handler development options, and build event handler templates. (The templates are code stubs that provide a starting point for your own event handler.) You can build, load, attach, and test event handlers within the Design Tool.

The current version of Host Integrator supports development of event handlers in Java. After you create an event handler template in the Design Tool, you can implement the handler with any appropriate third-party editor. Event handlers are packaged and deployed with a Host Integrator model.

Once event handlers are deployed, they are managed by the script manager, which runs with both the Design Tool and the Session Server. An event source is a specified point in the execution of a model, such as the reading of an attribute, the authentication of a user, or the writing of a recordset field. An event source generates an event only if the corresponding event method is implemented in the attached event handler. The event delivered to the event handler describes the environment in which the event occurred. The event handler code uses this description to accomplish its task.

In a Development Kit installation, you can build Java or .NET event handlers outside Design Tool.

The following batch files are located in the `C:\Program Files\Microsoft Focus\Verastream\HostIntegrator\bin` folder:

- `buildeventhandlers.bat`: Builds event handlers for a specified model. For example, `<model folder>\scripts\lib\vhi_model.jar` is created for Java event handlers.
- `cleaneventhandlers.bat`: Cleans executables from the specified model event handler subfolder, thus retaining source code only. Used to prepare the model for checking into a third-party source control system.

NOTE THE FOLLOWING

- Close the Design Tool application before running these batch files.
- Each batch file requires a single command line argument, the complete model directory path.
- For Java, the `cleaneventhandlers.bat` file requires the `build.xml` file to be present in the `<model folder>\scripts` folder, which is automatically generated for models created by Design Tool version 6.6.188 and higher. For existing older models, you can either copy `build.xml` from `<VHI>\lib\scripts\java\templates`, or edit your custom `build.xml` by adding the following commands to the clean model target:

```
<delete file="{vhi_model.jar}"/>
```

```
<delete file="tmp/vhi_model.jar"/>
```


- For .NET, the `cleaneventhandlers.bat` file requires the `build.proj` file to be present in the `<model folder>\scripts` folder, which is automatically generated for models created by Design Tool version 7.0. and higher.
- After running `buildeventhandlers.bat` and re-deploying an existing model on the session server, the updated event handlers do not take effect for existing idle pooled sessions (since

the model version is not updated) and new client connections until you do one of the following:

- Undeploy the model using the `deactivatemodel` command before re-deploying -or-
 - After re-deploying, stop and start the associated session pools in Administrative Console, or restart the session server
- or-
- After re-deploying, in the Administrative Console Sessions view, select idle sessions, and Terminate the Session from the Host. Idle pooled sessions are automatically re-connected as specified by each pool's configured minimum idle sessions.
 - Instead of using `buildeventhandlers.bat`, rebuild the event handler within the Design Tool before you re-deploy the model. This updates the model version. Idle pooled sessions are automatically re-connected.

Features

Host Integrator has tools and visual indicators to assist you as you create and incorporate event handlers into your model:

- Design Tool visual indicators
 - Within the Design Tool, objects with attached event handlers are identified with a lightning bolt  displayed to the left.
 - The status bar includes symbols that indicate if a script manager reset is needed or if event handlers should be updated.
 - The event handler toolbar offers shortcuts for standard Design Tool tasks associated with event handlers
- Event Handler configuration

Use event handler settings to configure your editing, building, debugging, and environment preferences.

You can modify other configuration options in properties files. You can also set up security for event handlers using configuration files.
- Developing event handlers

Use the options on the Events menu to build, edit, and attach event handlers to objects. From this menu, you can also open the libraries and sources folders for event handlers, reload and rebuild handlers, and disable event timeouts.
- API support

There is complete API support for event handlers, for both Java and .NET. See the [Event Handling API](#) topic.
- Examples

An event handler [example](#) is provided with Host Integrator.
- Debugging tools for event handlers

Additional tools are available for [debugging event handlers](#).

Best Practices

For best results within the event handler runtime environment, follow these guidelines for optimizing your event handlers:

[Design considerations](#)

[Do not implement unnecessary event handlers](#)

[Do not store state information in handler classes](#)

[Synchronize the model hash table](#)

[Use of multithreaded objects is discouraged](#)

[Limit use of output streams](#)

Use a class path folder for JAR files that do not contain event handlers

DESIGN CONSIDERATIONS

The runtime environment for Host Integrator event handlers is similar to that of JSP servlets.

Handlers are treated as stateless objects running in the context of a single thread. State information is stored in two system-administrated hash tables:

- One for each model, available to all client and host sessions using that model
- One per client session, where each client connection creates a new client session

The [ModelContext](#) and [ClientSession](#) interfaces provide methods to add, remove, and access objects stored in the respective hash tables.

Note

For Java, you can use java resources in Host Integrator event handlers. For .NET, you can use third party or your own custom assemblies in Host Integrator .NET event handlers by placing them in the model's `scripts\lib` folder

DO NOT IMPLEMENT UNNECESSARY EVENT HANDLERS

Event handlers are intended to override or augment default behavior. Don't implement an event handler that makes no change to the default behavior, as it will affect server performance.

DO NOT STORE STATE INFORMATION IN HANDLER CLASSES

Handler classes should not generally use member objects to store state information. This state information often spans client sessions, which creates the risk of crosstalk and makes state information vulnerable to unexpected behavior by the model in the context of an error. If a situation arises where state members are needed, access to the member object must be thread-safe (accessible by multiple threads at once, as in read-only objects, or making use of locks to prevent simultaneous access).

SYNCHRONIZE THE MODEL HASH TABLE

Objects stored in the model hash table must also be thread-safe, as multiple client sessions could gain simultaneous access to them. The only events that are guaranteed to have sole access to the model hash table are Model Loaded and Model Unloaded. Objects stored in the client hash table are guaranteed to only be accessed by a single thread at a time; a client session may have several events in progress, but only one can actually be executing code at a time.

USE OF MULTITHREADED OBJECTS IS DISCOURAGED

Just as in many JSP Servlet environments, Host Integrator does not actively prevent use of objects that start additional threads. However, as the event handlers are event driven via the controlling Host Integrator application, there is no explicit support for such objects because there is little direct advantage to be gained. In many cases, Host Integrator's behavior is dependent on host responses or client inputs; under these constraints, multithreaded event handlers can be easily placed in positions that rely on undefined behavior to function correctly. If you must use undefined threads, use sound Java programming concepts.

LIMIT USE OF OUTPUT STREAMS

Limit use of System.out and System.err streams. Blind calls to the output streams are wasteful and not a best practice. Using these outputs has a performance impact (often up to 15%). One recommendation is to code each call to check for applicability before the arguments are constructed, assembled, and the call made. The criterion for whether a call is applicable may vary, but the following example addresses the problem:

```
if (event.isDesignEnvironment()) {  
    System.out.println("Here I am. The count is " + count);  
}
```

```
if (vsEvent.IsDesignEnvironment())  
{  
    Console.WriteLine("Here I am. The count is " + count);  
}
```

This simple check prevents the allocation of a StringBuffer, a String append, conversion of an int to a String, another String append, entering/exiting of the stream's synchronization monitor, and a few other steps.

USE A CLASS PATH FOLDER FOR JAR FILES THAT DO NOT CONTAIN EVENT HANDLERS

JAR files that do not contain event handler classes should be placed on the user class path, not in the model's `\scripts\lib` directory, to avoid the overhead of introspection. Introspecting more than 1 MB of JAR files in the model's event handler directory may cause delays in the Design Tool and slow down Server startup.

More information

[Event Handler Guidelines](#)

[Troubleshooting Event Handlers](#)

4.11.2 Guidelines for Developing Event Handlers

Use this information on the structure and behavior of event handlers to develop event handlers for objects in a model. You should also consult the event handler examples (located in `Micro Focus\Verastream\HostIntegrator\examples\EventHandlers\java\documentation\index.html`) that are available as part of the Host Integrator Development Kit.

Defining an Event Handler

Use the New Event Handler dialog box in the Design Tool to generate Java source code that is a template for the new event handler class. Each handler is a class that is associated with a specific type of model object (attribute, operation, procedure, etc.) and can be attached to one or more components of that type.

By definition, the event handler class extends a base class specific to the type of model object. This base class is defined as part of Host Integrator.

For each object type, the base class defines a method signature for each supported event. For example, a model event handler has these defined events:

- Client Connected
- Client Disconnected
- Error Reported
- Execute Login
- Execute Logout
- Format Error
- Move Cursor
- Move Cursor Forward
- Move Cursor Backward
- Process String
- Unrecognized Screen

In the user derived class, the declaration of an override for an adapter method signals that the corresponding event is enabled. Otherwise, that event is considered disabled, which means any object that has its events mapped to the user class will not fire the event.

For example, the `AttributeEventHandler` base class defines two methods for events:

JAVA

```
void readAttribute(AttributeEvent e) throws ApptrieveException; void  
writeAttribute(AttributeEvent e) throws ApptrieveException;
```

.NET

```
public override string ReadAttribute(IReadAttributeEvent vsEvent) public override  
void WriteAttribute(IWriteAttributeEvent vsEvent)
```

If a user class overrides the writeAttribute method of the AttributeEventHandler base class, then the write event is considered enabled and the read event is considered to be disabled. You can also [extend the event handler definition](#).

SETTING TIMEOUTS

A model's default event handler timeout setting applies to all events. See information on [general timeout processing](#) for more information.

- Java - If a particular event requires a different value, a value can be assigned per event by declaring a static final member of the user's handler class. For example, to set the timeout for the Write Attribute event on a handler to 30 seconds, use this statement:

```
static final int writeAttributeTimeout = 30;
```

- .NET - If a particular event requires a different value, a value can be assigned per event by using the Timeout attribute. For example, to set the timeout for the Write Attribute event on a handler to 30 seconds, use this statement:

```
[Timeout(30)]
```

```
public override void WriteAttribute(IWriteAttributeEvent vsEvent)
```

EVENT HANDLER INTERFACES

The event handler user class has access to all relevant interfaces by calling `get<MethodType>` methods on the `<EventType>Event` interface passed into the method. These interfaces include:

ModelContext— information about the model, its environment (system name), and its state (handler-populated hash table visible to all model sessions).

ClientSession— data about the connected client, such as the user name and ID passed to Host Integrator, and its state (handler-populated hash table visible only to this client).

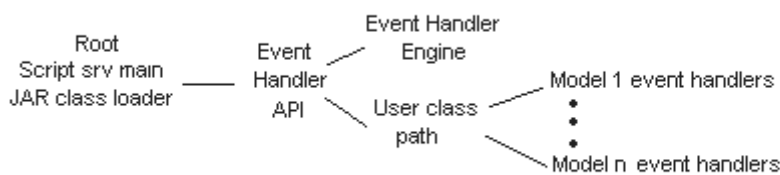
ScriptHostSession— access to the terminal and the model objects. This is an expanded version of the Host Integrator connector interface available to the client in prior versions of Host Integrator.

Handler-specific methods— access to metadata information on the object being scripted and to any default model behavior defined for it.

JAVA CLASSPATHS

A handler is permitted to use any available Java library to assist in meeting the requirements of its script. All libraries placed in the model's `scripts\lib` directory are included in the classpath for that model, and are introspected for event handler classes. In addition, you can specify a classpath that applies to the local Host Integrator installation with a `[properties file]#event-handler-properties-files`).

The script manager uses the class loader scheme shown below. The libraries in each model are isolated from each other, and the user class path will take precedence in loading classes. The model's class loader does not use manifest files to find nested JAR files; only class files included directly in the JAR are available.



PROCESSING JAVA CODE INTO AN EVENT HANDLER

After the source code is constructed, it is compiled into a Java class (.class file) and the class is placed in a Java Archive (.jar file). It is then available for use as an event handler. Host Integrator provides an Ant script that compiles all Java source files in the model's `\scripts\src` folder into Java classes in the `\scripts\classes` folder, and then stores the class files in a single Java Archive in the `\scripts\lib` folder. Once incorporated in the JAR file, all classes that extend handler adapters for a given object type can be mapped (attached) to any object in that model.

An available event handler can then be assigned to a model object. Most object types have an Advanced Properties option that allows you to attach the handler; the Attach Event Handlers dialog box provides access to all model object types and all available event handlers.

For a given model, all JAR files in the `scripts\lib` directory will be introspected using the Java reflection API for available event handlers. It is recommended that you use this folder only for event handlers.

MODIFYING THE JAVA EVENT HANDLER BUILD ENVIRONMENT

You can modify the ant script (build.xml), located in the `models\<modelname>\scripts` folder, but remember that the Design Tool uses the following ant targets:

`buildfile`—used to build a single source file and add it to `vhi_model.jar`. The file is specified by the `src.files` ant property.

`cleanbuild`—used to rebuild all source files and create `vhi_model.jar`.

In both cases, the Design Tool defines the following ant properties:

vhi.jar.dir—the location of the Host Integrator JAR files for event handling.

vhi.script.classpath—the user classpath as defined in script.properties.

.NET ASSEMBLIES

A handler is permitted to use any available .NET Application Extension (DLL) to assist in meeting the requirements of its script. All .NET Application Extensions placed in the `model's scripts\lib` directory are included in the assembly for that model, and are introspected for event handler classes.

PROCESSING .NET CODE INTO AN EVENT HANDLER

After the source code is constructed, it is compiled into a .NET Application Extension (DLL). It is then available for use as an event handler. Host Integrator provides a project file that msbuild uses to compile all .NET source files in the model's `\scripts\src` folder and subfolders into a .NET Application Extension in the `\scripts\lib` folder. Once msbuild has completed, all classes that extend handler adapters for a given object type can be mapped (attached) to any object in that model.

An available event handler can then be assigned to a model object. Most object types have an Advanced Properties option that allows you to attach the handler; the Attach Event Handlers dialog box provides access to all model object types and all available event handlers.

MODIFYING THE .NET EVENT HANDLER BUILD ENVIRONMENT

You can modify the project `build.proj`, located in the `models\<modelname>\scripts` folder. The Design Tool will invoke msbuild so that the default target is built.

ScriptHostSession API

The Event Handler [ScriptHostSession API](#), in combination with other event handler guidelines, will help you develop event handlers that override or extend standard model behavior.

An event handler performs a mix of custom logic and callbacks to Host Integrator to get information about or modify the state of the model or terminal.

An invocation takes place any time an event handler invokes a method on one of the `ScriptHostSession` interfaces. This may just be asking for information, or it may be a request to modify the model or terminal state.

Note the following properties of the `ScriptHostSession` API:

- No specific timeout. The governing timeout is that of the event itself. The time allowed per callback extends to the remaining time allotted to the event.
- The event can return the expected data, "normal errors" such as a bad parameter name or a runtime failure in the model, or an event timeout error if the event timeout expires during execution.

THE SCRIPTHOSTSESSION API

The ScriptHostSession API is documented in the [Javadocs for the Event Handler API](#) and the .NET Event Handler API (available in a Windows help file). You can use this interface to get information about the host session and to manipulate the terminal (whether at the terminal, model, or table level).

SCRIPTHOSTSESSION LIMITATIONS

- For events that rely on a specific terminal state, callbacks that could change terminal state are disabled within that event. If you attempt this sort of callback, you'll see an error message similar to the following:

```
Event handler callback not allowed in current state.
```

```
An event callback method was invoked at an inappropriate time.
```

Click Details in the error message dialog box to see the sequence that preceded this error. The method immediately preceding the error can often tell you the offending ScriptHostSession method. Many events within recordset event handlers (for example, Apply Filter, Parse Screen) and recordset field event handlers (Read Field) disable most callbacks:

- The ScriptHostSession interface cannot be used outside of the event from which it was obtained. In the following events, this interface is not available:

Entity Departure (entity event handler)

Format Error (model event handler)

- Most events with life cycle event handlers
- In addition to the limitations within the immediate event handler, `ScriptHostSession` restrictions apply when event handlers are nested. If an event with restrictions is currently open, this limitation will also be enforced for a nested event.

Nested Events

Event handler nested events, in combination with other event handler guidelines, will help you develop event handlers that override or extend standard model behavior.

Host Integrator's event handler architecture permits an event handler callback to invoke another event handler.

EXAMPLE

A procedure event handler can issue a request to execute an operation. If the model happens to map the operation to an event handler, then a nested event handler invocation takes place. In this runtime situation, the operation event is considered to be the child or descendant of the procedure event, while the procedure event is defined as the parent or ancestor of the operation event. This is a transitory relationship that exists only as long as both event handlers are in progress.

Further, the operation event handler can request to read data from an attribute, where the attribute has a handler with the Read Attribute event defined. This process of nesting can continue to whatever level is dictated by the callback and event handler constructs of the user model.

LIMITATIONS

Host Integrator imposes one limitation on nested events: the same event on the same object cannot be signaled more than once in the same nesting structure. A nested request to invoke the same event on a given object results in an error being reported to the event handler that issued the callback request.

ERRORS

Errors returned from a nested event are propagated to the parent event handler in the form of an `ApptrieveException` result on the parent event handler callback. Host Integrator may add one or more additional messages to the stack when formulating its callback response.

When a nested event returns an exception, that error is propagated to the immediate ancestor event. If the exception is not an `EventTimeoutException`, the ancestor event is permitted to catch the resulting Java exception and proceed with the logic in the "catch" portion of the try-catch block surrounding the callback request. If the event handler completes its task within the time allotted and does not return an exception, then the event is considered a success and normal processing continues.

TIMEOUTS

With nested events, the event with the smallest amount of remaining time governs the interval before a timeout notification takes place.

In the case discussed here, a procedure includes an operation that includes a request to read from an attribute. If a procedure with a 30-second timeout has been running for 15, an operation with a 10-second timeout has been running for 7, and the attribute has a 5-second timeout, the attribute event times out in 3 seconds (when the operation timer expires).

When a timeout occurs in a nested situation, some special processing rules apply.

At any given time, only one hierarchy of events can be in progress. The first event in the hierarchy is the primary event. All the errors reported from primary event handlers result in failure of the client request, but this is not the case for nested events.

If the operation times out in the example used here, both the nested attribute and operation event handlers receive and return `EventTimeoutException` results. For the procedure event handler containing the primary event, the operation timeout represents a simple failure. The procedure event handler receives an `ApptrieveException` result reporting the timeout. The procedure event can attempt recovery or otherwise continue processing to completion.

Session Startup/Shutdown

The initialization and shutdown sequence information below, in combination with other event handler guidelines, will help you develop event handlers that override or extend standard model behavior.

MODEL LOADING AND UNLOADING

The behavior for model loading and unloading within the Design Tool differs slightly from the same sequence in the Session Server.

MODEL LOADING

When a model is loaded in either the Design Tool or the Session Server, the following sequence occurs:

Java

The Design Tool copies all JAR files in the model's `\\scripts\\lib` directory to a temporary working area. On the Session Server, the deployment directory is the working area.

Host Integrator notifies the script manager of the new model configuration, passing the model name, model directory, and the location of the working area. The Session Server also passes a copy of all model metadata to be cached in the script manager.

An infrastructure, consisting of the created data accessible through the `ModelContext` interface and a class loader for the JARs in each model's working area, is built.

Each JAR file in the event handler working area is inspected for classes that extend an event handler. For any class that is found, the methods and applicable timeout overrides are catalogued for return to Host Integrator.

Upon return, if the model implements the `Model Loaded` event, that event is fired.

All errors returned by this event are reported and then ignored in the Design Tool.

In the Session Server, errors can cause the model load to fail. The model is then unavailable for pools or dedicated sessions.

.NET

The Design Tool copies all assemblies (.DLL files) in the model's `\\scripts\\lib` directory to a `\\scripts\\tmp`. On the Session Server, the deployment directory is the working area.

Host Integrator notifies the script manager of the new model configuration, passing the model name, model directory, and the location of the working area. The Session Server also passes a copy of all model metadata to be cached in the script manager.

An infrastructure, consisting of the created data accessible through the `IModelContext` interface and a `.NET AppDomain` for the assemblies in each model's working area, is built.

Each assembly in the event handler working area is inspected for classes that extend an event handler. For any class that is found, the methods and applicable timeout attributes are catalogued for return to Host Integrator. To be able to inspect the classes, the location from which they are loaded must be fully trusted. If you load classes from a network share, make sure this share is trusted.

Upon return, if the model implements the `Model Loaded` event, that event is fired.

All errors returned by this event are reported and then ignored in the Design Tool.

In the Session Server, errors can cause the model load to fail. The model is then unavailable for pools or dedicated sessions.

MODEL UNLOADING

- Design Tool - You unload a model when you close the Design Tool or open a new model. The act of opening a new model over an existing model causes the existing model to be

unloaded. Reloading event handlers (for example, after a rebuild) also causes a model unload.

- Session Server - The Session Server unloads a model during shutdown and when it is removed from or updated in the server configuration. The following steps take place:
- Each host session is instructed to log out when the current client finishes. The host session is then destroyed.

Once all sessions have been destroyed, the Model Unloaded event is fired.

The command to remove the model is sent to the script manager. Among other actions, this includes destroying the Java class loader that was used to access the model's script manager work area.

The deployment directory is made available for clean up.

A change in the version of a model deployed to a Session Server results in a model unload/load sequence. This means that model versions can be changed and new versions of event handlers loaded without restarting the server.

CREATING AND DESTROYING HOST SESSIONS

Whenever a host session is created by Host Integrator, the Host Session Created event is fired. Similarly, when a session is destroyed, the Host Session Destroyed event is fired.

- Design Tool - You create a host session when you load a new model. Whenever you unload a model, the existing host session is destroyed.
- Session Server - On the Session Server, all of the existing rules about configuration changes to models and session pools still apply. Host sessions are created at startup and destroyed on shutdown for session pools. When a new model version is loaded, any existing host sessions with that model are destroyed and recreated. Changing pool sizes up or down can also result in sessions being created or destroyed. Most other changes do not result in host sessions being created or destroyed.

Sessions are created when a client connects using the `connectToModel` API. The Host Session Destroyed event never fires when a client is actually connected to the host session.

CLIENT SESSIONS AND AUTHENTICATION

The client session represents a client program accessing Host Integrator.

- Design Tool - As long as there is a model loaded, there is always a single simulated client session. The Authenticate User and Client Session Created events are fired when you load the model, and the Client Session Destroyed event is fired when you unload the model. You are informed of any errors generated by the Authenticate User event via the Java console, but this does not prevent the client session from being created.

In the Design Tool, the Client Connected event and the Client Disconnected event are attached to login and logout buttons to approximate the behavior of a client connecting to and disconnecting from a session. In this setting, a login process may be executed twice without an intervening logout process. However, after an initial login and associated Client Connected event fires, no additional Client Connected event will fire until a logout occurs.

- Session Server - Each session spans one client connection. The Authenticate User event fires when a host session is allocated to an incoming client. If the Authenticate User event returns an error, the client session is not created. Otherwise, the client session is created and the Client Session Created event is fired. The Client Session Destroyed event is fired after the client has disconnected. If the Client Session Created event or Client Session Destroyed event fails, the client connection fails.

Error Processing

Event handler error processing, in combination with other event handler guidelines, will help you develop event handlers that override or extend standard model behavior.

In event handlers, errors are created when a Java exception is thrown. Here are a few examples:

The user code creates an exception to communicate an error to Host Integrator.

A runtime error returned from a Host Integrator callback is reformulated into an `ApptrieveException`.

A timeout condition communicated to the script manager causes any callback request to return a locally generated `EventTimeoutException`.

The user code has the ability to trap all exceptions thrown. The following rules apply to the user code for event handler errors:

- To communicate standard application errors, an event handler should create and throw an `ApptrieveException`. This exception can contain multiple messages, each of which is converted into an error message on Host Integrator's internal stack.
- Any exception thrown from an event handler that is not an `ApptrieveException` or a subclass results in a single error message being placed on the Host Integrator stack showing the exception class and the string returned from `getMessage()`.
- An event handler may trap and ignore any `ApptrieveException` it receives from a callback. In this case, however, no error is logged or returned to the client application. If informational logging is turned on, the only evidence of the error condition is an informational log message saying an error was returned to an event handler. Otherwise, Host Integrator does not generate any persistent record of the error.
- As an extension of bullet 3, an event handler may trap one exception and then throw an entirely new exception with one or more custom messages. This operates as if the event handler initiated the error report.
- An event handler may trap an `ApptrieveException`, append its own message, and re-throw the exception. The appended message appears with others generated during the client request.
- An event handler is obligated either to not catch or to re-throw an `EventTimeoutException`. This exception indicates a system state that requires the event handler to terminate and control to be returned to Host Integrator. In any case, any return from the event handler other than the `EventTimeoutException` is ignored in favor of creating a new `EventTimeoutException`.

In addition to the standard Host Integrator error stack reporting, any exception caught from user code that is not an `ApptrieveException`, a subclass of `ApptrieveException`, or an `EventTimeoutException` results in the Java stack dump being printed to the [Event Handler Console] (need link) or the [designated output file] (need link) (by default, this is the `traps.out` file in the `<VHI>\etc\output` subdirectory).

Timeout Processing

Event handler timeout processing, in combination with other event handler guidelines, to develop event handlers that override or extend standard model behavior. You can check timeout status using `checkForTimeout` or `IEvent.CheckForTimeout` for .NET.

When an event timeout expires, there are several possible processing states. The most common states are:

- Event handler is executing

This means that the script manager process has control of the client session to execute the logic in the user's event handler. Host Integrator sends a timeout notification that identifies the host session that has entered timeout processing. The event handler code can also periodically check for this state by invoking `checkTimeout()` on the callback interface.

In this state, most callback invocations result in an immediate return of an `EventTimeoutException`. The callbacks that do not result in this exception are:

Java

ScriptHostSession Interface: `getCursorPosition`, `getRectangularTerminalRegion`, `getLinearTerminalRegion`, `getTerminalScreenSize`, `getTerminalScreen`, `getStringAtCursor`, `getStringAtOffset`, `getStringAtLocation`, `getPatternRegion`, `getPatternRegions`, `getAttributeRegion`, `getAttributeRegions`, `getRecordSetRegion`, `getRecordRegion`, `getFieldRegion`, `getFieldRegions`

Event Interface: `getLogger` method for log level manipulation on the `Logger` interface (`isLogInfoEnabled`, `isLogWarningEnabled`, `logInfoMessage`, `logWarningMessage`).

.NET

IScriptHostSession: `CursorPosition`, `GetRectangularTerminalRegion`, `GetLinearTerminalRegion`, `TerminalScreenSize`, `TerminalScreen`, `GetStringAtCursor`, `GetStringAtOffset`, `GetStringAtLocation`, `GetPatternRegion`, `GetPatternRegions`, `GetAttributeRegion`, `GetAttributeRegions`, `GetRecordSetRegion`, `GetRecordRegion`, `GetFieldRegion`, `GetFieldRegions`

Event Interface: `Logger` method for log level manipulation on the `ILogger` interface (`LogInfoEnabled`, `LogWarningEnabled`, `LogInfoMessage`, `LogWarningMessage`)

- **Callback** is executing

This occurs when the event handler makes a request to Host Integrator, and that request is being processed when the timeout expires. In this case, the same processing applies except that the callback return is a timeout error that is converted into an `EventTimeoutException` in the script manager.

Additional error states

Once the timeout status has been communicated to the script manager, Host Integrator begins another timer that governs how long the event handler is given to complete and return control:

- If the event handler returns control within the time allotted, the errors are reported to the client and the session remains intact.
- Abort mode: Should this new timer also expire, Host Integrator enters into abort mode. Host Integrator returns an event handler failure message to the invoking object (attribute, operation, etc.) and communicates this new state to the script manager. The script manager "quarantines" the event handler and its script manager session in a processing space separate from normally functioning sessions and handlers. If this expired handler returns, the response is ignored in favor of an `AbortedSessionException` in the standard unhandled exception locations. Once the client has been notified of these fatal errors, the host session is terminated and restarted. The Design Tool always leaves the terminal intact and creates a single new session; the Session Server will destroy the entire host session, including the virtual terminal, and will only recreate it if a session pool minimum would not otherwise be met.

Abort mode may have some unpredictable side effects. Each abort results in at least one quarantined thread in the script manager that could theoretically exist for the life of the process. If the host session requires access to the script manager to log out of the host properly, this can also fail with unpredictable results for the host (for example, the host could lock the account for a period time). It is strongly recommended, therefore, that event handlers be written to conform to the timeout processing rules and that the handlers check the timeout status periodically during tasks that could take significant time to complete.

Note

Timeout processing for nested events differs.

Importing Events Handlers

You can use an event handler in more than one model. To import existing event handlers created in one model into another model, you can do either of the following:

Java

- Copy the `.java` files into the `\scripts\src` folder within your model folder. You must then rebuild event handlers in order to incorporate them into the `.jar` file for the model.
- Copy the `.jar` file into the `\scripts\lib` folder within your model folder. Since the default `.jar` file for all models is `vhi_model.jar`, you may need to rename the `.jar` file that you are importing. By default, you do not have to rebuild event handlers. Host Integrator automatically loads the `.jar` file as soon as it is detected.

Note

The build process will attempt to rebuild all event handlers. You must manually delete any event handler .java files from the `\scripts\src` folder if you do not want them included in the .jar file.

.NET

- Copy the source (.cs) files into the `\scripts\src` folder within your model folder. You must then rebuild event handlers in order to incorporate them into the .NET Assembly file for the model.
- Copy the `vhi_model.dll` file into the `\scripts\lib` folder within your model folder. Since the default Application Extension file for all models is `vhi_model.dll`, you will probably need to rename the .dll file that you are importing. By default, you do not have to rebuild event handlers. Host Integrator automatically loads the .NET Assembly Extension file as soon as it is detected.

Note

The default rebuild process will attempt to rebuild all event handler C# source files. You must manually delete any event handler C# source files from the `\scripts\src` folder if you do not want them included in the .NET assembly, or explicitly exclude them in the project file, `build.proj`.

Extending Event Handlers

The general guidelines for developing event handlers provide basic information on defining an event handler by defining a Java class that directly extends the Host Integrator-defined event handler base class. However, it is permissible to use more complex class hierarchies that extend the base class through multiple subclasses.

Java

```
`class A extends AttributeEventHandler
{
    public static final int readAttributeTimeout = 50;
    void readAttribute(ReadAttributeEvent e) throws ApptrieveException;
    public String myRandomExampleMethod();
}
class B extends class A
{
    void readAttribute(ReadAttributeEvent e) throws ApptrieveException;
    void writeAttribute(WriteAttributeEvent e) throws ApptrieveException;
}
class C extends class B
{
    public static final int readAttributeTimeout = 100;
}`
```

All three of the above classes are valid attribute event handlers that can be mapped into a model.

Class A implements only the Read Attribute event and uses a 50 second timeout.

Class B implements its own version of both attribute events. It will inherit the 50 second timeout for Read Attribute from class A and use the model's default timeout for Write Attribute. Following standard Java inheritance rules, it inherits the method `myRandomExampleMethod()` from class A and can invoke it.

Class C also implements both attribute events—it inherits those defined in class B. In addition, class C defines a timeout override for the Read Attribute event of 100 seconds. For class C, this new definition supercedes the override from class A.

.NET

```
public class A : AttributeEventHandler
{
    [Timeout(50)]
    public override string ReadAttribute(IReadAttributeEvent vsEvent)
    {
        return vsEvent.DefaultReadAttribute();
    }
    public String MyRandomExampleMethod()
    {
    }
}
class B : A
{
    public override string ReadAttribute(IReadAttributeEvent vsEvent)
    {
    }
    public override void WriteAttribute(IWriteAttributeEvent vsEvent)
    {
    }
}
class C : B
{
    [Timeout(100)]
    public override string ReadAttribute(IReadAttributeEvent vsEvent)
    {
    }
}
```

All three of the above classes are valid attribute event handlers that can be mapped into a model.

Class A implements only the Read Attribute event and uses a 50 second timeout. Note that the .NET event handlers use a `Timeout` attribute to set the timeout period. The `Timeout` attribute has inheritance enabled.

Class B implements its own version of both attribute events. This example will inherit the 50 second timeout for Read Attribute from Class A and inherit the model's default timeout (`EventHandlerConstant.DefaultTimeout`) for Write Attribute. Following standard .NET inheritance rules, it inherits the method `MyRandomExampleMethod()` from class A and can invoke it.

Class C also implements both attribute events—it inherits those defined in class B. In addition, class C overrides the value of the `Timeout` attribute for the Read Attribute event, setting it to 100 seconds.

Example: Attaching and Testing an Event Handler

Examples are available for Java and .NET event handlers.

Example models are located at `Program Files\Microsoft Focus\Verastream\HostIntegrator\`
`\models\CCSDemo`

This Java example attaches and tests a sample event handler for reformatting currency amounts in recordset fields. Before you begin, you should review the [basic steps for creating an event handler](#).

This example uses two Java files:

- `CurrencyFormatter.java` - performs the reformatting job of prepending a dollar sign. This file is in the `Micro Focus\Verastream\HostIntegrator\examples\EventHandlers\java\shared\src\shared\formatting` folder. The `CurrencyFormatter` can be reused to reformat currencies in other model elements, such as attributes (such as `CurrencyAttributeFormatter.java`, also included in the examples folder).
- `CurrencyFieldHandler.java` - reformats currencies in recordset fields. This file is in the `Micro Focus\Verastream\HostIntegrator\examples\EventHandlers\java\FieldEvents\FieldFormatting\src` folder.

Follow these steps to implement the currency formatting event handler:

From the Design Tool, open the CCSDemo model.

Save the model.

Click Connect on the toolbar.

Create a folder named `fieldformatting` under `<Documents>\Micro Focus\Verastream\HostIntegrator\models\CCSDemo\scripts\src\` and copy `CurrencyFormatter.java` and `CurrencyFieldHandler.java` to the folder.

From the Settings menu, click Event Handlers. On the Technology tab, verify that Java is selected, and then click OK.

On the Events menu, click Rebuild. The Build Output window indicates when the build is complete.

In the Entity window, select the AcctTransactions entity.

Click the Recordset tab, and then click the Fields subtab.

Select the Amount Field, then click the Advanced Properties button.

In the Advanced Recordset Field Properties dialog box, select `fieldformatting.CurrencyFieldHandler` from the Event Handler list. This dialog box also has options for reviewing the properties of the event handler or inspecting the code in your default Java editor.

Click Close to attach the event handler to the Amount Field. When you return to the Fields tab, the Amount field now has a lightning bolt displayed to the left, indicating that an event handler has been attached.

Click Apply to activate your changes.

To test the event handler, select Recordset Test from the Debug menu. Confirm that AcctTransData is listed as the Recordset and Fetch Records is listed as the Action.

Click Execute. After the test is complete, the returned values in the Amount column have a dollar sign and negative amounts are surrounded by parentheses, as they will be when returned to a client application.

.NET

This .NET example uses two C# files:

`CurrencyFormatter.cs` - Performs the reformatting job of prepending a dollar sign. This file is in the `Micro Focus\Verastream\HostIntegrator\examples\EventHandlers\dotnet\shared\formatting` folder. The `CurrencyFormatter` can be reused to reformat currencies in other model elements, such as attributes.

`fieldformatting.CurrencyFieldHandler.cs` - Reformats currencies in recordset fields. This file is in the `Micro`

`Focus\Verastream\HostIntegrator\examples\EventHandlers\dotnet\FieldEvents\FieldFormatting\src` folder.

Follow these steps to implement the currency formatting event handler:

Start the Design Tool and open the CCSDemo model.

Save the model.

Click Connect on the toolbar.

Create a folder named fieldformatting under `<Documents>\Micro Focus\Verastream\HostIntegrator\models\CCSDemo\scripts\src\` and copy CurrencyFormatter.cs and fieldformatting.CurrencyFieldHandler.cs to the folder.

From the Settings menu, click Event Handlers. On the Technology tab, verify that .NET is selected, and then click OK.

On the Events menu, click Rebuild. The Build Output window indicates when the build is complete.

In the Entity window, select the AcctTransactions entity.

Click the Recordset tab, and then click the Fields subtab.

Select the Amount Field, then click the Advanced Properties button.

In the Advanced Recordset Field Properties dialog box, from the Event Handler drop down list, select `fieldformatting.CurrencyFieldHandler`. This dialog box also has options for reviewing the properties of the event handler or inspecting the code in your default C# editor.

Click Close to attach the event handler to the Amount Field. When you return to the Fields tab, the Amount field now has a lightning bolt displayed to the left, indicating that an event handler has been attached.

Click Apply to activate your changes.

To test the event handler, select Recordset Test from the Debug menu. Confirm that AcctTransData is listed as the Recordset and Fetch Records is listed as the Action.

Click Execute. After the test is complete, the returned values in the Amount column have a dollar sign and negative amounts are surrounded by parentheses, as they will be when returned to a client application.

Removing an Event Handler

You can use options within the Design Tool to detach event handlers from Host Integrator components. If you also want to remove associated files, additional steps are required.

Click Events > Attach to open the Attach Event Handlers dialog box.

Navigate to the object that has the event handler you want to detach.

In the right panel, the Event Handlers column lists the event handler attached to an object. This option is available whether you view event handlers sorted by Structure or by Type.

When you click the event handler, a list is available. Select `<None>` from the list and click OK.

Before you delete an event handler, check whether any other component is using it. You can review components that may be still be using the event handler from the Type tab on the Attach Event Handler dialog box.

If you delete the .java file (Java) or the .cs file (.NET) when an event handler is still attached to a component, the event handler will be displayed in red in the Attach Event Handlers dialog box, and you will be warned when you try to save the model that it cannot be deployed in its current state.

The Java or .NET files associated with an event handler are located in `<your_model_folder>\<model name>\scripts\src`.

Delete the .java or .cs file for the event handler and then click Events > Rebuild to rebuild the model JAR or assembly (NET) file.

Event Handler Properties Files

Configure the default behavior for event handlers with settings available from the Event Handler Settings dialog box.

- Java Event Handlers

You can create other Java properties files for event handler settings. These settings are shared by all models in an installation and control aspects of runtime behavior. Those properties and their values are made available to event handlers via the callback interface method `getHandlerProperty(String name)` on the event interface passed as the event's parameter.

There are two separate properties files for event handlers: one contains properties for event handlers as they execute within the Design Tool, and the other contains properties for event handlers as they execute with the Host Integrator Session Server.

In the `<VHI_install_dir>/HostIntegrator/etc` folder, locate `dt_script.properties.sample` and `script.properties.sample`. Copy and rename these files as `dt_script.properties` and `script.properties` and modify them to your needs. There is additional documentation in the files.

- .NET Event Handlers

In the `HostIntegrator\lib\dotnet\clrscriptserver.exe.config` file you can modify properties used by the .NET script server. You cannot add arbitrary properties. You cannot use the .NET method `IEvent.GetHandlerProperty()` in this environment. Instead, consider reading properties from the environment or the registry.

DESIGN TOOL PROPERTIES

Create a `script.properties` file in the `<VHI_install_dir>\HostIntegrator\etc` folder. It can contain settings for the properties described below. This file has the form of a Java properties file. Entries have the form `name=value`, one per line. Backslashes (`\`) must be doubled (`\\`) to be interpreted literally. Open `<VHI_install_dir>/HostIntegrator/etc/script.properties.sample` for details.

Classpath: `vhi.script.classpath=<semicolon-separated list of JAR file names>`

Design Tool uses the `vhi.script.classpath` setting to locate Java classes that do not contain event handlers, but which must be available for use with the event handlers used by all models executing on the Server. This file is read at script manager startup. The script manager must be reset (by stopping and restarting it or by closing the Design Tool) before changes made to this file will take effect.

If this file is not present, or if the file does not contain a classpath definition, the classpath is assumed to be empty. No external Java libraries will be available to event handlers unless they are defined here.

Any other Java properties contained in this file are made available to event handlers via the [Event.getHandlerProperty\(\)](#) API.

SERVER PROPERTIES

The `script.properties` file defines properties used by the session server to control the script manager. It must be located in the `<VHI>\etc` folder, and is in the form of a Java properties file. Entries have the form `name=value`, one per line. Backslashes (`\`) must be doubled (`\\`) to be interpreted literally. Open `<VHI_install_dir>\HostIntegrator\etc\script.properties.sample` for more details.

Classpath: `vhi.script.classpath=<list of JAR file names>`

The value is a string. On Linux, it consists of a colon-separated list of JAR file names to be searched. On Windows, it is a semicolon-separated list of JAR file names.

You can define the Java classpath that the session server uses for locating Java classes that do not contain event handlers, but that must be available for use with event handlers. This file is read-only when the server starts. The script manager must be reset (using the Reload Event Handlers option on the Events menu in the Design Tool, or by restarting the session server) before changes made to this file take effect.

If this property is not present, the default classpath is assumed to be empty (no external libraries used).

- Script Output

Two properties control how event handler data written to the System output (System.out and System.err) is sent.

- `vhi.script.output.file= true | false`

Boolean property that enables or disables script output. The default is false (output disabled).

- `vhi.script.output.filename=pathname`

String that specifies a file in the `<VHI_install_dir>/HostIntegrator/etc/output` directory where System.out and System.err are written. The default filename is handlers.out.

No management is provided for any of the output generated by these properties. Any output recorded is simply appended to whatever content already exists in the output file.

- Trap Output

Two properties control how unhandled exceptions caught by the event handler environment will be written to a file.

- `vhi.script.output.traps= true | false`

Enables or disables trap output, recording information about unhandled exceptions. By default, Host Integrator writes stack dumps for unhandled exceptions caught from user classes.

- `vhi.script.output.trapfile= pathname`

String that specifies a file in the `vhi/etc/output` directory that holds the trap information. The default is traps.out.

No management is provided for any output generated by these properties. Any output recorded is simply appended to whatever content already exists in the output file

OTHER PROPERTIES

Any other Java properties contained in this file are made available to event handlers via the [getHandlerProperty](#) API.

4.11.3 Event Handler Troubleshooting

There are a variety of options available to test and debug event handlers.

See the [event handler API](#) documentation for more information.

Validating and Testing Event Objects

The Validator, available on the Debug menu, confirms that libraries associated with event handlers are up to date. It is good practice to confirm that the event handler is returning the results you expect.

- Click Connection Events Test on the Debug menu to confirm that the life cycle event handlers and model event handlers you have added to your model are performing as you expect.
- Click Execute on the Operation tab of the Entity window to test operation events.
- Test events that include recordset and recordset field events using the Recordset Test option on the Debug menu.
- Use the Procedure Test option, available from the Debug menu, to test the procedures for a table.
- Click Attributes Test on the Debug menu to test any event handler that uses Write Attributes.
- Use Web Builder, if debugging on the server is required. You must configure the server properties to enable debugging. Event Handler options are configured on the Server Properties panel.

Stream Output

You can view the output your Java or .NET event handler code sends to the System.out (stdout) and System.error (stderr) standard streams. The output also includes the trap exception stack dump for unhandled exceptions, which can be useful if your event handler code is calling other third-party packages or assemblies.

Design Tool

Output is displayed in Debug > Event Handler Console and is written to the console even when the console is hidden. You can open the console after unexpected behavior occurs. To display the console when the Design Tool is run, open Settings > Event Handlers > Debugging and select **Show when Design Tool starts**.

ABOUT THE EVENT HANDLER CONSOLE

To open the Event Handler Console, click the button on the Event Handler toolbar or click Event Handler Console on the Debug menu.

The Event Handler Console displays the output that event handlers have sent to the System.out and System.err streams. You can use the remote debugging features in your Java development environment to set breakpoints, step through code, and view runtime status.

Host Integrator writes to the console even when it is hidden, so you can open it for information after some unexpected behavior occurs. Any unhandled exception that is not an ApptrieveException or EventTimeoutException causes a stack dump to be written here.

- **Configuring and Using the Console**

You can open the console window yourself, or you can choose to open the console automatically when the Design Tool starts.

Each Design Tool instance (or Session Server instance) sends output to a distinct console window. When you are debugging your event handlers, use Alt+Tab to review console output.

- **Console Options**

- **Wrap Lines**— By default, output lines are wrapped at the right window edge. Select the **Wrap lines** check box to change this option.

- **Session Server Debug Settings** - In a production environment, the standard output or standard error stream information will, by default, be thrown away. You can use the [script.properties](#) file to designate a file where this information should be directed. Manual reduction of this file must be performed periodically.

Session Server

To provide maximum performance of the runtime session server, by default, the standard output and standard error stream information is discarded. However, you can redirect this information to a text file.

- For Java event handlers, edit the `<vhi>/etc/script.properties` file, setting the `vhi.script.output.file` property to true. See [property files](#) for more information.
- For .NET event handlers, edit `<VHI>\lib\dotnet\clrscriptserver.exe.config`, setting `RecordOutput` to true.

By default, the file, `handlers.out` is generated in the `Micro Focus/Verastream/HostIntegrator/etc/output` directory. However, you can set a different name with the `vhi.script.output.filename` property (Java) or `OutputFilename` setting (.NET).

No management is provided for any of the output generated by these properties. Any output recorded is simply appended to whatever content already exists in the output file.

See [Guidelines for Developing Event Handlers](#) for detailed information on event handlers in general and specifically on limiting the use of output streams to ensure performance efficiency.

Remote Debugging

Remote debugging enables you to use your IDE to set breakpoints in your event handler source code, step through code execution, and inspect variable values.

Load your event handler source code into the editor and set a breakpoint. Trigger the event and the breakpoint will be hit. Now you can step through the code and inspect the values of your variables.

JAVA

Verify that the debug port is enabled on the Java virtual machine that hosts the script manager. In the Design Tool, it is already enabled. For the Session Server, you must enable script manager debugging in the Administrative Console (server properties > General > Events) and restart the server.

Note which port number is used. In Design Tool, see the Debug port assigned on the Settings > Event Handlers > Debugging tab. For the Session Server, see the Assigned event handler debug port in server properties. If the requested port is in use (such as when running multiple instances of Design Tool), an increment is used.

Connect your Java IDE to this debug port

.NET

To debug a .NET script in the session server or Design Tool, attach the Microsoft Visual Studio debugger to the script server. The name of the .NET script server process is clrscriptserver.exe. In Visual Studio, you can attach to this process using Debug > Attach to process.

In a development environment, you typically have two instances of clrscriptserver.exe, one for the session server and one for the Design Tool. The script server for the session server runs as user SYSTEM. To select it, make sure you check the **Show processes from all users** check box. The script server for the Design Tool is running as the same user as the Design Tool.

Event Handler Timeouts

When timeouts are enabled (the default), if you leave execution suspended at a breakpoint for too long, you may trigger a timeout condition.

DISABLING EVENT HANDLER TIMEOUTS

You can disable event handler timeouts in both the Session Server and the Design Tool. When event handler timeouts are disabled, all timers used to calculate timeouts (for a single session) are suspended whenever that session has fired an event to the script manager. Once the script manager returns a response to the Design Tool or Session Server, the suspended timers will resume.

- In the Design Tool, to temporarily suspend all timeouts, click Events > **Disable Timeout**, or use the Disable Timeout button on the Event Handler toolbar.
- For the Session Server, you can disable event handler timeouts in Administrative Console (Server properties > General > Events).

Testing event handler timeouts

Use these steps to test event handler timeouts, when they have been left enabled in Design Tool.

Place a breakpoint in the event handler code activated by an event.

Set the event handler timeout and client timeouts to a value larger than the time expected for the event handler activity.

Cause the event to fire and stop execution at the breakpoint.

4. Click Cancel while the event is in progress to force a timeout exception to be sent to the event handler.
4. When an event takes more than a half second to complete, a progress window with a Cancel button displays. This is the same dialog box used for tracking the progress of procedures and operations. In some cases, this dialog box may already be open when the event handling condition occurs.
5. Begin single-stepping through the event handler code to determine how the event handler reacts to the timeout.

Note

You may encounter a problem attempting to reset the script manager after a breakpoint is encountered. After you click Events > Reload Handlers, you may see the following error: "[VHI 4302] Error connecting to the script manager. It is likely that the script manager's TCP Listening port (9654) is in use by a third-party application." To resolve this issue, resume execution at the breakpoint in the debugger before you reload handlers.

More information

[KnowledgeBase 7021562: Capturing Traces in Host Integrator](#)

Event Handler Communications Errors

While requesting a build of an event handler, the following message may be displayed if the script manager could not start:

```
An error occurred while communicating with the script manager.  
Socket Error - Socket not connected.
```

If you are debugging an event handler and the debugger stops at a breakpoint, you can resume the debugger and retry the communication. Otherwise, press Abort to restart the script manager.

If you press Abort and continue to see this error, it is usually due to a port conflict. The default port for the Script Manager is 9653, with a range of 9653 - 9669. Change the port number and try the event handler build again.

Connection Events Test Options

The Connection Events Test dialog box tests sequences of events that require a reset of the terminal session: loading a model, connecting to the host, and establishing a client connection. The startup and shutdown sequences below illustrate the differences between each of the test options.

DEDICATED SESSION: NEW CLIENT

This option simulates the sequence of one client application disconnecting and another client connecting to this model using the [Connect To Model method](#):

The Client Disconnected event is fired.

The terminal session navigates to the home entity, which could involve [Execute Operation](#) events.

The logout process is executed, which could involve firing the Execute Logout event.

The host session is disconnected from the host.

Steps 1-7 of standard reset processing are executed.

The host session is connected to the host.

The login process is executed, which could involve firing the Execute Login event.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The Client Connected event is fired.

Steps 1-4 can also be accessed by pressing the logout button, while steps 5-9 can be accessed by pressing the login button on the same toolbar.

POOLED SESSION: NEW SESSION

The New Session option simulates destroying and creating a pooled session, then connecting to the session using [ConnectToSession method](#).

With this option, a session pool host session is created, connected to the host, and logged in before a client session is actually created. This simulates the real-world behavior of a session pool session.

The Client Disconnected event is fired.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The logout process is executed, which could involve firing the Execute Logout event.

The host session is disconnected from the host.

The Client Session Destroyed event is fired.

The Host Session Destroyed event is fired.

The Host Session Created event fires.

The login process is executed, which could involve firing the Execute Login event.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The Client Connected event is fired.

The Authenticate User event is fired, with any resulting errors displayed in the Event Handler Console.

The Client Session Created event is fired.

POOLED SESSION: NEW CLIENT

This option simulates the sequence of one client application disconnecting and another client reconnecting to this model using the ConnectToSession method.

In this case, only the client session events are fired because a session pool host session is not normally logged out or destroyed between client invocations.

The Client Disconnected event is fired.

The terminal session is navigated home, which could involve Execute Operation events.

The Client Session Destroyed event is fired.

An Authenticate User event is fired.

The Client Session Created event is fired. The Client Connected event is fired.

Event Handler Update Needed


The Design Tool detects when the contents of the model-specific script libraries (in the `scripts\lib` directory) have changed.


If you have enabled the option to automatically reload event handler libraries when libraries change, on the Settings menu > Building tab, the Design Tool responds to a change by reloading the event handler JAR files (Java) or assembly files (.NET). There are cases when the reload does not occur:

The script manager is unable to respond to a Design Tool request (for example, if it is stopped at a breakpoint)

An operation or event is in progress

Automatic event handler reloading has been disabled.

The Design Tool displays a Update Needed indicator () in the status window. This symbol indicates that the event handler JAR/assembly files stored on disk no longer match what is being used by the script manager (in memory).

Click Reload Handlers on the Events menu. Clicking Rebuild  also updates the event handler libraries.

Additional Event Handler Troubleshooting Options

As you develop event handlers, use the debugging options in the Design Tool to identify problems. Here are additional conditions you may need to address when developing or maintaining models that have event handlers associated with them.

Debugging tools show results that do not match the objects definition

Event handler name is displayed in red

Reset or Update Needed icon is displayed on the status bar

Error message: An error occurred while communicating with the script manager

Callback method is not allowed within an event handler

Script manager does not start

Java startup synchronization problems when additional JAR files are in `scripts\lib` folder


NET Runtime errors result from crossing domain boundaries

Model is not recognizing source files that have been imported

Deployment error with saved model

Problems with Web Builder applications event handlers for recordsets and attributes

DEBUGGING TOOLS SHOW RESULTS THAT DO NOT MATCH THE MODEL OBJECTS DEFINITION

This is not a event handler error, and may in fact be the appropriate behavior if an object has an event handler attached to it. Check to see if a lightning bolt () is displayed next to the object in question. This event handler may include logic that overrides or extends the object's behavior as defined in the Design Tool. For example, the [recordset test used in the event handler example](#) returns values that include a dollar sign not defined within the recordset itself.


EVENT HANDLER NAME IS DISPLAYED IN RED

The name of an event handler is displayed in red if a model object has an event handler attached, but the event handler no longer exists (for example, the JAR or assembly file containing the handler has been removed, a class file is missing, or the source file was not available when handlers were rebuilt).


You cannot deploy a model with this problem, although the model itself can be saved. If the JAR file or the assembly file is the problem, simply rebuild the event handlers.

RESET OR UPDATE NEEDED

Under certain conditions, you may see one of the following symbols on the Design Tool status bar:

 [Update Needed] The event handler JAR or assembly files stored on disk no longer match what is being used by the script manager (in memory).

Click  (Reload Event Handlers) to update the JAR files.

 The script manager cannot respond. If this condition occurs on the Host Integrator Server, the Server initiates a shutdown process. No manual reset is needed.

ERROR MESSAGE

An error occurred while communicating with the script manager

If this event handler communication error is displayed while building event handlers, there may be a problem with the port being used by the script manager. Other problem sources are the version of the JRE or the JAR file being used by the script manager (Java), or an equivalent version discrepancy with .NET assembly files.

When a script manager communication error occurs on the Host Integrator Server, this information is sent to the log file and to the client application.

SCRIPT MANAGER DOES NOT START

The script manager is implemented as a separate process, and is required for normal server operation. The Design Tool and the Session Server each require a separate instance of the script manager, and the handling of script manager problems in each case differs.

Startup

- Session Server - If the script manager fails to start when the Session Server starts, the Session Server writes an error log message and exits.
- Design Tool - If the script manager fails to start with the Design Tool, a dialog box will indicate that the script manager failed to start. The Reset needed icon will also be displayed on the Design Tool status line.

Runtime

- Session Server - If the script manager fails at some time after startup, the Session Server will:

Log an error message describing the problem.

Terminate any sessions and models that need script manager services.

Refuse new client connections.

Wait up to 60 seconds for existing client connections to terminate.

Exit to the operating system. The exit status will indicate "normal termination" so that systems that can restart services will do so.

- Design Tool - If the script manager fails after having started successfully, the Design Tool will display a Reset Needed indicator on the status line. This condition may occur as part of a debugging process. Operations that require communication with the script server will

display a message indicating that the operation is being aborted because the script manager is not available and must be reset.

JAVA STARTUP SYNCHRONIZATION PROBLEMS

For a given model, all JAR files or in the `scripts\lib` directory will be introspected for available event handlers using the Java reflection API, an introspection utility for Java classes. JAR files that do not contain event handler classes can be placed on the user class path, not in the model's `scripts\lib` directory, to avoid the overhead of introspection. Introspecting more than 1 MB of JAR files in the model's event handler directory may cause startup synchronization problems.

NET RUNTIME ERRORS RESULT FROM CROSSING DOMAIN BOUNDARIES

When using .NET Event Handlers, objects you store and retrieve from VHI contexts must be made effectively serializable due to the .NET remoting that occurs over application domains.

The script server dynamically loads and unloads the event handler classes using an Application Domain. In the event handler API, you can store arbitrary objects in six contexts: host session, client session, model context, RecordSet, Record, RecordLocation. Each of these expose `Set...StateObject`, `Get...StateObject`, and `Is...StateObject` methods. The values that you pass to these `Set...StateObject` methods cross from the event handler AppDomain to the script server AppDomain. Later, if you use `Get...StateObject`, the value passes back to the event handler AppDomain.

If you store an object using these methods, you could see an exception similar to these:

```
[VHI 4389] An unhandled System.Runtime.Serialization.SerializationException exception occurred in the abbreviations.LifeCycleHandler.ModelLoaded() event handler: Type 'yourType' in Assembly 'example, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' is not marked as serializable.
```

```
[VHI 4389] An unhandled System.IO.FileNotFoundException exception occurred in the abbreviations.LifeCycleHandler.ModelLoaded() event handler: Could not load file or assembly 'example, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' or one of its dependencies. The system cannot find the file specified.
```

These exceptions occur because the Common Language Runtime uses remoting to pass values between AppDomains. All values must either be serializable or a reference (a subclass of `MarshalByRefObject`). Follow the guidelines below to avoid this problem:

Types that are built in to .NET, such as integers and strings, and collections of those types, such as a list of strings, can be stored without a problem, since they are inherently serializable.

Third-party types that are serializable can be used, but the definition of the class must be known on both sides. The assembly for the serializable class must be in both the model's `scripts\lib` and in the `%VHI_ROOT%\lib\dotnet\` folder. A class that is defined inside your `vhi_model.dll` can never be successfully serialized, because the assembly name is the same for all models.

Types that are a subclass of `MarshalByRefObject` can be used. To prevent the object from being handled prematurely for garbage collection, you will may need to override the `InitializeLifetimeService()`. It is possible to define a subclass of `MarshalByRefObject` inside your `vhi_model.dll`. This approach is demonstrated in the Abbreviations example.

You can avoid the problem by creating a static member or singleton object inside your event handler code to store your state.

When using mutable serializable objects, remember that changes made to the original object are not reflected in the remote copy. Refresh the copy after changing your original, or use a `MarshalByRefObject` if you need mutable semantics.

Exceptions are also objects, and throwing a third-party exception in an event handler might seem likely to produce a `SerializationException` here as well. However, in the case of exceptions, there is a catch clause in the event handler `AppDomain` that prevents the `SerializationException`. This catch clause translates the exception into a special `ApprievieException`, which is serializable and known on both sides.

MODEL IS NOT RECOGNIZING SOURCE FILES FOR IMPORTED EVENT HANDLERS

You can [import event handlers](#) from another model. Copying the source files requires a rebuild so that they can be added to the model JAR file.

DEPLOYMENT ERROR WITH SAVED MODEL

Resave a model if you make changes to one or more of the event handler files in the model package, even though you have not made changes to the model itself. Reload the model in the Design Tool, validate, and then save the model and redeploy or re-create the package for deployment. Event handler files are automatically included when you deploy a model.

PROBLEMS WITH WEB BUILDER APPLICATION'S EVENT HANDLERS FOR RECORDSETS AND ATTRIBUTES

If you use Web Builder to create web applications, there are [limitations on recordset and attribute event handlers](#).

4.11.4 Settings Menu: Event Handlers

Use the Technology tab in the Event Handler Settings dialog box to determine which language you want to use for event handlers. Each model can support event handlers written in either Java or .NET technology.

Java

.NET with C#

Default all models to the selected technology - All models created within the Design Tool will use the selected technology as the default.

Editing

Use the Editing tab in the Event Handler Settings dialog box to configure the source code editor you want to use for creating or modifying event handlers.

This setting is stored in the Windows registry as a per-user setting.





Note

Other properties for event handling can be stored in properties files.

Available Editors	Description
Source Code	Specify the command to launch the editor whenever you choose to edit source code for a script. Host Integrator searches for the presence of common Java editors and selects it as the source code editor if one is found. Otherwise, a text editor is selected as the default.
Java	With Java as the selected technology, if a Java editor is found on your machine it is displayed here. This option is the default.
C#	With .NET with C# as the selected technology, if a C# editor is found on your machine it is displayed here. This option is the default.
Text	This is the default if no Java editor is found on your machine. Notepad is the default.

Available Editors	Description
Other	You can specify another editor. Click Browse to locate it, or use the command syntax below. The %p is automatically appended to the string when you locate the editor by clicking Browse. This example uses an alternate editor: <code>"C:\Program Files\TextPad 4\TextPad.exe" "%p"</code>

 **Note**

If you're having trouble opening source code when you click the Edit button () , a quick workaround is to change your editor selection to a text editor such as Notepad. You may need to specify a custom string for the Other editor option to use the editor of your choice.

Command Syntax

The name of any external command should use Windows conventions:

- Either UNC syntax or drive:path syntax.

- Spaces in command or path names require that the entire command be in quotes.

The parameters used with commands support the following escape codes to further specify file and directory information for editing and compiling scripts:

- %f—source file name (no directory)
- %p—source file name as fully qualified path name
- %d—source file directory containing the file
- %s—root of the model source file directory
- %l—root of the model lib directory
- %%—literal "%" character

Building

Use the Building tab in the Event Handler Settings dialog box to determine what happens when you rebuild event handlers. Other properties for event handling can be stored in properties files.

- When event handler libraries change

Choose to automatically reload event handlers each time you make a change, or to reload only when a rebuild or reload command is executed.

This setting is stored in the model file and can be saved to a settings file (.dtool) with other configuration information.

- Event handler classpath

In Java, this is the non-event handler libraries classpath to be searched to satisfy external class references in the event handlers, both when compiling and at run time. The classpath is stored in a properties file; the path cannot be changed here.

Note

In .NET, event handler implementation does not include classpath support.

Debugging

Use the Debugging tab in the Event Handler Settings dialog box to configure your debugging options. These settings are stored in the Windows registry as per-user settings. For Java, Event Handler debugging allows a third-party Java IDE debugger to connect to the Host Integrator script server. Each Design Tool instance starts its own script server instance.

Java Debug Port

- Disable debug port

This option disables listening on the debug port. It is enabled by default.

- First port to try

Design Tool starts searching for an available remote debugging port at the port selected here. The default value is 5006. This value is saved as a per-user setting. If another Script Server instance is already running at the setting specified here, the next available port is used.

- Debug port assigned

This is the JVM remote debug port that is currently in use. This is a read-only setting. You may need to configure your Java IDE if the Debug port assigned is other than the default 5006 or if you will be doing remote debugging with the Java IDE running on a separate system instead of *localhost*. A value of zero indicates that the event handler has the debug port disabled.

Event handler console (Java and .NET)

Select **Show when Design Tool starts** to open the event handler console automatically when the Design Tool opens. You can open the console manually from the Debug menu or from the Event Handler toolbar. These debugging settings apply to script server instances started by Design Tool. The Host Integrator Session Server runs its own separate script server instance, but you must restart the server in order to change event handler debug settings.

Environment

Use the Environment tab in the Event Handler Settings dialog box to configure your event handler development environment.

Default event handler timeout

This is the event handler timeout used when a timeout is not specified for a particular event. The default is 10 seconds. If the event handler exceeds this timeout when processing an event, a timeout error is generated. This setting requires that event timeout be enabled. This setting is saved as part of the model.

Client application credentials

These options specify client application credentials sent to an event handler when a client connection event is simulated by the Design Tool. They are stored in the model file, but they cannot be saved as part of a settings file (.dtool).

The test parameters for User name and Password are used to supply the credentials for an Authenticate user event in a life cycle event handler. When using the Session Server, credentials are supplied by a client connection as the "userid" and "password" parameters to a Connect method.

4.11.5 Event Reference for Supported Object Types

Click on an event handler object type to see information about the events that are defined for that type.

- Java: Review the [API documentation for model events](#) for more information.
- .NET: Review the [ModelEventHandler documentation](#) for more information.

[Model](#)

[Life Cycle](#)

[Entity](#)

[Attribute](#)

[Operation](#)

[Recordset](#)

[Recordset Field](#)

[Procedure](#)

Model Event Handlers

The events available for creating model event handlers are listed below.

- [Client Connected/ Disconnected](#)
- [Error Reported](#)
- [Execute Login and Logout](#)
- [Format Error](#)
- [Move Cursor](#)
- [Process String](#)
- [Unrecognized Screen](#)

CLIENT CONNECTED

The Client Connected event is fired after a client session has been created and the terminal has been positioned on the model's home entity.

This event allows a new client session to have the terminal initialized in some customized manner. For example, based on an input parameter, the session could be automatically repositioned to another entity.

Note

This is the first event fired on behalf of a client session that has access to the terminal to fetch information or modify its state.

In the Design Tool, the Client Connected event is associated with a login. It will not fire if a disconnect has been performed without executing a logout.

Client Disconnected

The Client Disconnected event is fired when a client disconnects, but before Host Integrator attempts to reset the session to the home entity.

This event allows custom cleanup of the terminal session. For example, if you implement a pseudo-rollback of a failed procedure call, the event would use the state cache containing the audit trail of changes made by the client, visit the entities where the changes were made, and revert all attributes and fields to their prior values.

Note

This is the last event fired with access to the terminal and the client state cache. The subsequent Client Session Destroyed event does not have access to the terminal.

After the event completes, Host Integrator always attempts to reset the session to the model's home entity.

ERROR REPORTED

The Error Reported event fires whenever an initial error is reported in the Session Server and there is no event handler in progress. The parameters provide information about which error was reported.

This event handler can obtain information about the terminal state. In combination with the client state cache, the event handler can save information for auditing or debugging purposes. In the Design Tool, this event is unnecessary since the terminal window provides this feedback.

Note

If another event handler is in progress, this event does not fire, allowing an error to be returned for processing to that event handler.

If another error is reported subsequent to the reported error but prior to returning results to the client, this event does not fire. In other words, for a given error stack, at most only the initial error report triggers this event.

EXECUTE LOGIN AND LOGOUT

Note

By default, the Session Server invokes the login/logout sequence once per host session. Another event handler (Client Connected or Client Disconnected, for example) might call it again as part of recovering an otherwise dead session.

The Execute Login event is invoked when a host session requires a host login process.

- Include conditional logic in login process—Under some circumstances, hosts can alter their login sequence (such as after a forcible disconnect).
- Recover from login errors—In some cases, the first time Host Integrator becomes aware of an expired account or password is during login. One branch of conditional logic could obtain newer credentials for the userid and password model variables automatically, and then update the model variable list configuration.

In a dedicated session, the login sequence is performed after the client session has been created. In a session pool, Host Integrator calls the login event handler in the absence of a client session. Only the host session is created. This is an extension of the session pool behavior used in previous versions of the product.

The default callback executes the login command list defined in the model, if any.

Execute Logout

The Execute Logout event is invoked when a host session requires a host logout process.

This event allows you to handle conditional logic during a logout process. For example, you can detect a session that is stalled on a particular terminal screen and then enter special key sequences to "recover" the session and log out normally.

On the Session Server, Host Integrator always calls the logout sequence after a client session has been destroyed, but before the host session is destroyed.

The default callback executes the logout command list defined in the model, if any.

FORMAT ERROR

The Format Error event is called after all other processing for a client request is complete and Host Integrator is about to report an error. This event can alter the contents of an error report to better meet the requirements of the client application.

This event can alter the contents of the error stack reported. For example, this could be done by having a prior event handler stash a meaningful error in the client state hash table and then by using this event to remove the other errors and replace them with the more specific message.

Note

This event can add new user-defined messages and/or remove one or more messages from the existing stack. The event is not permitted to alter the contents of any given message. Such a case requires that the message be removed and then replaced with a new message.

See the [ScriptHostSession API](#) topic regarding restrictions when using this event.

If the event removes all messages, Host Integrator restores the reported error to the original error stack that would have been returned to the client.

The alterations of the error stack are not recorded in the Host Integrator log by default. If the event handler requires that the new messages be in the log, you must invoke the event handler logging interface and directly place the message in the log.

Upon returning from this event, the connector used by the client to connect to Host Integrator will often append an additional error message onto the stack. This error message provides information about which API method was invoked.

MOVE CURSOR

The Move Cursor event is invoked if the cursor needs to be moved to a defined location and there is no move cursor override on the current entity.

See the [Move Cursor](#) event for entities.

Move Cursor Forward

This event is invoked if the cursor needs to be moved forward and there is no tab override on the current entity.

This event can be used for special case logic in cursor movement. For example, in some cases a character mode host moves the cursor differently depending on the data transmitted. In these cases, the model-based method of a static set of tab stops may not be adequate to capture the logic of cursor movements.

This event is applicable only to VT and HP models. It never fires for 3270 or 5250 models. This event is invoked by the model when there is a simple request to tab (usually via the internal tab/update algorithm used to write data to the terminal) or when there is a specific cursor location requested, but there is no move cursor override defined.

If the current terminal screen is an entity and there are tab stops defined, the list is available from the callback interface.

If the event returns with the cursor unchanged, Host Integrator assumes that the cursor cannot be moved any more in that direction. If the reverse direction is defined and has not been tried, Host Integrator tries to reverse direction. Otherwise, Host Integrator expects the event handler to move the cursor to the next position on the screen.

This event can be invoked iteratively on a single screen instance as a way to write data or move the cursor to a defined location.

Move Cursor Backward

The Move Cursor Backward event is invoked if the cursor needs to be moved backward and there is no tab override on the current entity.

Uses for this event are parallel to those described for Move Cursor Forward.

PROCESS STRING

The Process String event is fired when a client invokes the `processString` API method.

- Process a custom XML document — Passing an XML document to an event handler enables complete encapsulation of processing an XML document inside the model. If the processing accesses entities directly (instead of through procedures), the local event handler callback interface is more efficient than a client application making the same calls.
- Process custom SQL statements — Host Integrator supports a subset of ANSI SQL-92. Keywords such as JOIN are recognized but not supported. You can extend this support by taking an SQL string, removing the portions not supported by Host Integrator, passing the remaining SQL to the model for processing, and then modifying the results based on the custom extensions from the original client string.

Tip

You can test a process string event handler using [Web Builder](#). On the Procedures and Model features panel of the project, select Process String Event Handler under Model Features. The resulting project will have an option to allow input for the event handler and return a result.

There is no restriction on the semantic meaning of the string passed to Host Integrator. The custom event handler is solely responsible for interpreting the input and deriving results.

If this event is not implemented, the `processString` API returns an error. There is no default processing in Host Integrator.

UNRECOGNIZED SCREEN

The Unrecognized Screen event is fired at specific times when an entity is expected but none is found.

Recovery – The event handler may restore the terminal to a known location by sending input directly to the terminal.

Capture information for debugging – The event handler has the ability to read data from the terminal. For example, capturing a screen dump to a file may assist in diagnosing unexpected behavior by the model.

This event fires only under the following circumstances:

An operation times out at an unknown location while Host Integrator is executing an operation that does not have an event handler attached, or is waiting for an entity arrival after an event handler has returned control.

The login sequence times out at an unknown location.

A remote command is issued (from a client or an event handler) that requires a current entity, but none is found. Example requests include setting the current entity, executing a procedure or operation, reading or writing data to or from attributes or recordset fields, or wait for cursor at attribute.

A host session is trying to reset to the home entity and/or log out of the host after client disconnect and cannot begin navigation because the current location is undefined.

The user selects a destination in the Navigator when the current location is undefined.

Timeouts and Unrecognized Screen Events

If an operation times out at an unknown location while its event handler is in progress as described in case 1 above, the existing event handler receives the error notification and should take corrective action. The Unrecognized Screen event is not invoked: by definition, the event is still executing.

If a login event handler times out as described in case 2, this is a simple error. The Unrecognized Screen event is not invoked. Distinct from the login event handler, the login sequence consists of connecting to the host, executing the login script/event, and recognizing the current location. This sequence is governed by the host connection timeout in model properties. If, however, this timer expires and there is no intervening error (such as an event handler timeout), then the Unrecognized Screen event is invoked.

Once a handler timeout has been reported, the abort interval is already ticking (see [timeout processing](#)). Firing additional events during this interval is not permitted. The original event has already failed and Host Integrator is in the process of getting the original event to halt.

It is highly likely, that the next action from the client will result in the situation described in case 3 (a new request that requires an entity) or case 4 (client disconnect). Even if the operation or login event does not recover, the model will still attempt a recovery at some point in the future. If knowledge of the model state at the point of failure is important to recovery, this information should be stored by the operation or login event prior to returning.

In all cases, failure to recover to a known location results in an error. Even if the session is restored to a known location, some requests still fail if the location is not the one expected.

Life Cycle Event Handlers

Note

None of the events on this interface have direct access to a terminal session. If initialization or cleanup with terminal access is needed, use model events. See [ScriptHostSession API](#) for restrictions when using these events.

- [Model Loaded/Unloaded](#)
- [Host Session Created/Destroyed](#)
- [Authenticate User](#)
- [Client Session Created/Destroyed](#)

MODEL LOADED

The Model Loaded event is fired whenever Host Integrator loads a new model, or an updated version of an existing model. There is one exception: when the Design Tool loads a model already loaded by another instance of the Design Tool, no events are fired for this model, including the Model Loaded event.

Create shared resources – If interaction with an external system is required, a pool of shared connections can be created and stored in the state hash table for the model.

Customize model environment - This could involve anything from altering Host Integrator's model variable list configuration to establishing a local output file for auditing or debugging purposes.

Note

With the Host Integrator Session Server, the Model Loaded event often fires for a new version of a model before the Model Unloaded event fires for the prior version. This occurs because new clients will be given access to the new version before all existing clients are finished with the prior version, and the prior version is not unloaded until all host sessions using the model have been destroyed.

Any exception returned from this event causes the model to fail to load. In the Design Tool, these errors are reported and the model remains loaded. However, any model that fails to load on the Session Server is unavailable for client requests. If this is not a desired outcome, then the event handler should catch all "throwable" exceptions, log its own message, and then discard the exception.

Model Unloaded

The Model Unloaded event is fired when a model is unloaded by Host Integrator. This occurs for the existing model when the user exits the Design Tool or when the user loads a model. On the Session Server, this occurs when a new version of a model is activated or the server process shuts down.

This event can be used to clean up any shared resources created during the Model Loaded event.

Note

On the server, this event may be fired after a new version of the model has already been loaded. Under normal circumstances, this should not present a significant issue because each model version has its own state hash table instance. However, statically loaded resources could be shared between model versions and should be managed accordingly.

HOST SESSION CREATED

The Host Session Created event is fired each time a host session is created. In the Design Tool, this happens each time a model is loaded and when the user invokes certain tests in the Connection Events Test dialog box. On the Session Server, this occurs when a session pool is created or augmented and when a dedicated session is created in response to a client request.

This event can be used to initialize state or other resources that will live for the lifetime of the host session.

Note

For each server instance, each host session is guaranteed to have a unique identifier (obtained from the model event callback interface). This can be the key, or part of the key, to store state information in the model state hash table. The host session may span multiple clients and thus does not have access to the yet-to-be-created client hash table.

If this event returns any exception, the Host Integrator Session Server fails to create the host session and it is not made available for client access. In the Design Tool, errors are reported, but the host session creation succeeds.

Host Session Destroyed

The Host Session Destroyed event is fired each time a host session is destroyed. In the Design Tool, this happens each time a model is unloaded and when the user invokes certain tests in the [Connection Events Test](#) dialog box. On the Session Server, this occurs when a session pool is destroyed or decreased in size and after a dedicated session is released by its client.

This event can be used to clean up any resources that were intended to live for the duration of the host session.

Note

This event is guaranteed to be fired after the last client attached to this session has been cleaned up and the session has been logged out or forcibly disconnected.

AUTHENTICATE USER

The Authenticate User event is fired when a client connects to a server host session to permit authentication or authorization beyond that provided by Host Integrator security. In the Design Tool, this event is fired when a model is loaded and when the user invokes certain tests in the Connection Events Test dialog box.

The normal Host Integrator security infrastructure is administered independently of this event.

Extend authentication – Set up the extended authorization to access a proprietary security system, use connection parameters beyond user id and password, or both.

Add authorization – For cases when user authentication to the larger Host Integrator system does not necessarily equate to authorization to access the model in question.

Prepare host session – This event can modify the values of the model variables to match the connection parameters or otherwise meet application requirements.

CLIENT SESSION CREATED

The Client Session Created event is fired after a client has been granted access to a host session, but before the client is allowed to make any requests.

Initialize state cache – This is the first event for a client session that has access to the client hash map. Populating the cache with needed data is the primary use of this event.

Define authorization tokens – After initializing the state cache, if you are implementing field-level security or other fine-grained authorization strategies, you can use this event to create the requisite tokens and store them in the cache.

As with other life cycle creation events, any exception returned from this event results in the Host Integrator Server failing to create the client session. The prospective client receives the error and is denied access to a host session. In the Design Tool, unhandled exceptions are reported but the client session is created anyway.

Client Session Destroyed

The Client Session Destroyed event is fired when the client has disconnected and all other client and terminal cleanup processing has been completed.

This event can be used to clean up any resources that were intended to live for the duration of the client session. The client cache will be made available for garbage collection after this event, so only orderly releases of external resources need be done with this event.

This event is guaranteed to be fired after all other client session cleanup has been completed.

Entity Events

The events available for creating event handlers associated with entities are described below.

ENTITY ARRIVAL

The Entity Arrival event is used to pre-process a terminal screen or define state information.

Validate terminal state – The event handler can verify that the expected data is on the terminal screen, and can confirm that the host has not placed any error messages or locked the keyboard.

Preprocess the terminal screen – Fields hidden from the client application may require static data entry or conditional logic to cause the screen to initialize properly.

Automatic recovery – In the case that the entity represents an error condition, the event handler may automatically alter the terminal state to bypass the screen.

This event fires whenever a transition is made from an unknown location or another defined location to the attached entity. The event fires after all signature patterns are recognized and the entity validation has been passed.

Entity Arrival events and Entity Departure events for a given entity are disallowed while the Entity Arrival event for that entity is in progress. If the current entity changes during an Entity Arrival event, the Entity Departure event will not fire.

ENTITY DEPARTURE

The Entity Departure event is used to delay actual departure from an entity or modify model state information to reflect the departure.

Define an unknown screen – The screen can be defined as an extension of the last known entity for data processing purposes. For example, a recordset scroll may lead to a screen with no static content for use in identification.

This event fires any time an entity changes from being recognized to being unrecognized. The Entity Departure event fires just before the rest of Host Integrator is informed of the entity departure. If the next location is unknown, the event handler can engage an entity override that forces the entity to remain current even though its signature is not found.

Once the override is engaged, the event fires again each time there is a terminal screen update if the entity signature is still not recognized. Depending on how the host updates the terminal screen, this may produce many iterations of this event. As soon as a different entity is found, the departure event is fired without the entity override. This state can be discovered by asking for the next entity: if the result is non-null, then an invocation of the entity override is ignored.

See the [ScriptHostSession API](#) topic regarding restrictions when using this event.

MOVE CURSOR EVENT

The Move Cursor event is invoked only when there is a predefined location requested. For example:

The MoveCursor or ShiftCursor command is issued by a command list.

The MoveCursor or ShiftCursor API method is invoked by a client or event handler.

Model commands that write data to attributes, such as TransmitToAttr, DefaultValue, WriteVarToAttr, and UpdateAttribute invoke this event if the cursor is not in the first terminal cell of its location. The same is true for corresponding commands that act on recordset fields.

The exception to the commands listed above is UpdateAttributes and UpdateRecordsetFields. These two commands have different behavior depending on the structure of the model:

If tab forward and/or backward definitions exist on the entity or the model, an internal algorithm that iteratively checks the current location for applicable input data, writes any data that is found, and then tabs to the next tab stop used to write the cached inputs to the terminal screen in order.

If there is no tab definition on the entity or the model, the iterative internal algorithm is bypassed (see information on the [cursor tab](#) for more details) in favor of issuing a TransmitToAttr command for each attribute with the input data as the argument. In this case, each attribute or field object that does not find the cursor in the correct location issues a MoveCursor command to move the cursor to the first terminal cell of its location.

Procedures in combination with client and event handler API invocations to write data to attributes or fields follow the same rules as UpdateAttributes and UpdateRecordsetFields commands.

The Move Cursor event would typically be defined when clients or operations are expected to request moving of the cursor to one or more absolute positions on the terminal screen represented by this entity.

This event can be invoked only by models built for VT or HP terminals. With 3270 and 5250 terminals, Host Integrator simply moves the cursor to the desired position in the local terminal, while VT and HP terminals often have the cursor position dictated by the host.

If this event fails to move the cursor to the expected location, Host Integrator sets a runtime error on return.

The default callback does one of the following:

- Invokes the internal algorithm built on model tab definitions to move to the desired tabstop
- Uses the arrow keys to try to position the cursor

For an entity, move the cursor using tab overrides by mapping operations for forward tab, backward tab, or both. This remains unchanged by the presence of the event handler. The mapped operations for tab overrides can themselves have event handlers attached. Host Integrator detects when a tab operation has an event handler and applies the same rules as for model object tab overrides.

Write Attributes

The Write Attributes event is invoked whenever a client API call, an event handler API callback, or a procedure writes attributes to the terminal. This event is also invoked by an operation issuing an UpdateAttributes command.

Possible uses:

Customize the order of update and/or terminal preparation—When one or more attributes need to be written to an entity in a character mode host, the cursor often must be moved in a particular manner. Additionally, terminal updates may be required in a specific order.

Validate inputs—In some cases, a set of attribute inputs requires validation as a whole instead of the more usual one-by-one basis. Alternatively, you may want to validate each entry before any data is actually written.

Typically, this event is used to handle the portions of terminal update that involve the entire set, such as validation, cursor movement, and ordering. Although the actual writing of the data can be done directly to the terminal, it may more often be done by calling upon each attribute in turn to perform the update.

As in the case of the Write Attribute event, Host Integrator accepts what is dictated by the event handler, even omitting some or all of the data if desired. Changing the current entity during this event is not recommended, as there may be additional actions that depend on the current entity remaining the same. Otherwise, there are no predefined conditions that result in an error.

The default callback behaves in one of two ways:

If a tab definition exists for the current context, then the internal tab/update algorithm is used to input the data.

Otherwise, each attribute object is invoked in an order determined by Host Integrator; each attribute ensures the cursor is in the first cell of its terminal location and then transmits the applicable data to the terminal.

Attribute Events

The two events available for creating event handlers associated with attributes, Read Attribute and Write Attribute, are described below.

READ ATTRIBUTE

This event is used to post-process data read from the terminal.

Reformat data – Examples include changing the data from internal codes to human-readable form, changing the format of a date from one locale to another, adding or removing characters to modify the meaning (such as a negative sign for a number less than zero).

Create composite data – Represent data from disparate regions of the screen as a single unit.

Enforce field-level security – Authorization for access to the field data can be based on the user credentials.

This event is invoked whenever a request is received to read attribute data. This request might be from a client API, an event handler callback, an operation, or a procedure.

The default result is "scraped" from the terminal, passed to the model-defined post-processing for white-space and symbol removal, and then sent to the event handler.

The value returned from the event handler is treated as the value of the attribute for the current request. Changing the current entity during this event is not recommended, as there may be additional actions pending that depend on the current entity remaining the same. Otherwise, there are no predefined conditions that result in an error.

WRITE ATTRIBUTE

This event is used to pre-process the data to be written to the terminal or to customize the process of writing the data.

Reformat data – Examples include changing the data from human-readable form to internal codes, changing the format of a date from the external locale to the host format, adding or removing characters or sending special terminal keys to correctly indicate meaning (removing a negative sign for a number less than zero, sending the raw data to the terminal, and pressing a special terminal key to communicate the negative value).

Split composite data – In contrast to the case of a Read Attribute event, a client may send data in a composite form, and the write processing requires special logic to split the data into constituent parts and transmit each to the terminal separately. For example, some calendar dates may need to be processed this way.

Enforce field-level security – Authorization to update the field data can be based on the user credentials.

Handle dynamic host applications – Some applications change the types of fields based on data inputs or user authorization. An event handler using Write Attribute can check for protected status and either omit the actual transmission of data or throw an informative exception.

Create audit trails of changes – All changes to an attribute can be logged directly to an external system. Alternatively, the changes can be cached for later recording or for use in a pseudo-rollback of a failed procedure. See the [Model event Client Disconnected](#).

This event is invoked whenever a request is received to write attribute data. This request might be from a client API call, event handler callback, an operation, or a procedure.

The input data is passed directly to the event handler with no pre-processing.

Examples of actions the event handler might take include:

- Modify the value and call the default

- Customize setting the terminal state with custom key sequences

- Make decisions about whether or not to attempt writing data

Host Integrator accepts whatever actions specified by the event handler, including omitting any terminal updates. Changing the current entity during this event is not recommended, as there may be additional actions pending that depend on the current entity remaining the same. Otherwise, there are no predefined conditions that result in an error.

The default callback performs the model-defined pre-processing for pre-pending, appending, and white-space padding, and then executes any pre-write operation, verifies the length of the data and state of the terminal, positions the cursor as needed, transmits the data to the terminal, and executes any post-write operation.

Operation Events

The event available for creating event handlers associated with operations is described below.

EXECUTE OPERATION

The Execute Operation event is fired whenever an operation execution is requested.

- **Conditional logic**—Host Integrator can internally handle alternative outcomes of operations. In order to perform conditional navigation, entities must be defined at each conditional branch point. As an alternative, an event handler could perform midstream adjustments in the list of commands to issue. This could decrease the number of entities required to navigate a host application.
- **Terminal validation** — In some cases, data can be entered from different sources that may be difficult to validate as a whole. As the first step in an operation, an event handler can validate the state of the terminal to detect potential errors before the underlying data causes corruption in the host data source.

Automated error recovery — For known error situations, an event handler can resolve the error cases in order to reach the desired destination. For example, some data combinations could cause a keyboard lock that can be resolved by simply pressing a terminal key.

Custom error reporting — Given a set of possible error conditions, an event handler can generate a specific error message that may be more helpful to a client application than automatically generated error messages.

Custom synchronization — Host Integrator provides a variety of means for deciding when a host has finished updating a terminal screen. However, there are occasions when the synchronization really involves branching logic based on inputs. The event handler is the only mechanism that supports this form of terminal-model validation.

The Execute Operation event is required to leave the terminal on a screen recognized as a valid outcome of the operation. Upon return, the operation iteratively checks for arrival at a valid destination a valid location is found, an invalid location is found, or the operation times out.

The default callback for an operation executes the command list, destination processing, and post-condition logic as defined by the model.

The Java [WaitForArrival](#) callback or the .NET [WaitForArrival](#) method (available in a Windows help file) processes primary, alternate, and error destinations, waiting for one of them to be found. This continues until the event times out, the callback-specified wait interval expires, or a destination is recognized.

Host Integrator does not process terminal input while the event handler is executing logic. The host data is processed into the terminal only during callbacks to Host Integrator. At the points where the operation needs to wait for the host to "catch up," inserting wait events or other callbacks is necessary.

Recordset Events

The events available for creating event handlers associated with recordsets are described below.

[Parse Screen](#)

[Parse Record](#)

Is Terminated

Get Record Type

Apply Filter

Get Current Host Record

Update Record

Insert Record

PARSE SCREEN

The Parse Screen event is fired during the reading of the current screen of the attached recordset. It determines the location and size of records within the recordset.

- Variable length records – A host application may vary the length of the records depending on content. This event allows for arbitrary record sizes as long as they are within the bounds of the recordset and rectangular in shape.
- Variable spaced records – A host application may group records together so that the records are not evenly spaced. This event allows for arbitrary record locations as long as they are within the bounds of the recordset.
- Partial records – The host may split records across columns or across screens. This event allows for these types of records to be identified and combined.

This event is fired on a screen by screen basis when the host session reads or rereads the contents of the current screen. Record locations must be rectangular and relative to the recordset.

Along with a location, each record can be designated as whole or partial. Upon returning from the event, Host Integrator automatically combines adjacent partial records into whole records. If the last record on the screen is marked as partial, Host Integrator retains it until the next screen is parsed.

The order in which the record locations are returned from this event dictates the order in which indices are assigned to the records. This is also the order that a client application sees the records when fetched.

PARSE RECORD

The Parse Record event is fired during the reading of the current screen of the attached recordset. It determines the location and size of fields within a record.

Variable field location and length – The location and length of fields may differ between records. This event allows for field locations and lengths to be specified on a record-by-record basis. The fields, however, must still be defined in linear fashion relative to the record.

This event is fired for each whole record resulting from the Parse Screen event. It is not fired for partial records until they are recombined. Field locations must be linear and relative to the record.

Changing the current entity during this event is not supported and causes the event to fail. Also, attempting any action on the recordset from which the event fired results in an error.

Callbacks that change terminal state are restricted when using this event.

IS TERMINATED

The Is Terminated event is fired during the reading of the current screen of the attached recordset. It determines if the current screen is the first or last screen of the recordset.

Scroll termination – The host may indicate the first or last page of the recordset in a way that cannot be handled by the built-in scroll termination criteria.

This event is fired once for scroll-up termination and once for scroll-down termination on a per screen basis. The event is fired after the records have been read from the screen. If this event returns True, the recordset is considered terminated in the direction queried.

Changing the current entity during this event is not supported and causes the event to fail. Also, attempting any action on the recordset from which the event fired results in an error.

Callbacks that change terminal state are restricted when using this event.

Host Integrator will not fire `isTerminated()` if the fetch terminates with either of the following:

Scroll operation results in the same recordset data.

Scroll operation results in entity change.

In both cases, the corresponding option must be selected in the RecordSet Termination Options dialog box and the action must occur during the fetch.

GET RECORD TYPE

The Get Record Type event is fired during the reading of the current screen of the attached recordset. The results of this event affect which records are visible to client applications and which records are eligible for insert. Records are categorized as normal, blank, or excluded and blank. This event fires after a record and all of its fields have been read from the screen.

Excluded records – Records marked as excluded are hidden from client applications.

Typically, this is used for records that are repeated from a previous screen or do not contain data. Filtered records are assigned an index, but client applications cannot access them.

Designate insertion locations – Records marked as blank can be used as insertion locations.

Blank records are visible to client applications unless they are also marked as excluded.

This event is fired after the records have been read from the screen. Host Integrator provides a default categorization based on the model definition.

Changing the current entity during this event is not supported and causes the event to fail. Also, attempting any action on the recordset from which the event fired results in an error.

Callbacks that change terminal state are restricted when using this event.

APPLY FILTER

The Apply Filter event is fired to apply a custom filter expression to a record.

Custom filter expressions – Many of the recordset API methods accept a filter expression. When this event is attached to a recordset, Host Integrator uses it to evaluate the filter expressions instead of using the built-in evaluation mechanism.

When this event is attached to a recordset, it replaces the built-in evaluation mechanism. Host Integrator does not attempt to parse or evaluate filter expressions from the client. Instead, any client filter expressions are handed to this event for evaluation against a record. This allows the event to define its own filter expression syntax for clients to use. This event only fires when the client uses non-empty filter expressions.

Callbacks that change terminal state are restricted when using this event.

GET CURRENT HOST RECORD

The Get Current Host Record event is fired when the host session needs to know the host's current record.

Customize current host record identification – A host application may identify the current record with an approach not supported by built-in Host Integrator mechanisms.

The host's current record is based on the host application current record status, while the recordset's current record is based on the client application's current record status. For performance reasons, these are not kept in sync unless the client application makes a request to update, insert or select a record. When one of these requests is made, the Get Current Host Record event is called to determine the current host record. If it does not match the recordset's current record, Host Integrator uses line up and line down operations to change the host's current record to match. The client request fails if they cannot be successfully synchronized.

The client application can also explicitly ask to set the recordset's current record to the host's current record. In this circumstance, the event fires to obtain the host's current record.

When this event is not attached to a recordset, the built-in evaluation mechanism is used.

Changing the current entity during this event is not supported and causes the event to fail. Also, attempting any action on the recordset from which the event fired results in an error.

Callbacks that change terminal state are restricted when using this event.

UPDATE RECORD

The Update Record event is fired whenever a client application or a procedure requests to update a record.

Specify the order in which fields are written — A host application may require the fields to be written in particular order.

Customize the cursor movement between fields — A host application may require customized cursor movement in order to move between fields in a synchronized fashion.

Customize before and after update actions — A host application may require special actions before or after updating a record.

Validate field values — A set of fields may require validation as a whole, instead of individually validating at the field write event level. Alternatively, there may be a need to validate each field value before any data is actually written to the terminal screen.

The Update Record event encompasses executing the before update operation, writing the fields, and executing the after update operation. Before the event fires, the record to be updated is made the recordset's current record and the host's current record.

Upon invocation, this event is responsible for orchestrating the record update. This may include performing before and after update actions, writing the fields, and moving the cursor between fields. The actual writing of the data can be done directly to the terminal, but each field should be called in turn to perform its write. Writing to the terminal directly would bypass the Write Field event for the fields.

The default callback executes the before update operation, writes the fields, and executes the after update operation. The fields are written in one of two ways:

- If a tab definition exists for the current context, then the internal tab/update algorithm is used to input the data.
- Otherwise, each field object is invoked in an order determined by Host Integrator; each field ensures the cursor is in the first cell of its terminal location and then transmits the applicable data to the terminal.

Host Integrator accepts whatever decisions are made by the event handler. This includes omitting some or all of the data if desired. Changing the current entity during this event is not recommended, as there may be additional actions pending that require that the current entity remain the same. Otherwise, there are no predefined conditions that result in an error.

INSERT RECORD

The Insert Record event is fired whenever a client application or procedure issues an insert record request.

Identify the correct insertion location — A host application may require special logic for identifying the correct insertion location.

Specify the order in which fields are written — A host application may require the fields to be written in a particular order.

Customize cursor movement between fields — A host application may require customized cursor movement in order to move between fields in a synchronized fashion.

Customize before and after insert actions — A host application may require special actions before or after inserting a record.

Validate the field values — A set of fields may require validation as a whole instead of individually at the field write event level. Alternatively, field value may need to be validated before any data is actually written to the terminal screen.

The Insert Record event encompasses searching for an insertion location, executing the before insert operation, writing the fields, and executing the after insert operation. When the event is fired, the event must first find the correct insertion location and make it the current record. This can either be done via a callback to find the default location, or done by simply making the correct location the current record. Either way, the insertion location must be the current record before any fields are written.

Once the insertion location is made the current record, the event is responsible for writing the fields to the record. This may include performing before and after insert actions, writing the fields, and moving the cursor between fields. The actual writing of the data can be done directly to the terminal, but each field should be called in turn to perform its write. Writing to the terminal directly would bypass the Write Field event for the fields.

The default callback locates the insertion location, executes the before insert operation, writes the fields, and executes the after insert operation. The fields are written in one of two ways:

- If a tab definition exists for the current context, then the internal tab/update algorithm is used to input the data.
- Otherwise, each field object is invoked in an order determined by Host Integrator; each field ensures the cursor is in the first cell of its terminal location and then transmits the applicable data to the terminal.

As in the Write Field event case, Host Integrator accepts whatever decisions are made by the event handler. This includes omitting some or all of the data if desired. Changing the current entity during this event is not recommended, as there may be additional actions pending that require that the current entity remain the same. Otherwise, there are no predefined conditions that result in an error.

Recordset Field Events

The events available for creating event handlers associated with recordset fields are described below.

READ FIELD

The Read Field event is fired during the course of reading the current screen of the recordset. It reads the contents of a field.

Reformat data – Examples include changing the data from internal codes to human-readable form, changing the format of a date from one locale to another, adding or removing characters to modify the meaning (such as a negative sign for a number less than zero).

Create composite data – Represent data from disparate regions of the screen as a single unit.

Enforce field-level security – Authorize access to the field data based on the user credentials.

This event is invoked for each instance of a field in a recordset. It is fired after the Parse Screen and Parse Record events.

The default result is "scraped" from the terminal, passed to the model-defined post-processing for white-space and symbol removal, and then sent to the event handler.

The value returned from the event handler is treated as the value of the field. Changing the current entity during this event is not supported and causes the event to fail. Also, attempting any action on the recordset from which the event fired results in an error. Otherwise, there are no pre-defined conditions that result in an error.

See the [ScriptHostSession API](#) topic regarding restrictions on changing terminal state using this event.

WRITE FIELD

The Write Field event is fired whenever a request is received to write data to a field.

Reformat data – Examples include changing the data from human-readable form to internal codes, changing the format of a date from the external locale to the host format, adding or removing characters or sending special terminal keys to correctly indicate meaning (removing a negative sign for a number less than zero, sending the raw data to the terminal, or pressing a special terminal key to communicate the negative value).

Split composite data – In contrast to the Read Field event, a client may send data in composite form, and the write processing requires special logic to split the data into constituent parts and transmit each to the terminal separately. Some calendar dates need to be processed this way.

Enforce field-level security**—Authorization to update field data can be based on the user credentials.

Handle dynamic host applications – Some applications change the types of fields based on data inputs or user authorization. An event handler using Write Field can check for protected status and either omit the actual transmission of data or throw an informative exception.

Create audit trails of changes – All changes to an attribute can be logged directly to an external system. Alternatively, the changes can be cached for later recording or for use in pseudo-transactional semantics (see [Client Disconnected] (need link) model event).

This event is fired whenever data needs to be written to a field. Typically, this occurs inside an update or insert record request, but can also occur as result of a command such as `TransmitToRecordsetField`.

The input data is passed to the event handler with no pre-processing.

Examples of event handler actions include:

- Modify the value and call the default

- Customize setting the terminal state with custom key sequences

- Decide whether or not to attempt writing data

Host Integrator accepts whatever actions are taken by the event handler as intentional, including omitting any terminal updates. Changing the current entity during this event is not recommended, as there may be additional actions pending that depend on the current entity remaining the same. Otherwise, there are no predefined conditions that result in an error.

The default callback performs the model-defined pre-processing for pre-pending, appending, and white-space padding. It then executes any pre-write operation, verifies the length of the data and state of the terminal, positions the cursor as needed, transmits the data to the terminal, and executes any post-write operation.

Procedure Events

The event available for creating procedure event handlers is described below.

EXECUTE PROCEDURE

The Execute Procedure event fires whenever a procedure execution is requested. This request can be from processing SQL input, a direct invocation of a procedure by a client or script, or a portion of a compound procedure.

- Custom procedure – You can implement a procedure using other procedures, low-level navigation, recordset manipulation, or external data sources.
- Validate inputs or filters – Host Integrator has limited support for validation, but an event handler can extend this capability.
- Transform inputs or filters – The internal host codes may differ from the way data is passed in from a client or script.
- Alter the results – Edit the result set by adding or removing records, transforming from internal to external representations of data, or storing the results in an external system (database) in lieu of returning them to the client.

The default callback executes the entire model-defined procedure with provided filters or inputs, returning any results to the event handler for post-processing. For select, update and delete procedures there can be only one set of input parameters. For insert procedures, there may be one or more sets of input parameters.

Compound procedures cannot have events attached to them.

4.12 Debugging Models

4.12.1 Debugging Your Models

The Design Tool provides several ways to test and debug host application models. Use the options on the Debug menu to confirm that your model is operating as you expect and is ready to deploy. The log monitor is a troubleshooting utility that you can run outside of the Design Tool.

Before You Deploy Your Model

Here are some additional ideas for validating that your model will work as you expect and will be easy to maintain.

Use offline mode to move to each entity you have created and use the operations tab to review each operation.

Confirm that your naming conventions are consistent with the operation performed. Are your constant values correctly spelled and specified? Review the [Modeling Tips](#).

Remove checkOperationConditions commands unless there is an actual condition to check. Removing any extra checks of this sort will improve model performance.

Make sure all constant values use TransmitToAttr (as opposed to DefaultValue) when an attribute is available.

Delete any unneeded entities, attributes, and operations. You may have extraneous commands that were inadvertently included when you were in Record mode.

Some setting preferences for a model when tested in the Design Tool may not be the same when it is deployed on the Host Integrator Server. Check the setting for Apply default value to server.

Validator

The Host Integrator Validator evaluates your host application model for errors. You must resolve all errors reported in the Validator before you can deploy it to a Host Integrator server. The Validator evaluates the following aspects of a host application model:

VARIABLES

All variables must have a unique name and be complete.

TABLES

Model tables are tested for the following:

All table names are unique.

The minimum value of all integer table columns is less than the maximum value.

The minimum length of all text table columns is less than the maximum length.

Type mismatches between filter parameters and output parameters that are mapped to each other. For example, a mismatch occurs if a filter parameter is a string type and the output parameter it is mapped to is an integer.

All filter, data, and output parameters are used in the procedure.

All branch entities in a procedure are either the primary or alternate destination of the selected operation.

A route exists between all entities in a procedure.

All attributes marked as required for an operation are mapped to input parameters in a procedure.

All attributes mapped to input parameters are writable.

All attributes mapped to output parameters are readable.

ENTITIES

All entities must be unique, and there must be a valid operation that is able to navigate to every entity in the model.

Note

A warning appears if navigation does not exist between an entity and its home entity.

EVENT HANDLERS

All event handlers must be up to date with the associated libraries. Click Rebuild on the Events menu to rebuild the event handler libraries.

VALIDATING A MODEL


1. Open the model you want to validate in the Design Tool and from the Debug menu, select Validate. All imported models are automatically validated unless you clear that option on the Import tab of the Preferences dialog box.
2. In Validator, select the model elements you want to evaluate. You can evaluate Variables, Tables, Entities, and Event Handlers.
3. Click Validate. When complete, a validation report appears in the results box.


Validation Report

The validation report details any problems found in your host application model that would prevent it from being stamped by the Host Integrator as deployable. The report lists any problems found with model variables, entities, tables, procedures, or paths. Before you can deploy your model to a Host Integrator Server, the Validator must report no problems.

Interpreting Validation Reports

If no errors are found in your host application model, the Validation Report is empty. If errors are found, the following symbols appear in the validation report:

 **Error.** An error that prevents the Design Tool from authenticating this model was detected.

 **Caution.** Alerts you to a possible error in your model. The Design Tool will authenticate a model that has cautions, but no errors.

The report is organized as a tree structure with variables, entities, tables, and paths at the top level, with any sub-categories of these model elements appearing below them, each describing the problem detected.

Signature Analyzer: Pattern Tab

Use the Signature Analyzer to verify that all entities are uniquely identified. Select Signature Analyzer from the Debug menu to open the Signature Analyzer dialog box. The options available on the Pattern and Validation tabs of this dialog box can be used to compare and validate the signatures of any two entities in your host application model. The Design Tool performs the evaluation by comparing the patterns that have been defined for any two entities.

PATTERN TAB

In the left portion of the dialog box, select the two entities to compare by clicking the down arrow next to the **Entity** and the **Compare to** lists and clicking the entities you want to compare. Miniature versions of the entities you select help you visualize the entities you are comparing. At the top center there are two icons representing the primary entity and the entity you're comparing it to. If the two entities match, both icons are green; if the entity you're comparing to the primary does not match, it appears in yellow.

- Save Log button

To create a log file of the selected entity compared to the current screen, click the Save Log button. By default, the Save As dialog box appears and prompts you to save your model log file as `<model name>_siglog.xhtml` in your `\<VHI install directory>\models\<model name>` folder.

- Compare All button

To compare all defined entities to the current screen, click the Compare All button. A log file is written that describes if any of the defined entities match the current screen or if two entities are defined for the current screen. By default, the Save As dialog box appears and prompts you to save this model log file as

`<model name>_siglogall.xhtml` in your `\<VHI install directory>\models\<model name>` folder.

The Signature Analyzer displays the following information about the entities you are comparing:

- Patterns from

The Patterns from box lists all the patterns defined for the primary entity. For each pattern, the Design Tool tells you whether it detects a match between any patterns in the primary entity and the one you are comparing it to. As you click the patterns in the list, notice that information about the selected pattern is displayed.

- Position

Specifies whether the pattern is in a specified region of the screen or relative to the cursor, and lists the location and size of the pattern (row, column, height, and width).

- Properties

Field type tells you whether the field type is protected or unprotected. If you incorporated the text color into the pattern definition, the color appears next to **Text color**.

The **Pattern evaluation** box specifies whether the selected pattern is defined as present or not present. If the pattern evaluation is defined as "Screen properties not present", then the screen signature is not considered to be complete unless this pattern is absent. The purpose of this setting is to allow you to specify the absence of a particular pattern as an indicator that an entity is unique.

The **Text** box displays the text that forms this pattern, and specifies whether it is case sensitive or defined as `<blank>`, `<Any text>`, `<Any number>` or `<User specified>`.

The Design Tool is unable to recognize patterns under the following conditions:

A pattern containing a decimal point with a text property type defined as `<Any number>`.

A pattern containing blank spaces with a text property type defined as `<Any text>`.

VALIDATION TAB

To review or modify Validation tab settings, select Signature Analyzer from the Debug menu, and click the Validation tab on the Signature Analyzer dialog box. Use the Validation tab to check that any validation patterns, conditions, or cursor positioning wait options defined as part of the entity signature are being validated. These options will only appear if they have been defined on the Validation tab of the Advanced Entity Properties dialog box.



Navigator



Select Navigator on the Debug menu to open the Navigator. The Navigator dialog box displays a graphical representation of the entities and operations in your model. Use the Navigator to review the structure of your host application and test traversal operations in your model to ensure that there is a valid traversal operation that can reach every entity in your model. You can use the Navigator either while you are connected to the host or running in offline mode.

Note

Select the **Use for navigation commands to this destination** check box on the Operation tab so the Navigator is aware of an operation between two entities. Only one operation between two entities can have this option selected.

The Navigator uses the following symbols to represent entities and operations in a model:

Symbol	Description
	Entity
	Selected entity

Symbol	Description
	There is no operation  available to navigate away from this entity

Use **Root view** at to change the root entity currently displayed in the Model window. Select either Home entity, the Current entity, or Custom to choose any entity in the model.

TESTING TRAVERSAL OPERATIONS

To test the Host Integrator's ability to navigate to an entity, click the entity in the Navigator window. If you are connected to the host, you will see the Host Integrator navigate to the entity in the host application in the model window.

IDENTIFYING OPERATIONS

To determine the name of an operation, click it in the Navigator window. The name of the operation appears next to **Selected operation** in the top portion of the Navigator dialog box.

Debug Command List

Select Debug Command List on the Debug menu or click the Debug button on the Operation tab to open the Command Debug dialog box. This dialog box is used for debugging an operation or command list (for example, a move cursor, login, or logout command list) created in the Design Tool.

To see the entire list, click **Copy** from the toolbar and paste the list into a text editor.

Tip

Click the Apply button before debugging a command.

TO TEST AND DEBUG A COMMAND LIST

Select the operation or command list to test from the Name field and click Run. The Design Tool runs through each command listed and then reports the operation's success or failure in the Status box.

Click Step to execute the highlighted command, or click Skip to pass over it.

To stop execution of the command list as it is running, click Stop.

Attributes Test

You can test the Host Integrator's ability to read data and to store and write attribute data to the terminal screen before deploying your model to a Host Integrator Server. Select Attributes Test on the Debug menu or click the Test button on the Attribute tab to open the Attributes Test dialog box.

READ ATTRIBUTES

When you click the Read tab, all the readable attributes for the current entity are included. By default, all attributes are enabled for the test. This test is particularly valuable if you are implementing event handlers and need to confirm that an attribute is being read successfully.

Click Execute. The checked attributes are read from the current entity and the resulting values are displayed.

Executing a Read Attributes Test

1. To perform a read attribute test, you must either be connected to the host and have access to the application the model is based on, or you must load the model in the Host Emulator and connect to it.
2. Select an entity that contains configured readable attributes, open the Attributes Test dialog box, and click the Read tab. The names of the readable attributes configured for the selected entity appear in the **Name** column.
3. Click Execute. If the correct value for the attribute is read from the screen, the model is working properly.

Note

Reading attributes associated with non-display fields such as passwords will return blank strings.

WRITE ATTRIBUTES

Click the Write tab. The attribute write test from within the Design Tool simulates the capabilities of the SetAttributes method provided with the [AppConn APIs](#) and uses UpdateAttribute and UpdateAttributes commands in conjunction with attribute input commands in operations. By simulating these calls, you can accurately test these methods with the host application model before using the connectors to create a client application.

Some preliminary steps may be required before running a write attribute test on a host application model. If you choose to enable **Cache all attribute writes**:

For a specific entity: Select the **Cache all attribute writes** check box on the General tab of the Advanced Entity Properties dialog box.

Throughout your model: select this same option on the **Entity** tab of the Preferences Setup dialog box.

Then, create operations that contain one or more the following commands:

- an UpdateAttributes command

- a series of UpdateAttribute commands

- attribute input commands such as DefaultValue and TransmitToAttr

Note

When the **Cache all attribute writes** check box is selected, the Design Tool will not write attributes to the terminal screen unless an UpdateAttributes command or a series of UpdateAttribute commands is included in the attribute input operation.

Executing a Write Attributes Test

To perform a write attribute test, you must either be connected to the host and have access to the application the model is based on, or you must load the model in the Host Emulator and connect to it.

Select an entity that contains configured attributes, open the Attributes Test dialog box, and click the Write tab. The names of the attributes configured for the selected entity appear in the **Name** column.

Select a configured operation from the **Operation** list and click the Execute button.

If the operation successfully navigates to its expected destination and writes the cached attribute values to the terminal screen, the model is working properly.

Recordset Test

The Recordset Test dialog box allows you to test recordsets before the model file is deployed to the Host Integrator Server and accessible to client applications written using the [connector APIs](#). All of the recordset actions listed below simulate the capabilities of recordset methods provided with the connectors. This dialog box allows you to test these methods with the host application model before using the connectors to create a client application.

Configure the following options to create a test of the current recordset:

RECORDSET

Displays a list of defined recordsets. By default, the recordset that appears in the **Name** box is selected.

ACTION

Displays a list of defined actions for the selected recordset. Use the model example, located in your `\<VHI install directory>\models` folder, to demonstrate recordset actions.

See [Adding Entities](#) for information on these actions:

Fetch Records

Select Record

Insert Record

Update Current Record

Set Current Record Index

Reset Current Recordset



Note

After executing the first **Fetch Records**, you must reset the recordset by executing the **Set Current Record Index** action. When Host Integrator arrives at an entity with a recordset, it will always indicate the current record is at a record index of 0, which means before the first record. Any subsequent fetch of data will then start with the first record and proceed normally. In addition, the current recordset index action available in this dialog box works independently of operations configured on the Operation tab. If you want to test an operation that is not related to a recordset, use the Execute button on the Operation tab instead of the options available in this dialog box.

FILTER STRING

To customize your test, you can create a string in the Condition Edit dialog box to narrow your fetch to certain record or records in the recordset. To open the Condition Edit dialog box, click the Edit button.

LIMIT FETCH TO __ RECORDS

Select this check box and type the number of records to fetch. The fetch will begin with the line number indicated by the **Current record index** at the bottom of this dialog box. **Example:** If you set your current record index to line 5, and you want to fetch 20 records beginning at line 5, select this check box and type **20** in the box. Select **Fetch Records** and click Execute; the first 20 records beginning at line 5 should be returned in the **Fetch returned** box.

SYNCHRONIZE BUTTON

Click this button to synchronize after an entity change. This button is available after a recordset fetch terminates due to a change in an entity.

COPY FETCH RESULTS BUTTON

Click this button to copy your fetch results onto the Windows Clipboard.

EXECUTE BUTTON

Click this button to execute the selected action.

EDIT BUTTON

Click this button to open the Condition Edit dialog box.

FETCH RETURNED

Displays the results of the data fetch test.

PERFORM SCROLLING OPERATIONS

Click any of the available buttons to test scrolling operations that have been defined for the current recordset.

Note

Testing recordsets will only function while you are connected to a host or using the Host Emulator.

Procedure Test

Use the Procedure Test to test the procedures for a table. This allows you to debug your procedure definitions before deploying your model. For a basic introduction, see the [procedure example](#) in the tutorial.

TO TEST A PROCEDURE

To test a table's procedure, you must either be connected to the host and have access to the application the model is based on, or you must load the model in the Host Emulator and connect to it.

1. Load the host application model containing the procedure to test into the Design Tool.
2. Connect to the host.
3. Select **Procedure Test** from the **Debug** menu to display the Procedure Test dialog box.
4. Click the down arrow next to **Table** and select the table containing the procedure you want to test.
5. Click the down arrow next to **Procedure** and select the procedure you want to test. All filter and data parameters for the selected procedure appear in the **Procedure filters** box. If the filter is defined as required, a check mark appears in the box to the left of the filter.
6. In the **Value** column, enter data for the parameter. If a parameter is required, you must enter a value for that parameter.
7. Click **Execute**.

The Design Tool will test the procedure. If this is a SELECT procedure, the Design Tool will display the output in the **Procedure outputs** dialog box. The record count is displayed at the bottom of the dialog box.

If the Terminal window is visible, you will see the Design Tool navigate to the appropriate host screen while it tests the procedure.

PROCEDURE OPTIONS

You can limit the number of records to be returned for the procedure test. Type in a number for **Limit results to <x> records**. The total record count is displayed with the procedure results.

After executing the procedure, click Copy to copy the procedure output to the Clipboard.

If an event handler that includes an [Execute Procedure] call is attached, a lightning bolt is displayed in front of the procedure name. When you click **Execute**, the associated event handler is invoked as part of the procedure test. The **Execute Default** button is available so that you can test the procedure without testing the event handler at the same time. Click

Execute Default to test the procedure as displayed in the Procedure Editor, without testing the associated event handler.

DEBUGGING A PROCEDURE

If you experience problems with a procedure used to satisfy an SQL query or you want to fully investigate the behavior of a procedure, you can debug it in the Debug Procedure dialog box.

Click the down arrow next to **Table** and select the table containing the procedure you want to test.

Click the down arrow next to **Procedure** and select the procedure you want to test. All filters for the selected procedure appear in the **Procedure filters** box. If the filter is defined as required, a check mark appears in the box to the left of the filter.

In the **Value** column, enter data to test against the procedure. All filters for the selected procedure appear in the **Procedure filters** box. If the filter is defined as required, a check mark appears in the box to the left of the filter.

Click **Debug** to open the Debug Procedure dialog box with the procedure displayed.

SQL Test

Use the Test SQL dialog box to test SQL queries on the tables you created for your host application model. This allows you to debug your table and procedure definitions before deploying your model. Before you can run an SQL test on a host application model, you must first create tables derived from the model that contain the data that you want to query.

When you test an SQL query, you simply enter a [supported SQL-92 statement](#) in the **SQL statement** box of the Test SQL dialog box and the Design Tool returns the results.

For a basic introduction, see the [SQL example](#) in the tutorial.

Tip

If you're using SQL, note the Tables dialog box setting for Allow SQL SELECT to return a subset of columns.

EXECUTING AN SQL TEST

To test an SQL query on your model tables, you must either be connected to the host and have access to the application the model is based on, or you must load the model in the Host Emulator and connect to it.

To run a test SQL query on a table, follow these steps:

1. Load the host application model containing the table to test into the Design Tool.
2. Select **SQL Test** from the **Debug** menu to display the SQL Test dialog box.
3. Connect to the host.
4. In the **SQL statement** box, type a supported SQL statement.
5. Click Resolve. The Design Tool determines the procedure or procedures that could be used to satisfy the query you entered and display them in the Procedures box.
6. Click Execute. The Design Tool executes the procedure or procedures and displays the output in the Output recordset dialog box. If the Terminal window is visible, you will see the Design Tool navigate to the appropriate host screen while it fulfills the SQL query.

Note

The total record count is displayed with the output recordset results. You can limit the number of results for your test. Type in a number for Limit results to `<x>` records.

7. Click Copy to copy the results to the Clipboard. This is handy if you need to evaluate and compare results from multiple SQL queries.
8. Click Reset to perform another test or Close to close the Test SQL dialog box.

TO DEBUGGING A PROCEDURE

If you experience problems with a procedure used to satisfy an SQL query or you want to fully investigate the behavior of a procedure, you can debug it in the Debug Procedure dialog box.

1. In the **SQL statement** box, type a supported SQL statement.
2. Click Resolve. The Design Tool determines the procedure or procedures that could be used to satisfy the query you entered and display them in the Procedures box.
3. Click Debug. The procedure or procedures display in the Debug Procedure dialog box.

Connection Events Test

Click Connection Events Test on the Debug menu to confirm that the life cycle event handlers and model event handlers you have added to your model are performing as you expect. Use this dialog box to test sequences of events that require a reset of the terminal session: connecting to the host and establishing a client connection. You can perform tests in both dedicated model and pooled session environments. In order to use the Connection Events Test, the model must be connected to the host, but login is not required.

- Dedicated Session

Click New Client to simulate the sequence of one client application disconnecting and another client connecting to this model using the [ConnectToModel](#) API.

- Pooled Session

- Click New Session to simulate destroying and creating a pooled session, then connecting to the session using the [ConnectToSession](#) API.

- Click New Client to simulate the sequence of one client application disconnecting and another client reconnecting to this model using the [ConnectToSession](#) API.

- Credentials

If the event handler requires a user name and password, specify the client user credentials here. An [Authenticate User](#) event will be sent to the event handler based on the credentials shown here. The initial values are based on those provided on the Environment tab in the Event Handler Settings dialog box, but you can change them. You can also update the Environment tab settings if you make changes here. The credentials you use for this test remain current until another test or a reset.

You can also specify or review host credentials. Click the Model Variables button to view or change other model variable values.

- Event Handlers

The Connection Events Test dialog box checks the event handlers currently associated with life cycle or model event handlers displayed here.

- Testing Event Handlers

After you have specified your test parameters, the Edit, Rebuild, and Reload options are available as you debug the event handlers.

- Edit Button

Click the Edit button to open the selected event handler in your default editor.

- Rebuild

After editing an event handler, click this button to recompile event handlers and to update them for the default JAR/assembly file.

- Reload

Click this button to reload all event handler libraries to match the latest copies stored with the model.

CONNECTION EVENTS TEST OPTIONS

The Connection Events Test dialog box tests sequences of events that require a reset of the terminal session: loading a model, connecting to the host, and establishing a client connection. The startup and shutdown sequences below illustrate the differences between each of the test options.

This option simulates the sequence of one client application disconnecting and another client connecting to this model using the [Connect to Model method](#):

The Client Disconnected event is fired.

The terminal session navigates to the home entity, which could involve [Execute Operation](#) events.

The logout process is executed, which could involve firing the Execute Logout event.

The host session is disconnected from the host.

Steps 1-7 of standard reset processing are executed.

The host session is connected to the host.

The login process is executed, which could involve firing the Execute Login event.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The Client Connected event is fired.

Steps 1-4 can also be accessed by pressing the logout button, while steps 5-9 can be accessed by pressing the login button on the same toolbar.

The New Session option simulates destroying and creating a pooled session, then connecting to the session using [ConnectToSession method](#).

With this option, a session pool host session is created, connected to the host, and logged in before a client session is actually created. This simulates the real-world behavior of a session pool session.

The Client Disconnected event is fired.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The logout process is executed, which could involve firing the Execute Logout event.

The host session is disconnected from the host.

The Client Session Destroyed event is fired.

The Host Session Destroyed event is fired.

The Host Session Created event fires.

The login process is executed, which could involve firing the Execute Login event.

The terminal session navigates to the home entity, which could involve Execute Operation events.

The Client Connected event is fired.

The Authenticate User event is fired, with any resulting errors displayed in the Event Handler Console.

The Client Session Created event is fired.

Pooled Session: New Client

This option simulates the sequence of one client application disconnecting and another client reconnecting to this model using the ConnectToSession method.

In this case, only the client session events are fired because a session pool host session is not normally logged out or destroyed between client invocations.

The Client Disconnected event is fired.

The terminal session is navigated home, which could involve Execute Operation events.

The Client Session Destroyed event is fired

An Authenticate User event is fired.

The Client Session Created event is fired.

The Client Connected event is fired.

4.12.2 Model Debug Messages

Model debug messages provide detailed information for diagnosing problems such as synchronization with the host and for repairing a malfunctioning model. They provide a review of a model interacting with a terminal datastream, allowing you to diagnose and repair screen recognition, data reads and/or writes, and synchronization flaws in a model. This feature is particularly helpful for those model designers working with VT hosts, where synchronization issues are more common.

A model debug message recording can be used for models under development in the Design Tool and for runtime models that have been deployed to the Host Integrator server.

You can debug models as you build them in the Design Tool.

- You can open the model debug messages file created by a runtime server. The naming convention for model debug messages (`.vmr`) is based on server startup time, model name or session pool, and session ID. You can customize a `.vmr` file name by adding a prefix to the default naming convention. This can simplify testing and working with `.vmr` files. To add a prefix to the `.vmr` file name:
 - Use the AppConn methods `setVMRFilePrefix` and `getVMRFilePrefix`. These methods are available for Java and .NET AppConn programs.
 - When implemented, a prefix will be added to the default file name (Connection_60.162.vmr) and will look similar to this; `aprefixConnection_60.162.vmr`.
- You can export a `.vmr` file in html or pdf format. Click Export, select the information you want to include and specify where to save the file. Select the extension (pdf, htm, or html) when prompted.
- File `vmr.css`, located in `c:/Program Files/Micro Focus/Verastream/HostIntegrator/bin`, determines the layout of the html and pdf export files. If desired, modify that file before exporting the `.vmr` file.
- There is a standalone utility available to view your model debug messages. Using the VMR Viewer you can open, save, and export your model debug messages without having to interact with the Design Tool. The VMR Viewer can be launched from either the Windows Start menu or using the command line option, `Vmrview.exe`. To export a `.vmr` file in pdf format from the command line, use the `-d` parameter with the `vmr.exe` command.

```
c:\Program Files\Micro Focus\Verastream\HostIntegrator\bin>vmr.exe
Print VMR debug file contents

Usage: vmr [-t] [-s] [-h] [-d] source [destination]

-t, -time           Include timestamps in output.
-s, -screenshots    Include screenshots in output.
-h, -html           Use html formatting in output.
-d, -pdf            Send output to a pdf file
source             Specifies the VMR file to process.
destination        Specifies the output file.
```

- Model debug message files (`.vmr`) and the session server log files do not contain data written to non-display terminal fields or stored in encrypted model variables. Values that are redacted in `.vmr` and session server log files are:

Values of variables that are marked encrypt value.

Encrypted values stored in the model, in operations such as `TransmitToAttrEncrypted` and `TransmitANSIEncrypted`.

Values written to a non-display field (3270/5250).

Values of attributes that are write-only.

There are rare exceptions where the system cannot determine if a value should be redacted. These can include: logging of model event handlers and callbacks, and specific host applications that may issue commands to read the entire terminal screen.

Warning

It is always important to verify that sensitive data has been removed from model debug message (`.vmr`) files and session server log files before making them available to others.

You can have up to fifty model debug message recordings open at the same time. This allows you to compare model behavior under different circumstances or after you have made changes to the model.

Configuring model debug message recording mode

Model debug message recording is always enabled for models you open in the Design Tool. When testing your model within the Design Tool, you can configure the message recording level for the test server.

You can record deployed models on the Host Integrator server. You can configure the recording option as you deploy the model.

Model Debug Message Components

VMR VIEWER

There is a standalone utility available to view your model debug messages. Using the VMR Viewer you can open, save, and export your model debug messages without having to interact with the Design Tool. The VMR Viewer can be launched from either the Windows Start menu or using the command line option, `Vmrview.exe`. To export a `.vmr` file in pdf format from the command line, use the `-d` parameter with the `vmr.exe` command.

THE MODEL DEBUG MESSAGES LIST

The left hand panel of the Model Debug Messages dialog box shows a list of the messages. The time-based list includes messages that have a subset or a series of nested subsets of additional messages, or descendants, associated with that action. Click the + symbol to view the message subset. These descendants typically display the underlying steps that transpired to produce the top level action.

Click an underlined link in a message to navigate to the entity in question. This option is not available when viewing a saved model debug message (`.vmr`) file when the model file is not present.

When you're viewing all messages (not datastream messages only), right-click a message for additional options:

When a + symbol is present, right-click the message to expand the message node.







Open the Signature Analyzer.


Create a new entity. If your model navigates to an unexpected host screen, use the New Entity option to capture the information for your model.

Move to the next error.

Save a node of the message list as a separate .vmr file. This option is available for root level messages.

Use the following symbols to manipulate the model debug message list:

Symbol	Meaning
	Display all messages
	Display datastream messages
	Display relative time for descendants
	Find a message
	Find next error
	Clear all messages (Actions that are in progress will not be cleared)

Symbol	Meaning
	If you have opened a model debug message file other than the one associated with the current model, click to close the file

MODEL DEBUG MESSAGE DETAILS

The right hand panel of the model debug message dialog box shows the details of the message highlighted on the left. Depending on what type of message is selected, it can include one or more of the following:

- Ending snapshot

A snapshot of the terminal at the time the message was generated. This is particularly helpful if you expect a full screen of data to have been processed, but the action and the associated snapshot shows only a subset of the host screen (with a gray overlay) was actually sent or received. The gray area of the snapshot represents the area of the screen that changed during the course of the action. This kind of feedback makes it much easier to identify where a synchronization issue needs to be handled.

- Starting and ending snapshots

A navigation message shows both the starting and ending snapshot.

- Message

When an error or operation message is selected, the message text is repeated in the lower-right panel. This text can be selected and copied to the Clipboard.

- Procedure data

When a procedure message is selected, the procedure filter parameters and procedure output are displayed. You can use the Copy button to copy the filter parameters or procedure results to the Clipboard.

- Datastream

When the message is associated with a datastream process, the datastream content displays in a text box in the lower right panel.

- Event handlers

When a fired event message or event callback message is selected, the input and output of the event or callback is displayed. You can view all messages associated with the runtime model or focus on datastream messages only.

DATASTREAM MESSAGES



Click the datastream messages icon to view datastream messages only in the model debug messages list.

When you select a datastream message (whether you're viewing all model debug messages or just datastream messages), select a message to see datastream details in the window at the lower right of the dialog box.

THE DATASTREAM WINDOW

The datastream window displays the real-time data sent and received from a host. This information is particularly valuable when debugging datastream issues on a character mode host. You can copy and paste text from this window into the [WaitForCommString](#) command dialog box or the [Host Communication String](#) event dialog box to configure operations that include raw host communications. This window contains errors, and operations in addition to the datastream messages.

Without this window, it is difficult to determine what effect a single key has on the host. With a character mode host, a simple cursor movement can use three packets of 50 or more characters. Lack of knowledge about these internal communications can often lead to missed synchronization points.

In addition, this view allows an easy view of the data that comes from the host, allowing you to choose which data to use for synchronization. Review the information about nonprintable characters, such as control functions, that appear in the Datastream view.

If you select a procedure in the Model Debug Message list, the details in the lower right include a Copy button. You can copy the procedure results to the Clipboard.

SEARCHING DATASTREAM CONTENT



When debugging a VT model, use the Suggest Escape Sequence button to have the model debug messages utility suggest a unique escape sequence (if present) that can be used to help synchronize the model. This option is available for VT models only.

You can copy text from this panel to the Clipboard by using the right-click menu or Ctrl+C keystroke. This is useful to paste into a WaitForCommString command (in the Operation Edit dialog box, or in the Host Event Edit dialog box after selecting Host Communication String event) to resolve synchronization issues.

Type a string into the text box at the bottom of the datastream window. All instances of that string in the datastream window are highlighted.



When debugging a VT model, the Next Packet option lets you move to the next packet using the same search string, simplifying the search process.



When debugging a VT model, the Previous Packet option lets you move to the previous packet using the same search string, simplifying the search process.



Use the Next Match button to move to the next occurrence of the string.



Use the Previous Match button to move to the previous occurrence of the string.

Using Model Messages

On the Debug menu, click Model Debug Messages to open the Model Debug Messages dialog box.

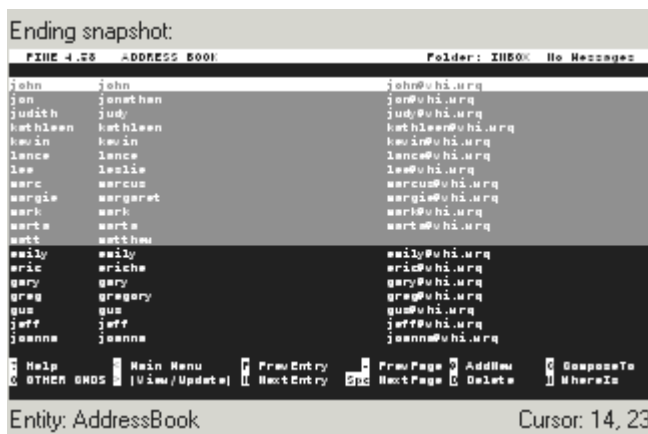
LOCATING AND SOLVING A SYNCHRONIZATION PROBLEM

If your model does not reach an expected entity or cursor location, review the model debug messages. The example below shows a sequence on a VT host for a PageDown operation.

Message	Time
Operation AddressBook.PageDown arrived at AddressBook (primary)	13:12:40.451
Executed command list	13:12:40.451
CheckOperationConditions	13:12:40.451
TransmitANSI " ", rcDontWaitForEcho	13:12:40.451
Transmitted 1 bytes to the host	13:12:40.451
Received and processed 1024 bytes from the host	13:12:40.491
Received and processed 695 unexpected bytes from the host	13:12:40.591

The final message, indicating unexpected bytes received from the host, is a strong indicator of a synchronization problem with the host. By reviewing the messages preceding this message, you can narrow down the source of the problem.

The highlighted message, "Operation AddressBook.PageDown arrived at AddressBook (primary)" has the associated ending snapshot on the lower right:



With a PageDown operation, a full screen of data should have been processed, but the action and the associated snapshot shows only a subset of the host screen (with a gray overlay) was actually received.

To ensure that all the expected data is processed before proceeding, you can add a Wait command to the operation.

 **Note**

There may be cases where a synchronization issue does not produce a message that unexpected bytes were received from the host, especially when running on the server. It's a good idea to look for partial operation snapshots similar to the one above whenever the model does not reach an expected entity or cursor location.

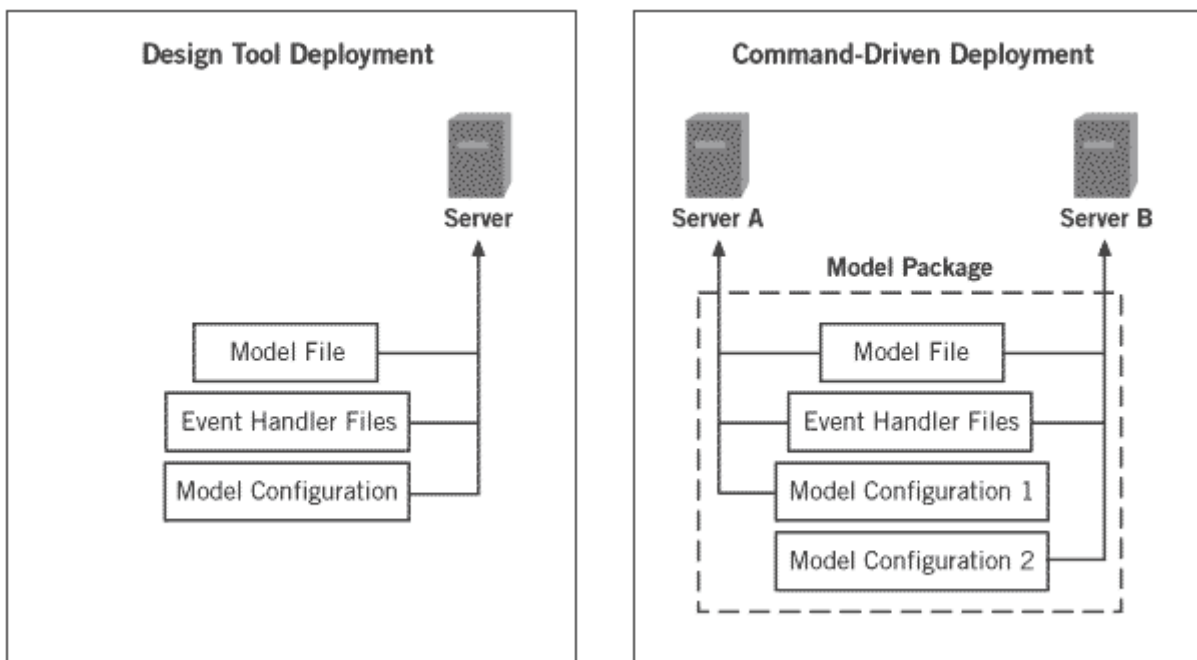
4.13 Deploying Models

4.13.1 Deploying Models

As you create a model, you can deploy it to the Host Integrator Server on your workstation or to a designated remote server for testing or for a single server production deployment. However, once you move into production testing and deployment, you need to deploy your model to the Host Integrator Servers that will handle the communication between the model and the host application.

If your final deployment is to multiple production servers and involves one or more model configurations, use Host Integrator's deployment commands to deploy a model package. These commands allow you to use scripts to automate a large-scale deployment.

The following figure illustrates Host Integrator's two deployment methods:



Using the Design Tool to Deploy a Model

Using Commands to Deploy a Model Package

4.13.2 Using the Design Tool to Deploy a Model

To use the Design Tool to deploy a model, select a deployment option: Deploy to the Local Server or Deploy to Remote Server. The Deploy to Remote Server option deploys the model to any designated remote server.

Web services are automatically provided by the session server as an embedded SOAP stack or REST service after a model package is deployed with the WSDL and metadata generated at deployment, for more information view [Deploy Web Services](#).

Note

If your integration solution includes a Web application created with Verastream's Web Builder, you must also deploy the Web application files. See [Deploying a Verastream Web Application](#) for more information.

Select Deployment Options

The *Deploy to Local Server* or *Deploy to Remote Server* commands on the Design Tool's File menu, deploy the model using the default host name and port number, with no session pool.

Note

The Deploy to Remote Server option provides the ability to use another Host Integrator server besides the one included with the Design Tool. This option is limited to the same basic settings as the local server deployment; it does not include all the options available when deploying to a non-local server using the `packagemodel` and `activatemodel` commands.

Early Adopter Program

Users can deploy to a Kubernetes environment using the server name. New Host Integrator Session Servers that register with the management server will deploy the model upon startup.

To modify these defaults or to use a session pool, set deployment options:

On the File menu, select Deployment Options.

The Host tab of the Deployment Options dialog box displays the host name and port number that the server connects to when the model is loaded. If you don't use the Deployment Options dialog box, the Deploy to Local Server command deploys the model using the default host name and port number, and without a session pool. Use the edit boxes on the Host tab, to change these to whatever is required for your server.

The *Sessions connect to host defined by model* will use the settings in the Session Settings dialog box.

To use host settings other than those specified in the Sessions Settings dialog box, click *Sessions connect to specified host* and type in a host name and port number.

3. Click the Sessions tab. To deploy the model without using a session pool, select *Deploy as a single session* and skip to step 7 below.
3. If your model uses a session pool, select *Deploy as a session pool* and provide the session pool name, the startup entity name, and the minimum and maximum numbers of sessions.
4. Open the Variables tab. If the default variable values stored in the model are suitable for all pooled sessions, clear the *Provide variable values for pool* check box and proceed to the last step. Otherwise, check the *Provide variable values for pool* check box and proceed to the next step.
5. The names of the variables defined in the model appear in the Variable column of the Variable names table.

Select *Include* for each variable which must have a non-default value.

Select *Unique* for each variable which must have a different value for every session in the pool. (At least one variable must be marked unique.)

Select *Hidden* for variable values that must be encrypted in the deployment and server configuration files. This check box is checked by default for variables marked as "hidden" in the model.

Use the Variable values table to assign values to your variables.

4. The table will contain a row for each session. The number of rows is equal to the maximum session pool size. The Session (leftmost) column numbers the sessions in the pool. Additional columns, each labeled with an "included" variable name, appear to the right. Enter a value for each variable name in each session.
6. The Recording tab controls model debug options. The default, *Record Nothing*, is appropriate for most cases. Change these settings if you want to take advantage of the Model Debug Messages feature. Click OK.

Resetting deployment options

Click the Defaults button to reset the model's deployment options back to the default settings.

View deployed model in Administrative Console

In the Administrative Console you can verify that the model, pool, and model variable lists have successfully deployed.

Open the Administrative Console and, if necessary, connect to your management server.

Open the Session Server Explorer. (Perspective > Host Integrator > Session Servers)

In the left pane, open Installation > Servers > [your server name]. Double-click on the Models, Model Variable Lists, or Pools items to view the configuration tab in the right pane, associated with each. You can right-click on an item and choose Properties to view the specific properties for each model, model variable list, or pool.

Note

Because any changes that you may make to model or session pool configurations (except pool scheduling) in the Administrative Console are overwritten by subsequent deployments from the Design Tool or by the `activatemodel` command, it is best to make those changes in the Design Tool.

4.13.3 Using Commands to Deploy a Model Package

Host Integrator provides deployment commands that you can use from the command line or in a script to deploy a model to multiple servers. With this capability you can combine model files with any event handler files they use, and also with configuration information that tells a Host Integrator Server how to provide access to the model (for example, using a session pool of a certain size, or by using specific requests for a new session).

[Using the `activatemodel` command](#)

[Using the `packagemodel` command](#)

[Using the `deactivatemodel` command](#)

About Model Packages

A model package is a .zip archive that you create using the `packagemodel` command. A model package can contain the following items:

Modelx file— The model package must include a modelx file, together with entityx files located in the entities directory. Tablex files, located in the tables directory, are needed if appropriate. The files can be located anywhere. You specify its location when you run the command to create a model package file (`packagemodel`).

Model file—A model file is the only file that must be in the model package. It can be located anywhere. You specify its location when you run the command to create a model package file (`packagemodel`).

Optional: Event handler files (*.JAR)—If your deployment includes event handlers, the event handler folders must be located in a folder named `scripts` beneath the folder that contains the model file.

Optional: A configuration descriptor(`deploy_desc.xml`)—If your deployment includes a configuration descriptor, it must be located in a folder named `deploy` beneath the folder that contains the model file.

Optional: A model variable list descriptor (`mvl_desc.xml`)—If your deployment includes a model variable list descriptor, it must be located in a folder named `deploy` beneath the folder that contains the model file. (See the documentation for the `packagemodel` command for exceptions.)

After you run the `packagemodel` command, the .zip file is created in your current working directory (unless you've specified a different name and location for the model package using the `-package` switch).

While a model package can only include one model, that model can, in turn, be deployed to multiple servers, optionally with different configuration information for each server.

How to Create and Deploy a Model Package

To create a model package for deployment, the only item you must have is a model file, unless your model uses event handlers, then the event handler .JAR files must also be present. If you're using session pools, or if your model uses model variable lists, you will also need to create configuration descriptor .xml files.

For each model that you intend to deploy using the new deployment commands, do the following:

Create a model.

(Optional) Create event handler .JAR files.

(Optional) Write descriptors: either just a configuration descriptor or a configuration descriptor and a model variable list descriptor. A configuration descriptor is required if you want to use session pools, or if you have multiple configuration targets. A variable list descriptor is required if your model uses one or more model variable lists. Because model variable lists contain model variable values for all sessions in a session pool, you cannot have a model variable list descriptor file unless you also have a configuration descriptor file.

Use the [packagemodel command](#) to create a model package. The packagemodel command can be executed from any folder, as long as `<VHI install folder>\bin` is in your PATH.

Deploy the model using the [activatemodel command](#). To deactivate an activated model, use the `deactivatemodel` command. You can open a command window set to the appropriate path by running `/hostintegrator/bin/deployCommandWindow.bat`.

Writing the Descriptors

Descriptors are .xml files that describe a model's configuration and how it should be deployed. For each model package that you create you can supply a configuration descriptor and a model variable list descriptor.

A configuration descriptor file specifies how a model should be deployed to a server. A configuration descriptor is an efficient means to deploy to multiple targets using a command line.

A variable list descriptor is required if your model uses one or more model variable lists. Because model variable lists contain model variable values for all sessions in a session pool, you cannot have a model variable list descriptor file unless you also have a configuration descriptor file.

A [sample configuration descriptor file](#) (deploy-ex3.xml) is in your `<VHI install folder>\Help\plugins\com.attachmate.vhi.help\html\` reference folder.

There are two ways that you can create a descriptor:

Using an XML schema: An XML schema for both descriptor files, named `vhi_deploy.xsd`, is located in the `lib\schema` folder of your Verastream installation. With an XML editor you can use this schema to help create descriptors files.

By hand: You can create the descriptors by hand using a text editor such as Notepad.

Configuration descriptor files have four main sections:

- [XML root tag](#)

The configuration descriptor header provides standard XML root tag information, such as XML namespace and schema information, and also provides the name of the model you are deploying.

- [Deployment targets](#)

The Deployment Targets section is where you specify a unique name to identify one or more configuration targets. This is a required attribute for any configuration descriptor file. The name you provide is used when you execute the `activatemodel` command.

 **Note**

If there is only one deployment target, you can use the name "default". You can then omit the target parameter when issuing the `activatemodel` command.

You can also specify one or more model configurations and session pool configurations, which you define later in the file. All of these names -- deployment target name, model configuration name, and session pool configuration name -- are unique names that you create in this file. They do not necessarily relate to names you provided while creating a model file.

- [Model configurations](#)

The Model Configurations section is where you provide the name or IP address of the host on which the modeled application is running, as well as the port number. This collection of information is called the model configuration. You can specify one model configuration or several, if the modeled application is running on several hosts. Model Configurations are referenced in Session Pool Configurations (required if using session pools) or in the Deployment Targets (optional, to override the host configured in the model). If you have no model configuration defined, the model deploys to the default host configured within the model (Design Tool > Connection Menu > Session Setup> Host name or IP address).

The Model Configurations section of the descriptor is where you also specify the Web service settings you want to use for the non-pooled model session. Model Web service settings are not inherited by the pool; you should configure each pool separately. These settings are described in detail in the Administrative Console help.

- [Session pool configurations](#)

Session Pools are sets of host sessions preconfigured for access by data objects. These host sessions can be configured with model variables and a starting entity, which allows faster host session allocation and access to the host system because the host session can be logged on and waiting at the host application's main entry screen (entity) when a data object is ready to use it. You can create and monitor pools in the Administrative Console.

If your model uses session pools, you can use the Session Pool Configurations section to define which session pools are configured. Each session pool configuration name, such as

`pool-one-cfg`, is also referenced in the Deployment Targets section by any target where you want to create the session pool on activation.

You can also configure Web service information for the pooled session. Each pool must be configured separately or session server defaults are used. The Web service elements are optional, but if they are included, they must be shown in the order below. If an element is not used, the session server default is used

Note

If you're upgrading from an earlier version of Host Integrator, you may have a significant amount of model or session pool configuration information. If you want to update a model without affecting existing configuration information, do not include descriptor files in your model package.

You should also not create descriptor files if you use an external service to handle your host passwords. In such a situation, a model variable list descriptor, which provides host user IDs and passwords, would be difficult to keep up-to-date.

EXAMPLE DESCRIPTOR FILES

XML header example:

```
<model-configuration-descriptor
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.wrq.com/vhi_schema/vhi_deploy.xsd"
  xmlns="http://www.wrq.com/vhi"
  model-name="AccountingSystem"
  version="1.0">
</model-configuration-descriptor>
```

Deployment targets example

In this example, executing the `activatemodel` command on deployment target `two-pools` specifies activation of the `AccountingSystem` model (defined under XML Root Tag) using 1) `accounting-host`, a model configuration you define in the Model Configurations section, and 2) `pool-one-cfg` and `pool-two-cfg`, session pool configurations you define in the Session Pool Configurations section.

```
<deployment-target name="two-pools">
  <pool-config-ref>pool-one-cfg</pool-config-ref>
  <pool-config-ref>pool-two-cfg</pool-config-ref>
</deployment-target>
```

For a given named model and session server combination, only one deployment target at a time can be active. You can configure multiple session pools in the same target, and you can run `activatemodel` multiple times to use the same target with multiple servers. However, you cannot run `activatemodel` multiple times with different targets on the same server -- only the last target specified is in effect.

Model configurations example

In this example, the user has defined host pincher and port 21 on pincher as the `accounting-host` model configuration. Using this example with the example under Deployment Targets, the model called `AccountingSystem` connects to port 21 of host `pincher`. The default model debug messages recording mode is `record-nothing`. The other options are `record-errors`, `record-errorsessions`, and `record-everything`. To start debugging on a production server, try `record-errors` first.

The Web service configuration is defined in the `ws-config` section. The `ws-config` section and the elements contained within the section are optional. Any elements you include must be listed in the order shown in this example. If a setting is not included, session server default values are used.

```
<model-configuration name= "accounting-host">
  <host-name>pincher</host-name>
  <host-port>21</host-port>
  <recording-mode>record-nothing</recording-mode>
  <ws-config>
    <publish-service-enabled>true</publish-service-enabled>
    <ws-resource-enabled>true</ws-resource-enabled>
    <execute-sql-statement-enabled>false</execute-sql-statement-enabled>
    <process-string-enabled>false</process-string-enabled>
    <method-timeout-msec>60000</method-timeout-msec>
    <suspend-timeout-minutes>60</suspend-timeout-minutes>
    <ws-namespace>urn:xmlns:model-namespace</ws-namespace>
  </ws-config>
</model-configuration>
```

More information

- Model configuration properties, including Web service settings, can be set in the Administrative Console.
- You can use the `ws-namespace` value to provide a custom namespace for a model or pool during deployment.
- To revert to the original default session server settings for a particular model or pool, you can:
 - Clear all values: `<ws-config xsi:nil="true"/>` or

Clear a specific value: `<publish-service-enabled xsi:nil="true">`

Session pool configurations example

```

<!-- A pool that uses an MVL from the MVL descriptor, model variable override, session pool, and Web service information -->
<pool-configuration name="pool-one-cfg">
  <pool-name>PrimaryAccounting</pool-name>
  <model-config-ref>accounting-host</model-config-ref>
  <min-sessions>5</min-sessions>
  <max-sessions>200</max-sessions>
  <max-free-sessions>150</max-free-sessions>
  <initial-free-sessions>20</initial-free-sessions>
  <max-pending-sessions>30</max-pending-sessions>
  <connect-delay>30</connect-delay>
  <max-retries>4</max-retries>
  <pool-started>true</pool-started>
  <startup-entity>Main</startup-entity>
  <model-variable-list>user-list-one</model-variable-list>
  <variable-override name="department" value="sales" hidden="true"/>
  <ws-config>
    <publish-service-enabled>true</publish-service-enabled>
    <ws-resource-enabled>true</ws-resource-enabled>
    <execute-sql-statement-enabled>false</execute-sql-statement-enabled>
    <process-string-enabled>false</process-string-enabled>
    <method-timeout-msec>60000</method-timeout-msec>
    <suspend-timeout-minutes>60</suspend-timeout-minutes>
    <ws-namespace>urn:xmlns:pool-namespace</ws-namespace>
  </ws-config>
</pool-configuration>

<!-- A pool that uses an MVL from the MVL descriptor -->
<pool-configuration name="pool-two-cfg">
  <pool-name>SecondaryAccounting</pool-name>
  <model-config-ref>accounting-host</model-config-ref>
  <min-sessions>10</min-sessions>
  <max-sessions>40</max-sessions>
  <startup-entity>Main</startup-entity>
  <model-variable-list>user-list-one</model-variable-list>
</pool-configuration>

```

For each session pool configuration, the sub-nodes `<pool-name>`, `<model-config-ref>`, `<min-sessions>`, `<max-sessions>`, and `<startup-entity>` are required. If the session pool uses a model variable list, `<model-variable-list>` is also required. Use one or more variable override names to set a variable's value for all sessions in the pool to the same value.

Note

When you enter values in this section of the file each entry must occur in the order laid out below.

Setting pool properties and scheduling is more easily accomplished using the user interface in the Administrative Console since it is handled by the management server. However, you can use this section to enter values.

Use the following table as a guide for filling out this section:

Item	Required	Details
<code><pool-configuration name></code>	X	The name of the pool configuration, referenced in the Deployment Targets section.
<code><pool-name></code>	X	How the pool is referenced when it's on the server.
<code><model-config-ref></code>	X	A reference to the model configuration name. The Deployment Targets section and one or more pools will refer to this name.

Item	Required	Details
<code><min-sessions></code>	X	The minimum number of sessions that the pool should contain.
<code><max-sessions></code>	X	The maximum number of sessions that the pool should contain.
<code><max-free-sessions></code>	Optional	Maximum idle sessions. Maximum idle sessions cannot exceed the Maximum concurrent sessions defined for the pool, or your licensed session limit, which is displayed as the maximum value for this option.
<code><initial-free-sessions></code>	Optional	Initial idle sessions. Specifies the number of sessions that are preloaded by the server. This is the number of sessions that a server should preconnect.
<code><max-pending-sessions></code>	Optional	Maximum pending sessions. The maximum number of sessions trying to connect to the host simultaneously. The actual number may be much less if the sessions start and login rapidly.
<code><connect-delay></code>	Optional	Connection throttle delay (seconds). The amount of time to wait since the last host connection before attempting another.
<code><max-retries></code>	Optional	Maximum retry backoff (n=0 is 1 second, n=1 is 2 seconds, n=2 is 4 seconds, n=3 is 8 seconds, and so forth). Each time there is an error starting host sessions in a pool, the pool waits for an increasing amount time before trying again (it backs off). This option limits the retry backoff so the delay doesn't become too long.
<code><pool-started></code>	Optional	Indicates if the pool should start automatically when deployed, true or false.
<code><startup-entity></code>	X	The entity in the model that will serve as the session pool's startup entity.
<code><model-variable-list></code>	Optional	If the pool has a model variable list, this is where you provide its name. This is the name of the model variable list, as the server knows it. Its contents are defined in the model variable list descriptor file.

Item	Required	Details
<code><variable-override-name></code>	Optional	If the pool has one or more model variable overrides, list them here.

MODEL VARIABLE LIST DESCRIPTOR

A model variable list descriptor file defines the contents of one or more model variable lists. These lists are referenced by name from the configuration descriptor file's Session Pool Configurations section.

A [sample model variable list descriptor](#) (mvl-example.xml) is in your `<VHI install folder>\help\plugins\com.attachmate.vhi.help\html\` reference folder.

Model variable list descriptor files have two main sections:

XML Root Tag

The model variable list descriptor header provides standard XML root tag information, such as XML namespace and schema information, as well as the name of the model you are deploying. Here's a sample of the XML header for a model variable list descriptor:

```
<mvl-descriptor
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.wrq.com/vhi_schema/vhi_deploy.xsd"
  xmlns="http://www.wrq.com/vhi"
  version="1.0">
</mvl-descriptor>
```

Model Variable List

The Model Variable List section is where you provide the contents of one or more variable lists for your model:

```
<model-variable-list name="user-list-one">
  <model-variable name="userID" hidden="false" unique="true"/>
  <model-variable name="password" hidden="true" unique="false"/>
  <value-row>
    <value>fred</value>
    <value>pass</value>
  </value-row>
  <value-row>
    <value>jimbo</value>
    <value>pass</value>
  </value-row>
</model-variable-list>
```

For each model variable list, the sub-nodes `<model-variable-list name>`, `<model-variable name>`, `<value-row>`, and `<value>` are required. The first `<value>` in each `<value-row>` corresponds to the value of the first `<model-variable>`, the second corresponds to the value of the second `<model-variable>`, and so forth.

Use the following table as a guide for filling out this section:

Item	Required	Details
<code><model- variable- list name></code>	X	The name of the model variable list.
<code><model- variable name></code>	X	How a model variable is referenced when it's on the server. Model variables can be hidden or visible (hidden=true or hidden=false) and unique or non-unique (unique=true or unique=false). A variable marked as hidden can be optionally encrypted in the model package file. A variable marked as unique must have entirely unique values.
<code><value- row></code>	X	A group of variable values, usually a user name and password for a host application.

Item	Required	Details
<value>	X	An individual value in a value row - usually a user name or password for a host application.

Using the activatemodel Command

Use the activatemodel command to deploy or redeploy a model, its event handlers, and its configuration information to a running Host Integrator Server

ABOUT THE ACTIVATEMODEL COMMAND

The activatemodel command references data in a model package file in determining how to deploy or redeploy a model. Any deployment target defined within the package can be activated on the specified server.

If a model that is being activated already exists on a server, a redeploy will occur automatically. Any configuration related to the existing model is deleted in favor of the configuration specified in the configuration target. New configuration information is not merged with previous configuration information. The only exception is for a model package that contains no configuration descriptors (as opposed to empty configuration descriptors) that is activated. In that case, the activatemodel command will simply update the model and (if provided) its event handlers, leaving any session pools or model variable list configurations that are specific to this model untouched in the Verastream Administrative Console.

Note

Before you use the activatemodel command, you must first create the model package using the packagemodel command.

COMMAND SYNTAX

```
activatemodel -package input_package_file
[ -target configuration_target]
[ -server vhi_server]
[ -user username with -password password]
[ -passphrase encryption_passphrase]
[ -mgmtserver management_server]
```

Option	Short version	Description
package	-p	The file name of the model package to activate. The file name specified can either be fully qualified or relative to the current working directory. Required.

Option	Short version	Description
target	-t	The configuration target defined within the model package you wish to activate. This switch is optional. If it is not supplied, the target default is implied.
server	-s	The network name of the Host Integrator Server where the model is to be activated. This switch is optional.
user and password	-u and -v	The user credentials of the Host Integrator administrator. Optional.

Option	Short version	Description
passphrase	-e	A pass phrase used to generate a decryption key for sensitive data contained within the model package. It must match the phrase used when the model package was created. This switch is optional.

Early Adapter Program

Users deploying a model in a Kubernetes environment can use the `-mgmtserver` or `-m` option to deploy a model to the Management Server. New Host Integrator Session Servers that register with the management server will deploy the model upon startup.

COMMAND EXAMPLES

The first command activates the model contained within `MyModel.zip` using the default configuration target. The model is activated on the same machine from which the command was executed:

```
activatemodel -package MyModel.zip
```

The next command activates the model contained within `MyModel.zip` in the current working directory, using the Zephyr configuration target, on the machine whose network name is `VHI_Server`. As demonstrated by the first command, above, if no `-server` switch is supplied, the model is activated on the VHI server where the command is executed. To activate a model on multiple machines you would execute the command multiple times, with different `-server` values.

```
activatemodel -package MyModel.zip -target Zephyr -server VHI_Server
```

The next command uses the short versions of the switches instead of the long versions (for example, `-s` instead of `-server`). Also, this command assumes that security is activated on the Host Integrator installation and provides the Host Integrator administrator user name and password. Finally, assuming that the pass phrase "In sequent toil all forwards do contend" was provided when the model package was created, it is provided here so that a decryption key for sensitive data can be generated.

```
activatemodel -p MyModel.zip -t Zephyr -s VHI_Server -u VHIadmin -w VHIPassWord
-e "In sequent toil all forwards do contend"
```

Using the `packagemodel` Command

Use the `packagemodel` command to combine a model file, its event handler files, and its configuration information into a single distributable file, the model package. Sensitive information included with the model package, such as password lists, can be encrypted.

ABOUT THE PACKAGEMODEL COMMAND

While everything contained within a model package relates to a single model, that model can be deployed to several servers, each with a different configuration, such as different session pool parameters and model variables. The `packagemodel` command is what you use to create the model package. To activate the model package, use the `activatemodel` command.

REQUIREMENTS

Before you use the `packagemodel` command, all necessary files should be present:

The model files

Optional. Event handler .JAR files. If your model uses event handlers, these files should be in a folder named "scripts", beneath the folder that contains your model file.

Optional. Descriptor files. A configuration descriptor file is required if you want to use session pools, or if you have multiple configuration targets. If your model uses one or more model variable lists, a model variable list descriptor file is also required, in addition to a configuration descriptor file. Because model variable lists contain model variable values for all sessions in a session pool, you can't have a model variable list descriptor file without also having a configuration descriptor file. Descriptor files should be in a folder named "deploy", beneath the folder that contains your model file.

Sample directory structure

Name	Date modified	Type
deploy	24-9-2019 12:17	File folder
entities	8-1-2020 12:45	File folder
scripts	24-9-2019 12:17	File folder
tables	8-1-2020 12:45	File folder
CICSAccts.modelx	8-1-2020 12:45	VHI Modelx File
modelx_1.xsd	21-2-2020 13:28	XML Schema File

The `packagemodel` command can be executed from any folder, as long as `<VHI install folder>\bin` is in your PATH.

PACKAGEMODEL COMMAND SYNTAX

```
packagemodel -model model_directory\model_file  
[ -package output_package file]  
[ -mvl mvl_descriptor -model model_directory\model_file]  
[ -passphrase encryption_passphrase]
```

Option	Short version	Description
model	-m	The model you want to package for deployment. The value for this option can either be a model file <code>modelName.modelx</code> or the older specification <code>modelName.model</code> , or a model directory name <code>modeldir</code> containing a model file of the same name <code>modeldir.modelx</code> or <code>modeldir.model</code> . See below for more information.
mvl	-l	Specifies the location of a variable list descriptor file. While any model variable list descriptor named <code>mvl_desc.xml</code> , located in the deploy directory, is automatically included in the model package, there may be circumstances where a model variable list is shared between multiple models. The use of this switch allows the administrator to override the default behavior and explicitly specify a model variable list descriptor file to package with the model. The path specified can either be fully qualified or relative to the current working directory. This switch is optional.
package	-p	This switch specifies the file name of the created model package. The file name specified can either be fully qualified or relative to the current working directory. This switch is optional. By default, the model package is named "modelName.zip" and placed in the current working directory.

Option	Short version	Description
passphrase	-e	This switch specifies a pass phrase used to generate an encryption key for encrypting sensitive data contained within the model package. Specifically, the values of any variables marked as "hidden" within the configuration descriptors are encrypted. This switch is optional. By default, encryption is not used. The phrase specified should either be a combination of eight or more random characters or a word phrase of five or more words. Spaces are significant; if they are used, the phrase should be placed in quotes.

More about the model option (-m)

If directory `modeldir` contains both a `.modelx` and a `.model` file, you must specify the file you want to use: `modeldir\modeldir.modelx` or `modeldir\model`. The path specified can either be fully qualified or relative to the current working directory. This switch is required.

If a subdirectory named `deploy` exists in the model directory, it is automatically searched for configuration descriptors. Specifically, if a file named `deploy_desc.xml` exists, it is assumed to be a configuration descriptor and included in the package. If a configuration descriptor was found, and a file named `mv1_desc.xml` exists, it is assumed to be a model variable list descriptor and included in the package.

If a subdirectory named `scripts` exists in the model directory, the model is assumed to use event handling. All JAR files (Java) or assembly files (.NET) are found in the `scripts/lib` folder and are included in the package.

EXAMPLES

The first command creates a package file for `MyModel` called `MyPackage.zip` and places it in the current working directory, which is the model directory. If a subdirectory named `deploy` exists in the model directory, it is searched for the descriptor files `deploy_desc.xml` and `mv1_desc.xml`. If these files exist, they are assumed to be the configuration and model variable list descriptor files and they are included in `MyPackage.zip`. You can also identify the model you want to package by using the `-m` or `-model` switch to specify the model directory instead of the actual model file.

It is possible to have both a model and modelx file in a model directory. This can happen when you use Save As and write a modelx file into the original model directory or vice versa. In this situation you will receive an error when running the package command and be prompted to supply the name of the file you want to use.

To specify what file you want to use, type:

```
packagemodel -model MyModel/MyModel.modelx -package MyPackage.zip
```

or

```
packagemodel -model MyModel/MyModel.model -package MyPackage.zip
```

The next command is equivalent, except it uses the model variable list descriptor file `MyModelMVL.xml` instead of `deploy\mvl_desc.xml`:

```
packagemodel -model MyModel -package MyPackage.zip -mvl MyModelMVL.xml
```

The final command uses short versions of the switches (for example, `-m` instead of `-model`) and uses the password phrase "In sequent toil all forwards do contend" to generate an encryption key for sensitive data contained within the model package. This means that variable values in the descriptor files that are marked "hidden" are encrypted:

```
packagemodel -m MyModel -p MyPackage.zip -l MyModelMVL.xml
```

```
-e "In sequent toil all forwards do contend"
```

Using the `deactivatemodel` Command

The `deactivatemodel` command allows you to remove a previously activated(deployed) model and its related configuration from a running Host Integrator Server.

Note

Before you use the `deactivatemodel` command, you must have an activated model running on a Host Integrator Server. Only the model name is needed. You do not need to provide the path to the model.

SYNTAX

```
deactivatemodel -model model_name  
[ -server vhi_server]  
[ -mgmtserver management_server]  
[ -user username -password password]
```

Option	Short version	Description
model	-m	This switch specifies the name of the model to deactivate. Required.
server	-s	The network name of the Host Integrator Server on which to deactivate the model. This switch is optional. If it is not supplied, the network name localhost is implied.
user	-u	The user credentials of the Host Integrator administrator. Required.

Option	Short version	Description
password	-w	The credentials of the Host Integrator administrator. Required.

Early Adapter Program

Users deactivating a model in a Kubernetes environment can use the `-mgmtserver` or `-g` option to deactivate a model to the Management Server. New Host Integrator Session Servers that register with the management server will not deploy the model upon startup.

DEACTIVATECOMMAND EXAMPLES

The first command removes the model `MyModel` from the Verastream server running on the local computer. It also deletes any session pools and model variable lists used solely by this model:

```
deactivatemodel -model MyModel
```

The following command removes the model `MyModel` from the server `VHI_Server`. It also deletes any session pools and model variable lists used solely by this model:

```
deactivatemodel -model MyModel -server VHI_Server
```

The final command uses the short versions of the switches. Since security is enabled by default on the Host Integrator Server, the Host Integrator administrator's user name and password must be provided to deactivate the model.

```
deactivatemodel -m MyModel -s VHI_Server -u VHIadmin -w VHIPassword
```

4.14 Using Web Services

4.14.1 About Verastream Web Services

Providing a Web service to your host applications enables rapid reuse of the business logic that exists on these hosts. Web services provide reusable APIs for creating portals, Web applications, and other business solutions. And, because they are technology independent, they can run on any platform. Developers can use utilities to generate specific files to locate and consume the Web service.

Web Service Standards

Host Integrator, at model deployment, generates both a SOAP (Simple Object Access Protocol) Web service and a REST (REpresentational State Transfer) service that uses JSON (Java Script Object Notation) to call VHI procedures from a Web application.

- VHI SOAP-based Web services adhere to WS-Standards, which includes WS-Resource, WS-Security, and WS-Addressing. These standards follow a third-party [specification](#). In most cases proper SOAP headers are generated to use these standards and it is unnecessary for you to write separate code, however some applications may not be able to consume these features for various reasons. The Web service standards are turned off by default. This setting can be changed on the Session Server Web Service Property panel in the Administrative Console.

See a [List of Web Service Specifications](#) for a description of WS-Security, WS-Addressing, and WS-Resource.

- VHI REST services use JSON as the data carrier, and, like the SOAP-based services, runs over HTTP.

Deploying the Web Service

In VHI, after a model is deployed using the Design Tool, Web services are automatically provided by the session server as both an embedded SOAP stack and as a REST service. The embedded Web service supports all model procedures and features, including `executeSQLStatement` and `ProcessString` event handlers.

METADATA REQUEST FORMAT

- SOAP request - `http://host-name:port/vhi-ws/model-or-session/model-or-pool-name?wsdl`
- REST request - `http://host-name:port/vhi-rs/model-or-session/model-or-pool-name?json`
- Native Web-Service - `http://host-name:port/models/model-or-pool-name`

About Model and Environment Variables

Model variables and environment variables are an optional part of every SOAP or REST document that a client submits to a VHI Web service to invoke a Web method. Although these elements are optional, meaning that the WSDL and schema, or REST metadata, do not require them in the syntax, a given application may require them to be specified to run successfully.

After you deploy your model to the session server, you can view and test the model to see if the Web service that is created works as expected, by clicking Test on the Deployment Successful message box to open the Web Services Explorer. In the Web Services Explorer you can add, remove and view model and environment variables.

Model variables

By default, VHI creates three model variables whenever a model is created. Any user-defined model variables that you create also are listed. How these user-defined variables are used and their semantics are application-dependent.

Model variable	Description
userID	Created by default. Specify a userID if needed when executing a VHI procedure
Password	Created by default. Specify a password if needed when executing a VHI procedure
Cursor position	Created by default. Specify the cursor position if needed when executing a VHI procedure

Model variable	Description
Additional user-defined model variables	You can specify other application-specific data when executing a VHI procedure

 **Note**

Within Web services there is a method, `getModelVariables`, that you can use to query the current settings.

Environment variables

VHI uses environment variables to communicate to the service how to run. For example, what server or domain to use.

Environment variable	Description
DomainName	Use this variable to have the client use the <i>connect via domain</i> feature when you execute a VHI procedure. If this option is not specified, VHI will invoke the procedure with using <i>connect via server</i> .
ServerName	Use this variable to have the client specify the management server for <i>connect via domain</i> , or to specify an alternate session server when not using the <i>connect via domain</i> option. The default is localhost.
SessionID	Use this variable to have the client specify a suspended session when using stateful Web services. SessionId is obtained from <code>wsResourceCreate</code> , and is valid until <code>wsResourceDestroy</code> is used to release the session.
Password	This variable has the client specify a password when executing a VHI procedure as required with session server security enabled.
Username	This variable has the client specify a user name when executing a VHI procedure as required with session server security enabled.

Environment variable	Description
VmrPrefix	Specifies a model debug file name prefix for this web method invocation. This variable is ignored unless model debug recording is enabled.

Note

When using SOAP Web services, if WS-Addressing, WS-Resource, or WS-Security are enabled in the Administrative Console, then, using the WS-Addressing, WS-Resource, and WS-Security specifications, the SessionID, Password, and Username information can be placed in the SOAP header instead.

Web Services Ports

Web Services deployed to the session server are set by default to run on HTTP port 9680 or HTTPS port 9681.

Using the WSDL Document

The Web Service Description Language (WSDL) document describes the interface to the Web service, the data types it uses, and the location of the service. Developers use this document to identify inputs, outputs, and methods needed to consume the Web service. Web services generation utilities use the WSDL document to create proxy objects and sample code for the services.

You can access your Web services WSDL documents here:

`http://<session server>:9680/vhi-ws` for a list of available WSDLs

For example:

`http://<session server>:9680/vhi-ws/model/<model name>?wsdl` for a non-pooled model

`http://<session server>:9680/vhi-ws/session/<pool name>?wsdl` for a pooled session

-or here-

`https://<session server>:9681/vhi-ws` for a list of available WSDLs

For example:

`https://<session server>:9681/vhi-ws/model/<model name>?wsdl` for a non-pooled model

`https://<session server>:9681/vhi-ws/session/<pool name>?wsdl` for a pooled session

Using the REST Document

Just like XML, JSON (using JavaScript syntax) is text only and thus can be read and used as data input by any programming language. However, unlike XML, JSON is very light-weight and easy to read. It is also very widely used for interchanging data on the Web.

JSON documents are an easier to use alternative to XML and follow this simple format:

```
{ "employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Betty", "lastName": "Brown" },
  { "firstName": "Michael", "lastName": "Jones" }
]}
```

You can access all available REST services for models or sessions deployed to the Host Integrator session server here:

```
http://<session server>:9680/vhi-rs
```

For example:

```
http://<session server>:9680/vhi-rs/model/<model name>?json for a non-pooled model
```

```
http://<session server>:9680/vhi-rs/session/<pool name>?json for a pooled session
```

-or here-

```
https://<session server>:9681/vhi-rs
```

For example:

```
https://<session server>:9681/vhi-rs/model/<model name>?json for a non-pooled model
```

```
https://<session server>:9681/vhi-rs/session/<pool name>?json for a pooled session
```

EXAMPLE

Take a look at a [REST Web application](#) that uses the CICSAcctsDemo model to demonstrate Host Integrator's REST service. It provides an example of both a stateful and stateless procedure call.

Prerequisites

Using the Design Tool, deploy CICSAcctsDemo to localhost. The model is located here: `.../Micro Focus/Verastream/HostIntegrator/examples/ModelSamples/CICSAcctsDemo`.

Verify that the security option is disabled in the Administrative Console. This option is disabled by default and can be found under Session Server Properties > General Properties > Security.

Metadata Web-Service

VHI provides metadata for deployed models and their procedures on a separate web service in the OpenAPI format. The [OpenAPI](#) specification describes REST APIs. Because it is language agnostic, its metadata can be used to generate code for REST clients, servers, and tests in many languages and frameworks.

The Metadata Web service runs on HTTP port 9682 and HTTPS on 9683. HTTP is disabled by default, but can be enabled by setting the environment variable `VHI_NATIVE_WEBSERVICE_HTTP` to true.

Certificates can be configured for HTTPS to provide secure connections. The CA certificate and private key must be in pem format. To configure certificates you must first define these environment variables:

`VHI_NATIVE_WEBSERVICE_CERT` should point to the pem, the default is `%VHI_ROOT%/etc/serverCertificate.pem`.

`VHI_NATIVE_WEBSERVICE_KEY` should point to the pem, the default is `%VHI_NATIVE_WEBSERVICE_CERT%`.

If your certificate and private key are in PFX format, you can use the OpenSSL command line utility or other conversion tool to convert it to standard PEM format.

For example: `%VHI_ROOT%/bin/openssl pkcs12 -import -in example.p12 -out example.pem -noenc`

USING SWAGGER UI FOR WEB-SERVICE TESTS

Swagger UI provides an easy interface to test out the endpoints.

Note

To test the service you must use a browser with CORS disabled.

If you are using HTTPS, the keys must be configured and stored in your system.

If you are using HTTP, the environment variable `VHI_NATIVE_WEBSERVICE_HTTP` must be set to true.

With CORS disabled, navigate to Swagger UI using HTTPS or HTTP:

`https://localhost:9683/swagger/ui`

`http://localhost:9682/swagger/ui`

In the search bar enter `/models/CICSAcctsDemo` and click Explore.

Select `/SearchByName`, and then Click `Try it out`.

Erase `modVars` and `envVars`

Change `string` to `W`

Execute and the data is returned from the web service.

EXAMPLE CODE GENERATORS

You can find sample generators for Java in `\HostIntegrator\examples\Generators`.

Using the Java Generator

1. Optional: Retrieve the metadata from the server using curl. `curl https://<url>:9683/models/<modelName> -k -o spec.json`. If you are using powershell, change curl to curl.exe.
2. Install Java and Maven and add them to your path
3. Navigate to `\HostIntegrator\examples\Generators\Java`, where there is a sample POM, OpenAPI specification in JSON format, Windows and Unix script for your use. Copy the files to a writable location outside of the install directory. If you do not do this, the code generation will fail and throw a `java.nio.file.AccessDeniedException`.
4. To generate code, execute the command `mvn install`. This generates and compiles Java source files into the `target/generated-sources/openapi` folder.
5. To use a different input:
Open the POM.
In the build section, change the value of the tag `<inputSpec>spec.json</inputSpec>` to match the name of your spec.
6. Using your IDE, this example uses IntelliJ IDEA, open the `target/generated-sources/openapi` directory. When prompted, select Maven, and then Trust the project. `target/generated-sources/openapi` contains a README.md with the

instructions on using the generated code. `target/generated-sources/openapi/docs` also contains a `DefaultApi.md` which include examples of how to use your endpoints.

Example of how to run the `/SearchByName` endpoint for `CICSAcctsDemo`:

Windows - `compile_and_run.bat`

Unix - `sh compile_and_run.sh`

Managing Web Service Settings

You use the Administrative Console to enable and disable all VHI Web services. In the Properties panels of the session server, model and session pool configurations you can set server-level policies as well as options for specific models and session pools.

Server level properties set Web service configurations, including enabling HTTP or HTTPS connections and other Web service elements. You can also decide whether to make Web services available for models and pools and link to the list of all available Web services.

Model level properties display information and options for the Web service associated with the selected model that is deployed to the session server. This is where you can override the session server values for a particular model.

Session pool level properties contain settings to configure the Web service for a particular session pool. This is where you can override the session server properties set for the Web service and view the Web service documentation.

4.14.2 Testing Web Services

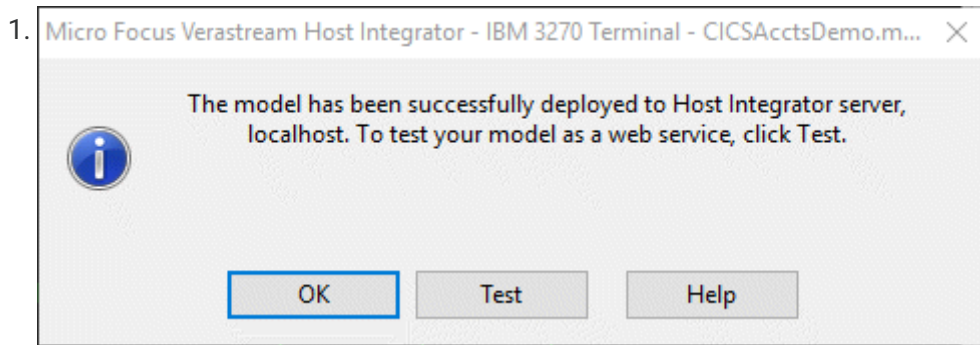
Testing your service is an important step before you run your process in a production environment. The Web Services Explorer, a browser-based tool, tests your model as a SOAP-based service.

Note

In the interest of increased security, the Web Services Explorer web service test tool is disabled by default. To enable it, open the file `install-dir\Micro Focus\Verastream\ManagementServer\conf\container.properties` and change the property `servletengine.port=0` to `servletengine.port=8095`, save the file and restart the Management Server. You can restart the Management Server from the Verastream Administrative Console. Click 'Management', then right click the server and select 'Restart'.

Testing is an easy two-step procedure:

1. After you deploy the model to the Host Integrator Server, a message box displays informing you that the model has been successfully deployed.

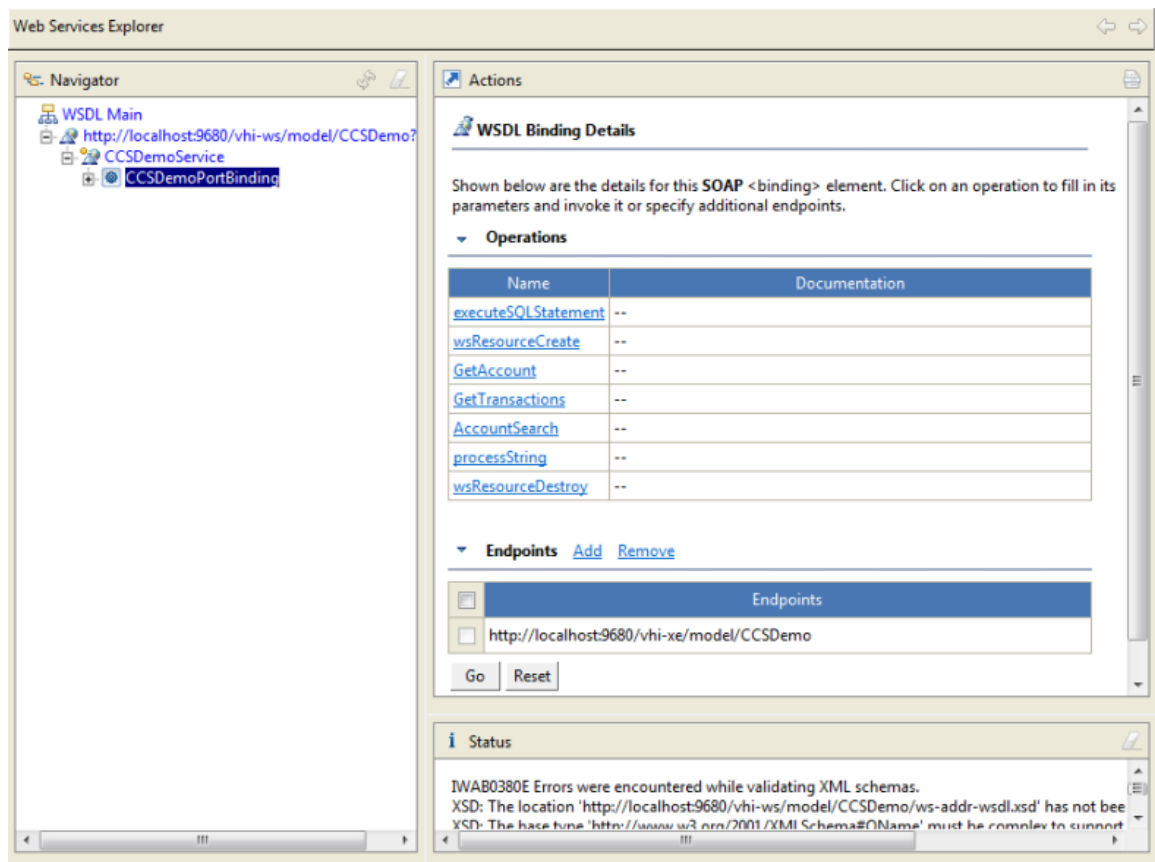


1. Click Test to launch the Web Services Explorer. If the Test button is disabled, verify that you have deployed a model that contains at least one procedure. Web services are not published for models that do not contain procedures.

 **Note**

To test SOAP Web services that are protected by WS-Security, Basic Authentication, or HTTPS use a third-party development environment.

2. In the Actions view of Web Services Explorer, enter the parameters of the WSDL in order to test the service. Click Go.



If your test is not successful

The Status section of the Web Services Explorer provides the first troubleshooting information.

Note

Benign errors may be generated during XML schema validation. If the Status section states that the WSDL was successfully opened, you can ignore these errors. If you are interested in more detailed information on these error messages, see [Technical Note 10118](#).

Under the Status section, review the error message. Always read status messages from the bottom up, addressing them in that order.

Click Source to expose the SOAP request and response envelope. The information in the response envelope is typically very useful in diagnosing the problem.

Always look carefully at the response from the server. When there is an error often there is a stack trace in the server console or log.

If you are getting a response, but not the response you expected, debug your Web service.

4.14.3 Web Service Security

SOAP-based Web services support the WS-Security standard which describes how security signatures and encryption are attached to headers of SOAP messages for access control. If you have session server security enabled in the Administrative Console, the user name and password credentials (for a user in the User security profile in Administrative Console) can be transmitted in the SOAP headers. If your client does not support WS-Security, inband equivalents are enabled by default, so information is passed in the data payload.

REST services use Host Integrator credentials that are sent using the environment variables in the request message. REST services use HTTPS for security and can be configured to use HTTP basic access authentication, preferably over HTTPS.

SSL Encryption

Web services automatically start HTTPS on port 9681 using SSL. To configure security features, edit the `HostIntegrator\sesssrvr\services\ws\META-INF\service-cfg.xml` file to change the `secure`, `transportLayerSecurity`, `authnMetadata`, or `authnService` properties.

To enable HTTP Basic Authentication, edit the `%VHI_ROOT%/sesssrvr/services/ws/META-INF/service-cfg.xml` file to set `authnMetadata` and `authnService` to `true`. The `authnService` enables basic authorization for execution, while `authnMetadata` enables basic authorization for accessing metadata. The credentials are cached by the web service and not passed to the session server until a subsequent SOAP or REST request is received. If the user is unauthorized, the initial HTTP authentication will appear to succeed but the subsequent request will fail.

Note

When Basic Authentication is enabled, you cannot test web services using VHI Web Services Explorer as the SOAP client. Authentication credentials sent via SOAP, REST, or HTTP Basic Authentication are transmitted over the network as clear text unless an HTTPS connection is used.

For more information see [Security Overview for Verastream 7.x](#).

In a default installation of Host Integrator, self-signed security certificates (`server.cer`) are generated. To add your own certificates issued by a trusted certificate authority (CA), see [Using Custom Keystores and Certificates](#).

4.14.4 About Stateful Web Services

Stateful Web services are those that can preserve their state for long running or distributed transactions. Typically, Web service calls are "stateless", which means that subsequent invocations are completely independent. Since the Host Integrator can suspend sessions, Verastream Web services uses stateful Web services to support this behavior. Stateful Web services automatically use the session server suspend/resume functionality.

Invoking stateful Web services

There are two different ways to invoke Verastream Host Integrator Web services.

The first, and preferred, method is:

- **Standards-based method** Host Integrator SOAP Web services implement the WS-Resource standard, which includes WS-Addressing. If the Web service client understands WS-Resource and WS-Addressing, it can automatically offer stateful behavior with minimal additional programming.

REST services do not understand WS-Resource or WS-Addressing. REST stateful services use environment variables to carry the session ID.

- `rsResourceCreate` is called to suspend the session (create a resource) and returns a session ID. The session ID can be used for any number of transactions.

`rsResourceDestroy` is called to free the session when it is no longer needed.

The second method is:

- **Inband support method - SOAP services** If the Web service client does not understand WS-Resource and WS-Addressing, or has difficulty importing WSDLs that contain references to WS-Resources and WS-Addressing, you can disable this feature on the session server Web service property page in the Administrative Console. However, disabling standards support does not disable stateful Web services in Host Integrator. You can use inband support.

What does "inband" support mean?

Inband support uses proprietary data types in the Web service document payload, instead of using the SOAP header which WS-Resources and WS-Addressing normally require. Because the Web service client tools don't automatically know how to use our proprietary inband support, you have to manually program stateful Web services.

- Use `wsResourceCreate` to allocate a resource (a VHI session) that you will use for multiple invocations on the Web service. The return document includes the session id. You will use this session id for subsequent Web service invocations.
- Invoke as many methods on the service, as many times as you need. For each invocation, you need to send the session id obtained in step one, in the session id field of the optional environment variables defined in the WSDL.
- When you are finished with the resource (VHI session), use `wsResourceDestroy` to free the session. For `wsResourceDestroy`, you need to send the session id obtained in step one, in the session id field of the optional environment variables defined in the WSDL.

The Web service configuration setting suspend timeout controls how long you can leave the session inactive. The Web service will return a SOAP fault, saying the requested resource is not available if you attempt to use the resource after you have destroyed it, or after the suspend timeout has expired.

Note

Stateful Web services suspend sessions with a specified timeout of 60 minutes; sessions not resumed within this period are automatically terminated by the session server.

Load Balancing and Failover

Web Services are not load balanced but rather the session server on which the Web service resides can be load balanced.

In order for clients to take advantage of the VHI load balancing and fail over feature when using Web services they must specify the "connect via domain" or "specify a domain server" in the SOAP request or use environment variables to ensure that the Web service calls the appropriate session server. Refer to the VHI domain documentation for further information on VHI load balancing and fail over, especially Configuring Management Servers for Failover under General Management Services.

4.15 Additional Resources

4.15.1 FAQs

The following provides the answers to some frequently asked questions about Verastream Host Integrator:

General Questions

- **What is the Verastream Host Integrator Development Kit?**

The Verastream Host Integrator Development Kit provides an innovative approach to host systems integration and re-engineering, which requires no changes to the existing host systems and causes no interruption to the end-user operating environment.

- **Which host character sets are supported?**

In the table below, there are pairs of code pages distinguished by Euro support, such as German and German + Euro. The Euro version of each pair reports that there is support for the Euro code page version to the host; the non Euro version can display the Euro, but does not report this to the host.

EBCDIC

Country	codepage	3270	5250
Austrian+Euro	1141	yes	no
Austrian	273	yes	no
Baltic	1142	yes	no
Baltic+Euro	1156	yes	no
Belgian (Old)+Euro	1148	yes	yes
Belgian (New)+Euro	1148	yes	no
Belgian	274/500	yes	yes
Belgian (New)	500	yes	no
Canadian French+Euro	1148	yes	yes
Canadian French	037/260	yes	yes
Custom	N/A	yes	yes

Country	codepage	3270	5250
Cyrillic+Euro	880	yes	yes
Danish+Euro	1142	yes	yes
Danish	277	yes	yes
East Europe+Euro	870	yes	yes
Finnish+Euro	1143	yes	yes
Finnish	278	yes	yes
French+Euro	1147	yes	yes
French	297	yes	yes
German+Euro	1141	yes	yes
German	273	yes	yes
Greek+Euro	875	yes	yes
Icelandic+Euro	1149	no	yes
Icelandic	871	no	yes
Italian+Euro	1144	yes	yes
Italian	280	yes	yes
Multilingual+Euro	1148	no	yes
Multilingual	870	no	yes
Netherlands+Euro	1140	yes	no
Netherlands	037	yes	no
Norwegian+Euro	1142	yes	yes
Norwegian	277	yes	yes
Portuguese+Euro	1140	yes	yes
Portuguese	037	yes	yes
Spanish+Euro	1145	yes	yes
Spanish	284	yes	yes
Swedish+Euro	1143	yes	yes

Country	codepage	3270	5250
Swedish	278	yes	yes
Swiss (French)+Euro	1148	yes	no
Swiss (French)	500	yes	no
Swiss (German)+Euro	1148	yes	no
Swiss (German)	500	yes	no
Swiss+Euro	1148	no	yes
Swiss	500	no	yes
Thai *	838	yes	yes
Turkish+Euro	1026	yes	yes
UK English+Euro	1146	yes	yes
UK English	285	yes	yes
US English+Euro	1140	yes	yes
US English	037	yes	yes

Some Thai characters (cent symbol, the split vertical bar, and the IBM not symbol) do not display correctly in Design Tool dialog boxes that display text for patterns or filtered text. These characters can be input via the keyboard and are displayed correctly in the Design Tool terminal window. The Java, COM, and .NET connectors handle these characters correctly.

VT host character sets
DEC Supplemental
ISO Latin-1 (8859-1)
ISO Latin-9 (8859-15)
PC 437 (English)
PC 850 (Multilingual)
Windows 1252

HP host character sets
HP Roman-8

HP host character sets

HP Roman-9

Modeling

- **Is it the connection or the applications that are block mode or character mode?**

A combination of the host and the application it is running dictates block mode versus character mode. VT/ASCII terminals are always character mode. IBM AS/400 terminals, which use the 5250 datastream protocol, and IBM mainframe terminals, which use the 3270 datastream protocol, are typically block mode. HP3000 terminals can be block mode or character mode, but the combination of the host and the host application determine the type of connection that is needed.

- **What's the difference between scrolling operations in character mode applications and block mode applications?**

Scrolling operations, such as page up or page down operations, are in no way unique to character mode or block mode applications. They are important for recordsets, but these types of screen constructs exist in all types of host applications that are accessible by Host Integrator. The only difference is that character mode applications require more specific recordset configuration since it is more difficult to determine when the host is finished sending recordset data after a query has been made.

TIP: Use Model Debug Messages to view the real-time data sent and received from a host.

- **Are there any hard coded timeouts in Verastream Host Integrator or should I expect to be able to configure my own timeouts for settings and operations?**

There are no hard-coded timeouts in the Host Integrator system. For example, you can configure timeouts for the host connection, global navigation, and "keep alive" inactivity in the Model Properties dialog box. Under Properties on the Operation tab, you can also configure a timeout for a selected operation. In addition, all of the Host Events commands provide timeout settings. If any timeout occurs in an operation, the whole operation is stopped—there is no "master timeout." For example, if a Host Events command has a timeout of 10 seconds, while an operation has a timeout of 30 seconds, they each get the

full extent of their time until they complete or time out. Since there are so many potential timeouts in Host Integrator, and you must adjust them all appropriately.

- **How do I make the current entity an alternate entity or error entity?**

See [Troubleshooting Error Patterns and Error Entities](#) for more information.

- **What is the difference between using the DefaultValue command and the TransmitToAttr command in operations?**

DefaultValue implies that the value argument will only be written if the client has not provided a value already. If the behavior you want is to always transmit the value to the terminal, regardless of the API client code, then use TransmitToAttr.

- **What's the difference between error patterns and error entities?**

Error patterns are often host messages that appear on the terminal screen when an error occurs during an attribute write. These types of messages are common when modeling character mode host applications. They cannot be part of an entity signature, but they can be contained in an operation condition. If an entity encounters an error pattern during an operation, a text or an attribute value can be returned. No action can be associated with an error pattern. An error pattern can consist of a particular host message or a non-blank host message.

Error entities are entities that only appear when a host error occurs. Error entities can be contained in operations destinations, and they can return text or attribute values in an exception. They cannot, however, be valid or intermediate destinations. Error entities should contain operations to move back to a navigable entity. Typically, the operation to move away from an error entity contains the logic to recover from the error, for example, a locked 5250 terminal can be released by pressing the Reset key. An error entity can be as simple as a copy of some other entity, when the host message line is non-blank.

Error entities typically have an action associated (an operation to get back to a navigable state), while error patterns cannot contain an action. For more detailed information, see [Troubleshooting Error Patterns and Error Entities](#).


- **How and when are error patterns detected?**

Error patterns are registered with the Host Integrator Server scanning component when the operation begins. At any time during the operation that a scan is performed (new data from the host is the usual reason a scan is performed), then the operation can trap the error

pattern. If the pattern is already present before an operation starts, it would not be recognized.

- **What about tables and error entities?**

If you are using procedures, you can add an error entity to your procedure. Defining error entities is not necessary if you've already defined them in your operations.

 **Note**

If you insert error entities into your procedures, you will not be able to use the custom error messaging option available when defining error entities within your operation. Instead, you have to define a custom error message within the Procedure Editor. Additionally, the error message feature within the Procedure Editor does not allow you to assign the contents of an attribute to the error message.

- **What about tables and error patterns?**

If you are using tables and error patterns, you don't have to modify anything within your procedures.

- **How can I fix the trailing spaces problem?**

If you have a procedure that requires that you send data to a numeric field (format: #####.##) and enter the value 6, for example, you may find that the application will not

accept your input. Running a trace reveals that instead of sending just "6", Host Integrator is sending "6" followed by 7 spaces. The spaces are wiping out the 6 and the result is an error. On the Attribute Properties tab, clear the Erase to end of attribute check box associated with writes.

4.15.2 Model Examples

Several model examples, along with readme instructions, are included in the Development Kit installation and are located in your `<VHI install directory>\examples\ModelSamples` folder. Each model emulates a different host application to provide an interactive example of how the Design Tool is used to create a model. To simulate this emulation, the Design Tool is used in conjunction with the Host Emulator. Before viewing a model example, make sure that the Design Tool is open and the Host Emulator is started.

The example models demonstrate strategies for modeling block mode 3270 applications and character mode VT applications. Some particularly important or difficult areas of these models are discussed to assist you in discovering solutions that will work for your host application. While each host application is unique, these techniques will help you avoid the most common problems. Look through all the examples for ideas about what you might want to use in your model.

You can also review examples for individual methods in the Visual Basic Methods Reference online help.

The following host application models are included in your installation and, with the exception of pine, can be used with the Host Emulator:

[CCSDemo](#) - An example of multi-column recordsets and accessing computer data.

[CICSAccts](#) - An example of operation conditions, recordsets, and compound procedures.

[SIDemo](#) - An example of recordset selection and insertion.

[Purchases](#) - An example of recordset implementation in a model.

[Pine](#) - An example of a VT terminal host application.

Note

If you see the message "Host refused to connect...", first confirm that the Host Emulator is running (this service is started automatically with the Developer Kit). You may have a port conflict with another application. You can try to close other applications that may be using that port. Otherwise, you can change the port number for model. This approach requires that you also change the port number in the Host Emulator.

Using CCSDemo

Provides a good example of multicolumn recordsets and accessing customer data.

The CCSDemo model is an example of a 3270 host application model. To access data using CCSDemo, use the following case sensitive search criteria:

UserID: bjones

Password: bjones

Last name: smith

First initial: c

State: ri

Account number for Carla Smith: 167439459

To access Carla Smith's customer profile in CCSDemo, follow these steps:

Open CCSDemo.modelx from the File menu.

On the Connection menu, click Connect to localhost via Telnet.

Select CustomerInquiryPanel from the Entity box in the Entity window.

Type 2 in the Selection field.

Type the above account number in the Account field and press Enter to display the account number.

Navigate to the AcctTransactions entity to view a multicolumn recordset.

Using CICSAccts

The CICSAccts model is an example of a 3270 host application model.

- After opening CICSAccts.modelx in the Design Tool, select Main from the Entity box. On the Main entity, view operation conditions such as preconditions, required attributes, and error patterns. Click the Operation tab and select an operation from the Name box to view details on any of the pre-configured operations. The Main entity is also a good example of demonstrating the difference between operations defined for use in tables and operations defined for direct model access.
- Navigate to the NameSearchResults entity to view how to define a recordset for read-only data. You may notice that even though Main and NameSearchResults are very similar screens, they have been designed separately. This is a good design strategy to prevent the recordset from having to be generated as a step within the Main entity. Differentiating these two screens makes use of the Screen properties not present check box for Pattern_2 on the Main entity.
- On the Model menu, click Tables to view the Tables dialog box. The Accounts table contains a good example of a compound procedure named CompoundNameSearch. Using compound procedures can be a good mechanism to provide access to more complete data sets that would otherwise be available in a single call. In this case, fetching details about all accounts that start with "w" would require two separate procedure requests (NameSearch followed by GetAccount). By connecting the two procedures, retrieving this information has been reduced to a single compound procedure.

To access data using CICSAccts, use the following case sensitive search criteria:

Navigate to the RecordAdd entity.

2. Click the Operation tab and select the FillWithValidData operation from the Name box. The attribute values to use are:

LastName: WILSON

FirstName: MARTIN

MiddleInitial: B

Title: MR

Phone: 9876543210

Address1: 1234 56TH ST.

Address2: HOUSTON, TX 98765

NumCardsIssued: 1

DayIssued: 01

MonthIssued: 01

YearIssued: 2001

Reason: N

CardCode: 1

ApprovedBy: 1

3. Navigate back to Main entity.

To do a name search, use "W" or "K" for the last name. "W" will provide you two screens to fetch, and "K" will fetch one screen.

Using SIDemo

The SIDemo model is an example of a 3270 host application model. This model can simulate the selection of multiple records in a recordset and the insertion of a new record into a recordset. To view examples of testing recordset selection and insertion, open the Test Recordset dialog box.

Using Purchases

The Purchases model is an example of a 3270 host application model. This model can simulate the updating of a current record in a recordset. Review an [example using Purchases](#). To view other examples of testing recordsets, open the Recordset Test dialog box.

Using Pine

The Pine model is an example of a VT host application model but includes several features that could also be applied to a 3270 host application model.

To open the pine model:

On the File menu, click Open and select pine.modelx from the Models\pine folder.

On the Connection menu, click Offline mode.

The Pine Model demonstrates the Design Tool capabilities discussed below. When you work with VT models, it is critical that you take host synchronization into account.

CURSOR MOVEMENT

This model demonstrates the Design Tool's ability to override the cursor movement of a character mode host. By default, the Host Integrator calculates and issues the correct number of arrow key commands to change the cursor position, but many character mode hosts do not respond to this behavior. In this version of Pine, issuing arrow keys in the default manner did move the cursor, but the peculiarities of the host application caused the cursor to be moved to the wrong location. The up and down arrows are actually sufficient to change fields, so the model designer who created pine decided to override the default mechanism. See the Cursor tab on the AddAddress and UpdateAddress entities.

Another common issue with character mode hosts is tabstops. For example, issuing a move cursor command may cause the host to move the cursor first to a status line to update the text, and then to its desired location. By default, Host Integrator will issue a move cursor command and then wait for the cursor to change position. If the scanning mechanism sees the cursor in the status line, Host Integrator will assume the status line location is the new position and issue another move command in an attempt to reach the desired location. This is a good example of how the Host Integrator might get ahead of the host. For a screen defined as an entity, you can get around this by defining tabstops. These constitute all the valid "at rest" positions for the cursor on that screen; therefore, if tabstops are defined, Host Integrator will wait for the cursor to appear in one of these locations after issuing a move cursor command and before evaluating whether the desired location has been reached. The Pine model uses the location of writable attributes as tabstops.

COMMAND LISTS

On the Model menu, click Properties to open the Model Properties dialog box to view the login and logout command lists created for Pine model. Command lists can help bypass the many host prompts encountered in the login sequence before actually getting to the host application. The command list is generic for any user based on WriteVarToTerminal. An API connector can set model variables with the correct userid and password at session connect, and those values would in turn be used by the login command list using the WriteVarToTerminal command.

ADDRESSBOOK ENTITY AND HOST SYNCHRONIZATION

The AddressBook entity demonstrates some techniques, such as including scrolling operations and setting termination conditions, to keep recordsets synchronized with the host. You can create scrolling operations that use the TransmitTerminalKey command assigned to the keystroke from which you expect to elicit a host response and then make Host Integrator wait for the host to respond with something other than a character echo. You can also verify the host response with one or more Host Events commands. For examples of scrolling operations, see the LineUp, LineDown, PageUp and PageDown operations defined on the AddressBook entity. Failure to wait would result in API calls, such as SelectRecordByFilter, executing the selection operation while the host is still trying to highlight the desired record, which may cause fetches or updates to access the wrong entry. Search for "SelectRecordByFilter" in the Host Integrator API Reference for more information.

In addition, the termination conditions for filtering the last screen allow Host Integrator to accurately detect the end of a recordset. By limiting it to non-blank records, Host Integrator correctly identifies the last record on the screen and remains synchronized with the host. Omitting this property causes Host Integrator to treat all entries as records. After a fetch call, Host Integrator assumes the host is at the bottom position of the screen instead of at the last non-blank record. Keeping the current record in line with the host is critical for character mode applications or any other recordsets that require line up and line down operations.

4.15.3 Settings

Settings Summary Overview

These settings, along with their descriptions, are available from the View Settings dialog box. The View Settings dialog box collects all settings from all Setup dialogs boxes, including:

- Settings that record the way that you have set up your screen
- Settings that can only be set from the View Settings dialog box (except by loading a settings file or executing a Design Tool command)

HOST INTEGRATOR SETTINGS

Emulation

Entity Design

Host Communication

Miscellaneous

Events

Setup

WCP Communication

Emulation

3270 TERMINAL

- **3270 Numeric Input Field Character set** Controls the set of characters that is allowed for 3270 numeric input fields. This setting handles non-standard 3270 applications that require non-numeric characters to be entered into numeric input fields. The default is Numeric and uppercase characters. Options include:

Numeric and uppercase characters

All characters

Numeric characters only

- **Country Extended Graphics Code** When this check box is selected, additional characters are available in the configured National character set. See your host documentation for details. By default the check box is not selected.

Delay After AID Specifies the amount of time Host Integrator should wait before processing keystrokes after an AID key (PF1-PF24, Enter, or Clear) has been pressed. The range of values is from 0 to 20,000 milliseconds. The default is 0.

Insert Protocol for 3270 You can specify what Host Integrator should do if you attempt to insert a character in a 3270 terminal session. The logic Host Integrator applies can be over a single field, multiple local fields, or all unprotected fields. The values are:

First Null (default) - Make room for the character being inserted by moving all characters to the right of the insertion point one character to the right until a null is encountered. The null is replaced by a character and all subsequent characters are unchanged. If no null is found, the insertion fails.

First Null or Trailing Space - Uses the same logic as First Null, except that a trailing space can be used if no null is found.

First Null or Trailing Character - Replaces the last character in the insert arena on an insert if neither a null nor a trailing space is found.

Preserve 3270 Terminal Insert State Specifies whether pressing Enter in 3270 terminal sessions resets the insert mode. When set to Yes, the insert state of the terminal is not reset when the Enter key is pressed. The default is No.

Preserve 3270 Partitions Specifies whether to send the partition-related information from the terminal to the host in response to a host query request. This setting is relevant for host

applications that are not designed to handle terminals that support partitions. You cannot change this setting during a connection.

Yes - the terminal includes partition-related information in its response to a host query request.

No (default) - the terminal does not include partition-related information in its response to a host query request.

5250 TERMINAL

5250 Status Line Type You can configure Host Integrator to display any of three different status lines at the bottom of the terminal window when connected to an AS/400:

3488 Status Line (default) - The 3488 status line uses symbols to represent various conditions and is based on the status line you see on newer 5250 terminals from IBM.

5250 Status Line - This status line uses character pairs to represent various conditions—the characters are always shown but appear in inverse video when the condition is true. For example, when the system is not available, the letters SA appear in regular video, but when the system is available, the letters appear in inverse video.

Debug Status Line - If you contact technical support, you may be asked to use the Debug Status Line to help diagnose the problem. The Debug Status Line is of use to users with an intimate knowledge of the 5250 datastream.

Aid Field Exit Specifies whether unrestricted sending of aid key (F1-F24 only) values to the host from restricted input fields can be sent. By default this checkbox is not selected; you cannot send aid key values to the host from restricted input fields. This is standard terminal behavior. Select this check box to send aid key values to the host from a restricted input field.

Cursor Progression Indicator Specifies how Host Integrator responds when the host queries to determine if End User Interface (EUI) enhancements are supported by the terminal. Host Integrator only supports the cursor progression enhancement. Set this value to Yes only if you are using a host application that uses cursor progression and queries to determine if EUI is supported. Do not change the value of this setting to Yes unless you are sure that the host uses only the cursor progression feature.

Ignore EUI Command Error This setting is used for testing purposes only.

Keyboard Error Alarm Specifies whether an alarm (a beep) is sounded when a keyboard error is detected. By default, this check box is not selected.

TN APPN Gateway Specifies whether the third PC Support header is added to the Telnet pass through header for a save screen command and removed for a restore screen command. This setting is valid for 5250 connections over Telnet to an Apertus gateway only.

Word Wrap Specifies whether text is divided in mid word at the end of the current line or wrapped to the next available line (or row) of the same field, and whether the host controls this behavior. The values are:

Host Word Wrap - text is wrapped to the next available line as controlled by the host.

Local Word Wrap - text is wrapped to the next available line as controlled by Host Integrator.

No Word Wrap - text is not wrapped to the next available line, instead it is divided at the end of the current line.

DEC/UNIX TERMINAL

Telnet VT Terminal ID See the Advanced VT Telnet topic in the online help for more information. This setting determines the following terminal characteristics:

Which screen control sequences the host sends to Host Integrator to format the screen.

The position of the cursor.

What characters to display in a host application.

VT Active Display Specifies which part of the VT Terminal window is the currently active area. The options are *Status Line Active Display* and *Main Screen Active Display*. This setting is read-only.

HP TERMINAL

HP Auto Linefeed/Auto Linefeed Default - Specifies whether Host Integrator appends a line feed character (LF) to each transmitted or received carriage return character (CR). Most hosts echo both a carriage return and a line feed when you press Enter, so in remote mode it isn't usually appropriate to add another line feed. In local mode, however, a carriage return does not execute a line feed automatically, so you may want to set AutoLinefeed to Yes. This setting reflects the current terminal state of HP Auto Linefeed, which can be changed by the host. The associated default setting, saved with the model, is HP Auto Linefeed Default.

HP Block Terminator/Block Terminator Default - Under certain conditions, the Host Integrator transmits a block terminator character at the end of each block of data transmitted. The value selected here specifies which ASCII character is sent to indicate that the end of the block has been reached. The values are ASCII decimal 0-127. The default is RS (ASCII decimal 30). The associated default setting, saved with the model, is HP Block Terminator Default.

HP Block Transfer Unit/Block Transfer Unit Default - When operating in block mode, a block of one or more characters is transmitted when you press Enter or when the host requests a block transfer from terminal memory. This option determines how much data Host Integrator transmits on each block transfer. When set to Line, data is transmitted one line at a time, or one field at a time in format mode. When set to Page, data is transmitted one page at a time. The associated default setting, saved with the model, is HP Block Transfer Unit Default.

Display Memory Lines - Specified the number of lines allocated for HP display memory. The default is 144. It is recommended that you do not change this setting.

Display Memory Response - Specifies the amount of memory to be reported to the host computer as part of a primary status response. Some host software may be able to use Host Integrator's extended display memory if you set this field to a value greater than 4K. The default is HP Display Response Maximum.

Enq/Ack (Pacing) Default - Some HP 3000 and 1000 computers use a form of handshaking called Enq/Ack (Enquire/Acknowledge) to prevent the terminal (or in this case, the Host Integrator) from falling too far behind the host system and losing data. With Enq/Ack pacing, the host system sends 80 characters followed by an ASCII Enq character and stops transmitting. When the Host Integrator has processed all of the characters preceding the Enq, it sends an ASCII Ack character, which tells the host it is ready for more data. This setting reflects the current terminal state of HP Enq/Ack, which can be changed by the host. The associated default setting, saved with the model, is HP Enq/Ack Default.

Field Separator (Default) - When transmitting in block, page, and format modes, Host Integrator sends a field separator character after each field of the formatted screen except the last one. The value selected here specifies which ASCII character is sent. The values are ASCII decimal 0-127. The default is US (ASCII decimal 31). This setting reflects the current terminal state of HP Field Separator, which can be changed by the host. The associated default setting, saved with the model, is HP Field Separator Default.

Host Character Set - The available host character sets for HP terminals are HP Roman 8 and HP Roman 9. The default is HP Roman 8.

Host Prompt Character - The values are ASCII decimal 0-127. The default is D1 (ASCII decimal 17). An HP 3000 sends a DC1 character to indicate that it is ready to accept a line or block of characters. This character is sent immediately after the MPE colon prompt is sent. This list allows you to change which character is expected. Most hosts either use the DC1 (^Q) character or no prompt (that shows up simply as a space). Select the appropriate host prompt from this list (turn on DISPLAY FUNCTIONS to see the control codes sent by the host before changing this value).

Inhibit DC2 (Default) - This check box, along with Inhibit Handshake and some other factors, determines the type of handshaking that precedes each block transfer of data from the Host Integrator to the host system. When selected, the DC2 handshake for block transfers is inhibited. This setting reflects the current terminal state of HP Inhibit DC2, which can be changed by the host. The associated default setting, saved with the model, is HP Inhibit DC2 Default.

Inhibit EOL Wrap (Default) - When not selected, the Host Integrator automatically returns the cursor to the left margin in the next line when the cursor reaches either the right margin or the right screen edge. When selected, the cursor is not automatically advanced when you reach the right margin. As you type additional characters, each one overwrites the character at the right margin until you explicitly move the cursor by pressing Return or using an arrow key. This

setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Inhibit EOL Wrap Default.

Inhibit Handshake (Default) - Determines the type of handshaking that precedes each block transfer of data from the Host Integrator to the host system. When selected, the DC1 handshake for block transfers is inhibited. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Inhibit Handshake Default.

Local Echo (Default) - When selected, each character typed at the keyboard is immediately displayed on the screen. In remote mode, each character you type is transmitted to the host computer. Most host systems immediately send the character back to your terminal (that is, they echo the character) and the character you see is the one that the host sent, not the one you typed. If you are communicating with a host computer that echoes characters, and this box is selected, each character you type appears on your display twice. If this happens, clear this check box. Select it only when communicating with host systems that do not echo each typed character (some public networks, for example). When the Host Integrator is in block mode or local mode, this check box is automatically selected. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Local Echo Default.

Modify All (Default) - MODIFY ALL is F2 on the modes keys (to display the modes keys, press Alt+M or select Modes from the Function key set list on the Function Keys tab in the Terminal Setup dialog box). When Host Integrator is in character/remote mode, MODIFY ALL lets you change text on the screen and retransmit the modified line to the host. This can save a lot of retyping. MODIFY ALL remains active until disabled by pressing F2 to remove the asterisk from the key label. Line Modify is disabled after you press Enter while Modify All stays active until you disable it by pressing F2. The default setting is saved as part of the model's default settings, and will be used when initially connecting or resetting a session. Modify All, reflects the current terminal state, which can be changed by the host.

National Replacement Set - Some host systems use national replacement characters to encode characters that are not available in the ASCII 7-bit character set. If necessary, set this list to match the set used by your host. In 8-bit operation, the value in this list has no effect. In 7-bit operation, however, the value assigned here limits characters to those that are defined for the configured set, and determines the replacement characters that are used during data communications.

Remote Mode (Default) - Specifies whether Host Integrator transmits each number, letter, or special character typed at the keyboard to the host. As characters are received from the host, Host Integrator displays them on your screen. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is Remote Mode Default.

Return Definition (Default) - Type a character in these boxes to be generated whenever Return is pressed. If the second character is a space, only the first character is generated. The values are ASCII decimal 0-127. The default is CR (ASCII decimal 13). This setting reflects the current

terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Return Definition Default.

SPOW Enabled (Default) - Ordinarily, the Spacebar overwrites and erases existing characters. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP SPOW Default. When the SPOW (SPace OverWrite) check box is selected, spaces entered from the keyboard (not spaces echoed from the host), move the cursor over existing characters, but do not overwrite them with spaces:

The SPOW latch is turned on by a carriage return.

The SPOW latch is turned off by a linefeed, tab, or home up.

Start Column (Default) - For every line in display memory, the Host Integrator attempts to remember the leftmost column that was entered from the keyboard, as opposed to that received from datacomm. This way, the Host Integrator can distinguish the host prompt portion of each line from the user-entered portion. This information is used when you enable LINE MODIFY or MODIFY ALL to determine the leftmost column that should be transmitted to the host when you press Enter or Return. Under some circumstances, it is impossible for the Host Integrator to tell which column was the first user-keyed column; when that happens, it uses the value you enter in this box to determine the leftmost column to be transmitted. When Display Columns are set to 80, enter a value from 0 to 79. When you're in 132-column mode, enter a value from 0 to 131.

Terminal ID Response - This setting specifies Host Integrator's response to a terminal ID status request from the host computer.

Terminal Type - Specifies which terminal the Host Integrator is to emulate.

Transmit Functions (Default) - Most of the keys on the keyboard have an associated ASCII character. Several keys, however, perform functions for which there is no character defined; for example, Home and PgUp. Certain host software programs need to be informed when you press one of these non-ASCII keys. Selecting this option signals the Host Integrator to inform the host system whenever you press one of these keys. When this check box is selected and the Design Tool is operating in character/remote mode, each time you press one of these keys the associated escape sequence is transmitted to the host. Most applications that require this feature automatically send the escape sequences to enable and disable the feature, so you probably will never need to enable it manually. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is Transmit Functions Default.

Use Host Prompt - Clear this check box if you want the Host Integrator to ignore the host prompt. Clearing this setting has the same effect as selecting the Inhibit Handshake and Inhibit DC2 check boxes. Ignoring the host prompt forces the Design Tool to behave as though both inhibits are on, thus preventing handshaking. Over an X.25 network, this prevents communications problems caused by applications that use handshaking. When cleared, the Design Tool always responds to a primary status request from the host that both Inhibit

handshake and Inhibit DC2 are enabled. This can affect a host application that explicitly changes one of these inhibits.

Warning Bell - Selecting this check box causes a bell to sound when the ASCII bell character (Bel, decimal 7) is received from the host or entered from the keyboard.

Show Function Keys - Select this check box to display the function key labels across the bottom of the Terminal window. Clear this check box if you want to hide the function key labels.

HP Margins/Tabs/Columns

Tab Definition - This setting specifies the character that is sent to the host when you press the Tab key in an HP terminal session. The options are Tab Equals Return and Tab Equals Tab. The default is Tab Equals Tab.

Tab Equals Spaces (Default) - Usually, pressing the Tab key sends a tab character to the host. Some HP 3000 applications, however, do not interpret the tab character correctly. This setting specifies whether pressing the Tab key sends the number of spaces needed to move the cursor forward to the next tab stop, rather than sending the tab character. In character/remote mode, the spaces are transmitted to the host computer. If there are no tab stops to the right of the cursor position when you press Tab, the cursor does not move, and no spaces are generated. This setting reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Tab Equals Spaces Default.

Tab Stops (Default) - Use this setting to set and clear individual tab stops. The default setting is saved as part of the model's default settings, and will be used when initially connecting or resetting a session.

HP Display

Inverse Video - This setting specifies whether the foreground and background colors for screen attributes are reversed.

Screen Columns - This setting controls the width of the scrolling region in the Terminal window. The values are either 80 or 132 columns. The default is 80 columns.

HP Keyboard

Destructive Backspace - By default, pressing Backspace moves the cursor to the left without erasing characters. Select this check box to erase the character to the left of the cursor when you press Backspace.

Margin Bell - Clear this check box to disable the margin bell. When selected the Design Tool beeps when the cursor is eight characters from the right margin or when, in format mode, the last character of an unprotected field is entered from the keyboard. To disable the format bell, clear the Warning Bell checkbox.

Return Equals Enter - To use Return rather than Enter in block mode applications, set this setting to Yes. This may cause problems if your host system expects a Return and you have configured Return to act as Enter. Setting Return Equals Enter to No for PCs without enhanced

keyboards specifies that only the numeric keypad "+" key or the Shift+F10 keystroke can be used as Enter. Use keyboard mapping to move this function to another key. This setting

reflects the current terminal state, which can be changed by the host. The associated default setting, saved with the model, is HP Return Equals Enter Default.

HP Default User Keys 1 - 8

Specifies the configuration of the user-defined HP function keys 1 through 8.

Telnet HP Terminal ID

This setting determines the following terminal characteristics:

Which screen control sequences the host sends to Host Integrator to format the screen.

The position of the cursor.

What characters to display in a host application.

When Host Integrator connects to a Telnet host, the Telnet protocol negotiates a Telnet terminal ID with the host. In general, this negotiation results in the correct terminal type. However, if you are having trouble running a host application, the negotiation between Telnet and the host could be the issue.

To override Host Integrator's selection of a terminal, select a terminal ID from the list (because this list box is editable, you can just type in any value). If you enter a terminal ID that the host does not recognize, Host Integrator reverts to a list of default values until one is found that the host supports. The default terminal ID values are HP and HP2392A. If you enter a terminal ID string, it may be up to 40 characters long taken from a set of uppercase letters, digits, and the two punctuation characters, hyphen and slash. It must start with a letter and end with a letter or digit.

CURRENT CURSOR COLUMN

Returns the current cursor column location relative to the left edge of the Terminal window. This setting is read-only.

CURRENT CURSOR ROW

Returns the current cursor row location relative to the top edge of the Terminal window. This setting is read-only.

CURSOR OFFSET

Returns the current cursor offset location relative to the top edge of the Terminal window. This setting is read-only.

ENABLE HOST ALARM

Specifies whether alarms (beeps) sent by the host are sounded. By default, this is enabled.

KEYBOARD LOCKED

Indicates whether the keyboard is currently locked. This setting is read-only.

SESSION TYPE

Specifies the type of session to configure. The options are:

IBM 3270 terminal

IBM 5250 terminal

VT terminal

HP terminal

TERMINAL MODEL

Specifies the terminal (also known as a display station) you want the Design Tool to emulate.

Entity Design

PATTERNS

Generate Patterns Automatically

Specifies whether or not the Design Tool creates patterns automatically as you add entities to your host application model. Patterns can be generated automatically for block mode applications only.

- **Generate Patterns Based On**

Specifies the physical characteristics of the pattern to be recognized during auto-generation. You can generate patterns based on field type and text color. Patterns can be generated automatically for block mode applications only. See Pattern Preferences.

- **Position Type for Patterns**

Defines the settings (Row, Height, Col, and Width) that describe the actual position of the pattern on the host screen. Row can be defined with the row number or you can select from one of the following options: Any Row, Current Row, First Row, or Last Row. Col can be defined with the column number or you can select from one of the following options: Any Column, Current Column, First Column, or Last Column.

Select either the Fixed or the Relative to cursor option. The default is Fixed and the coordinates of the pattern are automatically generated and can be viewed on the Pattern tab if pattern search is not defined relative to the cursor or in an expanded region.

When defining a pattern, set the Row box to Current Row then configure the options in the Properties box. After you've configured the pattern properties, set the Row box to Any Row to ensure accuracy in your pattern configuration.

- **Use Blink for Patterns**

Specifies that the text added as a pattern blinks on the terminal screen. The default is No. To change the color of a display attribute in the Design Tool, use the Display Setup dialog box on the Settings menu. This setting applies to VT and HP hosts only.

- **Use Brightness for Patterns**

Specifies the brightness level of a static area added as a pattern on the terminal screen. Brightness can be normal (most screens and text) or half (which makes the characters appear dimmer). The default is No. This setting applies to HP and VT hosts only.

- **Use Case Sensitivity for Patterns**

Specifies whether or not text, either typed or selected from the terminal screen, in a selected pattern is case sensitive. The default is Yes.

- **Use Display Attributes for Patterns**

Specifies what types of physical characteristics to look for on a host screen when adding patterns to entities. By default, the *Use all display attributes that are uniform across a selected area* option is selected. This means that the Design Tool will allow any display characteristics that are similar to be added as patterns on entities. The other option, designed mainly for VT and HP hosts, is *Use only specified display attributes*.

- **Use Field Types for Patterns**

Specifies how the host has set a terminal display field attribute on the terminal screen. For example, the host may recognize a text field as an "unprotected" field while areas that

cannot be edited by a user are "protected" fields. The default is No. This setting applies to IBM 3270 and 5250 hosts only.

- **Use Text Color for Patterns**

Specifies the terminal display attribute of the text color that appears on a pattern. The default is No. This setting applies to IBM 3270 and 5250 hosts only.

- **Use Text Color for Patterns**

Specifies that text characters, either typed into the Text box on the Pattern tab or selected from the terminal screen, be included when adding patterns to entities. The default is Yes.

- **Use Underscore for Patterns**

Specifies whether or not characters in a pattern appear to be underlined. The default is No. To change the color of a display attribute in the Design Tool, click Display on the Settings menu to open the Display Setup dialog box. This setting applies to VT and HP hosts only.

- **Use Video for Patterns**

Specifies the video type of a pattern on the terminal screen. There are two types of video: Normal (often white characters on a black background) and Reverse (black characters on a white background). The default is No. This setting applies to VT and HP hosts only.

ATTRIBUTES

• **Character Mode Data Entry for Attributes**

If you are working on a character mode host, such as HP or VT, direct the Host Integrator to treat an attribute in any one of the following ways:

- *Don't wait for echo (default)* - Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- *Wait for next tabstop* - Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- *Wait for next tabstop or string to echo at cursor* - Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- *Wait for same number of characters to echo* - Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- *Wait for string to echo at cursor* - Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.

• **Erase to End of Attribute for Attributes**

Specifies whether or not the Host Integrator is instructed to concatenate as many spaces as necessary to make the length of the input match the length of the attribute or field. The default is No. When writing an attribute value to an entity, there may or may not be a prefilled value from the host. If there is a prefilled value, just writing text to the screen will overwrite up to the length of the new data. If the old data is longer, the last few characters will be left behind, forming an odd input that was not intended. If this option is enabled, the Host

Integrator fills in the remainder of the host field with spaces, thus "erasing" the left over prefilled value.

- **Generate Attributes Automatically**

Specifies whether or not the Design Tool creates attributes automatically as you add entities to your host application model. The default is No. On the Attribute tab in the Preferences Setup dialog box, select the *No automatic generation* option.

- **Generate Attributes Based On**

Specifies what types of characteristics the Design Tool should look for when generating attributes automatically. On the Attribute tab of the Preferences Setup dialog box, select Every Unprotected field when the Generate based on option is selected. Attributes can be generated automatically for block mode applications only. The following options are available: - Every unprotected field (Default) - Every field - Every modified field

- **Name Type for Attributes**

Specifies the name or names of the currently defined attributes. See Attribute Preferences.

- **Position Type for Attributes**

Specifies the actual position of the attribute on the host screen. Choose from the options under Default position on the Attribute tab of the Preferences Setup dialog box. See Attribute Preferences.

- **Remove Leading Blanks for Attributes**

Gives you the ability to decide how the Host Integrator processes data after retrieving it from the terminal by allowing the you to remove blanks that come before the fetched data. The default is No.

- **Remove Trailing Blanks for Attributes** Gives you the ability to decide how the Host Integrator processes data after retrieving it from the terminal by allowing you to remove blanks that come after the fetched data. The default is No.

OPERATIONS

- **Apply Default Value to Server**

Specifies whether or not to apply the default value provided for the attribute on the Host Integrator Server and the Design Tool or on the Design Tool only. The default in the View Settings dialog box is Default value for Design Tool only. Additionally, to apply the default value to the server, select this option in the Operation Generation Preferences dialog box.

- **Generate Operations Automatically**

Specifies that operations are generated by the Design Tool automatically as you traverse between entities. The default is Yes. On the Operation tab of the Setup Preferences dialog

box, select the Automatically generate operations option. You can then choose Prompt me to approve each operation or Add operations silently.

- **Include CheckOperationCondition Command**

Specifies whether or not a CheckOperationConditions command is added to an operation being generated by the Design Tool. The default is Yes. In the Operation Generation Preferences dialog box, select the Include CheckOperationCondition command check box.

- **Prompt User When Generating Operations**

Specifies whether or not the Design Tool will prompt you with the Confirm Operation dialog box before automatically adding a new operation to the model. The default is Yes.

- **Record Attribute Contents with Command**

Specifies the command to use when recording attributes during operation generation. The default is Record attribute contents with DefaultValue command. The other option is to record attributes using the TransmitToAttr command.

- **Record Initial Cursor Position**

Specifies whether or not to record the first position of the cursor on the terminal screen during operation generation. The default is No. In the Operation Generation Preferences dialog box, select the Record initial cursor position check box. This setting is used with character mode applications.

- **Record the Contents of**

Specifies what type of records to record when generating operations automatically. The default is to record Modified Fields Only. On the Operation Generation Preferences dialog box, the default is Ignore unmodified fields.

- **Use Timeout for Commands**

Specify in seconds how long a command should take to execute before an error message is received. You can set the time out under Default properties in the Command tab of the Preferences Setup dialog box. The default is 5 seconds.

- **Use Timeout for Operations**

Specify in seconds how long a given operation should take to execute before you receive as error message. Under Default properties in the Operation tab of the Preferences Setup dialog box, the default timeout is 10 seconds.

By default, connector API methods timeout after waiting 30 seconds for the Host Integrator Server to respond. If the operation timeout exceeds 30 seconds, you'll see the error report "Timed out while waiting for data." You can use SetMethodTimeout on the connector API to

increase that default. Search for "SetMethodTimeout" in the Host Integrator API Reference for more information.

- **When Field Text is Hidden**

Specifies how the Design Tool should handle hidden field text during recording. On the Recording Preferences dialog box, the default is Record encrypted text. If you select Record clear text, the contents of the field will be recorded.

- **When the Field is Empty**

Specifies how to deal with an empty field that is recognized during operation generation. The default is Skip Recording. In the Operation Generation Preferences dialog box, the default is Ignore empty fields.

RECORDSETS

- **Cache All Field Writes**

Creates a temporary storage area for data assigned to recordset fields. To globally configure all recordset field data to be cached on new entities, select the Cache all field writes box on the Recordset Preferences dialog box. To cache recordset field data on a selected entity, select the entity in the Entity window and select the Cache all field writes box on the Recordset Fields tab.

When data is assigned to a field from within the Design Tool or from a Host Integrator API, it will be stored in this area rather than the default behavior of being written directly to the terminal. In an operation, field data can be assigned using write commands like `DefaultValue` and `TransmitToField`. When using a Host Integrator connector, field data is assigned with methods like `UpdateRecords` or `UpdateRecordByFilter`. When Cache all field writes is selected, the field data assigned by the Design Tool or the Host Integrator connector is temporarily stored until an operation issues an `UpdateAttribute` or `UpdateAttributes` command to write the stored data to the terminal screen. Alternatively, an operation can call `UpdateAttribute` for each field in the desired order, inserting other commands in between as needed to modify cursor positioning, synchronize with the host state, or meet any other requirements to be executed by the operation.

By default, an `UpdateAttribute` command will be inserted in each generated operation when Cache all field writes is selected.

By default, an `UpdateAttribute` command will be inserted in each generated operation when Cache all field writes is selected.

This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When field data assignments are arbitrarily mixed

between an operation and a Host Integrator connector API, the order in which data is actually written to the terminal can be important to the host application.

- **Character Mode Data Entry for Recordset Fields**

If you are working on a character mode host, such as HP or VT, direct the Host Integrator to treat a recordset field in any one of the following ways:

- Don't wait for echo (default)

Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.

- Wait for next tabstop

Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.

- Wait for next tabstop or string to echo at cursor

Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.

- Wait for same number of characters to echo

Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.

- Wait for string to echo at cursor

Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.

- **Erase to End of Field for Recordset Fields**

Specifies whether or not the Host Integrator is instructed to concatenate as many spaces as necessary to make the length of the input match the length of the attribute. The default is No.

- **Recordset Bottom Position**

The following settings describe the bottom position of the recordset on the entity:

- Fixed row - Select this option to specify the end row position of the selected recordset. By default, the Design Tool calculates the Row coordinate for you. This is the default setting for all sessions.
- Relative to top - Select this option to specify the end position for the selected recordset based on the recordset top position. The Design Tool calculates the Row coordinate for you. The position coordinates are set based on recordset screen region and position type.
- Relative to pattern - Select this option to choose an end position relative to a pattern created on the Pattern tab. Select the pattern from the list and the Design Tool will calculate the Row coordinate for you.

- **Recordset Record**

Specifies the size of the each recordset. The default is Fixed. Select this option if your recordset takes up a set number of rows. Select Variable to specify the beginning and end of a recordset on an entity. Use the Recordset Layout tab to define specific coordinates.

- **Recordset Top Position**

The following settings describe the top position of the recordset on the host screen:

- Fixed row - Select this option to specify the start position of the selected recordset. By default, the Design Tool calculates the Row and Col coordinates for you. This is the default setting for all sessions.
- Relative to cursor - Select this option to specify the start position for the selected recordset based on the position of the cursor. By default, the Design Tool calculates the Row and Col coordinates for you. The position coordinates are based on recordset screen region and position type. The recordset position will be set at the desired offset from the cursor's position when the entity is recognized and validated.
- Relative to pattern - Select this option to choose a start position relative to a pattern created on the Pattern tab. Select the pattern from the list and the Design Tool will calculate the Row and Col coordinates for you.

- **Remove Leading Blanks for Recordset Fields**

Specifies whether or not the Host Integrator removes blank spaces that come at the beginning of the fetched terminal data. The default is No.

- **Remove Trailing Blanks for Recordset Fields**

Specifies whether or not the Host Integrator removes blank spaces that come at the end of the fetched terminal data. The default is No.

OTHER ENTITY DESIGN SETTINGS

- **Automatically Create New Entity**

Specifies whether or not the Design Tool creates an entity automatically as it encounters undefined screens while navigating through a host application. The default is No.

- **Automatically Generate Tabstops**

Specifies whether or not the Design Tool creates tabstops automatically as you add entities to your host application model. This setting is designed specifically for character mode applications.

- **Cache All Attribute Writes**

Creates a temporary storage area for data assigned to attributes. To globally configure all attribute data to be cached on new entities, select the Cache all attribute writes box on the Advanced Entity Properties dialog box. To cache attribute data on a selected entity, select the entity in the Entity window and select the Cache all field writes box on the General tab of Advanced Entity Properties dialog box.

When data is assigned to an attribute from within the Design Tool or from a Host Integrator connector, it will be stored in this area rather than the default behavior of being written directly to the terminal. In an operation, attribute data can be assigned using write commands like `DefaultValue` and `TransmitToAttr`. When using a Host Integrator connector, attribute data is assigned with methods like `SetAttributes`. When Cache all attribute writes is selected, the attribute data assigned by the Design Tool or the Host Integrator connector is temporarily stored until an operation issues an `UpdateAttribute` or `UpdateAttributes` command to write the stored data to the terminal screen. Alternatively, an operation can call `UpdateAttribute` for each attribute in the desired order, inserting other commands in between as needed to modify cursor positioning, synchronize with the host state, or meet any other requirements to be executed by the operation.

By default, an `UpdateAttribute` command will be inserted in each generated operation when Cache all attribute writes is selected.

This is a solution to aid character mode applications in managing the order of screen updates. When attribute data assignments are arbitrarily mixed between an operation and a

Host Integrator connector API, the order in which data is actually written to the terminal can be important to the host application.

- **Echoed Character Timeout**

Specifies in seconds how long an echoed character should take to execute before an error message appears. The default is 5 seconds. This setting applies to character mode applications only.

- **Global Navigation Timeout**

Specify in seconds how long navigation between two entities should take to execute before an error message displays. Configuring a global navigation timeout is useful if a model gets into an infinite loop waiting for an event to occur on the host. The global navigation timeout monitors `SetCurrentEntity()` calls and `Navigate` commands in an operation. The detection of a global timeout will halt the navigation process. A well-designed model should never encounter a global navigation timeout error. The default is 30 seconds.

- **Keep Alive Timer Interval (in minutes)**

Specifies in minutes the amount of time before a keep alive operation is executed on the host. You can define a keep alive operation for any given entity in the Advanced Entity Properties dialog box.

- **MoveCursor Command List Timeout**

Specifies in seconds how long Host Integrator should wait for a move cursor command list to execute. If this setting is exceeded, an error is issued. The amount of time to wait is set under Settings Details on the View Settings dialog box. The default is 5 seconds. This setting is useful when connecting to a character mode host.

- **MoveCursor for Character Mode**

Specifies what mechanism to use to create a move cursor command list for a character mode host application. The default is Use arrow keys. To define a move cursor command

list, click Properties on the Model menu, and then click Advanced to open the Advanced Model Properties dialog box.

- **Record Cursor Position with Terminal Key Command**

Specifies whether or not to record a cursor position using a terminal key command. The default is No.

- **Use Writable Attributes as Automatic Tabstops**

Specifies whether or not to automatically include configured attributes as tabstop positions. The default is No. To configure this setting on the Cursor tab, select the Use writable attributes as automatic tabstops check box.

- **Wait for Cursor**

Specifies whether or not a command list will wait for the host to move the cursor to a specified location before an error appears. The default is No.

- **Wait for Host Writes**

Specifies whether or not a command list will wait for the host to send a response before validating its arrival at a destination. The default is No.

Host Communication

HOST COMMUNICATION TELNET

Assigned Device Name (setting)

Specifies which host device is assigned for your connection.

Assigned LU Name (setting)

Specifies which LU (session) is assigned for your connection. This setting is read-only.

Device Name

Specifies the Device name (also known as LU name) to which to connect. This box accepts up to 32 characters.

This option is only available in 3270 and 5250 sessions.

You cannot change this value while you are connected.

You can also map the Device Name setting to a model variable, which allows a different setting for each server runtime session. If a model variable is mapped to the Device Name setting, the Device Name configured in Session Setup will be used as a default unless changed by a session pool model variable list or a Host Integrator API SetModelVariable method.

Secure Host SSL Negotiated Cipher

Specifies the cipher negotiated for the host connection when Telnet SSL or Telnet Extended SSL is used as a transport. The value is based on text originating from the crypto library.

This is a read only setting.

To use FIPS 140-2 validated TLS encryption for SSL support, you must first define an environment variable, VHI_FIPS = 1. After this variable is set all SSL Telnet connections will use the FIPS 140-2 Crypto Libraries.

Telnet Attention

Specifies what happens when you press the following keys:

ATTN key (Ctrl-F1 by default) in a 3270 session

Ctrl-break in a VT or HP session

Telnet Average Keep Alive Roundtrip

Specifies the average amount of time Host Integrator has waited for a response to a Timing Mark Command.

This is a read-only setting.

The range of values is 0-999999 milliseconds, and the default value is 0.

Telnet Binary

Telnet defines a 7-bit data path between the host and the terminal (Host Integrator). This type of data path is not compatible with certain national character sets. Fortunately, many hosts allow for 8-bit data without zeroing the 8th bit, which resolves this problem.

In some cases, however, it may be necessary to force the host to use an 8-bit data path by selecting this check box.

Telnet Echo

This group box controls how the Design Tool responds to remote echo from a Telnet host:

Automatic (default): The Host Integrator Tool attempts to negotiate remote echo, but does what the host commands.

Yes: The Host Integrator negotiates local echo with the host, but always echoes.

No: The Host Integrator negotiates remote echo with the host, but does not echo.

Telnet Environment

This setting applies only to 5250 connections over Telnet.

When you establish a 5250 terminal session, specific information is sent to the AS/400 to configure your session. When you connect over Telnet, you can pass additional information at connect time as input to an Exit program on the AS/400. Use the Telnet Environment setting to send this additional information.

Enter a string of up to 260 characters, using the following format: `keyword=value;`
`keyword=value; keyword=value`

Using Telnet Environment with Auto SignOn

When Auto SignOn is set to **Yes**, Host Integrator automatically signs on to the host using the current values of the Username and Password settings. You can set Telnet Environment to specify additional, non-default sign-on options using any of the following keywords in the Exit program string:

Keyword	Sets	Equivalent SignOn Menu Option
IBMPROGRAM	Program to call	Program/procedure
IBMIMENU	Initial menu	Menu
IBMCURLIB	Current library	Current library

For example, the following statement specifies that when Host Integrator makes a Telnet connection using Auto SignOn, the host should display the menu called MAIN, run the program called QCL, and set QGPL as the current library: `Session.TelnetEnvironment =`

```
"IBMIMENU=MAIN;IBMPROGRAM=QCL;IBMCURLIB=QGPL"
```

Because this information is sent at connect time, you cannot change this setting while you are connected.

Initiate Option Negotiation

This option specifies whether certain connection options--including whether to always request a binary mode connection--should be negotiated when the Telnet connection is established.

Connections to some hosts on the Internet are expedited if this check box is cleared so that the terminal (meaning, the Design Tool) does not attempt to initiate negotiations for Telnet options.

Send Keep Alive Packets

In some cases, Host Integrator may become aware of Telnet communication problems only after a significant delay or when it attempts to send data to the host. This can cause problems if you enter a large amount of data on one screen or if you keep your connection open during periods of inactivity.

- To become aware of connection problems as they occur, you can configure your model to send Keep Alive packets. Four methods are available:

None. Host Integrator does not send any keep alive packets to the host. This is the default value.

Send NOP Packets. Host Integrator periodically sends a No Operation (NOP) command to the host. The gateway and host are not required to respond to these commands, but the TCP/IP stack can detect if there was a problem delivering the packet.

System. The TCP/IP stack keeps track of the host connection. This method requires fewer system resources than Send Timing Mark Packets or Send NOP Packets, but most TCP/IP stacks send Keep Alive packets infrequently.

- **Send Timing Mark Packets.** Host Integrator periodically sends a Timing Mark Command to the host to determine if the connection is still active. The gateway or host should respond to these commands. If a response is not received or there is an error sending the packet, it will shut down the connection.

To view the average amount of time waited for a response to a Timing Mark Command in the Design Tool, open the View Settings dialog box and select the Telnet Average Keep Alive Roundtrip setting.

Keep Alive Timeout

The **Keep Alive Timeout** is the interval between the keep alive requests sent by the Design Tool.

The range of values is 1-9999 seconds, and the default is 600 seconds.

Telnet Line Mode

The Host Integrator supports line mode and faux line mode for Telnet connections. Line mode allows Host Integrator to store characters in a buffer until a carriage return is entered, and then the characters are sent to the host in one packet (instead of sending each single character as an individual packet).

Line mode is useful for long network delays, and allows you to reduce costs on networks that charge on a per packet basis.

The Linemode list options: **Don't do line mode** (default), **RFC Compliant** (You need a host that supports line mode and it will get negotiated during connect), **During Local Echo** (Do line mode when the host tells Host Integrator to do the echoing), **When Not in SGA** (Do line mode when the host does not Suppress Go Ahead), **Local Echo no SGA, Always**

Note: All options other than RFC Compliant are known as "faux" line mode.

Telnet Location

Specifies to the host where the connection originated. Can also be used to provide informational messages to the host from the PC. Usage conventions vary by site.

You are not required to enter anything in this box.

This box accepts up to 260 characters.

You cannot change this value while you're connected to a gateway.

Telnet Ports

Specifies the host port or socket number that the Telnet session should use. Enter any number between 0 and 65,535 in this field.

You can configure **Port** in either the Session Setup or View Settings dialog boxes.

The default port number for Telnet is **23**.

Telnet Protocol

Specifies which Telnet protocol is being used to communicate with the host.

- This setting is read-only.
- The value of this setting can change during a session. The values are:

NVT: The Network Virtual Terminal protocol is in use.

TN3270: Standard TN3270 protocol is in use.

TN3270E: Extended TN3270 protocol (RFC 1647) is in use.

TN5250: Standard TN5250 protocol is in use.

Response Mode (Telnet Extended Option)

Specifies whether Verastream will support responses for Telnet Extended connections. The default is **Yes**. You should not need to change this setting.

If this setting is changed to No, Verastream will inform the host that it does not support responses when the connection is established.

The Telnet Extended option is configurable from the Session Setup or New Model dialog box. You cannot change this value while you are connected to a gateway.

Send LF after CR

A "true" Telnet host expects to see a CrNu (carriage return/null) character sequence to indicate the end of a line sent from a terminal (that is, the Design Tool).

Some hosts on the Internet are not true Telnet hosts, and they expect to see a Lf (linefeed) character following the Cr at the end of a line. If you're connecting to this type of Telnet host, check this box.

Set Host Window Size

This setting sends the number of rows and columns to the Telnet host whenever they change. This enables the Telnet host to properly control the cursor if the window size is changed.

Telnet Sys Request

Specifies what happens when you press the SYSREQ key (Alt+PrintScreen, by default). The definition of this key and its values vary by host application.

- This setting is valid for 3270 sessions over Telnet only. The values are:

Telnet Abort Output: Sends a TELNETABORTOUTPUT to the host.

Telnet Break: Sends a TELNETBREAK to the host.

Telnet Interrupt Process (default): Sends a TELNETINTPROCESS to the host.

Test Request Read: Sends a TESTREQUESTREAD to the host.

Telnet Terminal ID

This setting overrides the Model ID selected in Session Setup if the **Transport Type** is **Telnet**, and if the **Session Type** is **IBM 3270 Terminal**. It allows you to specify a Terminal Model that Host Integrator does not implicitly support (but might work with), such as Fujitsu.

For best results, specify one of the Terminal Models available in Session Setup and leave the **Terminal ID** box empty. When the **Terminal ID** box is empty, the Design Tool uses the Model ID defined in Session Setup by default. Only specify a Telnet **Terminal ID**, if after experimenting with other available Terminal **Models IDs** in Session Setup, you still cannot connect to the host.

If you must define a Telnet **Terminal ID**, specify a string you know is acceptable to the host. Otherwise, you may experience problems connecting to the host and emulation problems after connecting to the host. Typically, these types of problems occur if the host is not configured to recognize the terminal specified in the Telnet **Terminal ID** string.

Use Emulation Terminal ID

This setting is currently not used.

Telnet X System

- Specifies whether X SYSTEM is supported in 3270 sessions.

X SYSTEM is a keyboard-locked state that occurs when the host ends a transmission to the workstation without unlocking the keyboard. Some applications use this to notify you of a message written to the display, requiring you to reset the keyboard before you can type again. Other applications rely on SNA to unlock the keyboard. - The default is Yes.

TN APPN Gateway

Specifies whether the third PC Support header is added to the Telnet pass through header for a save screen command, and removed for a restore screen command.

This setting is only valid for 5250 connections over Telnet to an Apertus gateway.

The default is No.

SSH

SSH Compression Enabled

Change this setting to **Yes** to enable compression of all data.

Note: If you have a relatively fast connection to the host, compression can have a slightly negative impact on performance.

SSH Keep Alive Timer Interval

This setting specifies an interval, in seconds, for sending keep alive messages to the SSH server.

The default setting is 0, meaning that keep alive messages are disabled.

Change the value to reflect the number of seconds between messages. The maximum number is 86,400.

SSH Passphrase for the Private Key

If the key is protected by a passphrase, the client user is prompted to enter that passphrase to complete the connection using public key authentication. Public keys are not sensitive information and may be known to anybody, whereas the private key is protected very carefully by a strong passphrase.

This setting reflects whatever was set in the Design Tool or Model Variable Authentication dialog box; however, you can modify the value here.

Maximum length is 260 characters.

SSH Port

Specifies the host port or socket number that the SSH session should use. Enter any number between 0 and 65,535 in this field. You can configure **Port** in either the Session Setup or View Settings dialog boxes.

The default port number for SSH is **22**.

SSH Private Key File

SSH keys always come in pairs, one private and the other public. The **private key** is known only to you and it should be safely guarded. The public key, however, can be shared freely with any SSH server to which you would like to connect.

Although this setting reflects the value that was set in the Design Tool or Model Variable Authentication dialog box, you can modify it here.

Maximum length is 260 characters.

SSH Public Key Files

SSH keys always come in pairs, one private and the other public. The private key is known only to you and it should be safely guarded. The **public key**, however, can be shared freely with any SSH server to which you would like to connect.

Although this setting reflects the value that was set in the Design Tool or Model Variable Authentication dialog box, you can modify it here.

Maximum length is 260 characters.

SSH Selected Algorithms for Ciphers

This setting specifies the ciphers you want to allow for connections to the current host. When more than one cipher is selected, the SSH client attempts to use ciphers in the order you specify on the Advanced VT SSH dialog box.

If the value is modified, you must maintain the proper format, including commas.

Maximum length is 260 characters.

SSH Selected Algorithms for HMAC

Specifies the HMAC (hashed message authentication code) methods you want to allow. This hash is used to verify the integrity of all data packets exchanged with the server. When more than one HMAC is selected, the SSH client attempts to negotiate a MAC with the server in the order you specify on the Advanced VT SSH dialog box.

If the value is modified, you must maintain the proper format, including commas.

Maximum length is 260 characters.

SSH Selected Algorithms for Key Exchange

Specifies which key exchange algorithms the client supports. When more than one algorithm is selected, the SSH client attempts to negotiate key exchanges in the order you specify on the Advanced VT SSH dialog box.

If the value is modified, you must maintain the proper format, including commas.

Maximum length is 260 characters.

SSH Selected Host Key Algorithms

Specifies the hash algorithm the client uses in the process of proving possession of the private key. This hash is used during public key user authentication.

Maximum length is 260 characters.

SSH Selected User Authentication Types

Specifies an authentication method. Public key is selected by default.

Maximum length is 260 characters.

NS/VT

Host Network Address

- Specifies the IP address of the host for this model. This setting is read-only.
- Specifies the host port or socket number that the NS/VT session should use. Enter any number between 0 and 65,535 in this field.

You can configure **Port** in either the Session Setup or View Settings dialog boxes. The default port number for NS/VT is **1570**.

Note: NS/VT is a proprietary HP protocol that connects only to HP3000 hosts.

NS/VT Timed Reads

- The NS/VT protocol allows the host to set a time limit when it reads from the terminal. If the read has not been satisfied within the set time limit, the NS/VT client notifies the host and terminates the read.

Typically, a host application uses this capability for security. When notified that the read has timed out, it automatically logs the user off of the host. - A setting of **Yes** means that timed reads are treated normally and the host is notified when the time expires. - A setting of **No** means that the time parameter of reads is ignored by the NS/VT client and the reads will never time out. The default setting is No.

AUTO SIGNON (SETTING)

When this box is checked, Host Integrator tells the transport protocol to automatically log you on to the host as soon as you establish a connection in the Design Tool.

By default, this box is not checked.

CONNECTED (SETTING)

Specifies whether you are connected to a host.

This setting is read-only.

ENABLE ASYNCHRONOUS TRANSPORT BEHAVIOR (SETTING)

This setting is no longer used.

HOST COMMUNICATION TIMEOUT (FROM THE HOST)

Specifies how many seconds Host Integrator should wait for a host connection response. When there is no response from the host in the allotted time, an error results.

The range of values is from 1 to 9999.

The default is 9999.

HOST CONNECTION TIMEOUT (TO THE HOST)

Specifies how many seconds Host Integrator should wait for a successful connection to the host. When Host Integrator is unable to connect to the host in the allotted time, an error results.

The range of values is 0 to 86400. The default value is 30 seconds.

Note: When the timeout is set to zero, all host connection attempts time out immediately.

HOST NAME

Use this box to identify the host to which you want to connect. You can either select a host from the drop-down list or type one into the host name field.

When you start the Design Tool without a value for this setting, the Design Tool looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location.

On a Windows platform, the Design Tool searches for the Hosts file in this order:

- In the same path as Wsock32.dll

- In the \WINDOWS\system32\drivers\etc\Hosts folder

- In the System or System32 folder of the operating system root folder.

- In the folder where the Design Tool is installed.

- In the current folder.

- In the operating system root folder (typically C:\Windows).

- In the folders in your Path statement.

When it finds a Hosts file, the Design Tool changes the value of this setting.

If the Design Tool cannot find your Hosts file in any of those locations, the value of this field becomes blank.

To locate a Hosts file on your computer, use the Browse button in the View Settings dialog box (while the **Hosts File Name** setting is selected under **Settings**) or use the operating system's Find feature.

HOST PASSWORD

Enter your password for the host if you want to bypass being prompted for it at connection time.

You'll also need to enter your username.

HOST USER NAME

You must supply authentication credentials to establish a connection to the host.

Enter your username for the host sign-on screen if you want to bypass being prompted for it at connection time.

You'll also need to enter your password.

HOST FILE NAME

Specifies the path to the Hosts file, which maps assigned node names to internet addresses. The value is a string of up to 260 characters.

- The purpose of this setting is to allow Host Integrator to find a list of hosts to populate the Host name or IP address list box in the Session Setup dialog box.

When you start Host Integrator without a value for this setting, Host Integrator looks for the Hosts file in the same folder as Wsock32.dll. If you are using third-party TCP/IP software, your Hosts file may be in another location.

Host Integrator searches for the Hosts file in this order:

In the same path as Wsock32.dll

In the \WINNT\system32\drivers\etc\HOSTS folder

In the folder where Host Integrator is installed

The current folder

In the operating system's root folder

The folders in your Path statement

When it finds a Hosts file, Host Integrator changes the value of this setting.

If Host Integrator cannot find your Hosts file in any of those locations, the value of this field becomes blank.

To locate a Hosts file on your PC, use the Browse button in the View Settings dialog box (while the **Hosts File Name** setting is selected in the Host Integrator settings list box) or use the operating system's Find feature.

KEYBOARD LOCK TIMEOUT

The Keyboard Lock Timeout setting applies to terminals that the host system controls, whether or not keyboard input is permitted. The keyboard is said to be "locked" if input is disabled.

When the Host Integrator has input to send to the host, it will send the input immediately if the keyboard is unlocked. If the keyboard is locked, Host Integrator will wait up to the period specified by this setting for the keyboard to be unlocked.

If the keyboard is still locked after this time, Host Integrator will return a keyboard lock timeout error.

- This setting is an integer representing the number of seconds for the keyboard lock timeout. Values from 1-9999 seconds are accepted.
- The default is 5 seconds.

TRANSPORT PROTOCOL

- This setting specifies the transport type being used to connect to the host. The transport types available are related to the selected terminal type. You can use Telnet, Telnet

Extended, SSH, or NS/VT transport types as well as Secure Socket Layer (SSL) and Transport Layer Security (TLS) security protocols.

Telnet is available for 3270,5250, VT, and HP session types, while Telnet Extended is only available for IBM 3270 session types. Choose SSH to connect to a VT host using SSH security protocols. NS/VT is available for HP session types.

Telnet

For more information about using Telnet Secure Socket Layer (SSL) and Transport Layer Security (TLS) see [Using SSL/TLS](#).

Telnet and HP

If you are connecting to an HP terminal using Telnet as your transport type, use an event command such as `WaitForNewHostScreen` between HP `TransmitTerminalKey` commands in operations.

Telnet and Host Emulator

The Host Emulator cannot be used if you created `.hetrace` files while connected to a host using the Telnet Extended (TN3270E) transport type.

SSH and VT

You can configure SSH connections when you need secure, encrypted communications between a trusted VT host and your computer over an insecure network. SSH connections ensure that both the client user and the host computer are authenticated and that all data is encrypted. See "Using SSH: Overview."

Miscellaneous

- Autogroom the TracePlayer File

When this check box is selected (the default), the TracePlayer file used by Host Emulator is autogroomed after the recording is complete.

- Caps Lock State

Specifies whether the Design Tool is in Caps Lock mode. The default is No. This state is not saved as part of a model or settings file.

- Caption

Specifies the string that appears in the Design Tool title bar. This string is relevant only when the Help display mode is Status Bar (default). This string is also displayed on the taskbar when the Design Tool is running.

Select a predefined option from the list box, or enter up to 260 characters in the box. As you click on predefined options, shortcuts for these options are added to the box. The predefined options and their shortcuts are:

Shortcut	Option
&r	"Verastream Host Integrator"
&f	Model file name (or "Untitled" if a model file is not open)
&s	Session type
&t	Transport
&h	Host name
&d	Date
&c	Connection status
&v	Assigned device name
&&	Single ampersand

Shortcut	Option
&p	Script debugging port

So, for example, if you set the Caption to &s - &t - &c, you might see "IBM 5250 Terminal - Telnet - Connected" in the Design Tool title bar or on the taskbar, assuming the Design Tool is running but not minimized. The default is &f - &r - &s.

- Case Sensitive Search

Select this option to use case sensitive criteria when searching for tables and elements in both the Tables and Import Model Elements dialog boxes. By default searches are case insensitive.

- Close Progress Window on Completion

The Host Integrator progress window displays information about the progress of navigation or event handling sequences. The progress window is displayed when a sequence takes longer than one-half second. You can force the progress window to stay open by changing this setting to No. The default is Yes.

If set to No, a login or logout process can stall when a dialog box requires user input. You must click Close so that the login or logout process can continue.

- Command Line Switches (read-only)

Shows command line switches (also called command line parameters), if any, that were used to start the Design Tool.

- Confirm Exit

Select this option if you want the Design Tool to display a confirmation dialog box before exiting.

- Connect To Host on Open

Specifies whether you want the Design Tool to automatically connect to the host when you load a model file.

- Convert Alternate Font Characters to Spaces

Specifies how to handle characters that are not part of the default character set. For example, VT line drawing characters in a character mode application are not part of the ANSI character set. If this setting is set to Yes, Host Integrator will translate all such characters to spaces when reading them from the screen buffer. The default is No, which

means that Host Integrator will read and store the numerically equivalent ANSI character instead.

- Display small icons in the Navigator dialog (read-only)

Reflects the Host Integrator Navigator utility option to display small icons instead of the default large icons.

- Don't Show Discard Recording Dialog Box

Specifies whether or not to display the Discard Recording dialog box each time a record trace is deleted. The default is No.

- Don't Show the Hidden Variable in Descriptor Security Warning Dialog Box

Select Do not display the message again to hide the security warning regarding the display of hidden variable values as plain text within descriptor files. To secure sensitive information, it is best practice to use the Administrative Console to enter the variable values.

- Don't Show Save Entity Dialog Box

Specifies whether or not to display the Apply Changes dialog box each time an entity is modified. The default is No.

- Execute Host Login Sequence on Host Connect

Specifies whether the Design Tool will execute a login command list when your model connects to a host.

You can also enable this setting from the General tab of the Preferences Setup dialog box by selecting the Execute login commands check box. The default is No.

This option also navigates to the home entity and initiates associated event handlers.

- Execute Host Logout Sequence on Host Disconnect

Specifies whether to execute a logout command list when your model disconnects from a host.

You can also enable this setting from the General tab of the Preferences Setup dialog box by selecting the Execute logout commands check box. The default is No.

- Font Character Set

Specifies the character set that Host Integrator uses for its user interface. The character set values are:

- Ansi (default)
- Baltic
- Chinese Big5
- East European
- GB2312
- Greek
- Hangeul
- Hebrew
- Johab
- Oem
- Russian
- ShiftJis
- Turkish

- Max Trace File Length (KB)

Specifies the maximum size for an internal trace file (.cfgtrc), in kilobytes. The range of values is from 10 to 65,535 KB. The default is 10,000 KB.

- Maximum Number of Consecutive Session View Partial Updates Allowed

Specifies the maximum number of partial screen updates that will be accepted by a Host Integrator Server from a character mode host. Partial updates are commonly sent via a datastream of characters on character mode hosts unlike block mode hosts, which often transmit one screen's worth of data at a time. The default is 5.

- Maximum Size (KB) for Session View Buffered Data

Specifies the maximum limit of memory that is used when a Host Integrator Server stores host data for the Session Monitor. The default is 1,024 KB.

When a Host Integrator Server receives data from a host, it buffers that data until it can be used by the Session Monitor. During a host session, the Session Monitor requests the data from the Host Integrator Server and the buffered data is sent. Setting a maximum limit of memory on the Host Integrator Server creates a type of load balancing between the server and the Session Monitor that enables optimum performance. Once this memory limit is

exceeded, the memory is deleted and the following log message appears in the Session Server log:

```
Since the buffer size of the maximum session view exceeded its limit, one or more session view displays were lost.
```

- **Maximum Size for Debug Trace Output File**

Specifies the maximum size for debug trace output files. The range of values is from 0 to 1024 MB. If you specify 0, no size limit is enforced.

- **Maximum Size Design Tool Model Debug Message Store**

Specifies the maximum size for the model debug messages store. The range of values is from 1 to 1000 MB. The default is 128.

- **Mouse Last-Clicked Column**

Specifies the column number where the mouse was located the last time the left button was clicked.

- **Mouse Last-Clicked Row**

Specifies the row number where the mouse was located the last time the left button was clicked.

- **Numlock State**

Specifies whether Host Integrator is in Num Lock mode. The Design Tool does not change the state of Num Lock, although the model can affect the Num Lock setting if the state of

Num Lock is changed as part of building or loading a model. This state is not saved as part of a model or settings file.

- Path and Name of Executable (read-only)

Returns the full path and filename (including the drive letter) for the Host Integrator Design Tool executable file.

- Path to Executable (read-only)

Returns the full path (including the drive letter) for the Host Integrator Design Tool executable file.

- Pause Playback Trace on EOR

Specifies what happens when an end-of-record character is encountered during playback of a trace file. When set to Yes (default), the playback pauses at the end of each record. When set to No, the playback runs to completion without pausing.

- Procedure Navigates to Home Entity When an Error Occurs

Specifies whether or not a procedure automatically traverses to the home entity when an error occurs. The default is Yes.

- Read Hidden Text

Specifies whether or not the Design Tool should be able to recognize hidden text on the terminal screen.

- Record Initial Synchronization Commands

Specifies whether the recorder (used in manual recording and by character mode applications) starts each recording session with synchronization commands. If this setting is enabled, these commands will be recorded at the start of each generated operation in

character mode and in each manually recorded command list or operation. The default is Yes.

- Recording (read-only)

Specifies whether or not the command list recorder is on.

- Save Model

Specifies whether you want to automatically save your model when you exit the Design Tool.

- Scroll Lock State

Specifies the state of Scroll Lock in Host Integrator. This state is not saved as part of a model or settings file.

- Secondary Selection End Column (read-only)

Returns the ending column for the pattern definition location. This setting applies when defining a pattern in an expanded region.

- Secondary Selection End Row (read-only)

Returns the ending row for the pattern definition location. This applies when searching for a pattern in an expanded region.

- Secondary Selection Start Column (read-only)

Returns the starting column for the pattern definition location. This applies when defining a pattern in an expanded region.

- Secondary Selection Start Row (read-only)

Returns the starting row for the pattern definition location. This applies when defining a pattern in an expanded region.

- Selection End Column (read-only)

Returns the ending column for the current selection.

- Selection End Row (read-only)

Returns the ending row for the current selection.

- Selection Start Column (read-only)

Returns the starting column for the current selection.

- Selection Start Row (read-only)

Returns the starting row for the current selection.

- Serial Number

Returns the serial number of your copy of Host Integrator.

- Show Progress Details

The Host Integrator progress window displays information about the progress of navigation or event handling sequences. The progress window is displayed when a sequence takes longer than one-half second and will remain open after completion if an error occurs. By

default, this progress window shows a short progress statement. When Show Progress Details is set to Yes, the details of the sequence are also displayed. If Show Progress Details is set to Yes, you can also force the progress window to stay open by using the setting Close Progress Window on Completion.

- Startup Working Directory (read-only)

Specifies the working directory for the current session.

- Translation Character Set (read-only)

Indicates the type of translation taking place between the Host Integrator client and server, based on the type of terminal session and the international (National Character Set) setting. This setting is particularly useful for technical support personnel.

In the Design Tool, the setting indicates the international characters displayed in the terminal window and in certain internationalized dialog boxes.

On the server, the setting determines which single-byte to Unicode translation will be used when communicating with a Host Integrator client.

Possible values are:

ISO-8859—default. This is the character set used on Microsoft Windows for the US, Western Europe, and Latin America. All VT, HP, and most 3270 and 5250 Host Integrator sessions use this character set.

Thai—Used only for Thai 3270 or 5250 terminal sessions.

Baltic—Used only for Baltic 3270 sessions.

Greek—Used only for Greek 3270 or 5250 sessions.

Turkish—Used only for Turkish 3270 or 5250 sessions.

East Europe—Used only for East European 3270 or 5250 sessions.

Cyrillic—Used only for Cyrillic 3270 or 5250 sessions.

Custom—Used for custom sessions.

- Use model file host settings for design tool deployment

The host connection settings specified in the Session Settings dialog box (available by choosing Session Setup from the Connection menu) are used when deploying the model

from within the Design Tool. If set to No, a separate host name and port number must be specified on the Deployment Options tab available from the File menu. The default is Yes.

- User Recording (read-only)

Use to manually record commands in Host Integrator

- Version String (read-only)

Returns the version number of your copy of Host Integrator.

Events

Host Integrator event settings:

Action Taken after Reloading Event Handler Libraries

Determines whether to use an existing host session, or to shut down any existing host session and create a new one when rebuilding event handlers.

This setting is saved in the model file because it is specific to each host session.

The default is to use the existing session.

Automatically Send New Handlers to the Editor

Determines whether you automatically launch the default editor for newly-created event handlers or not.

The default is No.

Specifies whether the Design Tool automatically reloads the event handlers each time you make a change, or reloads only when a rebuild or reload command is executed.

This option is available on the Building tab of the Event Handler Settings dialog.

The default is Yes.

Default Event Handler Timeout

If a timeout is not specified for a particular event, this setting is used as the event handler timeout. This setting requires the event timeout to be enabled.

If the event handler exceeds this timeout when processing an event, a timeout error is generated.

You can also modify this setting (Default event timeout) on the Environment tab of the Event Handler Settings dialog.

The default is 10 seconds. The range is 1-9999 seconds.

Default Package Name for New Handlers

Specifies a package name that will prefix the class names of generated Java source files.

If you do not provide a package name, no package name is added to the class name.

The maximum length is 260 characters.

Disable Event Handler Timeouts in Design Tool

Controls the use of event handler timeouts. If event timeouts are enabled, the default or handler-specific event timeout setting is enforced.

If event timeout is disabled, no timeouts are enforced when an event handler is processing an event. All other running timeouts, such as operation timeouts, are suspended. This option is useful for debugging.

This setting is available from the Event Handler toolbar, and from the Events menu.

The default is No (timeout is enabled).

Event Handler Classpath

This is the classpath for non-event handler libraries that are searched to satisfy external class references in the event handlers, both when compiling and at run-time.

You cannot edit the classpath here. Specify classpath information in a script properties file.

Event Handler Language

This setting specifies Java as the event handler language and is read-only.

Event Handler Language Chosen

When set to No, event handlers have *not* been reviewed or implemented in the model.

When set to Yes, event handlers have been created.

This setting is read-only.

Event Handler Timeout Recovery

If an event handler times out, the timeout recovery interval, or abort interval, governs how long the event handler is given to complete and return control.

The default is 5 seconds.

Password Client Connect Test Parameter

The test parameter for "password" will be used to supply the credentials for an **Authenticate user** event in a lifecycle event handler. When using the Session Server, credentials will be supplied by a client connection as the "userid" and "password" parameters to one of the Host Integrator API Connect methods.

You can configure this setting on the Environment tab of the Event Handler Settings dialog.

The maximum length is 260 characters.

Script Manager State

Indicates the current condition of the script manager for event handlers.

This setting is read-only.

- The possible states:
- Ready: The server for event handler processing has all relevant updates.

Handler Update Needed: The session needs to be reset on the event handler server or event handlers need to be reloaded.

Reset Needed: This state occurs when the event handler server is blocked at a breakpoint or has been shut down. You can select Reload Handlers from the Events menu to change this state.

UserID Client Connect Test Parameter

The test parameter for UserID is used to supply the credentials for an **Authenticate user** event in a lifecycle event handler.

When using the Session Server, credentials will be supplied by a client connection as the "userid" and "password" parameters to one of the Host Integrator API Connect methods.

You can also configure this setting on the Environment tab of the Event Handler Settings dialog.

The maximum length is 260 characters.

Setup

Keyboard and display settings for Design Tool.

KEYBOARD

Auto Repeat (Enter Key Repeat)

Checking this box causes most keys to repeat when you hold down the key. The Shift, Return, and Ctrl keys do not auto repeat.

This setting is permanently on and cannot be turned off, even though it appears you can clear the selection.

Host 3270 Keyboard

Specify which terminal keyboard is displayed in the Keyboard Setup dialog box **and** on the floating terminal keyboard.

In the **Setting details** drop-down, choose your Typewriter. The default is US English 3270 Typewriter

Host 5250 Keyboard

Specify which terminal keyboard is displayed in the Keyboard Setup dialog box and on the floating terminal keyboard.

In the **Setting details** drop-down, choose your Typewriter. The default is US English 5250 Typewriter.

Host HP Keyboard

Specify which terminal keyboard is displayed in the Keyboard Setup dialog box and on the floating terminal keyboard.

In the **Setting details** drop-down, choose your Typewriter. The default is US English VT Typewriter

Host Keyboard Type

Specify which type of terminal keyboard you are emulating.

In the **Setting details** drop-down, choose your keyboard type. The default is Normal.

Host VT420 Keyboard

Specify which terminal keyboard is displayed in the Keyboard Setup dialog box and on the floating terminal keyboard.

In the **Setting details** drop-down, choose your Typewriter. The default is US English VT Typewriter.

Keyboard Error Alarm

Indicates whether an alarm (a beep) is sounded when a keyboard error is detected. By default, this box is not checked.

Note: This setting is applicable to 5250 terminal sessions only.

Keyboard Locked

Indicates whether the keyboard is currently locked. This setting is read-only.

Local Keyboard

Specify which PC keyboard is displayed in the Keyboard Setup dialog.

In the **Setting details** drop-down, choose a keyboard option. The default is US English Enhanced 101 Key.

Right Control Key Repeat (setting)

By default, the right Ctrl key acts as a modifier key. That is, used in combination with another key to send a function (for example, Ctrl+F1 is mapped to the Attention key in 3270 sessions). With this setting, you can specify whether the right Ctrl key is treated like any other key, where continuously pressing the key repeatedly sends its function.

Set this to Yes if you map the right Ctrl key to a function other than its default use as a modifier key. The default is No.

Show Host Mapped Keys

Specifies whether host keys (functions) that are mapped to PC keys are shown in cyan on the terminal keyboard in the lower half of the Keyboard Setup dialog. The default value is Yes.

Show Key Help

Specifies whether the name of the selected key in the Keyboard Setup dialog box is identified at the bottom of the dialog box in the Status line. The default value is No. Key help is available for keys on the PC keyboard and on the terminal keyboard.

Terminal Kbd Fix Upper Right Corner

Specifies which corner of the terminal keyboard remains tethered when the user resizes the Terminal Keyboard by clicking the up-arrow or down-arrow icon in the upper right corner with the right mouse button. When the value is **Yes**, the upper-right corner remains tethered. When the value is **No** (default), the upper-left corner remains tethered.

Terminal Kbd Mode

Specifies whether the graphical terminal keyboard is attached to one of the margins of the Terminal window and, if it is, which margin it is attached to. The values are:

- **Bottom Bar:** The terminal keyboard is attached to the bottom margin of the Terminal window.
- **Floating (default):** The terminal keyboard is free floating.
- **Left Bar:** The terminal keyboard is attached to the left margin of the Terminal window.
- **Right Bar:** The terminal keyboard is attached to the right margin of the Terminal window.
- **Top Bar:** The terminal keyboard is attached to the top margin of the Terminal window.

Terminal Kbd Shape

Specifies the shape of the graphical terminal keyboard. The values are:

- **Full Size (default):** This is the largest and most complete version—all keys, including alphanumeric keys, are shown.
- **Mini Size:** Similar to Full Size, but without the alphanumeric keys.
- **Vertical:** Displays a long, narrow keyboard, without alphanumeric keys.
- **Horizontal:** Displays a wide, short keyboard, without alphanumeric keys. This setting is relevant only if Terminal Keyboard Mode is set to Floating.

Terminal Kbd Tether

Specifies whether the location of the graphical terminal keyboard is tied to the Terminal window's location.

When the value is **Yes** (default), the terminal keyboard is tethered to the Terminal window—when you move or resize the Terminal window, the terminal keyboard moves, tracking the nearest corner of the Terminal window. When set to **No**, moving the Terminal window has no effect on the location of the terminal keyboard.

This setting is only relevant if Terminal Keyboard Mode is set to **Floating**—that is, if the terminal keyboard is not attached to one of the margins of the Terminal window.

Terminal Kbd Visible

Specifies whether the graphical terminal keyboard is visible. The default is **No**.

HOST EVENTS

Allow Keys During Wait Commands

Specifies whether the user can use the terminal keyboard while an operation containing a Host Events command is executed on the host. The default is **Yes**.

DISPLAY

All Font Character Sets

Specifies whether non-ANSI fonts can be used for the display font. The default is **No**. Selecting **Yes** means that non-ANSI fonts are available to use as the display font.

After setting this value, click **Display** on the Settings menu, and then click the Fonts tab. The **Font** box will list all the ANSI and non-ANSI fonts that are available for use as the display font.

Automatic Font Sizing (setting)

Specifies that the Design Tool automatically adjusts the font size to fit all text in the Terminal window for whichever font or style you select.

By default, Auto Font Size is enabled. To change the font size, set this setting to **No**.

BDT Ignore Scroll Lock (setting)

Specifies whether the Design Tool ignores the state of scroll lock when connecting to the host. The default is **No**.

Caps Lock State (setting)

Specifies whether the Design Tool is in Caps Lock mode. The default is **No**. This state is *not* saved as part of a model or settings file.

Caption

Specifies the string that appears in the Design Tool title bar. This string is relevant only when the Help display mode is **Status Bar** (default). This string is also displayed on the taskbar when the Design Tool is running.

Select a predefined option from the list box, or enter up to 260 characters in the box. As you click on predefined options, shortcuts for these options are added to the box.

The predefined options and their shortcuts are:

Shortcut	Option
&r	"Verastream Host Integrator"
&f	Model File Name (or "Untitled" if a model file is not open)
&s	Session Type
&t	Transport
&h	Host Name
&d	Date
&c	Connection Status (whether you are connected and over what transport)
&v	Assigned Device Name
&&	Single Ampersand

Shortcut	Option
&p	Script debugging port

So, for example, if you set the Caption to **&s - &t - &c**, you might see "IBM 5250 Terminal - Telnet - Connected" in the Design Tool title bar or on the taskbar, assuming the Design Tool is running but not minimized. The default is **&f - &r - &s**.

Current Cursor Column (setting)

Returns the current cursor column location relative to the left edge of the Terminal window. This setting is read-only.

Current Cursor Row (setting)

Returns the current cursor row location relative to the top edge of the Terminal window. This setting is read-only.

Current Display Height (setting)

Specifies the number of rows in the current display height. This setting is read-only.

Note: The value of this setting includes two non-addressable rows dedicated to the Operator Information Area (OIA), so the display height's actual number of addressable rows is the number of addressable rows minus two (2).

Current Display Width (setting)

Specifies the number of columns in the current display width. This setting is read-only.

Current Screen Selection Mode

Specifies what mode to use for the region selected on the terminal screen.

Select **Linear** to use word processor style selection regions that wrap lines, or **Rectangular** to use rectangular regions. The default is **Rectangular**.

This setting is read-only.

Default Screen Selection Mode

Specifies what mode to use when you select a new region on an entity.

Select **Linear** to use word processor style selection regions that wrap lines, or **Rectangular** to use rectangular regions. The default is **Rectangular**.

Display Margins

Specifies whether to display margins around the Terminal window. If the **Display margins** box is checked (default), text will not touch the sides of the Terminal window. Instead, a margin or border is displayed. Displaying margins may improve readability.

If you want text to fill the entire screen, clear this check box. If this check box is clear, text can touch the sides of the Terminal window, which may be useful for smaller monitors and notebook computers.

Depending on the screen resolution and the size of your Terminal window, you may not see an effect in the Terminal window when you clear this check box. This is because each character takes up the same number of pixels and if your Terminal window is sized so that its width is not evenly divisible by the number of columns, the Design Tool centers the text on the screen.

Display Variable Width Fonts

Specifies whether proportionally-spaced font types appear in the **Fonts** box. The default is **Yes**.

Event Handler Toolbar Visible

Specifies whether the event handler toolbar is visible. The default is Yes.

Font Character Set (setting)

Specifies the character set that Host Integrator uses for its user interface. The values are:

Ansi Character Set (default)

Baltic Character Set

Chinese Big5 Character Set

East European Character Set

GB2312 Character Set

Greek Character Set

Hangeul Character Set

Hebrew Character Set

Johab Character Set

Oem Character Set

Russian Character Set

ShiftJis Character Set

Turkish Character Set

Font Name

Specifies the display font in the Design Tool session window. You can use any font, including TrueType fonts. Arial is an example of a TrueType font that is not monospaced, and therefore is not supported by the Design Tool. By default, the Design Tool uses the `r_ansi` font. This provides a 24 x 80 display that accurately emulates the terminal. When you resize the Terminal window, the Design Tool chooses a new font size so the correct number of rows are displayed on the screen. If the Design Tool cannot display the font you have chosen, the default (`r_ansi`) is displayed instead.

The available fonts can change as you change the model ID in the Session Setup dialog box because not all model IDs support all fonts.

Note

When you print all or part of a host screen from a terminal session, the font used is the currently configured display font.

Font Size

Specifies a font size that the Host Integrator uses in its Terminal window. The range is from 6 to 32 points. The default is **12** points.

Font Style

Specifies a font style: regular or bold. The Design Tool does not support italicized fonts. The default is **Regular**.

Height of Window

The Design Tool can display a brief message about the currently selected menu, menu command, or button on the title bar (above the Terminal window) or in the status bar (below the Terminal window). By default, help messages are shown in the status bar.

Specifies the height (in pixels) of the Terminal window. The range of values is from 0 to 4095. The range of acceptable values and the default vary depending on your screen resolution and video driver.

Help Display Mode

The Design Tool can display a brief message about the currently selected menu, menu command, or button on the title bar (above the Terminal window) or in the status bar (below the Terminal window). By default, help messages are shown in the status bar.

Left of Window

Specifies the location of the left Terminal window margin (in number of pixels from the left margin of the screen). The range of values is from -2147483648 to +2147483647. The default is 0.

Model Toolbar Visible

Specifies whether the toolbar is visible. The default is Yes.

Mouse Shape

Specifies the appearance of the mouse pointer in the Terminal window. The values are:

- Default arrow: The mouse pointer appears as the Windows default arrow.
- IBeam (default): The mouse pointer appears as an I-shape.
- Rectangular: The cursor appears as a rectangle.

National Character Set

The values identify the various host character sets that Host Integrator supports for 5250 and 3270 terminal emulation. The list varies slightly between 3270 and 5250 sessions. Host Integrator uses this setting to choose a conversion table that it uses to convert host characters (EBCDIC) into PC characters (ANSI).

This setting should match the national character set used by your host system. If it doesn't match, then some characters, such as accents, may not display correctly. See your host documentation for definitions of the characters in each set. The default value is US English.

Num Lock State

Specifies whether Host Integrator is in Num Lock mode. The Design Tool does not change the state of Num Lock, although the model can affect the Num Lock setting if the state of Num Lock is changed as part of building or loading a model. *Note:* If the model changes the state of Num Lock, you may need to toggle your workstation's Num Lock state after closing the model. This is especially critical for laptops. This state is not saved as part of a model or settings file.

Scroll Lock State

Specifies the state of Scroll Lock in Host Integrator. This state is not saved as part of a model or settings file.

Secondary Selection End Column

Returns the ending column for the pattern definition location. This setting applies when defining a pattern in an expanded region. This setting is read-only.

Secondary Selection End Row

Returns the ending row for the pattern definition location. This applies when searching for a pattern in an expanded region. This setting is read-only.

Secondary Selection Start Column

Returns the starting column for the pattern definition location. This applies when defining a pattern in an expanded region. This setting is read-only.

Secondary Selection Start Row

Returns the starting row for the pattern definition location. This applies when defining a pattern in an expanded region. This setting is read-only.

Selection End Column

Returns the ending column for the current selection. This setting is read-only.

Selection End Row

Returns the ending row for the current selection. This setting is read-only.

Selection Start Column

Returns the starting column for the current selection. This setting is read-only.

Selection Start Row

Returns the starting row for the current selection. This setting is read-only.

Status Bar Text

Specifies the string that appears in the Design Tool status bar. Select a predefined option from the list box, or enter up to 260 characters in the box. As you click the predefined options, shortcuts for these options are added to the box. The predefined options and their shortcuts are:

Shortcut	Option
&r	"Verastream Host Integrator"
&f	Model File Name (or "Untitled" if a model file is not open)
&s	Session Type
&t	Transport Type
&h	Host Name
&d	Date
&c	Connection Status (whether you are connected and over what transport)
&v	Assigned Device Name
&&	Single Ampersand

Shortcut	Option
&p	Script debugging port

So, for example, if you set the status bar text to **&s - &t - &c**, you will see "IBM 5250 Terminal - Telnet - Connected" in the Design Tool status bar.

The default is **&s - &c**.

Substitute Display Chars (Substitute "0" with "Ø")

Specifies whether zeroes displayed in the Terminal window contain slashes (like this: Ø). Selecting this option may make it easier to work with numeric data. By default, this check box is cleared. This check box has no effect on fonts that, by design, already display zeroes with slashes.

Toolbar Visible

Specifies whether the toolbar is visible. The default is Yes.

Top of Window

Specifies the location of the top Terminal window margin (in number of pixels from the top margin of the screen).

The range of values is from -2147483648 to +2147483647. The default is 0.

Width of Window (setting)

Specifies the width (in pixels) of the Terminal window.

The range of values is from 0 to 4095. The range of acceptable values and the default vary depending on your screen resolution and video driver.

Window State

Specifies the state of the Terminal window. The values are:

- Maximized: The Terminal window is maximized.
- Minimized: The Terminal window is minimized.
- Restored (default): The Terminal window is neither maximized nor minimized.

Help Display Mode

The Design Tool can display a brief message about the currently selected menu, menu command, or button on the title bar (above the Terminal window) or in the status bar (below the Terminal window). By default, help messages are shown in the status bar.

Model File

Specifies the name of the model file currently open in the Design Tool. This setting is read-only.

Save Window Position

Specifies whether you want to save the position of the Entity window when you exit. The values are:

- Disabled: Does not save or load the previous position. This is the default option.
- Last position used: Uses the last position of the model window.
- Position saved with model: Saves the position of the model to the model file and only loads that position.

Settings File

Returns the name of the open settings file. If no settings file is open, an empty string ("") is returned. This setting is read-only.

WCP Communication

WCP Timeout: Specifies how long the Design Tool will wait for WCP to make a connection before an error message appears. The default is 60000 milliseconds.

Compatibility

Compatibility settings apply to upgrades from Verastream Host Integrator 4.5, 5.0, 5.5, and higher.

When you open a model from a previous version in the Design Tool, you'll see a message noting that "This model was created with a prior version of Verastream Host Integrator." Compatibility settings enable your models to maintain the same behavior as they had in earlier versions of Host Integrator.

4.15.4 Operation Command Summary

Click the command for more information on Host Integrator operation commands.

Terminal input commands - Terminal input commands in an operation transmit function calls directly to the host. When the host receives the command, it is executed on the terminal screen. For an example of terminal input commands, see the Pine host application model.

[MoveCursor](#)

[ShiftCursor](#)

[TransmitANSI](#)

[TransmitANSIEncrypted](#)

[TransmitToLocation](#)

[TransmitToLocationEncrypted](#)

[TransmitToOffset](#)

[TransmitToOffsetEncrypted](#)

[TransmitTerminalKey](#)

Host Events - Host Events commands transmit function calls that wait a specified amount of time before executing the next command in a command list. Use these events to synchronize the Host Integrator server with the state of the host application. Don't confuse these host synchronization events with event handlers, which perform or extend actions in the model file itself.

Note

You can change the default timeout period for these commands on the [Preferences Setup/Command tab](#) from 5 seconds, to a value that works better for your operation.

Wait
WaitForCommString
WaitForCondition
WaitForCursorAtAttr
WaitForCursorAtLocation
WaitForCursorAtRecordsetField
WaitForCursorAtTerminalField
WaitForCursorNotAtLocation
WaitForCursorNotAtTerminalField
WaitForDisplayString
WaitForHostSilence
WaitForKeyboardEnabled
WaitForMultipleEvents
WaitForNewHostScreen
WaitForUpdate
WaitMS

Attribute/Field inputs commands - Attribute/field input commands directly modify the value of an attribute or recordset field and write that value directly to the terminal screen.

DefaultValue
DefaultValueEncrypted
TransmitToAttr
TransmitToAttrEncrypted

Since managing the order of screen updates can be problematic in character mode applications, the following commands are designed to enable control of this ordering process. When using these commands, make sure to enable the caching feature available on the Advanced Entity Properties dialog box and the Recordset Fields tab. By enabling the caching feature, Host Integrator will store any attribute or recordset field data until one of the following commands are included in an operation, which orders the write of the cached data to the terminal screen. By default, an UpdateAttribute command is inserted into each generated operation when Cache all attribute writes is selected. Since attribute data assignments can be arbitrarily mixed between an operation and a Host Integrator API, the order in which data is actually written to the terminal can be important to the host application.

UpdateAttribute

UpdateAttributes

UpdateRecordsetField

UpdateRecordsetFields

Data exchange commands - Data exchange commands in an operation enable you to write variable data to or read variable data from an entity attribute, a recordset field, or data on the terminal screen.

ReadVarFromAttr
ReadFromMappedAttr
ReadVarFromLocation
ReadVarFromField
ReadVarFromTerminal
WriteVarToAttr
WriteToMappedAttr
WriteVarToLocation
WriteVarToField
WriteVarToTerminal
CheckOperationConditions
DefaultValue
DefaultValueEncrypted
ExtendSelection
DefaultValueEncrypted
ExtendSelectionLine
ExtendSelectionPage
ExtendSelectionRect
ExtendSelectionWord
LightPen
MoveCursor
Navigate
PlayBackTrace
ReadFromMappedAttr
ReadVarFromAttr
ReadVarFromField
ReadVarFromLocation
ReadVarFromTerminal
ResetRecordset
SelectLine
SelectWord
SetColorMap
SetMousePos

SetNumeric
SetScanCodeName
SetSelectionStartPos
SetTerminalKbdPos
SetUpdateOffset
SetUpdateRegion
SetupSession
SetWindowPos
ShiftCursor
StartRecording
StartTrace
SwitchToWindow
TerminalMouse
Toggle
TransmitANSI
TransmitANSIEncrypted
TransmitTerminalKey
TransmitToAttr
TransmitToAttrEncrypted
TransmitToField
TransmitToFieldEncrypted
TransmitToLocation
TransmitToLocationEncrypted
TransmitToOffset
TransmitToOffsetEncrypted
UpdateAttribute
UpdateAttributes
UpdateRecordsetField
UpdateRecordsetFields
Wait
WaitForCommString
WaitForCondition
WaitForCursorAtAttr

[WaitForCursorAtLocation](#)
[WaitForCursorAtTerminalField](#)
[WaitForCursorNotAtLocation](#)
[WaitForCursorNotAtTerminalField](#)
[WaitForCursorAtRecordsetField](#)
[WaitForDisplayString](#)
[WaitForHostSilence](#)
[WaitForKeyboardEnabled](#)
[WaitForMultipleEvents](#)
[WaitForNewHostScreen](#)
[WaitForUpdate](#)
[WaitMS](#)
[WriteVarToAttr](#)
[WriteVarToField](#)
[WriteVarToLocation](#)
[WriteVarToTerminal](#)

.CheckOperationConditions

Syntax

```
CheckOperationConditions
```

Description

Checks the required attributes and the condition string and verifies that any pre-or post-conditions configured for the current operation are satisfied. To configure pre- or post-conditions for any given operation, click Conditions on the Operations tab to open the Operation Conditions dialog box.

Note: Error patterns and post-operation conditions are evaluated separately.

More information

[Operation conditions](#)

.DefaultValue

Syntax

```
DefaultValue "Attribute name", "Value specified in Value box", specifies on which application this default value should be executed
```

Description

Transmits a default value to an attribute when a data input is not recognized.

The DefaultValue command guarantees that some value will be written as part of the operation. However, if an attribute has been modified by some other command, then the default value has no effect. If you need to write a value in all cases, then use the [TransmitToAttr](#) command instead.

The command applies per visit to an entity. If you leave the entity, all attributes have their modified flag reset.

Command Parameters

Attribute Select a configured attribute.

Value Type a value for the attribute.

Application Select from the following options to specify how this default value should be handled in the Design Tool and during Host Integrator Server execution:

Default value for Design Tool only - Select this option to make this a default value only during model production on the Design Tool.

Default value for server and Design Tool - Select this option to make this default value active during server execution as well as model production on the Design Tool.

More information

[Configuring variables](#)

[Attribute tab](#)

.DefaultValueEncrypted

Syntax

```
DefaultValueEncrypted "Attribute name", "Value specified in Value box", specifies  
which application this default value should be executed on
```

Description

Transmits an encrypted default value to an attribute when a data input is not recognized.

Command Parameters

Attribute Select a configured attribute from this list.

Value Type a value of the attribute selected in the Attribute list. The Design Tool will encrypt the value with asterisks as you type it.

Application Select from the following options to specify how this default value should be handled in the Design Tool and during Host Integrator Server execution:

Default value for Design Tool only - Select this option to make this a default value only during model production on the Design Tool.

Default value for server and Design Tool - Select this option to make this default value active during server execution as well as model production on the Design Tool.

More information

[Configuring variables](#)

[Attribute tab](#)

.ExtendSelection

Syntax

```
ExtendSelection Row, Column
```

Description

Selects all text in the Terminal window between the selection start position (or row 1, column 1, if there is no selection start position) and the specified coordinates. To set the selection start position, use the [SetSelectionStartPos](#) command or click with the mouse in the Terminal window.

Command Parameters

Row Argument type: Integer

The row in which the selection is to end. You can use `rcMouseRow` to specify the row where the mouse was last clicked.

Column Argument type: Integer

The column in which the selection is to end. You can use `rcMouseCol` to specify the column where the mouse was last clicked.

.ExtendSelectionLine

Syntax

```
ExtendSelectionLine Direction
```

Description

Selects all text between the selection start position (or row 1, column 1, if there is no selection start position) and either the beginning or the end of the line. To set the selection start position, use the [SetSelectionStartPos](#) command or click in the Terminal window. For example, this pair of commands selects all the text on the 5th row, starting with column 10 and continuing to the end of the row:

```
SetSelectionStartPos 5,10 ExtendSelectionLine rcForward
```

Command Parameters

Direction Argument type: Enumeration

Indicates whether text should be selected from the selection start position to the beginning or the end of the line. The values are as follows:

Backward

Extends the selection to the beginning of the line. The character under the cursor is not included in the selection.

Forward

Extends the selection to the end of the line. The character under the cursor is included in the selection.

[.ExtendSelectionPage](#)

Syntax

```
ExtendSelectionPage Direction
```

Description

Selects all text in the Terminal window between the selection start position (or row 1, column 1, if there is no selection start position) and the point at which the page ends. To set the selection start position, use the [SetSelectionStartPos](#) command or click with the mouse in the Terminal window.

Command Parameters

Direction The direction in which the selection is to end. You can select either **Backward** or **Forward** from the list.

[.ExtendSelectionRect](#)

Syntax

```
ExtendSelectionRect Row, Column
```

Description

Selects all text in the Terminal window between the selection start position (or row 1, column 1, if there is no selection start position) and the specified coordinates. The selection is made in a rectangular fashion. To set the selection start position, use the [SetSelectionStartPos](#) command or click with the mouse in the Terminal window.

Command Parameters

Row Argument type: Integer

The row in which the selection is to end. You can use `rcMouseRow` to specify the row where the mouse was last clicked.

Column Argument type: Integer

The column in which the selection is to end. You can use `rcMouseCol` to specify the column where the mouse was last clicked.

[.ExtendSelectionWord](#)

Syntax

```
ExtendSelectionWord Direction
```

Description

Selects all text in the Terminal window between the selection start position (or row 1, column 1, if there is no selection start position) and the point at which the text ends. To set the selection start position, use the [SetSelectionStartPos](#) command or click with the mouse in the Terminal window.

Command Parameters

Direction The direction in which the word selection is to end. You can select either **Backward** or **Forward** from the list.

[.LightPen](#)

Syntax

```
LightPen Row, Column
```

Description

Simulates a light pen selection at the specified location in the Terminal window.

Parameters

Row Argument type: Integer

Specifies a row on the host screen. You can use `rcMouseRow` to specify the row where the mouse was last clicked.

Column Argument type: Integer

Specifies a column on the host screen. You can use `rcMouseCol` to specify the column where the mouse was last clicked.

More information

[Adding an operation](#)

.MoveCursor

Syntax

```
MoveCursor Row, Column
```

Description

Moves the cursor to an absolute location on the terminal screen. Use the `[CursorRow]` and `[CursorColumn]` properties to verify the cursor location.

Command Parameters

Row Argument type: Integer

The row to which the cursor is to be moved. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

Column Argument type: Integer

The column to which the cursor is to be moved. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

More information

[Operation destinations](#)

.Navigate

Syntax

```
Navigate "Entity selected from the Destination box"
```

Description

Navigates to a specified entity in the model, automatically using the shortest path from the current entity. This is equivalent to selecting an entity in Design Tool or the Navigator while connected and online with the host.

Navigation across entities uses the operations defined with "Use for navigation commands to this destination" enabled.

Command Parameters

Destination Select the entity to navigate to from this list.

More information

[Model Properties dialog box](#)

[Navigator](#)

[Operation destinations](#)

.PlayBackTrace

Syntax

```
PlayBackTrace Type, Filename
```

Description

Plays back a recorded trace of host commands or data.

Parameters

Type DataStream Trace

Filename Type the filename of the .hst file to open. Supply complete path information.

More information

[Adding an operation](#)

.ReadFromMappedAttr

Syntax

```
ReadFromMappedAttr "Attribute"
```

Description

If you select this command as part of an operation, you must map an attribute to a variable on the [Attribute Variables](#) tab to enable an API developer to read this value but not write to it, using the Host Integrator connectors.

Command Parameters

Attribute Select a defined attribute to create a value for the selected variable.

Notes:

- This command is only available if you configured the Attribute Variables tab in the Entity window.
- ReadFromMappedAttr and WriteToMappedAttr commands are disabled in table procedures.

More information

[Configuring variables](#)

[.WriteToMappedAttr command](#)

.ReadVarFromAttr

Syntax

```
ReadVarFromAttr "Attribute", "Variable"
```

Description

Saves an attribute in a variable.

Command Parameters

Attribute Select a defined attribute from the list to create a value for the selected variable.

Variable Select a variable that defines the value of the attribute selected above.

More information

[Configuring variables](#)

.ReadVarFromField

Syntax

```
ReadVarFromField "Recordset", "Field", "Variable"
```

Description

Copies the data from a field in the selected recordset into a variable.

Command Parameters

Recordset Select the recordset in which the field exists.

Field Select a configured field from this list. Select the Fields tab on the Recordset tab to create a field.

Variable Select a configured variable value to be used for the field and recordset specified above.

More information

[Configuring variables](#)

.ReadVarFromLocation

Syntax

```
ReadVarFromLocation "Variable", Row, Column, Length
```

Description

Transmits the terminal data as a variable to the specified row and column coordinates. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Command Parameters

Variable Select a variable to be represented as the value of the terminal data entered into the specified location on the screen.

Row Argument type: Integer

The row to which the cursor is to be moved. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

Col Argument type: Integer

The column to which the cursor is to be moved. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

Length Argument type: Integer

Transmits the length of the variable to the row and column coordinates specified above.

More information

[Configuring variables](#)

.ReadVarFromTerminal

Syntax

```
ReadVarFromTerminal "Variable", Length
```

Description

Saves terminal data of a specified length in the selected variable.

This command reads terminal screen text into a Host Integrator variable, starting at the terminal's cursor position, and reading up to [length] characters. This can include the OIA and divider line (IBM hosts), VT status line, or HP function key definitions.

Note: When the OIA (operator information area), divider line characters, and other uncommon special characters are copied to the variable, they will not be displayed the same way that they appear in the Design Tool terminal window. The terminal text character's byte values are copied to the Verastream Host Integrator variable without regard to the font used to render them in the design tool's terminal window.

Command Parameters

Variable Select the variable to be used when this length is recognized on the terminal screen.

Length Argument type: Integer

The length of the data string.

More information

[Configuring variables](#)

.ResetRecordset

Syntax

```
ResetRecordset "Recordset"
```

Description

Clears the cached record data and resets the specified field of a recordset.

Command Parameter

Recordset Select the defined recordset to reset.

More information

[.TransmitToFieldEncrypted command](#)

.SelectLine

Syntax

```
SelectLine LineNumber
```

Description

Selects the line specified in the **LineNumber** box.

Command Parameters

LineNumber The row of the selection. You can select **Mouse Row** (rcMouseRow) to specify the row where the mouse was last clicked or select Cursor Position Row (rcCursorPosRow) to specify where the cursor was last positioned. You can also use the down and up arrows to select a line using an integer.

.SelectWord

Syntax

```
SelectWord Row, Column
```

Description

Selects the word in the Terminal window between the selection start position (or row 1, column 1, if there is no selection start position) and the specified coordinates. To set the selection start position, use the [SetSelectionStartPos](#) command or click with the mouse in the Terminal window.

Command Parameters

Row Argument type: Integer

The row in which the selection is to end. You can use rcMouseRow to specify the row where the mouse was last clicked.

Column Argument type: Integer

The column in which the selection is to end. You can use rcMouseCol to specify the column where the mouse was last clicked.

.SetColorMap

Syntax

```
SetColorMap Attribute, Foreground, Background
```

Description

Assigns a color to a terminal character attribute.

**Command Parameters8*

Attribute Select a terminal attribute. The default is **Error line**.

Foreground Argument type: Enumeration

The text (foreground) color being assigned. Colors are based on 24-bit RGB format. The possible values are:

- White
- Grey
- Red
- Blue
- Green
- Yellow
- Cyan
- Magenta
- Black
- Dark Grey
- Dark Red
- Dark Blue
- Dark Green
- Dark Yellow
- Dark Cyan
- Dark Magenta

Background Argument type: Enumeration

The background color being assigned. Colors are based on 24-bit RGB format. The possible values are the same as for **Foreground**.

.SetMousePos

Syntax `SetMousePos Row, Column`

Description

Sets the mouse position to the specified row and column coordinates.

Command Parameters

Row Argument type: Integer

The row to which the variable will be written. The default is Cursor Position Row. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

Column Argument type: Integer

The column to which the variable will be written. The default is Cursor Position Column. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

.SetNumeric

Syntax

```
SetNumeric Setting = Value
```

Description

Provides a way of linking a property value to an action or event (such as a toolbar button, keystroke, or hotspot) that can be mapped or linked to a Host Integrator command. Use SetNumeric to change the value of any property whose data type is Boolean, numeric, or enumeration.

Command Parameters

Setting Argument type: Enumeration

Select a setting from the list. On the Settings menu, click View Settings to open the View Settings dialog box for more information.

Value Select a value to assign to the numeric setting selected. The allowable values vary, depending on what setting is chosen as the first command argument.

.SetScanCodeName

Syntax

```
SetScanCodeName ScanType, ScanCode, Key
```

Description

Assigns a name to a key's scan code. You only need to use SetScanCodeName if you're using a nonstandard keyboard—that is, a keyboard with different key names than a standard keyboard, or a keyboard whose keys generate scan codes that differ from those generated by a standard keyboard.

Command Parameters

ScanType Argument type: Enumeration

The type of scan code generated by the key.

There are three possible values for this argument:

- **Extended**—An extended scan code is one that starts with E0.
- **Extended E1**— Like Extended, except the scan code starts with E1 instead of E0.
- **Normal**— If the scan code does not start with E0 or E1, use Normal as your ScanType.

ScanCode Argument type: Integer

The decimal value of the scan code.

Key Argument type: String

Specifies a string that you create for the key name. One possible name is the name of the key in question, but Host Integrator accepts any alphanumeric string.

.SetSelectionStartPos

Syntax

```
SetSelectionStartPos Row, Column
```

Description

Sets starting coordinates for selecting text. You can also set the selection start position by clicking in the Terminal window with the mouse. If no selection start position has been set, the default is row 1, column 1.

Command Parameters

Row Argument type: Integer

The row in which the selection begins. You can use rcMouseRow to specify the row where the mouse was last clicked.

Column Argument type: Integer

The column in which the selection begins. You can use rcMouseCol to specify the column where the mouse was last clicked.

.SetTerminalKbdPos

Syntax

```
SetTerminalKbdPos Left, Top, Width, Height
```

Description

Adjusts the location and size of the terminal keyboard window. The effects of SetTerminalKbdPos are only apparent when the terminal keyboard window is visible and not attached to the Terminal window margin.

The terminal keyboard can be resized with `SetTerminalKbdPos`, but not reshaped. If the width and height values that you assign do not match the current shape of the terminal keyboard, Host Integrator generally uses the specified width and then automatically sets height to preserve the shape.

Command Parameters

Left Argument type: Integer

The new location for the left margin of the terminal keyboard (in number of pixels from the left margin of the screen).

Top Argument type: Integer

The new location for the top margin of the terminal keyboard window (in number of pixels from the top margin of the screen).

Width Argument type: Integer

The new width (in pixels) for the terminal keyboard window.

Height Argument type: Integer

The new height (in pixels) for the terminal keyboard window.

.SetUpdateOffset

Syntax

```
SetUpdateOffset Offset, Length, Relative
```

Description

Transmits an active cursor to the offset location after the `WaitForUpdate` command is satisfied. Use the `CursorRow` and `CursorColumn` properties to verify the cursor location.

Command Parameters

Offset Argument type: Integer

The row and column coordinates on a terminal screen. The offset is zero-based and is the equivalent of the following formula:

$$\text{offset} = [(\text{row} - 1) \times \text{width}] + (\text{col} - 1)$$

The reverse formulas (offset to row/col) are as follows:

$\text{row} = (\text{offset} / \text{width}) + 1$, where the division operation uses integer logic (remainder of the division operation is ignored)

col = (offset % width) + 1, where % means modulo (use only the remainder of a division operation)

Example: For an 80 column terminal screen:

row 1, col 1 = offset zero (always true)

row 1, col 2 = offset 1 (always true)

row 2, col 1 = offset 80 *(depends on the screen width)

Length Argument type: Integer

The length of the active offset.

Relative Argument type: Enumeration

Select whether or not to make this offset relative to the cursor's current location by choosing either Yes or No.

More information

[.SetUpdateRegion command](#)

.SetUpdateRegion

Syntax

```
SetUpdateRegion Row, Column, Height, Width, Relative
```

Description

Sets up an active area on the terminal screen after the [WaitForUpdate](#) command has been satisfied. Use the [CursorRow] ([../set_summary/#current-cursor-row](#)) and [CursorColumn](#) properties to verify the cursor location.

Command Parameters

Row Argument type: Integer

The row to which the cursor is to be moved. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

Column Argument type: Integer

The column to which the cursor is to be moved. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

Height Argument type: Integer

The height of the region.

Width Argument type: Integer

The width of the region.

Relative Argument type: Enumeration

Select whether or not to make this region relative to the cursor's current location by selecting either **Yes** or **No**.

More information

[.SetUpdateOffset command](#)

.SetupSession

Syntax

```
SetupSession SessionType, TerminalModel, TransportType
```

Description

Defines the necessary settings for establishing a session. These settings must be supplied with this command, rather than independently, with properties, because they are interdependent. The following command sets up a 3270 terminal session emulating an extended Model 2 terminal using a Telnet transport:

```
SetupSession rc3270Terminal, rc3270MODEL2E, rcTelnet
```

This method fails if you run it when Host Integrator is connected to a host or if the arguments you use are not compatible with each other.

Command Parameters

SessionType Argument type: Enumeration

The type of device to emulate. The possible values are:

HP Terminal * Configures the Design Tool to emulate an HP terminal.

IBM 3270 Terminal

* Configures the Design Tool to emulate a 3270 terminal.

IBM 5250 Terminal * Configures the Design Tool to emulate a 5250 terminal.

VT Terminal * Configures Design Tool to emulate a VT terminal.

TerminalModel Argument type: Enumeration

The terminal to emulate. The possible values vary according to the value of both the `SessionType` and `TransportType` arguments. To determine if a particular model is available for the transport you have selected, open the [Session Setup](#) dialog box on the Connection menu.

TransportType Argument type: Enumeration

The transport to use. The options include **Telnet**, **Telnet Extended**, and **NS/VT**.

Note: NS/VT is only applicable when connecting to HP3000 hosts.

.SetWindowPos

Syntax

```
SetWindowPos Left, Top, Width, Height
```

Description

Adjusts the location, shape, and size of the Terminal window. If you attempt to set the Terminal window to a size larger or smaller than is feasible, `SetWindowPos` sets the Terminal window to the largest or smallest size possible.

Command Parameters

Left Argument type: Integer

The new location for the left margin of the Terminal window (in number of pixels from the left margin of the screen).

Top Argument type: Integer

The new location for the top margin of the Terminal window (in number of pixels from the top margin of the screen).

Width Argument type: Integer

The new width (in pixels) for the Terminal window.

Height Argument type: Integer

The new height (in pixels) for the Terminal window.

.ShiftCursor

Syntax

```
ShiftCursor Row, Column
```

Description

Shifts the cursor to a location relative to the current position of the cursor. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Command Parameters

Row Argument type: Integer

The row to which the cursor is to be shifted relative to its current position. Mouse Row (`rcMouseRow`) is the default. The minimum value is 1. The maximum value varies according to the number of rows supported by the host the Design Tool is emulating.

Column Argument type: Integer

The column to which the cursor is to be shifted relative to its current position. Mouse Column (`rcMouseCol`) is the default. The minimum value is 1. The maximum value varies according to the number of rows supported by the host the Design Tool is emulating.

More information

[Creating command lists](#)

.StartRecording

Syntax

```
StartRecording
```

Description

Starts recording a command list. On the Model menu, point to Record and click Start Recording to begin recording commands. To stop the recorder, click Stop Recording.

Command Parameters Mode Argument type: Enumeration

The **User** option is always in effect when mapping keys to the StartRecording command. The **System** option is no longer used.

.StartTrace

Syntax

```
StartTrace Type, ExistOption, FileName
```

Description

Sets a new value for a numeric setting.

Command Parameters

Type Argument type: Enumeration

The type of trace file. The only option is **Datastream Trace**.

ExistOption Argument type: Enumeration

Specifies an action to take if the file you specify already exists:

Ask User (rcAskUser)

Prompts the user to overwrite the existing file, append new data to the existing file, or cancel the save. An error results if the user then clicks Cancel.

Open Error (rcOpenError)

Causes StartTrace to return an error.

Overwrite (rcOverwrite)

Replaces the existing file.

FileName Argument type: String

The file to which commands are recorded.

.SwitchToWindow

Syntax

```
SwitchToWindow Window
```

Description

Switches to the specified window. This command executes asynchronously (that is, it does not stop command list execution—the calling procedure continues to run).

Command Parameters

Window Select Next Window (rcNextWindow) to switch to the next window.

.TerminalMouse

Syntax

```
TerminalMouse Type, Row, Column
```

Description

Equivalent to clicking in the specified location with the terminal mouse. Use the SetMousePos command to position the mouse before using TerminalMouse to simulate a click.

Command Parameters

Type Argument type: Enumeration

Specifies which mouse action to simulate. The values are:

Left Button Down (rcLeftDown)

Simulates pressing down the left button.

Left Button Up (rcLeftUp)

Simulates releasing the left button.

Left Click (rcLeftClick)

Simulates a single-click with the left button.

Left Double Click (rcLeftDbClick)

Simulates a double-click with the left button.

Middle Button Down (rcMiddleDown)

Simulates pressing down the middle button.

Middle Button Up (rcMiddleUp)

Simulates releasing the middle button.

Middle Click (rcMiddleClick)

Simulates a single-click with the middle button.

Middle Double Click (rcMiddleDbClick)

Simulates a double-click with the middle button.

Right Button Down (rcRightDown)

Simulates pressing down the right button.

Right Button Up (rcRightUp)

Simulates releasing the right button.

Right Click (rcRightClick)

Simulates a single-click with the right button.

Right Double Click (rcRightDbClick)

Simulates a double-click with the right button.

Row Argument type: Integer

Specifies a row in the host screen. To simulate a click of the terminal mouse in the same location as the PC mouse, use rcMouseRow for this argument.

Column Argument type: Integer

Specifies a column in the host screen. To simulate a click of the terminal mouse in the same location as the PC mouse, use rcMouseCol for this argument.

Note: In AS/400 sessions, you can simulate a light pen selection by using rcLeftClick for the Type argument, and specifying row and column values for a pen-detectable field. (For 3270 sessions, use the LightPen command to simulate a light pen selection.)

.Toggle

Syntax

```
Toggle Setting
```

Description

Changes the value of a Boolean property, such as ConditionCheckCommand. (A Boolean property is one that takes True and False as its only values. The data type for each property is given immediately below the syntax in that property's Help topic.)

You can explicitly set the value of a property. For example:

```
ConditionCheckCommand = True ConditionCheckCommand = False
```

In these examples, the results are not influenced by the current setting. Using the Toggle method always changes the current setting to its opposite. For example:

```
Toggle ConditionCheckCommand
```

This will turn this setting from on or off, depending on the current setting.

Command Parameters

Setting Argument type: Enumeration

Select from any of the available settings listed.

.TransmitANSI

Syntax

```
TransmitAnsi "AnsiString", Echo
```

Description

Transmits an ANSI string to the host.

Command Parameters

String Argument type: String

Specifies an ANSI string. This argument can be up to 260 characters long.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for same number of characters to echo**— Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. When TransmitANSI is received, Host Integrator takes the current cursor position and adds the number of characters it is going to transmit. To detect the number of echoed characters, Host Integrator transmits the data and waits until the cursor is at this calculated position. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. Note: With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for next tabstop**— Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for cursor to arrive at specific location**— Select this option to wait for the cursor to arrive at a specified location.

More information

[.TransmitTerminalKey command](#)

.TransmitANSIEncrypted

Syntax

```
TransmitANSIEncrypted "EncryptedAnsiString", Echo
```

Description

Transmits an obfuscated ANSI string to the host.

Strings are obfuscated not encrypted. Obfuscated strings are not as secure as encrypted strings. You should not store host passwords in your model files.

Command Parameters

String Argument type: String

Specifies an encrypted ANSI string. This argument can be up to 260 characters long.

Echo

If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop**— Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo**— Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. **Note:** With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**—Select this option to wait for the cursor to arrive at a specified location.

More information

[.TransmitAnsi command](#)

.TransmitTerminalKey

Syntax

```
TransmitTerminalKey Key
```

Description

Transmits a terminal key to the host.

Command Parameters

Key Argument type: Enumeration

The terminal key transmitted to the host.

The terminal keys for 3270, 5250, VT, and HP terminals are listed below. The listing is alphabetical.

Note: The **HP Terminal Key** list contains several Hp keys, specifically the HpCtrl keys, that are not listed in the Terminal keys list accessible from the right arrow button in the Operation Edit dialog box. To access these keys, use the **Key** list in the **Command parameters** box.

3270 Terminal Keys	5250 Terminal Keys	VT Terminal Keys	HP Terminal Keys
rcIBMAltCursorKey	rcIBMAttnKey	rcVt0Key	rcHpBackspaceKey
rcIBMAttnKey	rcIBMAU1Key	rcVt1Key	rcHpBackTabKey
rcIBMBackspaceKey	rcIBMAU2Key	rcVt2Key	rcHpBreakKey
rcIBMBackTabKey	rcIBMAU3Key	rcVt3Key	rcHpClearDisp
rcIBMClearKey	rcIBMAU4Key	rcVt4Key	rcHpClearLine
rcIBMClearPartitionKey	rcIBMAU5Key	rcVt5Key	rcHpConfigKey
rcIBMCursorBlinkKey	rcIBMAU6Key	rcVt6Key	rcHpCtrl@
rcIBMCursorSelectKey	rcIBMAU7Key	rcVt7Key	rcHpCtrl[
rcIBMDeleteCharKey	rcIBMAU8Key	rcVt8Key	rcHpCtrl]
rcIBMDeleteWordKey	rcIBMAU9Key	rcVt9Key	rcHpCtrl^
rcIBMDestrBackSpaceKey	rcIBMAU10Key	rcVtBELKey	rcHpCtrl_
rcIBMDeviceCancelKey	rcIBMAU11Key	rcVtBreakKey	rcHpCtrlA
rcIBMDownKey	rcIBMAU12Key	rcVtBSKey	rcHpCtrlB
rcIBMDupKey	rcIBMAU13Key	rcVtCANKey	rcHpCtrlC
rcIBMEndOfFieldKey	rcIBMAU14Key	rcVtComposeKey	rcHpCtrlD
rcIBMEnterKey	rcIBMAU15Key	rcVtCRKey	rcHpCtrlE
rcIBMEraseEofKey	rcIBMAU16Key	rcVtCursDKey	rcHpCtrlF
rcIBMEraseInputKey	rcIBMBackSpaceKey	rcVtCursLKey	rcHpCtrlG
rcIBMFieldDelimKey	rcIBMBackTabKey	rcVtCursRKey	rcHpCtrlH
rcIBMFieldMarkKey	rcIBMClearKey	rcVtCursUKey	rcHpCtrlI

3270 Terminal Keys	5250 Terminal Keys	VT Terminal Keys	HP Terminal Keys
rcIBMHomeKey	rcIBMDeleteKey	rcVtDC1Key	rcHpCtrlJ
rcIBMIdentKey	rcIBMDestrBackSpaceKey	rcVtDC2Key	rcHpCtrlK
rcIBMInsertKey	rcIBMDowndoubleKey	rcVtDC3Key	rcHpCtrlL
rcIBMKeyClickKey	rcIBMDownKey	rcVtDC4Key	rcHpCtrlM
rcIBMLeftDoubleKey	rcIBMDuplicateKey	rcVtDecimalKey	rcHpCtrlN
rcIBMLeftKey	rcIBMEndOfFieldKey	rcVtDelKey	rcHpCtrlO
rcIBMNewLineKey	rcIBMEnterKey	rcVtDisconnectKey	rcHpCtrlP
rcIBMNextWordKey	rcIBMEraseEOFKey	rcVtDLEKey	rcHpCtrlQ
rcIBMPa1Key	rcIBMEraseInputKey	rcVtEMKey	rcHpCtrlR
rcIBMPa2Key	rcIBMExtgrKey	rcVtENQKey	rcHpCtrlS
rcIBMPa3Key	rcIBMF1Key	rcVtEnterKey	rcHpCtrlSlash
rcIBMPanLeftKey	rcIBMF2Key	rcVtEOTKey	rcHpCtrlT
rcIBMPanRightKey	rcIBMF3Key	rcVtEscKey	rcHpCtrlU
rcIBMPartitionJumpKey	rcIBMF4Key	rcVtETBKey	rcHpCtrlV
rcBMPf1Key	rcIBMF5Key	rcVtETXKey	rcHpCtrlW
rcBMPf2Key	rcIBMF6Key	rcVtF1Key	rcHpCtrlX
rcBMPf3Key	rcIBMF7Key	rcVtF2Key	rcHpCtrlY
rcBMPf4Key	rcIBMF8Key	rcVtF3Key	rcHpCtrlZ
rcBMPf5Key	rcIBMF9Key	rcVtF4Key	rcHpDeleteCha
rcBMPf6Key	rcIBMF10Key	rcVtF5Key	rcHpDeleteCha
rcBMPf7Key	rcIBMF11Key	rcVtF6Key	rcHpDeleteKey
rcBMPf8Key	rcIBMF12Key	rcVtF7Key	rcHpDeleteLine
rcBMPf9Key	rcIBMF13Key	rcVtF8Key	rcHpDownKey
rcBMPf10Key	rcIBMF14Key	rcVtF9Key	rcHpEnterKey
rcBMPf11Key	rcIBMF15Key	rcVtF10Key	rcHpEscapeKey
rcBMPf12Key	rcIBMF16Key	rcVtF11Key	rcHpExtendKey

3270 Terminal Keys	5250 Terminal Keys	VT Terminal Keys	HP Terminal Keys
rcIBMPf13Key	rcIBMF17Key	rcVtF12Key	rcHpF1Key
rcIBMPf14Key	rcIBMF18Key	rcVtF13Key	rcHpF2Key
rcIBMPf15Key	rcIBMF19Key	rcVtF14Key	rcHpF3Key
rcIBMPf16Key	rcIBMF20Key	rcVtF15Key	rcHpF5Key
rcIBMPf18Key	rcIBMF22Key	rcVtF17Key	rcHpF6Key
rcIBMPf19Key	rcIBMF23Key	rcVtF18Key	rcHpF7Key
rcIBMPf20Key	rcIBMF24Key	rcVtF19Key	rcHpF8Key
rcIBMPf21Key	rcIBMFieldExitKey	rcVtF20Key	rcHpHardRese
rcIBMPf22Key	rcIBMFieldMinusKey	rcVtFFKey	rcHpHoldKey
rcIBMPf23Key	rcIBMFieldPlusKey	rcVtFindKey	rcHpHomeDov
rcIBMPf24Key	rcIBMHelpKey	rcVtFSKey	rcHpHomeLeft
rcIBMPrintKey	rcIBMHexKey	rcVtGSKey	rcHpHomeRigh
rcIBMPrintPartKey	rcIBMHomeKey	rcVtHoldScreenClearKey	rcHpHomeUpk
rcIBMResetKey	rcIBMInsertKey	rcVtHoldScreenKey	rcHpInsertCha
rcIBMRightDoubleKey	rcIBMLeftDoubleKey	rcVtHoldScreenSetKey	rcHpInsertCha
rcIBMRightKey	rcIBMLeftKey	rcVtInsKey	rcHpInsertLine
rcIBMScrollDownKey	rcIBMNewLineKey	rcVtKp0Key	rcHpInsertWra
rcIBMScrollUpKey	rcIBMNextWordKey	rcVtKp1Key	rcHpLeftKey
rcIBMSysReqstKey	rcIBMPA1Key	rcVtKp2Key	rcHpLeftArrow
rcIBMTabKey	rcIBMPA2Key	rcVtKp3Key	rcHpMenuKey
rcIBMUpKey	rcIBMPA3Key	rcVtKp4Key	rcHpModesKey
	rcIBMPageDownKey	rcVtKp5Key	rcHpNextPage
	rcIBMPageUpKey	rcVtKp6Key	rcHpPageUpK
	rcIBMPlusCRKey	rcVtKp7Key	rcHpPageDow
	rcIBMPrintKey	rcVtKp8Key	rcHpReturnKey
	rcIBMRightDoubleKey	rcVtKpCommaKey	rcHpRightKey

3270 Terminal Keys	5250 Terminal Keys	VT Terminal Keys	HP Terminal Keys
	rcIBMRightKey	rcVtKpDotKey	rcHpRightArrowKey
	rcIBMRollDownKey	rcVtKpMinusKey	rcHpRollDownKey
	rcIBMRollUpKey	rcVtLFKey	rcHpRollLeftKey
	rcIBMRuleKey	rcVtNAKKey	rcHpRollRightKey
	rcIBMSlpautoEnterKey	rcVtNextPageKey	rcHpRollUpKey
	rcIBMSOSIGenerateKey	rcVtNextScrKey	rcHpScrollDownKey
	rcIBMSysReqstKey	rcVtKpNULKey	rcHpScrollUpKey
	rcIBMTabKey	rcVtPf1Key	rcHpSelectKey
	rcIBMTestKey	rcVtPf2Key	rcHpSmartEnterKey
	rcIBMUpDoubleKey	rcVtPf3Key	rcHpSoftResetKey
	rcIBMUpKey	rcVtPf4Key	rcHpStopKey
		rcVtPrevPageKey	rcHpSystemKey
		rcVtPrevScrKey	rcHpTabKey
		rcVtPrtDKey	rcHpUpKey
		rcVtRemoveKey	rcHpUserKey
		rcVtRetrnKey	rcHpUserConfKey
		rcVtRSKey	
		rcVtScrollDownKey	
		rcVtScrollUpKey	
		rcVtSelectKey	
		rcVtSendNullKey	
		rcVtSIKey	
		rcVtSOHKey	
		rcVtSOKey	
		rcVtStopKey	
		rcVtSTXKey	

3270 Terminal Keys	5250 Terminal Keys	VT Terminal Keys	HP Terminal Keys
		rcVtSUBKey	
		rcVtSYNKey	
		rcVtTabKey	
		rcVtUdk6Key	
		rcVtUdk7Key	
		rcVtUdk8Key	
		rcVtUdk9Key	
		rcVtUdk10Key	
		rcVtUdk11Key	
		rcVtUdk12Key	
		rcVtUdk13Key	
		rcVtUdk14Key	
		rcVtUdk15Key	
		rcVtUdk16Key	
		rcVtUdk17Key	
		rcVtUdk18Key	
		rcVtUdk19Key	
		rcVtUdk20Key	
		rcVtUSKey	
		rcVtVTKey	

.TransmitToAttr

Syntax

```
TransmitToAttr "Attribute", "String"
```

Description

Transmits a specified String to an attribute value.

Command Parameters

Attribute Select a defined attribute from the list to create a value for the selected variable.

String Argument type: String

Specifies a data string.

More information

[Configuring variables](#)

.TransmitToAttrEncrypted

Syntax

```
TransmitToAttrEncrypted "Attribute", "String"
```

Description

Transmits an obfuscated data String to a configured attribute.

Strings are obfuscated not encrypted. Obfuscated strings are not as secure as encrypted strings. You should not store host passwords in your model files.

Command Parameters

Attribute Select a defined attribute to create a value for the encrypted data string.

String Argument type: String Specifies an encrypted data string.

More information

[Configuring variables](#)

.TransmitToField

Syntax

```
TransmitToField "Recordset", "Field", "String"
```

Description

Transmits the string to the specified field of a recordset.

Command Parameters

Recordset Select the defined recordset on which the field exists.

Field Select the defined field on which to copy the string value. Use the Fields tab to add fields to recordsets.

String Argument type: String

Transmits this string to the field location specified above.

More information

[How to add a recordset](#)

.TransmitToFieldEncrypted

Syntax

```
TransmitToFieldEncrypted "Recordset", "Field", "String"
```

Description

Transmits the obfuscated string to the specified field of a recordset.

Strings are obfuscated not encrypted. Obfuscated strings are not as secure as encrypted strings. You should not store host passwords in your model files.

Command Parameters

Recordset Select the defined recordset on which the field exists.

Field Select the defined field to copy the string value to. Use the [Fields](#) tab to add fields to recordsets.

String Argument type: String

Transmits this encrypted string to the field location specified above. The value is encrypted with asterisks when it's saved to the [model file](#).

More information

[How to add a recordset](#)

.TransmitToLocation

Syntax

```
TransmitToLocation Row, Column, "String", Echo
```

Description

Transmits the string to the specified row and cursor location on the terminal screen. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Command Parameters

Row Argument type: Integer

The row to which the cursor is to be moved. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

Column Argument type: Integer

The column to which the cursor is to be moved. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

String Argument type: String

Transmits this string to the row and column coordinates specified above.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop**—Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the [Cursor](#) tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor** —Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo** —Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**—Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. Note: With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**—Select this option to wait for the cursor to arrive at a specified location.

More information

[.TransmitToLocationEncrypted command](#)

[.TransmitToLocationEncrypted](#)

Syntax

```
TransmitToLocationEncrypted Row, Column, "String", Echo
```

Description

Transmits an obfuscated string to the specified row and cursor location on the terminal screen. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Strings are obfuscated not encrypted. Obfuscated strings are not as secure as encrypted strings. You should not store host passwords in your model files.

Command Parameters

Row Argument type: Integer

The row to which the cursor is to be moved. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

Column Argument type: Integer

The column to which the cursor is to be moved. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

String Argument type: String

Transmits this encrypted string to the row and column coordinates specified above. Asterisks are used to encrypt the string as you type.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop** —Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the [Cursor](#) tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**—Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo** —Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor** —Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. Note: With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location** —Select this option to wait for the cursor to arrive at a specified location.

More information

[.TransmitToLocation command](#)

[.TransmitToOffset](#)

Syntax

```
TransmitToOffset Offset, "String", Echo
```

Description

Transmits the string to the specified offset location. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Command Parameters

Offset Argument type: Integer

The row and column coordinates on a terminal screen. See the status bar at the bottom of the terminal screen for the offset value of the cursor position.

The offset is zero-based and is the equivalent of the following formula:

$$\text{offset} = [(\text{row} - 1) \times \text{width}] + (\text{col} - 1)$$

The reverse formulas (offset to row/col) are as follows:

row = (offset / width) + 1, where the division operation uses integer logic (remainder of the division operation is ignored)

col = (offset % width) + 1, where % means modulo (use only the remainder of a division operation)

Example: For an 80-column terminal screen:

row 1, col 1 = offset zero (always true) row 1, col 2 = offset 1 (always true) row 2, col 1 = offset 80
*(depends on the screen width)

String Argument type: String

Transmits this string to the row and column coordinates specified above.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will **not** be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop**— Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo**— Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. **Note:** With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**— Select this option to wait for the cursor to arrive at a specified location.

More information

[.SetUpdateOffset command](#)

.TransmitToOffsetEncrypted

Syntax

```
TransmitToOffsetEncrypted Offset, "String", Echo
```

Description

Transmits the obfuscated string to the specified offset location. Use the [CursorRow](#) and [CursorColumn](#) properties to verify the cursor location.

Strings are obfuscated not encrypted. Obfuscated strings are not as secure as encrypted strings. You should not store host passwords in your model files.

Command Parameters

Offset Argument type: Integer

The row and column coordinates on a terminal screen. See the status bar at the bottom of the terminal screen for the offset value of the cursor position.

The offset is zero-based and is the equivalent of the following formula:

$$\text{offset} = [(\text{row} - 1) \times \text{width}] + (\text{col} - 1)$$

The reverse formulas (offset to row/col) are as follows:

row = (offset / width) + 1, where the division operation uses integer logic (remainder of the division operation is ignored)

col = (offset % width) + 1, where % means modulo (use only the remainder of a division operation)

Example: For an 80-column terminal screen:

row 1, col 1 = offset zero (always true) row 1, col 2 = offset 1 (always true) row 2, col 1 = offset 80
*(depends on the screen width)

String Argument type: String

Transmits this encrypted string to the row and column coordinates specified above. As you type the value, asterisks will appear to encrypt the value of the string.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will **not** be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop**— Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**—Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo**— Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. **Note:** With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**— Select this option to wait for the cursor to arrive at a specified location.

More information

[.SetUpdateOffset command](#)

.UpdateAttribute

Syntax

```
UpdateAttribute "Attribute name"
```

Description

Writes cached attribute data to the terminal screen. In an operation, attribute data can be assigned using write commands like DefaultValue or TransmitToAttr. This command is used in conjunction with the [Cache all attribute writes](#) check box on the General tab of the Advanced Entity Properties dialog box.

Notes:

- By default, an UpdateAttribute command will be inserted into each generated operation when Cache all **attribute writes** is selected.
- This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When attribute data assignments are arbitrarily mixed between an operation and a Host Integrator API, the order in which data is actually written to the terminal can be important to the host application.

Command Parameters

Attribute Select a configured attribute.

More information

*.DefaultValue command

.UpdateAttributes

Syntax

```
UpdateAttributes
```

Description

Writes all cached attribute data to the terminal screen. In an operation, attribute data can be assigned using write commands like DefaultValue or TransmitToAttr. An operation can call the UpdateAttributes command for all attributes that have assigned data. This command is used in conjunction with the [Cache all attribute writes](#) check box on the General tab of the Advanced Entity Properties dialog box.

Notes:

- By default, an UpdateAttribute command will be inserted in each generated operation when **Cache all attribute writes** is selected.
- This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When attribute data assignments are arbitrarily mixed between an operation and a Host Integrator API, the order in which data is actually written to the terminal can be important to the host application.

More information

[.DefaultValue command](#)

.UpdateRecordsetField

Syntax

```
UpdateRecordsetField "Recordset", "Field"
```

Description

Writes the specified cached recordset field data to the terminal screen. This command is used in conjunction with the [Cache all field writes] (../entity-recordset-tab/#cache-all-field-writes) option. To globally configure all recordset field data to be cached on new entities, select the Cache all field writes box on the Recordset Preferences dialog box. To cache recordset field data on a selected entity, select the entity in the Entity window and select the **Cache all field writes** (../settings-menu-preferences/#cache-all-field-writes) box on the Recordset Fields tab.

Notes:

- By default, an UpdateAttribute command will be inserted in each generated operation when **Cache all field writes** (../settings-menu-preferences/#cache-all-field-writes) is selected.
- This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When field data assignments are arbitrarily mixed between an operation and a Verastream connector API, the order in which data is actually written to the terminal can be important to the host application.

Command Parameters

Recordset Select the configured recordset that contains the cached recordset field data to be written to the terminal screen once this command executes.

Field Select the configured recordset field to be written to the terminal screen.

More information

[.WaitForUpdate command](#)

.UpdateRecordsetFields

Syntax

```
UpdateRecordsetFields "Recordset"
```

Description

Writes all cached recordset field data to the terminal screen. In an operation, field data can be assigned using attribute/field input commands like DefaultValue or TransmitToAttr. This command is used in conjunction with the [Cache all field writes](#) check box on the Recordset Fieldstab in the Entity window.

Notes:

- By default, an UpdateRecordsetField command will be inserted into each generated operation when **Cache all field writes** is selected.
- This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When field data assignments are arbitrarily mixed between an operation and a Host Integrator API, the order in which data is actually written to the terminal can be important to the host application.

Command Parameters

Recordset Select the configured recordset that contains the cached recordset field data to be written to the terminal screen once this command executes.

More information

Attribute/field input commands

.Wait

Syntax

```
Wait Duration
```

Description

Waits the specified period of time before executing the next command in the operation. The operation containing the command is paused until the duration expires.

Command Parameters

Duration Argument type: String

Specifies an interval of time in HH:MM:SS format.

More information

[.WaitForCommString](#) command

[.WaitForDisplayString](#) command

[.WaitForUpdate](#) command

.WaitForCommString

Syntax

```
WaitForCommString "CommString", "Timeout", "Count"
```

Description

Begins a wait that is satisfied by the reception of the specified string from the host. The operation containing the command is paused until the data communication string is received or the timeout expires.

When connecting to VT and HP character mode hosts, you can wait for datastream sequences that include [nonprintable characters](#). As the terminal processes characters, you can copy and paste text from the Model Debug Messages dialog box into the **String** box.

Notes:

Using the right mouse button, open the shortcut menu to copy and paste text from the Model Debug Messages dialog box. You can also use the Ctrl-C keystroke to copy the text; however, do not use the Ctrl-V keystroke to paste the text. Instead, use the Paste button provided next to the String box.

Often the best synchronization for VT and HP is in the datastream itself. Coupled with the WaitForMultipleEvents command, it will enable you to be very deterministic about synchronization.

Use WaitForCommString only if you are connecting to your host using a host application that uses character mode datastreaming rather than block mode. This applies to VT and HP character mode models only.

If you are connecting directly to a host, use WaitForDisplayString.

You can also record a login command list to determine which command is appropriate for your connection.

Command Parameters

Count Argument type: Numeric

Specifies the number of times a string is received from the host. The default value is 1.

String Argument type: String

The string that can terminate the wait. This string must be received from the host.

Note: Use the right-mouse button menu to copy and paste text instead of the Ctrl-C and Ctrl-V keystrokes. A Paste button appears when the Host Integrator accepts special characters.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForCondition

Syntax

```
WaitForCondition "Entity", "Condition", "Timeout"
```

Description

Waits for the condition to be satisfied. The operation containing the command is paused until the condition is satisfied or the timeout expires.

Command Parameters

Entity Argument type: String

Specifies the entity on which the condition will be satisfied.

Condition Click the Edit button to open the Condition Edit dialog box for the command argument. Use this dialog box to create a condition that is specific to the entity selected above.

For example, create the following condition that addresses cursor syncing when the cursor is expected to arrive at one of two possible locations:

```
(Variables.CursorRow = 5 And Variables.CursorColumn = 27) Or (Variables.CursorRow = 24 And Variables.CursorColumn = 1)
```

Timeout Specifies an interval of time to wait for the condition in seconds.

More information

[Editing a condition](#)

.WaitForCursorAtAttr

Syntax

```
WaitForCursorAtAttr "Entity", "Attribute", "Timeout"
```

Description

Begins a wait that is satisfied when the specified attribute data is received from the host. The operation containing the command is paused until the wait expires or is satisfied.

Use `WaitForCursorAtAttr` only if you are connecting to your host using a host application that uses character mode rather than block mode. (In character mode, incoming characters appear on screen in sequence, and existing lines of text scroll out of sight as new lines appear. In block mode, you see complete screen displays, beginning with the login display screen. You can also record a [login command list](#) to determine which command is appropriate for your connection.

Command Parameters

Entity Argument type: String

The entity on which the attribute is defined.

Attribute Argument type: String

The attribute data that can terminate the wait.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForCursorAtLocation

Syntax

```
WaitForCursorAtLocation Row, Column, "Timeout"
```

Description

Begins a wait that is satisfied when the cursor is at a position. The operation containing the command is paused until the cursor is at the position or the timeout expires.

Command Parameters

Row Argument type: Integer

For the wait to be satisfied, the cursor must be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Column Argument type: Integer

For the wait to be satisfied, the cursor must be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for the cursor to be at the specified coordinates. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForCursorAtTerminalField

Syntax

```
WaitForCursorAtTerminalField Row, Column, "Timeout"
```

Description

Waits for the cursor to be at the specified terminal field (as identified by the Row and Column arguments). The operation containing the command is paused until the cursor is at the field or the timeout expires.

Command Parameters

Row Argument type: Integer

For the wait to be satisfied, the cursor must be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Column Argument type: Integer

For the wait to be satisfied, the cursor must be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for the cursor at the specified coordinates. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForCursorNotAtLocation

Syntax

```
WaitForCursorNotAtLocation Row, Column, "Timeout"
```

Description

Waits for the cursor to be at any screen coordinates other than specified coordinates (as identified by the Row and Column arguments). The operation containing this command is paused until the cursor is no longer at the specified coordinates or the timeout expires.

Command Parameters

Row Argument type: Integer

For the wait to be satisfied, the cursor must not be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Column Argument type: Integer

For the wait to be satisfied, the cursor must not be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for the cursor to not be at the specified coordinates. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForCursorNotAtTerminalField

Syntax

```
WaitForCursorNotAtTerminalField Row, Column, "Timeout"
```

Description

Waits for the cursor to not be at the specified terminal field (as identified by the Row and Column arguments). The operation containing the command is paused until the cursor is no longer at the field or the timeout expires.

Command Parameters

Row Argument type: Integer

For the wait to be satisfied, the cursor must not be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Column Argument type: Integer

For the wait to be satisfied, the cursor must not be at the field at the specified row and column coordinates. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for the cursor to not be at the field at the specified coordinates. If this method times out, an error occurs.

More information

[.WaitForDisplayString command](#)

[.Wait command](#)

.WaitForCursorAtRecordsetField

Syntax

```
WaitForCursorAtRecordsetField "Entity", "Recordset", "Timeout"
```

Description

Begins a wait that is satisfied when the specified recordset field data is received from the host. The operation containing the command is paused until the wait expires or is satisfied.

Use WaitForCursorAtRecordsetField only if you are connecting to your host using a host application that uses character mode rather than block mode. (In character mode, incoming characters appear on screen in sequence, and existing lines of text scroll out of sight as new lines appear. In block mode, you see complete screen displays, beginning with the login display screen.) You can also record a [login command list](#) to determine which command is appropriate for your connection.

Command Parameters

Entity Argument type: String

The entity on which the attribute is defined.

Recordset Argument type: String

The recordset that can terminate the wait.

Field Argument type: String

The record data that can terminate the wait.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForDisplayString

Syntax

```
WaitForDisplayString "String", "Timeout", Row, Column, Relative
```

Description

Begins a wait that is satisfied by the appearance of the specified string at the specified location on the terminal screen. The string can be received from the host or typed by the user. The operation is paused until the display string is received or the timeout expires.

**Command Parameters*

String Argument type: String

The string that can terminate the wait.

Timeout Argument type: String

Specifies a time interval in HH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

Row Argument type: Integer

The row in which the string must be received. You can specify Any Column or Any Row (which shows us as -1 in a trace).

Column Argument type: Integer

The column in which the string must be received. You can specify Any Column or Any Row (which shows us as -1 in a trace).

Relative

Specifies whether or not the position of the display string is determined relative to the terminal cursor position. The default is **No**.

More information

[Wait command](#)

.WaitForHostSilence

Syntax

```
WaitForHostSilence "Duration", "Timeout"
```

Description

Begins a wait that is satisfied by the absence of data traveling to or from the host. The operation containing the command is paused until the host silence occurs or the timeout expires. By specifying a duration of time, Host Integrator will expect the silence to last that long to satisfy the wait.

Command Parameters

Duration Argument type: String

Specifies an interval of time in HH:MM:SS format. This is how long the period of silence must last.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for a period of silence. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForKeyboardEnabled

Syntax

```
WaitForKeyboardEnabled "Duration", "Timeout"
```

Description

Begins a wait that is satisfied by the keyboard being unlocked for the specified duration. The operation containing the command is paused until the keyboard is enabled or the timeout expires. When you specify a duration of time, Host Integrator will expect the keyboard to be enabled that long to satisfy the wait. This command applies to 3270/5250 models only.

Command Parameters

Duration Argument type: String

Specifies an interval of time in HH:MM:SS format. Use Duration to specify how long the keyboard must remain enabled. The specified event must last this long to satisfy the wait.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. Use the Timeout argument to specify how long to wait for the keyboard to be enabled. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForMultipleEvents

Syntax

```
WaitForMultipleEvents "EventExpression", "Timeout"
```

Description

Begins a wait that is satisfied by the occurrence of a series of global events. This command enables Host Integrator to rely on an order of events rather than the number of data packets that are sent from a character mode host. Since Host Integrator handles only one command per packet of data, the WaitForMultipleEvents command is designed to encapsulate several global events into one unique command. The operation containing this command is paused until all of the waits expire or are satisfied.

To define global events that can be used with this command on any entity, click Events on the Model menu to open the Event Edit dialog box. To create conditional expressions and sequences using global events, click the Edit button to open the Event Expression Editor dialog box. To view the datastream coming from the host, open the Model Debug Messages dialog box.

Command Parameters

EventExpression Click the Edit button to open the [Event Expression Editor](#) dialog box and create conditional expressions and sequences using defined global events. Use the buttons in this dialog box to create a string of global events that is specific to the selected entity.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. The wait terminates if this amount of time passes without the specified events occurring. If this method times out, an error occurs.

More information

[.Wait command](#)

[.WaitForNewHostScreen](#)

Syntax

```
WaitForNewHostScreen "Timeout"
```

Description

Begins a wait that is satisfied with a new screen from the host. A new screen is recognized in a 3270/5250 environment by an End of Chain indicator, and by a minimum of 4 bytes of data in a character mode environment. The operation containing the command is paused until the new host screen is returned or the timeout expires.

Command Parameters

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format. The wait terminates if this amount of time passes without the new host screen being returned. If this method times out, an error occurs.

More information

[.Wait command](#)

.WaitForUpdate

Syntax

```
WaitForUpdate TopRow, LeftCol, BottomRow, RightCol, RegionType, Relative, "Timeout"
```

Description

Waits for the host to update the specified region on the terminal screen before the timeout expires.

Command Parameters

TopRow Argument type: Integer

The top row of the updated region. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

LeftCol Argument type: Integer

The left column of the updated region. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating. You can also specify Any Column or Any Row (which shows us as -1 in a trace).

BottomRow Argument type: Integer

The bottom row of the updated region. You can specify Any Column or Any Row (which shows us as -1 in a trace).

RightCol Argument type: Integer

The right column of the updated region. You can specify Any Column or Any Row (which shows us as -1 in a trace).

RegionType

Specifies what mode to use for the region selected on the terminal screen. Select either the **Linear** or **Rectangular** position. The default is **Rectangular**.

Relative Argument type: Enumeration

Select whether or not to make this region relative to the cursor's current location by selecting either **Yes** or **No**. The default is **No**.

Timeout Argument type: String

Specifies an interval of time in HH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

More information

[Wait command](#)

.WaitMS

Syntax

```
Wait Duration
```

Description

Waits the specified period of time in milliseconds before executing the next command in the operation. The operation containing the command is paused until the duration expires.

Command Parameters

Duration Argument type: String

Specifies an interval of time in milliseconds.

More information

[.WaitForCommString command](#)

.WriteToMappedAttr

Syntax

```
WriteToMappedAttr
```

Description

If you select this command as part of an operation, you must map an attribute to a variable on the Attribute Variables tab to enable an API developer to change this value using the Host Integrator connectors.

With this command, the value will always be based on the variable associated with the attribute as shown on the Variable tab in the lower portion of the Attribute tab. This allows you to remap the variable-attribute relationship without making similar changes to the related operations.

Command Parameters

Attribute Select a defined attribute to create a value for the selected variable.

Notes:

This command is only available if you configured the Attribute Variables tab on the Entity window.

ReadFromMappedAttr and WriteToMappedAttr commands are disabled in table procedures.

More information

[Adding an attribute](#)

[.ReadFromMappedAttr command](#)

.WriteVarToAttr

Syntax

```
WriteVarToAttr "Attribute", "Variable"
```

Description

Writes the value of the variable to the specified attribute.

Command Parameters

Attribute

Select a configured attribute.

Variable

Select a defined variable to be copied to the attribute's location.

More information

[Mapping an attribute to a variable](#)

[Configuring variables](#)

.WriteVarToField

Syntax

```
WriteVarToField "Recordset", "Field", "Variable"
```

Description

Writes the value of the variable to the specified field.

Command Parameters

Recordset

Select the defined recordset on which the field exists.

Field

Select the defined field to copy the string value to. Use the Fields tab to add fields to a recordset.

Variable

Select a defined variable to be copied to the specified field.

More information

[Mapping an attribute to a variable](#)

[Configuring variables](#)

.WriteVarToLocation

Syntax

```
WriteVarToLocation "Variable", Row, Column, MaxLength, Erase, Echo
```

Description

Writes the value of the variable to the specified row and column coordinates.

Command Parameters

Variable

Select a defined variable to be copied to the row and column specifications.

Row Argument type: Integer

The row to which the variable will be written. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

Column Argument type: Integer

The column to which the variable will be written. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of columns supported by the terminal model the Design Tool is emulating.

MaxLength Argument type: Integer

The maximum number of characters to write to the terminal.

Erase

Represents whether or not to erase the variable contents after navigating to another entity. Select **Yes** or **No**.

Note: If you select **Yes** and the length of the variable contents is less than MaxLength, all other characters up to MaxLength will be deleted.

Echo If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop**— Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**- Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop.
- **Wait for same number of characters to echo**— Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**— Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. **Note:** With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**— Select this option to wait for the cursor to arrive at a specified location.

More information

[Mapping an attribute to a variable](#)

[Configuring variables](#)

.WriteVarToTerminal

Syntax

```
WriteVarToTerminal "Variable", MaxLength, Erase, Echo
```

Description

Copies the variable value to the current terminal cursor position.

Command Parameters

Variable Select a defined variable to be copied to the terminal position.

MaxLength Argument type: Integer

The maximum number of characters to write to the terminal.

Erase

If you select **Yes** and the length of the variable contents is less than MaxLength, all other characters up to MaxLength will be erased.

Echo

If you are modeling a block mode application, select the **Don't wait for echo** option. If you are modeling a character mode application, select from the following options:

- **Don't wait for echo** (Default)—Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for next tabstop** —Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the [Cursor](#) tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor**—Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.
- **Wait for same number of characters to echo**—Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor**—Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. **Note:** With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for cursor to arrive at specific location**—Select this option to wait for the cursor to arrive at a specified location.

More information

[Mapping an attribute to a variable](#)

[Configuring variables](#)

4.15.5 Troubleshooting Tips and Techniques

Click these links to reference troubleshooting information for different aspects of your host application model.

Troubleshooting:

[Timeout Errors](#)

[Error Patterns and Error Entities](#)

[Tracing](#)

[Recordsets](#)

[Tables](#)

[Host Connections](#)

Timeout Errors

A timeout is the default length of time that a computer will wait for a task to be completed before the task is canceled. The cancellation of a task is usually indicated by an error message. Timeouts can almost always be adjusted by the user or system administrator.

Host Integrator uses multiple timeouts; client connection timeouts, connector method timeouts, host connection timeouts, and so forth. If you encounter a timeout error, it does not necessarily mean that you should increase the duration of the timeout. For example, if the host application is not arriving at an expected entity within 10 seconds, waiting longer is not going to correct the situation. You should first investigate the error logs and make targeted model improvements.

CONNECTOR TIMEOUT

If a table procedure is returning a large amount of data (such as navigating too many host screens or scrolling a large recordset), then the connector timeout may need to be increased so the client-side doesn't give up too early.

CLIENT METHOD TIMEOUT

A client method timeout may occur if you have a procedure returning a large number of records, or any operation or event handler that takes a long time to complete. For information on revising the method timeout to address client timeouts. See [KnowledgeBase Article 10044](#).

Error Patterns and Error Entities

There are useful techniques associated with using error entities and error patterns when modeling a host application, specifically when dealing with character mode host applications. Using error conditions helps you to detect when an error has occurred while you're modeling and testing your host application model. Error patterns catch error messages displayed on the current entity, while error entities handle errors that result in a host error screen being displayed.

When creating both error patterns and error entities, it's possible to create a customized error message that enables you to detect what kind of error has occurred during an operation.

Note

In the Design Tool, if an error occurs and an operation completes before an error pattern is displayed or an error entity is reached, then error processing will not occur. For example, if an operation's destination entity is the same as its start entity and an error message is expected to display when the destination is reached, the action triggered by that error message may be bypassed. The operation needs to include a command, like a `WaitForCursorAtLocation` command, that ensures that the error pattern is displayed or the error entity is reached before the operation completes.

ERROR PATTERNS OR ERROR ENTITIES

When handling an error condition, you sometimes have the choice of setting up an error pattern or an error entity. When evaluating which to use, remember that error patterns require less processing overhead and are therefore recommended. If you're dealing with a problem associated with a starting screen, you will need to use an error entity.

Trapping error messages that result from writing attribute data to the terminal screen is often helpful in identifying errors. This is usually applicable to character mode host applications that produce error messages while you are writing data to the terminal screen.

To create an error pattern to trap a host application error message:

Encapsulate the error message in a pattern and add it to the entity.

On the Pattern tab, clear the Use in entity signature check box.

Click the Operation tab in the Entity window.

Click the Conditions button to open the Operation Conditions dialog box.

Under Error patterns, move the pattern from the Available list to the Error patterns list.

Under Error pattern, you can configure a text or an attribute value to be returned if an entity encounters an error pattern during an operation.

Click Close to close the Operation Conditions dialog box and save any changes.

Remember:

Operation conditions can contain error patterns.

No action can be associated with an error pattern.

An error pattern can be a particular host message or a non-blank host message.

Error patterns can be recognized any time during the operation processing.

Error Entities

A user-defined error entity enables you to handle a host error screen by defining a custom error message or returning attribute data as your error message when the error entity is encountered. Make sure to define an operation on this error entity that enables you to navigate back to your original entity. Do not define an error entity as your home entity; if error handling is required, use error patterns instead.

Select the entity from the Name box on the Entity window and click the Operation tab.

Click the Destinations button to open the Operation Destinations dialog box.

Select an entity from the Unassigned entities box and click the right arrow button to add it to the User-defined error entities box.

You can also select one of the following options from the Operation Destinations dialog box to customize an error message to appear when your error entity is reached:

Remember:

- Return text — Select this option to create any return text that will be displayed if the error entity is encountered during execution of this operation.
- Return attribute contents — Select one or more attributes to return as the contents of your user defined error message.



Note

You can choose to define error entities in operations or in procedures. If you are using procedures, you can choose to add the error entity to your procedure. If you've already defined an error entity in an operation, it is not necessary to also define an error entity in your procedure. To define a custom error message when using procedures, use the Procedure Editor. Additionally, the error message feature within the Procedure Editor does not allow you to assign the contents of an attribute to the error message.

Tracing

The Design Tool provides the following types of tracing to aid you in the modeling process:

Host Emulator recording

Datastream tracing (for 3270 and 5250 block mode datastreams only)

Debug tracing

HOST EMULATOR RECORDING

Host Emulator recordings capture negotiations between entities in your model and can capture the data sent to the host at the transport level. When you use the Host Emulator, these files can be played back to simulate the host environment without having a direct connection to the host. This feature is helpful if you plan to use the Host Emulator as a model testing tool during production and for offsite model development.

To create trace (.trc) files for the Host Emulator

On the Connection menu, point to Host Emulator Recording and choose Start. You must start the recording before you make the host connection.

Choose the file seed name and location for the recording. Recordings are saved, by default, in `<My Documents>\Micro Focus\Verastream\HostIntegrator\recordings`. The file is saved with a .trc or .trc_GROOMED extension.

Click Trace. For every new session a new recording file is created.

Step through your model entities including any login and logout command lists. This records all the data traffic with the host.

Click Stop to terminate the recording. The trace file is groomed automatically, which makes it suitable to be played back immediately.

Recordings are played back using the options in the Administrative Console. You can disable automatic grooming using the View Settings option available from the Settings menu.

DATASTREAM TRACING

Datastream tracing interprets 3270 and 5250 datastreams by capturing data transmissions to and from the host during a live connection. Tracing is intended primarily for diagnostic purposes. Use the model debug messages feature for more options on interpreting the datastream.

On the Connection menu, point Datastream Trace, and select Start Trace.

Type a name in the File name box and click Trace. By default, the .hst file is saved in the `\<VHI install directory>\etc\tracing` folder.

Navigate through the entities in question, and select Stop Trace.

To play back the datastream trace:

On the Connection menu, point to Datastream Trace, and select Play Back Trace.

Select the .hst file and click Play.

The trace navigation displays on the Terminal window, which enables you to see the sequencing of host writes to the terminal screen to track navigation in offline mode.

Playing back a trace does not redisplay the user's input and does not require a live host connection.

DEBUG TRACING

Debug tracing is a more intricate form of configuration tracing that is used to help debug the product. This form of tracing should only be used if the other forms of tracing have failed. Make sure to contact a technical support representative before proceeding with a debug trace. A trace configuration file (.cfgtrc file) and an actual trace file (.trace file) results from a debug trace.

Some reasons to create a debug trace:

- Possible transport difficulties.

- Session problems on the Host Integrator Server

To start and stop a debug trace:

Click the Debug menu, point to Debug Trace and select Start Debug Trace. The Start Debug Trace dialog box opens.

Enter a name for trace file in the File name box.

Click Start. From this point forward, all data transmissions to and from the host are recorded.

To stop the trace, click the Debug menu, point to Debug Trace and select Stop Debug Trace.

Recordsets

Modifying records in a recordset from the Design Tool can create synchronization problems between the Design Tool and the host. For example, if Host Integrator inserts the first record in a recordset and there are no indicators that it should wait for the host to commit to a modification and make a new blank record appear, Host Integrator reports this as an error. The issue is that the Design Tool refreshes the internal view of the screen in the Terminal window before the host has received the new data.

Use the Model Debug Messages utility to compare what the model expects the screen to be against the host data stream.

INSERTING AND UPDATING RECORDS

To fix this modification issue, create an operation using one or more Host Events commands that makes Host Integrator wait until the host is done returning the new input record. Then, configure Host Integrator to execute this operation after a record insert or record update. This operation will finalize a recordset change and write the inserted or updated record to the host. This can be accomplished in the Design Tool by configuring the options in the Inserting records box or the Updating records box of the Recordset Operations dialog box or with a performOperation method using an AppConn data object.

TESTING RECORDSET FETCHING AND SELECTING

The following provide examples of recordset debugging that can be accomplished with the Test Recordset dialog box:

Fetch Records

- Only one screen of data is returned

On the Recordset Tab, open the Recordset Operations dialog box. Check to see if you are missing a Page down operation.

- No data is returned

Check the Current record index. If it is not zero, then set the current record index to zero and retry the fetch.

- Fetching records does not terminate. The Break button must be clicked.

Check the termination condition of your recordset in the Recordset Termination dialog box.

If the termination condition is correct, check the Page down operation to ensure that it is truly paging down on the host. A Page down operation that doesn't actually page through the host data may cause the termination condition to fail.

- First screen is returned twice.

The Page Down operation should not complete until the entire screen is displayed. The type of Wait operation for this situation varies by terminal type. - A number of blank records are returned along with the data.

In the Recordset Options dialog box, select Skip blank records.

ANY ACTION

- After executing an action, the Test Recordset dialog box just displays the message, "The current entity does not contain a recordset."

The action that you performed navigated you to a different screen. For selection, this is generally the expected result. If this happens as the result of a fetch or an insert, there may be a problem with a scrolling operation or an insertion operation resulting in unexpected navigation.

SELECT RECORD

- A selection operation did not reach the expected destination. AND It did navigate to a different screen.

There is probably something wrong with the selection operation itself. Check to make sure that the expected destination and alternate/intermediate destinations for the selection operation are correct.

- A selection operation reached the expected destination, but the wrong record was selected.

Open SIDemo.modelx in the Design Tool.

Select CustomerPurchases from the Entity box.

Select the Recordset tab, and click the Operations button.

In the Scrolling box, notice that LineDown is selected from the Line down box.

To view details about the LineDown operation configured here, open the Operation tab and select LineDown from the Name box.

Tables

- Executing an SQL query results in no records found

If you execute an SQL query against a table and do not find any records even though they exist, the Design Tool is unable to resolve your query into a valid procedure: You need to create one or more procedures that can access the table data.

- A valid procedure returns no data

It's possible that a post-fetch filtering operation removed the data you were trying to retrieve. Table data is case sensitive: Be sure you are using the correct case when trying to retrieve data

Host Connections

You may not be able to connect to the host for the following reasons:

- The host and transport settings you specified in the Session Setup dialog box are not being recognized by the Design Tool. Verify your settings are correct and try connecting again.
- If you see a blank screen after establishing a host connection, the problem may be that the host has sent a disconnect to the Design Tool. Some hosts do this if they don't recognize the terminal type you have selected. When the host sends a disconnect, the Design Tool displays the following message in red on the bottom of the terminal screen:

```
Your connection to <host name> has been terminated
```

- To solve this, select a different model ID in the Session Setup dialog box and try connecting again.
- The host may be unavailable temporarily or the network hardware may be malfunctioning. Check with your system administrator or try connecting again later.
- The network hardware may be malfunctioning. Check with your system administrator or try connecting again later.

If you suspect a host problem could be associated with running the model itself, try deleting a login command list if present. This will allow you to see the individual steps involved in the sequence and may resolve any issues in timing associated with the command list.

4.15.6 Using Custom Keystores and Certificates

When Host Integrator is installed it generates and stores a key and certificate in a keystore for secure access to the Host Integrator session server and Web server.

HTTPS to Host Integrator Web services - `https://vhiserver:9681`

HTTPS to the Host Integrator Web server that runs applications generated by Web Builder -
`https://vhiserver:8443`

To resolve browser/client certificate security warnings, if you do not want to trust the self-signed certificates, you can provide custom keystores and CA-signed security certificates.

Note

The key and certificate chain provided by your Certificate Authority (CA) must use FIPS validated algorithms and strengths.

To use a CA-signed certificate in Host Integrator Web services

The SOAP stack uses the certificate for authenticating itself to HTTPS clients.

The key and certificate chain provided by your CA must be in a keystore in either BCFKS format or a PKCS12 format with strong encryption (PBE-SHA1-3DES). Rename the file `server.bcfks` and copy it over the existing `server.bcfks` file in folder `%VHI_ROOT%/sesssrvr/etc`.

2. Locate the Java `keytool.exe` utility in the following directory:

3. Windows: `C:\Program Files\Micro Focus\Verastream\java\bin`

4. Linux: `/opt/microfocus/verastream/java/bin`

5. Run `keytool` with an appropriate command line, including the following parameters:

`-importcert` to store the certificate in the keystore

`-keystore` to specify the `server.bcfks` file name, including path from step 1 above

`storetype bcfks` to specify the keystore type

`alias server-container` to specify the alias used inside the keystore

6. When prompted for a password enter `not-secure`. Both the key and keystore must use that password.

Restart the session server.

For more information on using the Java `keytool`, see the [Oracle documentation](#).

To use a CA-signed certificate in the Host Integrator Web server

This certificate is used for HTTPS to the Host Integrator Web server.

The key and certificate chain provided by your CA must be in a keystore in BCFKS format.

The password for the key and for the keystore must be the same.

- Open the `%VHI_ROOT%/servletengine/conf/container.properties` file and add the following three lines:

```
servletengine.ssl.keystore =full path to keystore
```

```
servletengine.ssl.keystoretype =format name of keystore,, either BCFKS or PKCS12
```

```
servletengine.ssl.keystorepassword =password for the keystore file you specified
```

Restart the Web server.

More information

[About Web Services](#)

4.15.7 Entity Window Options

Entity Settings

The Entity window contains all of the settings used to define the host screens that make up the model. The Entity box contains all of the currently defined entity names with visual indicators to indicate whether each entity listed is "reachable" from the current location via dynamic traversal or navigation.

When you are in offline mode, all screens are considered reachable and display a green icon since the screen shots are loaded from a .snapshot file which has also recorded the navigation of the model. Next to the Entity box, there are three buttons that allow a user to add a new entity, delete the current one, or view the Advanced Entity Properties dialog box.

There are five tabs available from the Entity panel:

[Pattern](#)

[Attribute](#)

[Operation](#)

[Recordset](#)

[Cursor](#)

See [Adding Entities to a Model](#) for information on how to add and define your entity.

Entity Pattern Tab

Note

Auto-generating patterns simplifies the modeling process, but you may need to delete some auto-generated patterns later to enhance server performance. If you choose to have the Design Tool auto-generate patterns, you can configure properties recognition on the Pattern tab of the Preferences Setup dialog box. In addition, patterns cannot be generated automatically for a VT host.

- Name - Defines the name or names of the currently defined patterns. When you add a new pattern to an entity, a default name, such as *Pattern_1*, is generated and default settings are applied in the Position, Properties, and Pattern use boxes. If a pattern is no longer valid (for example, the layout of the screen on the host has changed), the pattern is marked with a red X in the list after running the Entity Repair utility or setting the model to offline mode.
- Position - Defines the position settings (Row, Height, Column, and Width) that describe the actual position of the pattern on the host screen. First, select a Selection mode (Linear or Rectangular).

By default, the pattern is assumed to always be located in the same place on the terminal window. If the pattern can occur in another position, use one or both of the following to describe where to look to for the pattern:

- Search for pattern relative to cursor

Relative to cursor means relative to initial cursor position, not relative to the current Terminal window cursor position. If you select Relative to cursor, click Update Initial Cursor Position on the Model menu to change the initial cursor position to the current cursor position. If you have configured any entity properties relative to a cursor position, these property coordinates automatically update. Click Apply to save your changes or click Cancel to revert your initial cursor position back to its original state.

- Search for pattern in expanded region

The actual pattern or "text" can float inside a larger region defined in the Position box. In these instances, the positioning noted in the **Position** box functions more as a search region than an exact position. In default color configuration, select the **Position** settings for region using the green selection rectangle. Select the pattern definition using the red selection

rectangle, or use the settings on the Location tab. At runtime, the Host Integrator searches the entire region outlined in green for the text defined by the settings in the Location tab.

If both of these options are enabled, the expanded region is defined relative to the cursor position specified.

- Definition tab

Describes the physical characteristics of the pattern and provides a view of any characters appearing on that area of the host screen. The Design Tool displays different property settings for different types of terminals.

- Field type and text color

For IBM 3270 and 5250 hosts, the Text check box is selected by default when a new pattern is created. To view the field type or the text color of the pattern as recognized by the host, select the Field type or Text color check boxes.

These options vary based on your host type.

- Text

Displays the text in the selected area. You can also select < Any text >, < Any number >, < Blank >, or < Regular Expression >. Select the Case sensitive check box if you would like the data stored in your model as case-sensitive text.

The Design Tool cannot recognize patterns under the following conditions:

- A pattern containing a decimal point with a text property type defined as < Any number >.

- A pattern containing only blanks with a text property type defined as < Any text >.

Regular expressions example

You can use [regular expressions](#) to have screen recognition based on text that meets the criteria you specify. You may want to define an entity that handles a number of different errors on the host application status line.

To differentiate error messages from other host messages on the status line, create a regular expression that only matches when error messages are displayed on the status line, for example: `ERROR [0-9]{1, 4}: .*`

- Use in entity signature - Defines whether or not a pattern is included in the entity signature. By default, the **Use in entity signature** check box is selected, which requires that a pattern be present as part of a defined entity. To view details about a specific entity signature, click Signature Analyzer on the Debug menu.
- Screen properties not present - Defines whether or not screen properties for this pattern are recognized by the Design Tool. Select the **Screen properties not present** check box to make sure the properties described above for the pattern are not included in the pattern's entity signature.

By default, a pattern is considered "present" when Host Integrator finds its properties on the terminal screen in the correct location. In some cases, however, a pattern is "present" when

its properties are not on the terminal screen in the defined location. For example, when modeling the process of scrolling through a set of records such as a bank statement, the host may use a text prompt to tell the user how to scroll when there are more records, then remove the prompt on the final screen. In Host Integrator, a good way to define a **NoMoreRecords** pattern would be to select the host prompt text, define the pattern, and then select the **Screen properties not present** check box. The NoMoreRecords pattern is then considered "present" when the host prompt text is not.

- Location tab

If you selected *Search for pattern in expanded region*, the settings on this tab identify the location of the pattern definition. Select the region in the terminal window using the red selection rectangle or fill in the settings for row, column, height, and width (Rectangular mode), or offset and length (Linear mode). In default color configuration, this pattern definition area is outlined in red, while the Position settings for region are outlined in green.

Attribute Tab

Auto-generating attributes simplifies the modeling process, but you may need to delete unneeded auto-generated attributes before copying your model to the Host Integrator Server.

Attributes cannot be generated automatically for a VT host.


When you add a new attribute to your model, the following options will be available for configuring an attribute:

- Name

Specifies the name or names of the currently defined attributes. Under Name on the Attribute tab of the **Preferences Setup** dialog box, select from the following options:

- Start position

The options in the Start position box specify the actual beginning position of the attribute on the terminal screen.

 **Note**

The Relative to cursor option means relative to initial cursor position, not relative to the current Terminal window cursor position. If you select Relative to cursor, click Update Initial Cursor Position on the Model menu to change the initial cursor position to the current cursor position. If you have configured any entity properties relative to a cursor position, these property coordinates will be automatically updated.

- End position

The options in the End position box specify the actual ending position of the attribute on the terminal screen.

Make sure to select the Relative to pattern option if you have selected the Relative to pattern option in the Start position box.

The following tabs are available to configure attributes:

[Attribute Properties tab](#)

[Attribute Variables tab](#)

[Attribute Errors tab](#)

[Attribute Operations tab](#)

[Attribute Echo tab](#)

PROPERTIES TAB

The Attribute Properties tab contains the following options for configuring the basic properties of an attribute:

Property	Description
Type	Since configuring type options is possible for both attributes and recordset fields, select either the Attribute Properties tab or the Recordset Fields tab to view the Type box. These options allow you to specify whether or not data can be accessed by one of the Host Integrator connectors. If both Read and Write are cleared, the Host Integrator will reject any attempt to save the entity.
Read	When selected, Host Integrator reads the host data contained in this attribute or recordset field at runtime. This option is selected if protected fields exist on the entity. The Read check box should not be selected if the attribute or recordset field contains or is to contain secure information such as a password.
Write	When selected, Host Integrator writes host data to this attribute or recordset field at runtime. This check box is available even when an attribute or field is not writable to cover cases the protected state can change dynamically based on the data sent from the host. When configuring a write attribute, do not use the Any Row or Any Column to specify start position or end position.
Hidden	Select this check box to hide the existence of the attribute or recordset field from one of the Host Integrator connectors.

Property	Description
Enable terminal attributes	Select this check box to make terminal attributes accessible from a Host Integrator connector. You must select this check box to enable and read terminal attributes using one of the Host Integrator APIs, specifically when using the EnableTerminalAttributes method.

Advanced Attribute Properties

The Advanced Attribute Properties dialog box provides the following options:

Event Handler

Use the options below to create, edit, attach, and view properties of an attribute event handler.



- Click this button to create a new event handler for the attribute. Use the list to select an existing attribute event handler. The selected event handler is attached to the attribute when you click OK.



- Click the Edit button to open the event handler in your default editor.



- Click the Properties button to view event handler properties.

Attribute description

Type a description of the selected attribute. This data is included in Web Builder projects or in any exported XML- or HTML-based documentation documentation.

VARIABLES TAB

Use the Attribute Variables tab to map attributes to and from variables.

Attribute mapping to variables

Allows a user to map attributes to and from variables, so these attributes can be referenced as part of operations. This enables the user to gain a level of abstraction from attributes or share data among several entities. When a user wants to be able to traverse to several screens using one command, any data that needs to be returned can be automatically stored in a variable by selecting it from these two lists:

When executing a ReadFromMappedAttr command in an operation, update this variable list

When executing a WriteToMappedAttr command in an operation, obtain value from this variable list



Note

ReadFromMappedAttr and WriteToMappedAttr commands are disabled in procedures.

By default, the Design Tool provides the following variable options: **None**, **cursorPosition**, **password**, and **userID**. To create a new variable, click the New Variable button to open the Variables dialog box.

To update the data contained in the selected variable each time a new entity is encountered, select the **Always update this variable on arrival at entity** check box. If the **Override any prior attribute inputs** check box is selected, the Design Tool will overwrite any previous attribute inputs.

ERRORS TAB

The Attribute Errors tab is only available if an attribute is relative to a pattern. For example, if an error pattern is used to position an attribute, that pattern will not be part of the entity signature; consequently, the entity will be recognized by the host, but the pattern will not be found until the error is caused. Since the attribute's location is relative to this pattern, any request to read the attribute will fail prior to the error condition. Select one of the following options from the **If unable to locate a relative attribute during fetch** box to resolve this case:

Return blank string

Return fetch error

Return entered string - Use the text box provided to enter text to be returned on the host screen.

ATTRIBUTE OPERATIONS TAB

Configure operations you want to execute when writing to an attribute. Options are:

- Execute operation before writing to attribute

Select a configured operation to be executed before data is written to this attribute.

Create an operation that clears an attribute field before data is written to it. To see an example of a Clear operation, open the [Pine model](#) and navigate to the AddressUpdate entity. Select **Cutline** from the **Nickname** list and view the Command list box.

- Execute operation to autotab on underfill

Select a configured operation to be executed when attribute data does not completely fill in the length of the selected attribute.

Create an operation that automatically tabs to the next input field or an operation that traps an error. For example, if the data input is always a set length, such as a social security number, create an operation that includes an error pattern as an operation condition. When the error pattern is recognized, the operation will stop.

This option is typically used with character mode hosts. When data doesn't fill the attribute, the behavior of a character mode host application can often be less predictable than a block mode host application.

- Execute operation after writing to attribute

Select a configured operation to be executed after data is written to the selected attribute.

ECHO TAB

If you are working with a character mode host, such as HP or VT, direct the Host Integrator to treat this input data in any one of the following ways:

- **Don't wait for echo** (Default). Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character-mode applications, use this option only when you know that data will **not** be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for same number of characters to echo** Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- **Wait for string to echo at cursor** Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character-mode hosts.

 **Note**

With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.

- **Wait for next tabstop** Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the [Cursor](#) tab, then Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor** Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.

Operation Tab

Use the Operation tab to edit or define operations on a selected entity. If you've just added a new operation to your model, the following options are available:

Option	Description
Name	The name or names of the currently defined operations. When adding a new operation, a default name, such as <i>Operation_1</i> , is generated. You can edit the name. You can add, copy, delete, check advanced operation properties, or test your new operation using the toolbar buttons associated with the Name field. If you want to test an operation that is related to a recordset, use the Test Recordset dialog box instead of the Execute button.
Destination	Select the default destination for the current operation from this list. Once you select a destination entity, the Design Tool expects that the selected operation will traverse to it. If the commands in the operation do not lead to the specified destination, a runtime error will occur unless any alternative destinations are selected in the Operation Destination dialog.
Use for navigation commands to this destination	Two entities can have several operations defined to traverse between them; however, for a given entity, there should be at least one "default" operation defined to all adjacent entities. This means that when a dynamic traversal is requested and that traversal needs to jump from the current entity to an adjacent entity, the corresponding default operation will be used. To designate this default operation for a given destination entity, select the Use for navigation commands to this destination check box when the desired operation is selected in the Name field.
Command list	The list of commands used to navigate between entities. You can edit, test and debug, or copy the list. When you copy the list and paste it in a text editor, the entire list is visible.

Option	Description
Properties	Timeout - Specify in seconds how long a given operation should take to execute before a error message appears. Wait for host update when selected (default), the operation waits for the host to update any area of the screen before trying to recognize the screen and test for its arrival at a destination.

OPERATION CONDITIONS

Click **Conditions** to open the Operation Conditions dialog. Use this dialog to configure options that verify that certain conditions, such as pre-conditions, required attributes, and/or error patterns, have been satisfied before an operation is evaluated during execution.

The [CheckOperationConditions](#) command will fail if the condition fails during evaluation, which results in an operation exception at runtime. In order for these conditions to be verified, the [CheckOperationConditions](#) command is required as the first command in the operation. To cancel any changes and return to the default settings, click the Revert button.

The following options are available:

- Pre-condition

Assigning a pre-condition to an operation means that the pre-condition will have to be satisfied in order for this operation to be executed. To create a pre-condition for this operation, click the Edit button to open the Condition Edit dialog.

- Required attributes

To require that certain attributes are written to before validation of an operation, select the Required check box for one or more attributes.

- Error patterns

Error patterns enable you to set up user-defined errors in your model. A pattern is a candidate for being an error pattern if it is not considered to be part of an entity signature. By default, all auto-generated and manually created patterns are configured to be a part of the entity signature. To change this setting, clear the **Use in entity signature** box on the Pattern tab.

A good candidate for an error pattern might be an error message that appears on the terminal screen when a nonexistent account number is sent to the host.

- Select a pattern and add it to the Error patterns list, creating return text that displays when the error pattern is encountered.
- Select **Return attribute contents** to return one or more attribute values as the contents of your error message.

You will see an exception of the type 'user defined' indicating that an error pattern was trapped, followed by the name of the pattern and the contents of the attribute. Attributes are returned in listed order.

- Post-condition

Assigning a post-condition to an operation means that the post-condition must be satisfied after the operation has been executed so that it can be evaluated during operation execution. If the post-condition fails during evaluation, an operation exception is issued at runtime. To create a post-condition for an operation, click the Edit button to open the Condition Edit dialog.

Note

Post-conditions can only be used when the origin and destination of an operation are the same entity, for example, a page down operation. Defining a post-condition allows a final check to make sure the operation has navigated to the correct location.

Advanced Operation Properties

The Advanced Operation Properties dialog box provides the following options:

Event Handler

Use the options below to create, edit, attach, and view properties of an operation event handler.



- Click this button to create a new event handler for the attribute. Use the list to select an existing attribute event handler. The selected event handler is attached to the attribute when you click OK.



- Click the Edit button to open the event handler in your default editor.



- Click the Properties button to view event handler properties.

Operation description

Type a description of the selected operation. This data is included in Web Builder projects or in any exported XML- or HTML-based documentation documentation.


OPERATION DESTINATIONS

The Operation Destinations dialog contains settings that supply the Design Tool with methods of reaching alternative entities using the selected operation. Click the Destinations button on the Operation tab to open this dialog box.

Setting	Description
Unassigned entities	The names of all entities in the model that are not assigned to be navigated to with the selected operation. To enable any of these entities to be navigated to with the selected operation, move them to the Valid alternates box.
Right/Left arrows	Use to add or remove entities from unassigned list.
Valid alternates	Valid alternates are entities that can be traversed to during navigation of an operation. If an operation is executed and an entity in this box is its destination, the operation will be considered successful. It's an alternative outcome that is still valid.

Setting	Description
Intermediate entities	<p>Intermediate entities are considered to be a part of the operation but not a destination of the operation. For example, a Splash screen that is encountered during the operation, but is not the origin or destination of an operation, could be a good candidate for an intermediate entity. You can further define intermediate entities using the Execute operation and Issue these commands options. Select the Execute operation option to choose an operation to execute when this entity is encountered. Select Issue these commands to define commands that will bypass the intermediate entity in order for the operation to reach its destination. For example, define a terminal key command, such as <code>rcIBMEnterKey</code>, for the intermediate entity so the host recognizes that when the Enter key is pressed, it should immediately bypass this screen.</p>

Setting	Description
User-defined error entities	An error entity is a location that constitutes an error condition if it's detected during an operation. You can create a custom error message by selecting Return text and entering the text to be displayed when the error entity is encountered. Select Return attribute contents to return an attribute value as the contents of your error message.


 **Notes**

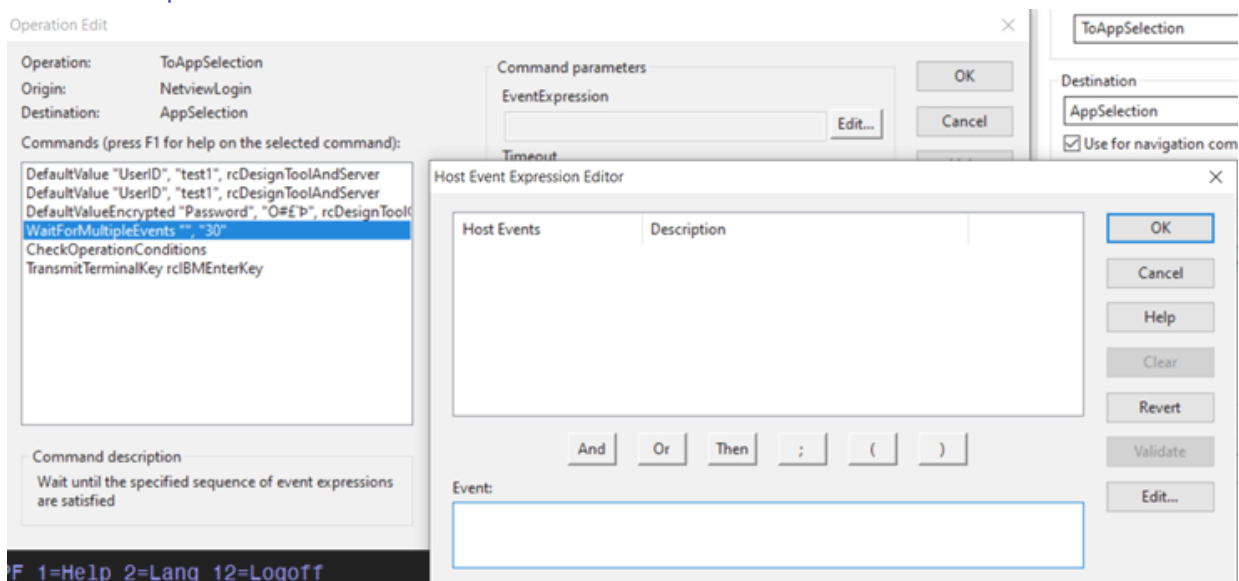
If you are using table procedures, you can choose to add the error entity to your procedure. To define a custom error message when using table procedures, use the Procedure Editor. The error message feature within the Procedure Editor does not allow you to assign the contents of an attribute to an error message. If the starting entity for your operation is also your error entity, be sure to clear the **Wait for host update** check box on the Operation tab. Otherwise, the error entity will not be recognized and you will not get an error until the operation times out. In this scenario, error patterns are strongly recommended as an alternative solution.

OPERATION EDIT

Use the Operation Edit dialog, available from Operation > Command list > Edit, to add, edit, copy, or delete commands in a command list. See [Operation Command Summary](#) for a description of each command.

HOST EVENT EXPRESSION EDITOR

To open this dialog: In the Operation Edit dialog, select a command and click  Host Events > WaitForMultipleEvents > Edit.



The events described are used to synchronize the Host Integrator server with the state of the host application. These host synchronization events should be distinguished from event handlers, which perform or extend actions in the model file itself.

Use the Host Event Expression Editor dialog box to string together several events into one `WaitForMultipleEvents` command using the following event operand buttons:

Button	Description
And	Use And to conjoin two events to create a conditional event that is not satisfied until both events have occurred.
Or	Use Or to conjoin two events to create a conditional event that is not satisfied until at least one of the two events has occurred.
Then	Use Then to conjoin two events or expressions to create a conditional expression that is not satisfied until both events have occurred in order from left to right.
;	Use ; to separate one statement from the next. Statements are evaluated sequentially.
()	Use () to group or combine events.

For example, if the following three global events are listed as follows:

Events	Description
Event A	<code>WaitForCommString \033EnterData</code>
EventB	<code>WaitForCursorEnterPosition 5,10</code>

Events	Description
EventC	WaitForCursorExitPosition 24, 2

You could then select one or more of the above events to formulate the following WaitForMultipleEvents command using the event operand buttons **And**, **Or**, **Then**, **,**, and **()**:

```
WaitForMultipleEvents "EventA and (EventB or EventC)", "30"
```

This means that Host Integrator will wait for EventA, and either EventB or EventC. If this series of events is not satisfied within the 30 second timeout period, an error will occur. It does not matter whether these events occur in a single host packet or across any number of multiple packets. These events may be received in any order.

Host events are processed in the order in which they are received from the host. If the order of events is critical, use the **Then** operator. The **Then** operator will not change the order of events received from the host, but it will ignore events or expressions on the right before events or expressions on the left are satisfied. For example:

```
EventA Then EventB
```

In the above example, the host could send events that might satisfy EventB, but those events will be ignored until EventA is satisfied. To further illustrate: `EventA Then EventA Then EventA` will be satisfied on the third occurrence of EventA.

And, **Or**, and **Then** expressions may be combined in any arbitrary order in a statement, for example:

```
(EventA Then EventB) And (EventC Or EventD) Then (EventE Then EventF Then EventG)
```

This is a valid statement.

The semicolon can be used to separate statements, as in Java and C++. Statements will be considered separately and sequentially. By separating statements, a statement separator may behave somewhat like the **Then** operator. However, the **Then** operator can be used within statements and combined with the other operators, **And** and **Or**.

Other options:

Validate - Click to perform syntax checking of the expression.

Event Edit – Click to close this dialog and open the Host Event Edit dialog.

IMPORT

This dialog, from Operation Edit > Import, provides a comprehensive list of existing model templates that can be reused during the modeling process.

To import an existing command list template:

1. Expand the **Command list** directory tree and select either an existing **Operation**, **Command List**, or **Template** to copy into the **Commands** box on the Operation Edit dialog box. If a green dot appears in front of the template name, it is ready to use.
2. Click OK to return to the Operation Edit dialog box and complete the configuration of the current command list.

Recordset Tab

The Recordset tab contains all of the settings used to define recordsets on a selected entity. If you've just added a new recordset to your model, the following options will be available:

The **Name** box contains the name or names of the currently defined recordsets. When you add a new recordset to an entity, a default name, such as `Recordset_1`, is generated. Use the buttons next to the **Name** box to add and delete recordsets. To further configure your new recordset, click the Advanced Properties or the Test button.

POSITION TAB

This tab contains options used to determine the position of a recordset on the terminal screen.

Top - Choose either Fixed Position, Relative to Cursor, or Relative to Pattern to describe the top position of the recordset on the host screen. Fixed Position is selected by default and the Row and Col coordinates are calculated for you.

Bottom - Choose either Fixed Row, Relative to Top, or Relative to Pattern to determine the bottom position of the recordset on the entity. Fixed Row is the default selection and the row coordinate is calculated for you.

Width - The width of the recordset in terminal screen columns. By default, the Design Tool provides this information automatically when a new recordset is added.

To configure advanced recordset features, click:

[Operations](#)

[Termination](#)

[Options](#)

Advanced Recordset Properties

The Advanced Recordset Properties dialog box provides the following options:

Event Handler

Use the options below to create, edit, attach, and view properties of an event handler for the recordset.



- Click this button to create a new event handler for the attribute. Use the list to select an existing attribute event handler. The selected event handler is attached to the attribute when you click OK.



- Click the Edit button to open the event handler in your default editor.



- Click the Properties button to view event handler properties.

Recordset description

Type a description of the selected recordset. This data is included in Web Builder projects or in any exported XML- or HTML-based documentation documentation.

RECORDSET OPERATIONS

Operations can be defined for recordsets on the Operation tab and then configured in this dialog.

Scrolling- Configure how to scroll through recordsets on the terminal screen by matching already defined operations with the following options:

Selection - Certain recordsets allow you to select a record and then move to another entity for more information about that record. To encapsulate this required action, create a selection operation.

To view an example of a selection operation:

Open the SIDemo example and navigate to the CustomerPurchases entity.

Open the Recordset Operations dialog box.

Under Selection, view the choices in the Selection operation list.

To view the commands in these operations, click the Operation tab and select an operation from the Name box.

View the commands in the Command list box.


From a data object or tables, the API developer can choose to `getCurrentRecord` (scroll to a specific record in a recordset and return its contents) or `fetchRecords` (fetch up to the maximum rows of data from the Host Integrator Server for the current recordset of the current entity). The expected result of selection is to navigate to the entity that contains the additional information. For more information, search for `getCurrentRecord` or `fetchRecords` in the Host Integrator API Reference.

If the host supports this navigation by selection, select the Host allows navigation by record selection check box and select a defined operation from the Selection operation list. To test this operation, click the Test button from the Recordset tab.

Inserting records - Some hosts allow you to directly insert records as part of a recordset on an entity. Use the options in this dialog box to configure your model to execute an operation before and after a single record insertion. If your host supports multiple record inserts, select the Host supports multiple record inserts check box described below and configure the model to instantiate the insertion using a combination of an operation stored in the model file and the new `InsertRecords` method available in your preferred Host Integrator connector.

Select the Host supports direct insertion check box and choose from the options described below.

Option	Description
Insert at first blank record	Select this option if the host allows you to insert a new record into the first blank record that appears on the terminal screen.
Insert at first record with these properties	Select this option and click the Edit button to open the Condition Edit dialog box. Specify certain properties to look for in a record, so the Host Integrator can alert the user that direct insertion of data is now possible. If you select this option, you will not be able to exit this dialog box until a condition is specified.
Execute operation before or after insert	Select this check box and map an operation to execute before or after inserting a new record. Executing an operation before or after a record is inserted is often helpful when working with character mode applications. For example, you may want to create an operation that uses the MoveCursor command to position the cursor on the entity before a new record is inserted. After the record is added, create another operation that transmits the cursor to the next position. You could also create an operation that transmits a terminal key using the TransmitTerminalKey command to alert the host that a new record is about to be added. Then, create another operation that tells the host the insertion is complete.
Host supports multiple record inserts	Select this check box if the host allows you to insert more than one record in a recordset of an entity.

If you attach an event handler to the recordset that has an Insert Record event implemented, a lightning bolt  displays to the left of this option. This is a reminder that the event handler may include logic that overrides or extends any of the settings you configure.

Updating records - Some hosts allow you to directly update a new record as part of a recordset on the terminal screen. To deal with this scenario, select the Host supports direct record update check box and choose from one or both of options below.

Option	Description
Execute operation before update	Select this check box and choose an operation to execute before changing the value of the new record. Example: Create an operation that clears the recordset field prior to writing using a combination of TransmitTerminalKey commands and MoveCursor commands.

Option	Description
Execute operation after update	Select this check box and choose an operation to execute after changing the value of the new record. Example: Create an operation that controls how the cursor is moved around the screen using one or more Host Events commands, such as <code>WaitForMultipleEvents</code> , to signal completion.

If you have configured an operation that executes prior to or after a group of records is to be inserted, invoke the operation using the new `PerformEntityOperation` method available with your preferred Host Integrator connector instead of the using the options in this dialog box. Use the Recordset Test dialog box to make sure your records are being properly added to the recordset.


Note

Modifying records in a recordset from the Design Tool can create synchronization problems between the Design Tool and the host. For example, if Host Integrator inserts the first record in a recordset and there are no indicators that it should wait for the host to commit to a modification and make a new blank record appear, Host Integrator reports this as an error. The problem is that the Design Tool refreshes the internal view of the screen in the Terminal window before the host has received the new data. To fix this, create an operation that uses one or more Host Events commands that makes Host Integrator wait until the host is done returning the new input record. Then, select the Execute operation after insert check box described above and choose the configured operation from the list.

RECORDSET TERMINATION OPTIONS

Set the options needed to recognize when the end of a recordset has been reached. On the **Scroll Up** tab configure the information needed to define how to recognize when the first page of the recordset is reached. Once the first page is reached, it no longer makes sense to continue scrolling up. Choose **Scroll Down** to configure how to determine when the end of a recordset has been reached and what to return from the data in the last page of the recordset.

Scrolling operations must be configured on the Operation tab before you can configure items on the following tabs. To view an example of a **PageDown** and a **PageUp** operation, open the **Pine** model and select the **AddressBook** entity.

If you attach an event handler to the recordset that has an `Is Terminated` event implemented, a lightning bolt  is displayed to the left of this option. This is a reminder that the event handler may include logic that overrides or extends any of the settings you configure below.

Scroll Up

Use this tab to define how Host Integrator is to recognize that it has reached the first page of a recordset (at which point it no longer makes any sense to continue scrolling up).

****Scroll termination criteria (Scroll Up)**** - The options in this group define a set of behaviors that a host application might use to indicate the first page of a recordset. The options you select from this set are defined as "termination criteria," which let Host Integrator know that it has reached the beginning (first page) of a recordset. If you select multiple termination criteria, Host Integrator assumes the beginning of the recordset is reached as soon as any one of the selected criteria is satisfied.

Scroll operation results in same recordset data – The records displayed after the Scroll Up operation are the same as before the operation.

Screen contains __ blank record(s) – The screen after the Scroll Up operation contains the specified number of blank records.

Screen contains __ repeated record(s) – The screen after the Scroll Up operation contains the specified number of identical records.

Screen contains pattern – The specified pattern is present in on the screen after the Scroll Up operation.

Screen has condition – Select this check box and click the Edit button to open the Condition Edit dialog box. Use this dialog box to construct a logical statement that defines the beginning of a recordset.

Scroll Down

Scroll termination criteria (Scroll Down) - The termination criteria define a set of behaviors that you set to determine how the Host Integrator knows that it has reached the end (last page) of a recordset. If you select multiple termination criteria, Host Integrator assumes the end of the recordset is reached as soon as any one of the selected criteria is satisfied. The criteria are tested in the order they are listed in this dialog box, for example **Scroll operation results in same recordset data** is first, and so forth.

Scroll operation results in same recordset data – The records displayed after the Scroll Down operation are the same as before the operation.

Scroll operation results in entity change – The Scroll Down operation triggers the recognition of a new entity.

Screen contains __ blank record(s) – The screen after the Scroll Down operation contains the specified number of blank records.

Screen contains __ repeated record(s) – The screen after the Scroll Down operation contains the specified number of identical records.

Screen contains pattern – The specified pattern is present in on the screen after the Scroll Down operation. The Learning to Use Host Integrator tutorial employs this strategy to recognize the end of the AccountList recordset (in the Terminating the page-down operation section).

Screen has condition – Select this check box and click the Edit button to open the Condition Edit dialog box. Use this dialog to construct a logical statement that defines the end of a recordset.

Max number of Page Down operations – This value sets the maximum number of Page Down operations that are executed before an error message is sent. The default number of operations is 1000, which in normal usage should never be reached, but setting a maximum value ensures that if a termination option on the same page is in error, an endless loop cannot occur.

Exclude records on last screen

Use these options to determine how Host Integrator will handle recordset data when it recognizes that it has reached the last page in a recordset.

Select **Exclude records repeated from previous screen** to stop Host Integrator from returning records from the last page, if they are identical to those records on the previous page.

The following options specify where you want Host Integrator to stop returning data from the last page in a recordset. If you select more than one of these options, Host Integrator stops returning data after one of the options is satisfied.

Read until __ blank record(s) are found – No records are returned after the specified number of blank records is found.

Read until pattern – No records are returned after the specified pattern is found.

Read until record with condition – Select this option and click Edit to open the Condition Edit dialog box. Using this dialog box, construct a logical statement describing a condition to indicate where to stop returning records.

RECORDSET OPTIONS

Click the Options button on the Recordset tab to open the Recordset Options dialog box. Configuring filters and host scrolling behavior gives you more flexibility in sorting through recordsets when developing a host application model.

Current record preferences

Configure the options below to describe the properties of the current record in a recordset. The process of finding the current record is also referred to as recordset synchronization. When Host Integrator arrives at an entity with a recordset, it will always indicate the current record is at a record index of 0, which means before the first record. Any subsequent fetch of data will then start with the first record and proceed normally.

In some cases, you may not be primarily interested in fetching data. For example, you may instead want to select a record and navigate to another entity to view or edit details about it. In these cases, the record that the host claims is current may be important information for Host Integrator since the host's current record may not always be the first record.

If the host's current record is not always the first record, the model needs to provide the ability for Host Integrator to discover which record is current, so Host Integrator can calculate the number of LineDown or LineUp operations to execute.

This is most applicable to character-mode hosts; in block mode, the host does not typically designate a current record and Host Integrator can freely choose the record its client desires. *Note:* The current record definition defined here can also be used for recordset synchronization in procedures.

Use first record on the current screen — Default. Select this option to assume the first viewable record on the terminal screen as the host's current record.

Use current cursor position — Select this option to designate the record that contains the cursor as the host's current record.

- **Record contains pattern** — Select a pattern to use as the indicator of the host's current record location. See the AddressBook entity of the [Pine](#) model for an example.

Tip

On a character mode host, define a pattern with reverse video in the recordset region, then select that pattern from this list to define it as a way to discover the current record.

If you want Host Integrator to make its current record match that of the host, use the `SetCurrentIndex` method and pass the `SyncToHost` parameter in your client application. Search for `SetCurrentIndex` in the Host Integrator API Reference for more information.

Excluding records

Configure the following options to create recordset filters that remove unneeded data sent by the host:

Exclude consecutive repeated records — Select this check box to filter out records that appear at the bottom of one terminal screen and then repeat at the top of the next screen. *Note:* If the host you are creating your model for never repeats records, clear this check box.

Exclude blank records — Select this check box to filter out blank records that appear on a terminal screen. It is possible that some field delimiters will not be seen as blank and will therefore prevent the record from being excluded. Use **Exclude records matching condition** instead and test if each individual field is blank.

Exclude records matching condition — Select this option to specify an exclusion condition for the recordset using the condition editor. You can specify a regular expression as part of the condition.

Scrolling behavior

Configure the following options to indicate in the model file how the host handles recordsets that overlap between two screens:

Host starts each screen with new record — Select this option if the host begins each new screen with a new record after scrolling down to the next screen. This indicates that the result of a PageDown operation will be a new set of records without any repeated records from the previous screen.

- **Host repeats records on new screen** — Select this option if the host repeats records as a way of having continuity between screens when scrolling is executed. For example, the last record on one screen may appear as the first record on the next screen after a PageDown operation has been executed. If the host uses this pattern consistently, select this option and configure one of the following options in the **Repeated record preferences** box:
- **Assume __ repeated records** — Select this option and enter the number of expected repeated records.
- **Dynamically determine repeated records** — Select this option to instruct the Host Integrator to recognize repeating records.
- **Host overlaps records across screens** — Select this option if the host begins a record on one screen and ends that same record on the next screen after a PageDown operation has been executed. If the host uses this pattern consistently, select this option and configure one of the following options in the Overlapping record preferences box:
- **Assume no repeated data** — Select this option if the host application does not repeat data when a record overlaps from one screen to the next.

Dynamically determine repeated data — Select this option to instruct the Host Integrator to recognize if there are any repeating records when records overlap from one screen to the next.

Skip records not contained on a single screen — Select this option to instruct the Host Integrator to ignore records that are not visible on a single screen.

Warning

If you select the **Host overlaps records across screens** option, be sure the **End of screen is always a delimiter** check box under Record size in the Recordset Layout tab is *not* checked.

LAYOUT TAB

This tab contains options used to define the layout of a recordset. Click one of the following for more information:

[Column location](#)

[Record size](#)

Column Location

Use the following options to define a recordset that displays more than one entry per line:

Single Default. One entry per line.

- **Multiple** Select this to enable the up and down arrows and three-column grid (**Column**, **Start**, and **End**) controls. As you change the total number of columns in a selected recordset, the grid control will list the changed entries under **Column**.

The **Start** and **End** controls represent the start and end terminal screen columns for each recordset column. As you increase or decrease the total columns with the up and down arrows, the grid will be updated to divide the width of the recordset evenly among all columns. The terminal screen will also be adjusted as the number of columns changes.

Note

The columns can have their start and end terminal screen columns set independently of each other.

Record size

Allows you to define the size of the each recordset with the following options:

Fixed - Default. Select this if your recordset takes up a set number of rows. The default is one row.

Variable - Select this option to specify where the beginning and end of a recordset are on an entity. When selected, other delimiter options are available.

Delimit by - Select an option to specify delimiters with text and the horizontal placement of the text within a recordset. The following options are provided: `<Blank line>` (default), `<Any text>`, `<Any number>`, or `<User specified text>`.

If you select an option other than `<Blank line>`, click the **Advanced Delimiter Properties** button



to open the Advanced Recordset Delimiter Properties dialog.

Select the **Delimiting text is also data** check box to treat the delimiting text as data. By default, the Host Integrator assumes the delimiter is a visual separator between records rather than data. If the delimiting text is data, select one of the following options:

Last field in current record

First field in next record

Advanced Delimiter Properties dialog

Under Record size in the Recordset Layout tab, click the Advanced Delimiter Properties button to open the Advanced Recordset Delimiter Properties dialog box. Select the Delimiting text is also data check box to treat the delimiting text as data. By default, the Host Integrator assumes the delimiter is a visual separator between records rather than data. If the delimiting text is data, select one of the following options:

Start col - If you select `<Any text>` or `<Any number>` in the Delimit by box, select an option from this list to specify how the delimiter should be handled by the Design Tool. The following options are provided: **Any Column** or **First Column** (default). You can also numerically select your start column by using the up and down arrows.

End col - Specify the end column of your recordset to match the End specified in the Column location box.

Text (type or select it from the terminal screen) - If you selected `<User specified text>` above, use this text box to type the text you want to use for your delimiter. If you select a portion of the terminal screen that falls within the recordset and click the ! button, the Design Tool will update and transfer the written contents to the text control and the horizontal location to the column control.



Note

The Design Tool displays different host display settings for different types of terminal sessions. For 3270 and 5250 sessions, the Design Tool displays options to select field type and text color.

For VT sessions and some character mode HP sessions, the **Field type** and **Text color** check boxes are replaced with these options:

Brightness - Specifies the brightness level of a static area added as a pattern on the terminal screen. Brightness can be normal (most screens and text) or half (which makes the characters appear dimmer). The default is No.

Blink - Specifies that the text added as a pattern blinks on the terminal screen. The default is No. To change the color of a display attribute in the Design Tool, use the Display Setup dialog box on the Settings menu.

Video - Specifies the video type of a pattern on the terminal screen. There are two types of video: Normal (often white characters on a black background) and Reverse (black characters on a white background). The default is No.

- **Underscore** - Specifies whether or not characters in a pattern appear to be underlined. The default is No. To change the color of a display attribute in the Design Tool, click Display on the Settings menu to open the Display Setup dialog box.
- **Case sensitive** - Specifies whether or not text, either typed or selected from the terminal screen, will be stored as case sensitive text in your model.
- **End of screen is always a delimiter** - Select this check box if you want the Host Integrator to always start a new record after iteration.

Warning

If you select this option, be sure the **Host overlaps records across screens** option under **Scrolling behavior** in the Recordset Options dialog is *not* checked.

RECORDSET – FIELDS TAB

This tab contains options used to define fields within recordsets. Click one of the following for more information:

[Cache all field writes](#)

[Name, Start, End, and Key](#)

[Type](#)

[Read](#)

[Write](#)

Cache All Field Writes

Creates a temporary storage area for data assigned to recordset fields. To globally configure all recordset field data to be cached on new entities, select the **Cache all field writes** box on the Recordset Preferences dialog box. To cache recordset field data on a selected entity, select the entity in the Entity window and select the **Cache all field writes** box on the Recordset Fields tab.

When data is assigned to a field from within the Design Tool or from a [Host Integrator API](#), it will be stored in this area rather than the default behavior of being written directly to the terminal. In an operation, field data can be assigned using write commands like [DefaultValue](#) and [TransmitToField](#).

When using a Host Integrator connector, field data is assigned with methods like [UpdateRecords](#) or [UpdateRecordByFilter](#).

When [Cache all field writes](#) is selected, the field data assigned by the Design Tool or the Host Integrator connector is temporarily stored until an operation issues an [UpdateAttribute](#) or [UpdateAttributes](#) command to write the stored data to the terminal screen.

Alternatively, an operation can call [UpdateAttribute](#) for each field in the desired order, inserting other commands in between as needed to modify cursor positioning, synchronize with the host state, or meet any other requirements to be executed by the operation.

Note

By default, an [UpdateAttribute](#) command will be inserted in each generated operation when **Cache all field writes** is selected.

This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When field data assignments are arbitrarily mixed between an operation and a Host Integrator connector API, the order in which data is actually written to the terminal can be important to the host application.

Name Start End and Key

Before you can define a field on a recordset, you need to add fields to your recordset.

Note

If the size of a record in a recordset is not consistently a fixed number of rows, you can create variable records in a recordset. Variable records enable you to dynamically access records that change in size throughout your host application. Using the [Recordset Layout](#) tab, configure a distinguishing characteristic of a record in a recordset and then, using the [Recordset Fields](#) tab, identify certain record fields in a recordset that are unique.

To add a field to a recordset:


Select an area within the recordset on the terminal screen that contains specific data and click the [New Field](#) button. By default, a name and the start and end offsets of the field are provided by the Design Tool. To change the name of the field, place your cursor in the [Name](#) text field and type a new name.

To delete the field, place your cursor within the field definition in the [Entity](#) window and click the [Delete](#) button.

Use the following options to further define the field you just created:

Name - The name of the field.

Start - The start offset within a field. By default, the Design Tool provides the start coordinate of the field you have selected.

 **Note**

This is **not** a position on the terminal screen but rather a position within the record. **Example:** If a terminal screen is 80 columns wide and a record has two rows, the Start position on the second row will be 1 rather than 81.

End - The end offset within a field. By default, the Design Tool provides the end coordinate of the field.


Key - Select this check box if the name of this field returns one unique record. If you mark multiple fields as a key, the combination forms a key that appears in the exported documentation to aid the API developer in fetching specific pieces of data.

To create variable records in a recordset:

Add a recordset to your model.

On the Recordset Layout tab, select **Variable** under **Record size**.

Select an option from the **Delimit by** box. For example, select `<Any text>` and then specify the start and end column of the text that appears in the first row of a recordset.

Open the Advanced Delimiter Properties  dialog box. If you selected a delimiting property that is also data in the recordset, like the `<Any text> *` option described above, configure the options in this dialog.

Click the Recordset Fields tab, and add fields to your recordset.

On the last field of the recordset, delete the column number in the **End** box and type the letters **eor**, which notifies the Host Integrator that this is the end of the variable record that you've created.

Type

Since configuring type options is possible for both attributes and recordset fields, select either the Attribute Properties tab or the Recordset Fields tab to view the **Type** box. These options allow you to specify whether or not data can be accessed by one of the Host Integrator connectors. If both **Read** and **Write** are cleared, the Host Integrator will reject any attempt to save the entity.

The choices include the following:

- **Read** - Select this check box to allow Host Integrator to read the host data contained in this attribute or recordset field at runtime. By default, this option is selected if protected fields exist on the entity. *Note:* The **Read** check box should not be selected if the attribute or recordset field contains or is to contain secure information such as a password.
- **Write** - Select this check box to allow Host Integrator to write host data to this attribute or recordset field at runtime. This check box is available even when an attribute or field is not writable to cover cases the protected state can change dynamically based on the data sent from the host. *Note:* When configuring a Write attribute, do not use the Any Row or Any Column to specify start position or end position.
- **Hidden** - Select this check box to hide the existence of the attribute or recordset field from one of the Host Integrator connectors.
- **Enable terminal attributes** - Select this check box to make terminal attributes accessible from a Host Integrator connector. *Note:* You must select this check box to enable and read terminal attributes using one of the Host Integrator APIs, specifically when using the EnableTerminalAttributes method. Search for "EnableTerminalAttributes" in the [Host Integrator API Reference](#) for more information.

Read

Configuring read options is possible for both attributes and recordset fields. Select either the Attribute Properties tab or the Recordset Fields tab to view the **Read** box. These options give you the ability to configure how Host Integrator processes terminal data that is read by allowing you to remove any set of spaces or symbols from the retrieved data.

Use the following options to configure how data is read:

- **If empty, fill from last record with valid data.** Select this check box to add a value from the last record to an empty field in the retrieved data. *Note:* This option appears on the Recordset Fields tab only.

Example: An empty retrieval might be a field, such as "Mr.," that is programmed to be saved in one record in order to save space when consecutive records are encountered. This field is copied by the host application until an instance of "Mrs." appears. When this data is fetched by an API developer using one of the Host Integrator connectors, an empty field is returned because the host application has only saved "Mr." for one record—all of the other records return an empty field.
- **Remove trailing blanks.** Select this check box to remove trailing blanks from data read by Host Integrator.
- **Remove leading blanks.** Select this check box to removing leading blanks from date read by Host Integrator.
- **Advanced.** For more extensive substitution options, click [Advanced](#).

Write

Since configuring write options is possible for both attributes and recordset fields, select either the Attribute Properties tab or the Recordset Fields tab to view the Write box. These options allow you to erase to the end of an attribute or or configure support for scrollable text.

- **Erase to end of attribute.** Select this check box to concatenate as many spaces as necessary to make the length of the input match the length of the attribute. For IBM terminal models, an Erase to end of field operation is performed if the attribute's location coincides with a terminal field. If the location does not coincide with a terminal field, Host Integrator uses spaces. This option appears on the Attribute tab only.
- **Erase to end of field.** Select this check box to concatenate as many spaces as necessary to make the length of the input match the length of the recordset field. This option appears on the Recordset Fields tab only.
- **Scrollable text.** Select this check box to add support for writing more characters to an attribute than the host would normally allow. This means that the length validation of the attribute or recordset field is bypassed by Host Integrator.
- **Advanced.** For more extensive substitution options, click Advanced. For example, you can prepend or append a constant string to a data input. The resulting string will be checked for length and type restrictions.

ADVANCED RECORDSET FIELDS: GENERAL PROPERTIES

- **Event Handler.** Use the options below to create, edit, attach, and view properties of a recordset field event handler(/event-handler-reference/#recordset-field-events).



Click this button to create a new event handler for the recordset field. Use the list to select an existing recordset field event handler. The selected event handler is attached to the recordset field when you click OK.



Click the Edit button to open the event handler in your default editor.



Click the Properties button to view event handler properties.

- **Field description.** Type a description of the selected recordset field. This data is included in Web Builder projects or any documentation generated using Export options.

ADVANCED RECORDSET FIELDS: CHARACTER MODE PROPERTIES

The Character Mode tab of the Advanced Recordset Field Properties dialog box provides the options below. The settings on this tab are unavailable if your model is for a 3270 or 5250 terminal type.

Character mode data entry.

If you are working on a character mode host, such as HP or VT, direct the Host Integrator to treat this field in any one of the following ways:

- **Don't wait for echo (Default)** – Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will **not** be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- **Wait for same number of characters to echo** – Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted.

For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.

- **Wait for string to echo at cursor** – Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- **Wait for next tabstop** – Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then the Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then the Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- **Wait for next tabstop or string to echo at cursor** – Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.

ADVANCED RECORDSET FIELDS: OPERATIONS PROPERTIES

The Operations tab for the Advanced Recordset Field Properties dialog box provides the following options for configuring operations to be executed before or after writing to a recordset field:

- **Execute operation before writing to field** — Select a configured operation to be executed before data is written to the selected field.
- **Execute operation to autotab on underfill** — Select a configured operation to be executed when recordset field data does not completely fill in the length of the field.

 **Tip**

Create an operation that automatically tabs to the next input field or an operation that traps an error. For example, if the data input is always a set length, like a social security number, create an operation that includes an error pattern as an operation condition.

The option is typically used with character mode hosts. When data doesn't fill the recordset field, the behavior of a character mode host application can often be less predictable than a block mode host application.

- **Execute operation after writing to field** — Select a configured operation to be executed after data is written to the selected field.

Cursor Tab

The Cursor tab contains cursor movement settings to be used with character-mode hosts. (If you are not working with a character-mode host, Cursor tab controls are unavailable.) Defining cursor movement is helpful when the host and the host application are working in character mode and there are multiple pieces of data to enter on the terminal screen.

To enable Host Integrator to control cursor movement, a key must be mapped in a move cursor command list to create this action. For example, configure tabstops to navigate around attributes or recordset fields on a selected entity by creating a move cursor command list, so that Host Integrator recognizes that the Tab key is mapped to move the cursor on the screen.

You can define cursor movement differently on each entity by using the options on the Cursor tab or you can choose to define a global definition to be used throughout your model using the options in the Advanced Model Properties dialog box. Under **Character mode cursor positioning**, use the following options to configure a move cursor command list for the selected entity:

- Use global definition from Model Properties dialog

Default. Select this option to use the global configuration that you defined in the Advanced Model Properties dialog box to move the cursor between attributes on an entity.

- Execute operations to move cursor

This option allows you to select existing operations to move the cursor around the attributes and/or recordset fields of the selected entity. The entity can use one operation to move the cursor forward and another to move the cursor backward.

Each time Host Integrator executes one of these operations, it expects the cursor to move to a new tabstop. Host Integrator will iteratively execute a cursor operation as needed to reach a desired location on the screen.

Note

Use tabstops to synchronize the host state instead of waiting for a host update.

Notes:

- In block mode, the cursor location is managed by the local terminal, so Host Integrator can typically direct the terminal to place the cursor in the desired location on the terminal screen. In character mode, however, the host controls all of the data exchange mechanisms.
- At runtime, Host Integrator uses the configuration hierarchy of entity, global command list, and then global default to decide which direction to move the cursor. For example, if you decide to configure cursor movement for a specific entity on the Cursor tab, this configuration will be used even if you have a global command list defined in the Advanced Model Properties dialog box.
- To set a timeout by which Host Integrator should expect the host to move to another tabstop, use the [MoveCursor Command List Timeout](#) setting. If the operation is still executing when this timeout expires, the operation will terminate.
- See the [Pine](#) model example for more information about configuring cursor movement for character mode applications.

To configure the **MoveCursor Command List Timeout** setting:

On the Settings menu, click View Settings to open the View Settings dialog box.

Under **Host Integrator** settings, expand the **Entity Design** directory tree and select **MoveCursor Command List Timeout**.

Under **Setting details**, specify how long (in seconds) to wait for a move cursor command list to execute before an error appears.


CONFIGURING TABSTOPS

Use the options in the Tabstops box to configure a list of tabstops that constitute all of the possible valid "at rest" positions on the terminal screen. By default, when executing a move cursor operation or command list, Host Integrator will assume that any location other than the current position constitutes the host's next tabstop or new "at rest" position for user interaction.

In most cases, the host requires additional information from Host Integrator to ensure tabstop accuracy. The host often updates status lines or performs other terminal updates before moving the cursor to the next tabstop. Because character mode terminals always write data at the cursor position, the host has to move the cursor to the status line or another update location to make the desired screen changes. Without additional configuration, Host Integrator may issue more move cursor requests than is necessary and get out of sync with the host.

To prevent Host Integrator from getting into this state, define tabstops at the locations for which Host Integrator should wait after issuing move cursor requests. For example, from a given location on the screen, Host Integrator may issue a move forward operation or command list that waits for the cursor to appear at any valid tabstop that is not the starting cursor position. If this new location matches the desired destination, the process terminates.

Otherwise, Host Integrator repeats the process until it reaches the correct location. It is essential that all locations where the host can leave the cursor "at rest" for user interaction be defined as tabstops, regardless of whether the model has an attribute or field defined in that location. If even one location is missed, Host Integrator will issue timeout exceptions when the cursor is moved to the omitted location.

 **Note**

If moving through the fields of your host application requires application interaction, you may have to define the tabstops manually.

The following three lists contain tabstop positioning information:

Num - the number of tabstops on the screen.

Row - the screen row in which the tabstop begins.

Column - the terminal column of the tabstop.

The following options allow you to configure an ordered list of tabstops for the selected entity:

Insert - Click to manually insert a tabstop at the current cursor position. To have the Design Tool identify the tabstop positions automatically, click the Generate Next button.

Add Current Pos - Click to add the tabstop order number (Num), the screen row, (Row), and the terminal column (Col) of the current cursor position on the Terminal window.

The Apply and Cancel buttons on the Cursor tab are unavailable if the Design Tool recognizes that you have not made any other changes since the last apply was made. Even though the Apply button is unavailable, any changes that you add will be automatically applied to your model.

Delete - Removes the selected entry.

Move Up - Moves the tab position up one row in the ordered list of tabstops.

Move Down - Moves the tabstop down one row in the ordered list of tabstops.

Clear - Deletes the entire list.

Generate Next - Executes the move cursor command list until all attributes have been recognized and recorded. Once this list is generated, it appears in the Character mode applications box and is saved in the model file.

Notes

When generating tabstops, temporarily clear the Wait for cursor check box on the Validation tab of the Advanced Entity Properties dialog box. If you do not clear this check box, the screen snapshot will fail because the entity is set to match attributes according to cursor positioning and generating a tabstop changes the initial cursor position.

The Generate Next button is disabled in offline mode.

Tabstops are only generated for attributes that appear after the currently defined tabstop on the terminal screen.

Test - Click to test the tabstops in the list. If a tabstop in the list does not accurately reflect an attribute on the entity, an error message will appear. Note: This button is disabled in offline mode.

Use writable attributes as automatic tabstops - Select this option to automatically include configured attributes as tabstops. This eliminates the need to enter all tabstops manually. If there are other tabstops besides those defined as writable attributes, those tabstops will still have to be entered manually.


- **Tabstops are in specified order** - Select this option to make sure that the order of the tabstops reflected in the Design Tool is the order that they will execute at runtime. By default, tabstops are not assumed to be in any specific order.

If both move forward and move backward options are defined above, Host Integrator uses the screen offsets of the cursor destination location relative to the current cursor location to determine whether to move forward or backward. On some terminal screens, the cursor offset is not a good indicator of direction; therefore, listing tabstops in the same order that the host accesses them can be helpful. Host Integrator then uses the relative location of the tabstop in the list to choose a move cursor direction.

Note

This does not mean Host Integrator will wait for the next or prior tabstop in the list. This ordered list is only used to determine the direction to move. Once the operation or command list has been executed, Host Integrator will still wait for any valid tabstop to become the current cursor position.

Advanced Entity Properties

There are several advanced entity properties available. With Entity selected, click  and select the property.

GENERAL

Use the options below to create, edit, attach, and view properties of an entity event handler.



Click this button to create a new event handler for the recordset field. Use the list to select an existing recordset field event handler. The selected event handler is attached to the recordset field when you click OK.



Click the Edit button to open the event handler in your default editor.



Click the Properties button to view event handler properties.

Immediate arrival processing

This option is available when an event handler is attached to the current entity. When available, it is disabled by default.

When immediate arrival processing is *disabled*, the entity arrival event for this entity is not fired while any active command list (for example, an operation) is active. All synchronization commands in the operation will first be completed so that the event handler has all patterns, attributes, and recordsets within the entity available for use.

When immediate arrival processing is *enabled*, an entity arrival event is fired as soon as the patterns used to recognize the entity are found and the entity is validated. An entity arrival event can start before any other commands (such as "wait" commands for synchronization) have been completed. Enable this option if you want to handle this entity as a notification or an error message, dismissing it as a "pop-up" that might prevent an operation from reaching its destination entity.

Entity description

Type a description of the selected entity. This data is included in Web Builder projects or any documentation generated using Export options. In the sample models provided with the Design Tool, a description is provided for all preconfigured operations.

Interacting with Host Application

Configure host application interaction properties for the selected entity with the following options:

- **Hidden** — Select this check box to disable this entity from being used with a Host Integrator connector.
- **Cache all attribute writes** — Select this option to create a temporary storage area for data assigned to attributes.
 - When data is assigned to an attribute from within the Design Tool or from a Host Integrator connector, it will be stored in this area rather than written directly to the terminal (the default). In an operation, attribute data can be assigned using write commands such as `DefaultValue` and `TransmitToAttr` or `WriteVarToAttr`.
 - When using a Host Integrator connector, attribute data is assigned with methods such as `SetAttributes`.

When **Cache all attribute writes** is selected, the attribute data assigned by the Design Tool or the Host Integrator connector is temporarily stored until an operation issues an `UpdateAttribute` or `UpdateAttributes` command to write the stored data to the terminal screen. Alternatively, an operation can call `UpdateAttribute` for each attribute in the desired order, inserting other commands in between as needed to modify cursor positioning, synchronize with the host state, or meet any other requirements for the operation.

Notes

By default, an `UpdateAttributes` command will be inserted in each generated operation when **Cache all attribute writes** is selected.

This feature is specifically designed for character mode applications as a solution for managing the order of screen updates. When attribute data assignments are arbitrarily mixed between an operation and a Host Integrator connector API, the order in which data is actually written to the terminal can be important to the host application.

"Keep Alive" operation — Select a keep alive operation to execute on the host when an active session on the Host Integrator Server is not in use. This operation is executed periodically when the host session is idle to prevent the session from being disconnected from the host prematurely. Use the **Default "Keep Alive" inactivity timeout** option on the Model Properties dialog box to specify how long to allow a session to be inactive before executing the keep alive operation.

On the Host Integrator Server, you can create a pool of sessions that remain at their home entity and wait for a user to connect to them. When a session is in that state, the host may disconnect it due to inactivity.

The Design Tool lets you define an operation that performs a command that stays within the same entity but generates communication with the host (for example, you can send an Aid key to a mainframe, or you can send a space/backspace combination to a host with VT emulation). This ensures that the session is not disconnected due to inactivity.

SNAPSHOT

With Entity selected, click  > Snapshot tab.

This tab provides an image of the selected screen to be used in offline mode. To update the screen to reflect its current state, click Update.

- The Update button is disabled during offline mode.
- Make sure to update your screen snapshot regularly. Patterns that are configured on the Validation tab are compared to this snapshot when switching to offline mode, and if one of the validation patterns is not present on the snapshot, you will receive an error message about defined patterns not matching the snapshot.

VALIDATION

With Entity selected, click  > Validation tab.

Entity validation enables the Design Tool to verify that a terminal screen has been added to a model as an entity. This validation is checked every time you arrive at that entity from another entity. While performing operations within an entity, the validation will not be used again unless you navigate to another entity. Review **entity validation with character mode hosts** for specific information about character mode hosts.

Arrival validation - Configure arrival validation with the following options:


- **Wait for patterns** - Select this check box to direct the model to wait to find a pattern specified in the **Check** box before validating the arrival at the current entity. To move a pattern from the **Unused** box to the **Check** box, use the right arrow button.

Patterns define the unique signature of a screen in the host application model. Define a pattern to be used in the entity signature by selecting the **Use in entity signature** check box on the Pattern tab in the Entity window. In addition, the only patterns that will appear in the **Unused** box will be patterns that are not considered to be a part of the entity signature.
- **Wait for cursor** - Select this check box to direct the model to wait for the cursor to be at the specified row and column on the screen before validating the current entity.

When enabling the **Wait for cursor** check box, enter a valid cursor position only.

Wait for condition - Select this check box to direct the model to check the specified condition iteratively until it is satisfied. If the condition cannot be satisfied, a timeout will occur. To create a condition like the one described, click the Edit button to open the Condition Edit dialog.

TAGS

With Entity selected, click  > Tags tab.

Click the Advanced Properties icon for the selected entity to configure advanced settings on the Tags tab. When you create tags for a particular entity, using an identifying keyword, you have the ability to search and illuminate only those entities that match the search criteria.

Creating tags

In the New Tag field, type the keyword you want to use to identify an entity.

Click New (or press Enter) to add the tag to the Current tag panel and continue creating other tags.

Available tags are those that can be associated with other entities. Use the navigation buttons to move tags between panels.

When you have finished creating tags, click Close and then on the Entity panel click Apply.

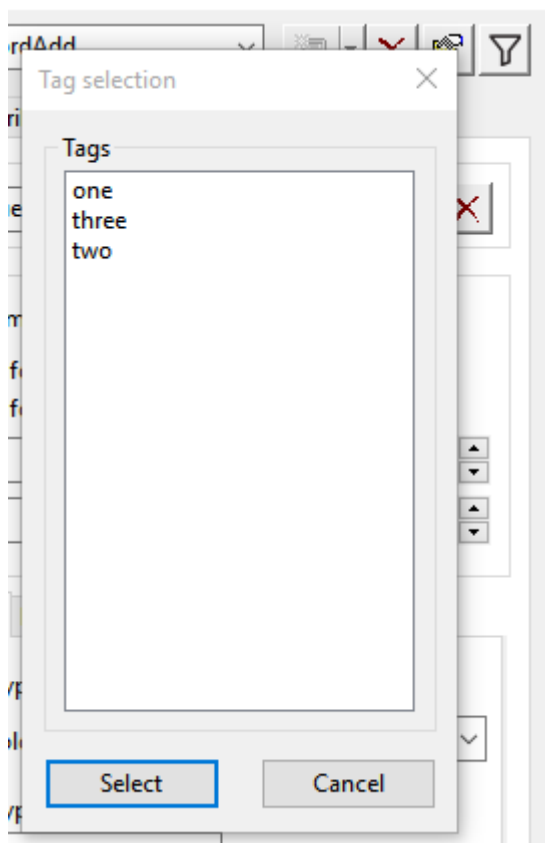
Using tags

After creating your tags:

Open the Tag selection panel using the Tag Selection icon on the Entity toolbar.

Choose the tags you want to apply to the list of entities, click Select. Alternatively, you can select a single tag by simply double-clicking the tag.

- The Tag Selection icon, when marked with an **X**, indicates that a filter is being used. Click the icon to revert the entity list to its original state.



4.15.8 Terminal Window Menus

Terminal Window Menus

The terminal window contains all of the menu items that can be used to configure host screens that make up the model.

File

Edit

Connection

Settings

Events

Model

Debug

Window

Help

FILE

On the File menu you can create a new model, open an existing model, save models and settings, as well as open Web Builder and deploy models. Some of the more advanced options available from this menu are:

- Reload model - Click Reload Model to update the current model. This option incorporates any changes that were made to the model outside of the active Design Tool.

- Deployment options

You can set deployment options for specific hosts, sessions, variables and recordings on the Deployment Options dialog box.

- Host - The host connection settings you specify here are used when you deploy a model from the Design Tool. You can deploy a model to either a local or a single remote server. When you deploy a model from the Design Tool a deployment descriptor file (`deploy_desc.xml`) is created and saved in the model's `deploy\design_tool` folder.

Sessions - Use the Sessions tab on the Deployment Options dialog box to indicate whether your model will use a single session or a session pool. If your model will use a session pool, this is also where you provide the pool and startup entity names.

Variables - Use the Variables tab of the Deployment Options dialog box to include a model variable list in your deployment, exclude or include model variables, and to require the value of certain variables to be unique and/or hidden.

- Recording - Use the Recording tab of the Deployment Options dialog box to configure model debug messages recordings for deployed models. This setting is associated with the Model Debug Messages (`.vmr`) feature available from the Debug tab within the Design Tool.

Note

This option is associated with the model only when it is deployed from the Design Tool to the local server or to another single remote server. When deploying to a production server, use deployment descriptors.

- Deploy to Local or Remote Servers - Deploy to Local Server automatically deploys the current model to your local Host Integrator Server. By default, when both the Design Tool and the server are on the same machine, the Design Tool uses the same folder for models that the Host Integrator Server uses.

You can use a Host Integrator server other than the one associated with your copy of the Design Tool. To deploy to a single remote server use the Deploy To Remote Server option. This option is limited to the same basic settings as the local server deployment; it does not include all the options available when deploying to a non-local server using the `packagemodel` and `activatemodel` commands.

- Importing - Use the Import Model Elements dialog box to bring elements of an existing model into the model you currently have open. See [Importing Model Elements](#) for instructions. You can import XML-based documentation about your model, click Import XML Model.

Exporting - Use these options to generate XML- or HTML-based documentation about your custom model or one of the sample models provided with the Design Tool.

To interact with model project files in a XML format, save the file as .modelx. The files generated using this process are validated using an XML schema (.xsd) and provide an accurate representation of the project files, which is not true of the XML generated for documentation purposes using this dialog box.

EDIT

Use the Edit menu to copy selected text, paste the copied contents at the cursor position (row, column) of the Design Tool, and select all the text in the terminal window. Select **Linear** to use word processor style selection regions that wrap lines, or **Rectangular** to use rectangular regions. The default is Rectangular.

CONNECTION

The Connection menu is where you connect, disconnect, configure and define your host connections. You can also start (before you connect to the host) and stop Host Emulator recordings and datastream traces.

To review host connection properties, click **Connection Properties** on the Connection menu. This option is available after you connect to the host using the settings specified in the New Model or Session Setup dialog box.

More information

[Configuring a Host Session](#)

About Traces

With the Design Tool's trace feature, you can record data transmissions to and from the host. Although trace is intended primarily for diagnostic purposes, you can use the playback feature to redisplay the host's output to your screen. Playback does not redisplay the user's input.

Enter a file name for the trace in the File name box. By default the extension for trace files is .hst. Click Trace to begin the trace. To end the trace and close the file, click Connection, point to Datastream trace and click Stop trace.

SETTINGS

On the Settings menu you can configure your display options; colors, fonts, margins and status bar text. On this menu you also can set up preferences, event handlers, and other more advanced options.

From the Display menu:

[Display Setup Options](#)

Other Settings menu options:

[Preferences](#)

[Event Handlers](#)

[View Settings](#)

EVENTS

You can implement event handlers to extend the capabilities of your model, to represent the host user interface within a higher-level procedural (table) interface, or to extend error handling to make host application models more robust.

Use the commands on the Events menu to attach event handlers, locate libraries and source code, and change the behavior of your event handling environment.

[Attach event handlers](#)

[New event handler](#)

[Event handler reset](#)

ATTACH EVENT HANDLERS

The Attach Event Handlers dialog box, available by clicking Attach from the Events menu, is used to attach an event handler to a component in your model, to create event handlers, to launch an editor for an event handler, and to view event handler properties.

Note

You can also attach event handlers while working with any individual object that supports them. For example, the Advanced Properties button for any selected entity, attribute, operation, recordset, field, or procedure provides the same options available here.

Viewing Sources

The left-hand portion of the Attach Event Handlers dialog box provides two options for viewing components that support event handling: by the model's structure, or by the types of objects that support event handlers. Any component or component group that has an event handler already attached is displayed in bold. Any specific component type that includes an attached event handler displays a yellow lightning bolt to the left.

- Structure

The Structure tab shows a hierarchical listing (tree view) of available components for attaching event handlers. This is a convenient way to work with a set of event sources grouped by the model structure (for example, all fields on a recordset, or all attributes on an entity). When you click a component in the Structure list, the right hand portion of the dialog box shows event sources belonging to that component, as well as any currently attached event handlers. Any type that has an event handler already attached displays a yellow lightning bolt.

- Type

Use the Type tab to work with all components that generate the same kinds of events, regardless of where they are in the model. When you click a component in the Type list, the right hand portion of the dialog box shows event sources of that type, as well as any currently attached event handlers.

Selecting Event Handler Sources

When you select an event object on the left side of this dialog box, the associated Event Sources available in the model are displayed on the right. If you have already created event handlers for this object, click the Event Handlers column to select from the list. This list always includes a `<None>` option; use this to remove the attachment of an event handler to an event source.

You can select more than one event source to attach or remove a handler that applies to both.

Attaching Event Handlers

After selecting or creating an event handler, click OK to attach the event handler to the event source. If you create an event handler but do not attach it to an event source or click Cancel, the event handler is still available to attach to other objects of the same type. Use standard control key selection procedures to attach multiple event sources to the same event handler. If you want to discard the changes you have made since you opened the Attach Event Handlers dialog box, click Revert.

You can also detach event handlers and remove associated files.

NEW EVENT HANDLER

Use this dialog box to generate template code for new event handlers. The template includes the basic code for the events associated with a specific event handler type. See the [event handler reference](#) for a list of event handler object types.

Adding an event handler

1. You can open the New Event Handler dialog box from several locations.

- Click New on the Events menu. You can select among object types when you use this option. In this case, the event handler is not attached to any particular object in the model; you must then attach it using one of the options below.
- Click the Add Event Handler button in the Attach Event Handlers dialog box or in the Advanced Properties dialog box for a specific entity, attribute, operation, recordset, recordset field, or procedure. For model events and life cycle events, click Model Event Handlers on the Model menu. With these options, the object type is already selected when you open the New Event Handler dialog box.

2. After providing a name or confirming the provided name, add any necessary description and comments. Click OK to create a Java source file template based on the handler type. The name you provide must follow Java naming conventions. You'll see an error message when you click OK if your name does not meet these requirements.

2. The source file is compiled and the output is added to a JAR file used with the model (by default, vhi_model.jar in the `\models\\scripts\lib` folder)

3. After creating and compiling the code, edit the event handler source code. Click Open for Editing to have the editor open automatically when the compilation is complete.

Host Integrator includes an ant script (Java) or msbuild script (.NET) that compiles source files located within the Design Tool directory structure. You can modify the ant build.xml or the msbuild script.

New event handler settings

- Java class name or C# class name

The Class Name can be either a simple class name or a fully qualified name (prepending package name). A default name associated with the event handler type is provided. If a package name is supplied, it becomes the default package name for this model.

- Description

Use Description to provide additional information about the event handler in the Design Tool user interface.

- Comments

Any information entered here is included as a block comment near the top of the generated event handler source file.

- Open for editing

Select this check box to open a source code editor for the event handler after the source code has been edited and compiled and the class file has been added to the event handler file.

EVENT HANDLER RESET

An event handler reset occurs when you click Reload Handlers on the Events menu and a reset is needed. This forces a complete reset of the host and client sessions in the script manager.

The Design Tool displays a Reset Needed indicator (R) in the status window whenever the script manager cannot respond. Some conditions that produce this indicator are:

An event handler is stopped at a breakpoint, and the debugger has suspended all threads

- An error in the event handler code is preventing the script manager from responding
- Java: The JVM process has been terminated.

.NET: The clscriptserver.exe process has been terminated

Click Reload Handlers on the Events menu to force a reset.

The order in which events are fired in a event handler script manager reset sequence:

Client Session Destroyed

Host Session Destroyed

Model Unloaded

Model loaded

Host Session Created

Authenticate User. Any resulting errors display in the Event Handler console.

Client Session Created

If a reset causes the script manager to be recreated when an existing script manager cannot be contacted, steps 1-3 are omitted.

MODEL

On the Model menu you can add entities, patterns, and attributes, along with many other options.

On the Model menu, point to Add and select Entity, Pattern, or Attribute for your model.

[Variables](#)
[Tables](#)
[Event Handlers](#)
[Host Events](#)
[Properties](#)
[Templates](#)
[Record](#)
[Repair](#)
[Execute Login Command](#)
[Execute Logout Command](#)
[Update Initial Cursor Position](#)

TABLES

Use the Tables dialog box to create and modify tables and procedures for the current host application model. Depending on what is selected in the tables and procedures box, the contents of this dialog box change to reflect these views. Tables and procedures enable you to create a database abstraction of host data.

See:

[Tables Overview](#)
[Creating Tables](#)
[Creating Procedures](#)
[Creating Compound Procedures](#)

Table Properties Pane

When a table is selected in the Tables and procedures pane, you can view and configure the following properties:

Table properties	Description
Name	Specifies the name of the currently selected table.
Description	Displays a description of the currently selected table.

Table properties	Description
Columns	Click the <i>plus sign</i> button on the right to add a new table column or the <i>minus sign</i> button to delete the selected column. Then supply a Name, Data Type, Key value, and Description for each column. Table columns defined in the Tables dialog box are initially not associated with specific procedures. For step-by-step instructions on creating table columns, see Defining Table Columns .
Column properties	Set minimum and maximum values for a column; this is an optional setting.
Allow SQL SELECT statements to return a subset of columns when all columns are requested	Select this option to have Host Integrator return a partial set of data to a querying application. If you leave this check box clear, Host Integrator will return an error to a querying application if it cannot return a full set of data.

Procedure Properties Pane

When a procedure is selected in the Tables and procedures pane, you can view and set these properties:

Procedure properties	Description
Name	The name of the procedure.
Home entity	The entity where the procedure begins. If the Navigate back to starting point upon completion option is selected, the procedure will also navigate back to this entity upon completion.
Type	Specifies the procedure's type. Choose to create one of these types of procedures: Select, Update, Insert, and Delete.
Parameters	Set these parameters for the type of procedure you are creating. See Mapping Procedure Parameters .
Available for SQL queries	If this option is selected, the procedure is available for Host Integrator to fulfill SQL queries. If this option is not selected, the procedure is only available using the PerformTableProcedure method.

Procedure properties	Description
Navigate back to starting point upon completion	If this option is selected, the procedure returns to its home entity upon completion. This gives you greater flexibility in creating sequences of procedures, but also increases the difficulty of assuring that Host Integrator is able to navigate from where one procedure ends off to where another begins.

Compound Procedure Properties

A compound procedure is a procedure that consists of one or more SELECT subprocedures and an UPDATE or a DELETE subprocedure. A compound procedure cannot contain an INSERT procedure. By combining one or more subprocedure into a compound procedure, you can perform more than one query level task at the same time, like selecting several records and updating or deleting them. The order in which subprocedures appear in a compound procedure is the order in which they are invoked. The first subprocedure in a compound procedure must be a SELECT subprocedure.

When you select a compound procedure in the Tables and procedures pane, you can view and set these properties:

Compound procedure properties	Description
Name	The name of the compound procedure.
Description	The compound procedure's description.
Type	Specifies the compound procedure's type. The options are SELECT, UPDATE, and DELETE.
Subprocedures	The SELECT subprocedures are listed in the Select subprocedures box, and the UPDATE or DELETE subprocedure, if any, is listed in the Update/Delete procedure box. The order the subprocedures are listed is the order they are invoked when the compound procedure is run. The first subprocedure in a compound procedure must be a SELECT subprocedure. Any output parameters from a subprocedure are used as the filter or data parameters for the next listed subprocedure.

Compound procedure properties	Description
Filter parameters	Specifies which attributes are valid in incoming query requests. Only those attributes that are selected (checked) are valid. If the Req box is checked, all queries MUST contain this attribute or the Host Integrator will return an error.

Additional parameters:

The column on the right differs according to the procedure type:

- Update— The Data parameters list specifies which attributes are updated on the host during the compound procedure. Only those attributes that are selected (checked) can be updated. If the Req box is checked, all queries MUST contain this parameter or the Host Integrator will return an error.
- Select— The Output parameters list specifies where the attribute is written from during the compound procedure. The Data Source column indicates the subprocedure that includes the attribute.
- Delete— No additional information is required for this option.

Available for SQL queries

When you select this check box, you make this procedure available for the Host Integrator to use to fulfill SQL queries. If you clear this check box, this procedure will only be available using the PerformTableProcedure method.



EVENT HANDLERS


You can add, edit, or view properties of a Life Cycle or Model event handler.

- [Life cycle event handlers](#) are those that occur outside the context of a host screen.
- [Model events](#) are those that occur when a host session is active, but are not limited to a particular component such as an entity or attribute.

Adding and viewing event handlers

Use the options below to create, edit, attach, and view properties of an model event handler or a life cycle event handler.

Option	Description
	Click this button to create a new event handler for the life cycle or model. You can also select among other event handlers for the life cycle or model that have been created. The selected event handler is attached to the life cycle or model when you click OK.
	Click the Edit button to open the event handler in your default editor.


Option	Description
	Click the Properties button to view event handler properties.

Event handler properties

After you have created an event handler, you can review its properties.

- Handler type - The handler type of the selected event handler. All event handlers correspond to a specific object type within a model.
- Java class name - If you provide a fully qualified name (with a prepended package name) when you created the event handler, the class reflects the final (unique) portion of the name.
- Package - The package name is displayed here if you provided a fully qualified name (with a prepended package name) when you named the event handler. When you supply a package name, it is remembered as the default package name for this model.
- Description - Shows additional information about the event handler that was entered when you created the event handler.
- Events handled - This information is based on handler metadata. All events that can be handled by this type of handler are listed, along with an indication if the event is currently enabled in this handler. The Timeout column shows any nondefault event timeout for an event, as listed in the event handler metadata.

TEMPLATES

Click Templates on the Model menu or click the  Templates button on the standard toolbar to open the Model Templates dialog box. Use this dialog box to create and edit command list templates that can be reused throughout the modeling process. These templates are then stored in the modelx file. For example, create a command list containing the CheckOperationConditions command, the WaitForDisplayString command, and a TransmitToAttr command that can be used on various entities. By default, CheckOperationConditions command is automatically added to each template.

Notes

Templates are stored in the model file.

To create a model template:

Click the Add button. By default, the name OpTemplate_0 is provided as well as a CheckOperationConditions command.

To change the name of the template, select it and type the new name in the Name list.

To create the template, click the Edit button to open the Command List Edit dialog box.

Create a command list using the command options and then click OK.

The Model Templates dialog box opens and displays the command list in the Commands box. Click OK.

To use an existing model template in your model:

Open the Operation Edit dialog box and click Import to open the Import dialog box.

Expand Template in the Command list directory tree and select a template to use. If a green dot appears in front of the template name, it is ready to use.

RECORD

On the Model menu, point to Record and select from the following:

Start Recording to start manually recording commands. This option is available before you have loaded a model into the Design Tool.

- Stop Recording

The Stop Recording dialog box allows you to specify what happens to recorded commands. When you click Save, the Design Tool automatically generates an operation using the keystrokes recognized after Start Recording was selected on the Model menu. When you click Continue, the Design Tool closes this dialog box and continues recording commands. If you click Discard, the Discard Recording dialog box appears and verifies whether or not you want to delete the recorded operation. The following options are available for defining recorded commands:

- Save as new operation

This option indicates that the previous keystrokes will be saved as an operation if you click Save. By default, this option is only available if the recording began and ended at defined entities.

- Origin

This is a read-only setting that displays the name of the entity on which the recording started.

- Destination

This is a read-only setting that displays the name of the entity on which the recording stopped.

- Name

The name of the new operation. By default, the Design Tool generates a name for your new operation. To change the name, type it in this box.

- Save as login command list

Select this option to save your recorded commands as a login command list.

- Save as logout command list

Select this option to save your recorded commands as a logout command list.

REPAIR

The Entity Repair dialog box allows you to repair an entity whose signature is not recognized by the Design Tool. This option will be enabled whenever the current screen is recognized as undefined.

To use the dialog box, select the entity that should correspond to the current terminal screen from the list. You will then see that entity appear in the Entity window for editing. You can use the Signature Analyzer to determine why the Design Tool does not recognize the entity and then take corrective action.

If the entity signature still does not match after you click Apply, you will receive a warning message. If you save changes anyway, the Entity window will once again be reset to empty.

If you find an unexpected entity is recognized by the Design Tool, you must first define a new signature property, such as a new pattern, that differentiates the current screen from the unexpected entity. The Design Tool then automatically loads the desired entity in the Entity window. If the Design Tool does not automatically load the desired entity, use this dialog box to correct the problem.

EXECUTE LOGIN COMMAND

Specifies whether you want the Design Tool to execute a login command list when your model connects to a host. For instructions, see creating a [login command list](#).

You can also enable this setting from the General tab of the Preferences Setup dialog box by selecting the Execute login commands check box. The default is No.

This option also navigates to the home entity and initiates associated event handlers.

EXECUTE LOGOUT COMMAND

Specifies whether you want the Design Tool to execute a logout command list when your model disconnects from a host. For detailed instructions, see creating a [logout command list](#).

You can also enable this setting from the General tab of the Preferences Setup dialog box by selecting the Execute logout commands check box. The default is No.

UPDATE INITIAL CURSOR POSITION

Changes the initial cursor position to the current cursor position. You can also click the Update Initial Cursor Position button on the model toolbar. View the current cursor position and the initial cursor position in the status bar in the lower left corner of the Terminal window.

If you have configured any entity properties relative to a cursor position, these property coordinates will be automatically updated. Click Apply to save your changes or click Cancel to revert your initial cursor position back to its original state.

DEBUG MENU

The Design Tool provides several ways to test and debug host application models. Use the options on the Debug menu to confirm that your model is operating as you expect and is ready to deploy.

The log monitor is a troubleshooting utility that you can run outside of the Design Tool. See [Debugging Models](#) for descriptions of the various commands available from the Debug menu.

WINDOW MENU

On the Windows menu, select which toolbars to display. You can also choose to display the terminal keyboard.

HELP MENU

On the Help menu you can find out more about Host Integrator, including version numbers and links to Technical Support.

Display Setup Options

Click the Options tab in the Display Setup dialog box to set these options:

Fonts

Colors

Help display mode

Caption

Status bar text

Display margins

Substitute 0 with Ø

CONFIGURING DISPLAY FONTS

Use this dialog box to specify the display font and style in the Design Tool. Changing the Design Tool's font has no effect on the font used by your printer, the status line, menu commands, or dialog boxes.

- Font

Specifies the display font in the Design Tool session window. You can use any font, including TrueType fonts. Arial is an example of a TrueType font that is not monospaced, and therefore is not supported by the Design Tool.

By default, the Design Tool uses the `r_ansi` font. This provides a 24 x 80 display that accurately emulates the terminal.

When you resize the Terminal window, the Design Tool chooses a new font size so the correct number of rows are displayed on the screen. If the Design Tool cannot display the font you have chosen, the default (`r_ansi`) is displayed instead.

The available fonts can change as you change the model ID in the Session Setup dialog box, because not all model IDs support all fonts.

!!!note When you print all or part of a host screen from a terminal session, the font used is the currently configured display font.

- Font style

Specifies a font style, regular or bold. The Design Tool does not support italicized fonts. The default is Regular.

- Display variable width fonts

Specifies whether proportionally-spaced font types appear in the Fonts box. The default is Yes.

- Auto sizing

Specifies that for whichever font or style you select, the Design Tool automatically adjusts the font size to fit all text in the Terminal window. To change the font size, set this setting to No. By default, Auto Font Size is enabled.

CONFIGURING COLORS

The Colors tab in the Display Setup dialog box lets you associate colors with screen attributes in the Terminal window.

Color settings can be saved to a settings file (`.dtool`) with other configuration information.

You can set colors in either of the following ways:

[Setting colors with a mouse](#)

[Attribute by attribute](#)

HELP DISPLAY MODE

The Design Tool can display a brief message about the currently selected menu, menu command, or button on the title bar (above the Terminal window) or in the status bar (below the Terminal window).

By default, help messages are shown in the status bar.

CAPTION

Specifies the string that appears in the Design Tool title bar. This string is relevant only when the Help display mode is **Status Bar** (default). This string is also displayed on the taskbar when the Design Tool is running.

Select a predefined option from the list box, or enter up to 260 characters in the box. As you click on predefined options, shortcuts for these options are added to the box.

The predefined options and their shortcuts are:

Shortcut	Option
&r	Verastream Host Integrator
&f	Model File Name (or "Untitled" if a model file is not open)
&s	Session Type
&t	Transport
&h	Host Name
&d	Date
&c	Connection Status (whether you are connected and over what transport)
&v	Assigned Device Name
&&	Single Ampersand

Shortcut	Option
&p	Script debugging port

So, for example, if you set the Caption to **&s - &t - &c**, you might see "IBM 5250 Terminal - Telnet - Connected" in the Design Tool title bar or on the taskbar, assuming the Design Tool is running but not minimized.

The default is **&f - &r - &s**.

STATUS BAR TEXT

Specifies the string that appears in the Design Tool status bar. Select a predefined option from the list box, or enter up to 260 characters in the box.

As you click the predefined options, shortcuts for these options are added to the box.

The predefined options and their shortcuts are:

Shortcut	Option
&r	Verastream Host Integrator
&f	Model File Name (or "Untitled" if a model file is not open)
&s	Session Type
&t	Transport
&h	Host Name
&d	Date
&c	Connection Status (whether you are connected and over what transport)
&v	Assigned Device Name
&&	Single Ampersand

Shortcut	Option
&p	Script debugging port

So, for example, if you set the status bar text to **&s - &t - &c**, you will see "IBM 5250 Terminal - Telnet - Connected" in the Design Tool status bar. The default is **&s - &c**.

DISPLAY MARGINS

Specifies whether to display margins around the Terminal window.

If the Display margins box is checked (default), text will not touch the sides of the Terminal window. Instead, a margin or border is displayed. Displaying margins may improve readability.

If you want text to fill the entire screen, clear this check box. When this check box is clear, text can touch the sides of the Terminal window, which may be useful for smaller monitors and notebook computers.

Depending on the screen resolution and the size of your Terminal window, you may not see an effect in the Terminal window when you clear this check box.

This is because each character takes up the same number of pixels and if your Terminal window is sized so that its width is not evenly divisible by the number of columns, the Design Tool centers the text on the screen.

SUBSTITUTE 0

Specifies whether zeroes displayed in the Terminal window contain slashes: Ø. Selecting this option may make it easier to work with numeric data.

This check box has no effect on fonts that, by their design, already display zeroes with slashes.

By default, this check box is cleared.

Settings Menu - Preferences

On the Settings menu, click Preferences to configure unique settings before you begin modeling:

General

Import

Entity

Pattern

Attribute

Operation

Command

Recordset

GENERAL PREFERENCES

Preferences control the behavior of the Design Tool. These settings do not have any effect on a model deployed on a server.

The following options are available on the General tab of the Preferences Setup dialog box:

Option	Description
When model file opened - Connect to host	Automatically connect to the host when you load a model file
When connected to host - Execute login commands	Execute a login command list when your model connects to a host. This option also navigates to the home entity and initiates any associated event handlers.
When disconnected from host - Execute logout commands	Execute a logout command list when your model disconnects from a host.
When Host Integrator exits	Save model - automatically save your model when exiting the Design Tool. Confirm exit - display a confirmation dialog box before exiting
Edit Options - Show Table Wizard	Launch the Table Wizard whenever you open the Tables dialog box

Option	Description
Initial Design Tool window position	Select Position saved with model if you want the Design Tool to use the window position that was current when you last saved the model. Select Last position used to open the model with current window position the next time you open the Design Tool. The Disabled option does not save any window position (Default).

IMPORT PREFERENCES

The following options are available on the Import tab of the Preferences Setup dialog box and are selected by default:

- Automatically select referenced elements

When an element is selected in the left tree pane, all of the dependencies or references from that element to others are selected to be imported into the destination model.

For example: If you select attribute A in the left pane that refers to a variable V at the bottom of that pane, this preference makes sure variable V is also checked automatically. If Automatically select referenced elements is not selected, variable V_notImported will be created after import, unless you manually select V. All references need to be resolved, either manually or automatically.

- Automatically select all sub items

When an element is selected in the left tree pane all the children of the element are automatically selected to be imported into the destination model.

- Validate after import

After importing the model, the model is validated and the results are displayed. It is important to validate your model and resolve any discrepancies that might have occurred during the import. There is a Validate option available on import dialog box.

- Confirm before overwriting existing element

When you select an element that is already in the destination model the element is replaced. Select this option to confirm your decision before the element is overwritten. You can choose to stop displaying the confirmation message on the dialog box.

ENTITY PREFERENCES

The following options are available on the Entity tab of the Preferences Setup dialog box:

- On undefined screen (except VT)

Select Automatically create new entity if you want the Design Tool to automatically create an entity for each screen you traverse across in a host application. Whenever a new screen displays, you must define at least one pattern for the entity before you can move to the next screen. Note: This setting does not apply to character mode host application models.

- Character mode applications

- **Automatically generate tabstops**

Specifies whether or not the Design Tool creates tabstops automatically as you add entities to your host application model. Note: This setting is designed specifically for character mode applications and will only function if you create a user-defined command list to move the cursor from the Advanced Model Properties dialog box. On the Model menu, click Properties and then click the Advanced button to open this dialog box.

- **Use writable attributes as automatic tabstops**

Select this option to automatically include configured attributes as tabstops when creating new entities. If you want to use this option on an existing entity, select the Use writable attributes as automatic tabstops check box on the Cursor tab.

- Validation defaults

Select Wait for cursor to direct the model to wait for the cursor to be on a specified row and column on the screen before validating the current entity.

- Other preferences

Select Cache all attribute writes to temporarily store attribute values after they are written to the terminal screen. To erase the cached memory, issue an UpdateAttribute or UpdateAttributes command. Attributes are guaranteed to be written in the same order each time an entity is encountered.

PATTERN PREFERENCES

The following options are available on the Pattern tab of the Preferences Setup dialog box:

Default properties

- Field type (3270/5250 only)

Specifies how the host has set a terminal display field attribute on the terminal screen. For example, the host may recognize a text field as an "unprotected" field while areas that cannot be edited by a user are "protected" fields. The default is No.

- Text color (3270/5250 only)

Specifies the terminal display attribute of the text color that appears on a pattern. The default is No.

- Text

Specifies that text characters, either typed into the Text box on the Pattern tab or selected from the terminal screen, be included when adding patterns to entities. The default is Yes.

- Case sensitive

Specifies whether or not text, either typed or selected from the terminal screen, in a selected pattern is case sensitive. The default is Yes.

- Use display attributes for patterns

Specifies what types of physical characteristics to look for on a host screen when adding patterns to entities. By default, the **Use all display attributes that are uniform across a selected area** option is selected. This means that the Design Tool will allow any display characteristics that are similar to be added as patterns on entities. The other option, designed mainly for VT and HP hosts, is **Use only specified display attributes**.

Default position

- Fixed, Relative to cursor

Defines the settings (Row, Height, Col, and Width) that describe the actual position of the pattern on the host screen. Row can be defined with the row number or you can select from one of the following options: Any Row, Current Row, First Row, or Last Row. Col can be defined with the column number or you can select from one of the following options: Any Column, Current Column, First Column, or Last Column.

Select either the Fixed or the Relative to cursor option. The default is Fixed and the coordinates of the pattern are automatically generated and can be viewed on the Pattern tab if pattern search is not defined relative to the cursor or in an expanded region.

Tip: When defining a pattern, set the Row box to Current Row then configure the options in the Properties box. After you've configured the pattern properties, set the Row box to Any Row to ensure accuracy in your pattern configuration

Automatic generation (except VT)

- Generate patterns based on...

Specifies the physical characteristics of the pattern to be recognized during auto-generation. You can generate patterns based on the following:

Field type

Text color

Field type and text color

ATTRIBUTE PREFERENCES

The following options are available on the Attribute tab of the Preferences Setup dialog box:

[Default position](#)

[Name](#)

[Automatic generation](#)

[Advanced attribute preferences](#)

Default Position

Specifies the actual position of the attribute on the host screen. Choose from the following options under Default position on the Attribute tab of the Preferences Setup dialog box:

Option	Description
Fixed (default)	Specifies the start and end position on the entity for the selected attribute. By default, the Design Tool calculates the Row and Col coordinates for you on the Attribute tab.
Relative to cursor	Specifies that the position for the selected attribute is based on the position of the cursor. With this option, the Design Tool calculates the Row and Col coordinates for you on the Attribute tab. The position coordinates are set based on attribute screen region and position type.

Option	Description
Select relative pattern	Specifies that the positioning of the attribute is relative to a pattern created on the Pattern tab.

Name

Specifies the name or names of the currently defined attributes. Under Name on the Attribute tab of the Preferences Setup dialog box, select from the following options:

Option	Description
Generate default	When the user adds a new attribute to an entity, a default name, such as Attribute_1, is generated and default settings are applied in the Position field on the Attribute tab. To edit an attribute name, select it on the Attribute tab in the Entity window and type it in the Name field.
Use text in preceding field	Select this option to create an attribute name based on the protected field text that appears before the attribute field on the terminal screen. This setting only applies to hosts that have defined fields on their terminal screens.

Option	Description
None - prompt for input	Select this option to be prompted with an Enter Attribute Name dialog box.

Generate Attributes Based on

Note

Attributes can be generated automatically for block mode applications only. It is not possible to generate attributes on character mode applications like VT.

Specifies what types of characteristics the Design Tool should look for when generating attributes automatically. The following options are available:

- Every unprotected field (Default)
- Every field
- Every modified field

On the Attribute tab of the Preferences Setup dialog box, select Every Unprotected field when the Generate based on option is selected.

Advanced Attribute Preferences

Click the Advanced button on the Attribute tab of the Preferences Setup dialog box to open the Advanced Attribute Preferences dialog box. Use this dialog box to configure how the Host Integrator processes data after retrieving it from the host.

Fetch

Remove blanks—Select from the Leading, Trailing, or All check boxes to remove spaces that come before, after, or appear anywhere in fetched data.

Update

Erase to end of attribute— Select this check box to instruct the Host Integrator to concatenate as many spaces as necessary to make the length of the input match the length of the attribute. For more information on this attribute, see [Erase to End of Attribute](#) for Attributes.

- Character mode data entry
- Don't wait for echo (Default)— Select this option to transmit the string to the host and immediately move to the next action. Always use this option for block mode applications. For character mode applications, use this option only when you know that data will not be echoed back to the terminal screen. For example, many times alphanumeric keys are used to move between screens and are not echoed.
- Wait for same number of characters to echo—Select this option to wait for the same number of characters to be sent back to the terminal screen after data has been transmitted to the host. To detect the number of echoed characters, Host Integrator transmits the data and waits for the cursor position to move the same number of columns as the length of the data transmitted. For example, if you transmit the name George as your password, a host will often echo six spaces instead of the original text. With this option selected, Host Integrator waits for the cursor to move six columns from the cursor position when the name George was transmitted to the host before moving on to the next command.
- Wait for string to echo at cursor—Select this option to wait for the exact data string to be written back to the terminal screen after data has been sent to the host. This is the most robust option to select when working with character mode hosts. Note: With character mode applications, the cursor must be wherever characters are being written to the screen. For this reason, use this option to synchronize with the host application.
- Wait for next tabstop—Select this option to wait for the cursor to appear at the next tabstop after transmitting data. If tabstops have been defined on the Cursor tab, then the Host Integrator waits for the cursor to appear at one of the defined locations. If no tabstop definitions are applicable, then the Host Integrator waits for the cursor to be anywhere on the screen that is outside the attribute or recordset field to which the data is being written.
- Wait for next tabstop or string to echo at cursor—Select this option to wait for the exact data string to be written back to the terminal screen or to wait for it to appear at the next appropriate tabstop. This option is not recommended.

Operation Preferences

The following options are available on the Operation tab of the Preferences Setup dialog box:

- Default Properties - You can set the Timeout for operations. The default is 10 seconds.
- Operation generation - The default is to automatically generate operations, and prompt to approve each new operation. If you automatically generate operations, you can choose to add operations silently. You can also choose not to generate operations at all.

Operation Generation Preferences

On the Operation tab in the Preferences Setup dialog box, click the Advanced button to open the Operation Generation Preferences dialog box.

Block mode applications

Ignore unmodified fields - When generating operations automatically, only record modified fields

Ignore empty fields - When generating operations, ignore fields that are empty

Record attribute contents with command - Specifies the command to use when recording attributes during operation generation.

Default value command

TransmitToAttr command with an option to Apply default value to server

- Include CheckOperationCondition command - Specifies whether or not a CheckOperationConditions command is added to an operation being generated by the Design Tool.

- Character mode applications

Record initial cursor position - Specifies whether or not to record the first position of the cursor on the terminal screen during operation generation.

See [Operations](#) for more information on these options and [Operation Command Summary](#) for descriptions of commands.

Operation Command Preferences

You can set this option on the Command tab of the Preferences Setup dialog box:

Default Properties - Set the Timeout for commands. The default is 5 seconds.

Recordset Preferences

The following options are available on the Recordset tab of the Preferences Setup dialog box:

[Default top position](#)

[Default record size](#)

[Default bottom position](#)

[Cache all field writes](#)

[Advanced recordset field preferences](#)

[Recording preferences](#)

Recordset Top Position



Note

By default, the Design Tool calculates the Row and Col coordinates for you.

Setting	Description
Fixed position	Specifies the start position of the selected recordset.
Relative to cursor	Specifies the start position for the selected recordset based on the position of the cursor. The position coordinates are based on recordset screen region and position type. The recordset position will be set at the desired offset from the cursor's position when the entity is recognized and validated.
Relative to pattern	Select this option to choose a start position relative to a pattern created on the Pattern tab. Select a pattern from the list and the Design Tool will calculate the Row and Col coordinates for you.

Recordset Record

Specifies the size of the each recordset. The default is Fixed. Select this option if your recordset takes up a set number of rows. Select Variable to specify where the beginning and end of a recordset are on an entity. Use the Recordset Layout tab to define specific coordinates.

Recordset Bottom Position

Note

By default, the Design Tool calculates the Row and/or Column coordinates for you.

Setting	Description
Fixed position	Specifies the end row position of the selected recordset.
Relative to top	Specifies the end position for the selected recordset based on the recordset top position. The position coordinates are set based on recordset screen region and position type.

Setting	Description
Relative to pattern	Select this option to choose a start position relative to a pattern created on the Pattern tab.

Cache All Field Writes

- Creates a temporary storage area for data assigned to recordset fields. To globally configure all recordset field data to be cached on new entities, select the **Cache all field writes** box on the Recordset Preferences dialog box.
- To cache recordset field data on a selected entity, select the entity in the Entity window and select the Cache all field writes box on the Recordset Fields tab.
- When data is assigned to a field from within the Design Tool or from a Host Integrator API, it will be stored in this area rather than the default behavior of being written directly to the terminal. In an operation, field data can be assigned using write commands like `DefaultValue` and `TransmitToField`. When using a Host Integrator connector, field data is assigned with methods like `UpdateRecords` or `UpdateRecordByFilter`. When Cache all field writes is selected, the field data assigned by the Design Tool or the Host Integrator connector is temporarily stored until an operation issues an `UpdateAttribute` or `UpdateAttributes` command to write the stored data to the terminal screen. Alternatively, an operation can call `UpdateAttribute` for each field in the desired order, inserting other commands in between as needed to modify cursor positioning, synchronize with the host state, or meet any other requirements to be executed by the operation.

Advanced Recordset Field Preferences

On the Recordset tab in the Preferences Setup dialog box, click the Advanced button to open the Advanced Recordset Field Preferences dialog box.

If you are not working with a character-mode host, the controls in the Advanced Recordset Field Preferences dialog box are unavailable.

The following options are available:

Fetch: Remove blanks

Select the Leading, Trailing, or All check boxes to remove spaces from fetched data.

Update: Erase to end of field

Select this check box to concatenate as many spaces as necessary to make the length of the input match the length of the field.

Character mode data entry

Recording Preferences

This option is available on the Recording tab of the Preferences Setup dialog box: **When field text is hidden**. This setting is designed for block mode applications.

Variables

You can initialize model variables in several ways using the Host Integrator:

Using the Design Tool, you can use this dialog box to define and initialize model variables.

Using the Host Integrator Server, you can override a variable value stored in the model file using session pooling.

Using an AppConn API, you can change the value of a variable using the SetModelVariable method.

Note

If host data is accessed through a procedure, you should avoid using variables in combination with a procedure unless there is no other alternative. You may find you need to use ReadVarFromAttr and WriteVarToAttr commands to pass data between screens, but all client data should be passed in and returned through the table itself.

The primary advantages of using variables are to provide abstraction from entities and to enable attributes to be accessible from all parts of the Host Integrator. See the Verastream Administrative Console help for information about initializing variables from a session pool model variable list during server execution.

USING VARIABLES

The Variables dialog box contains a list of defined variables. By default, the Design Tool provides the following predefined variables:

cursorPosition

password

userID

Note

The userID and password variables should be initialized at connection since the data for these variables will change each time a new user connects to a host. Under Initialization in the Initialize variable box, select Initialize variable at host connect for both the userid and password variables.

To add/delete a new variable

- Click New. The Design Tool provides a default name, such as Variable1. To change the default name, select it in the Variable name box under Properties.
- Click Delete. If you try to delete a variable that is in use, the Design Tool warns you that the variable in question is in use and cannot be deleted.



Note

A variable can be associated with an attribute and used for conditional operations and data fetching. For more information, see the Attribute Variables tab in the Entity window.

Properties tab options

- Variable name - Type a unique name to programmatically identify the variable in the model file. When you rename a variable, the Design Tool checks all references to it and updates them.
- Current value - Type the value currently assigned to the selected variable. This is the value used by the Design Tool for testing the model file before deploying it to the Host Integrator Server. If a model variable is mapped, the Initialize Variable setting uses the current value as the initialization default.
- Map to setting - If the value of the selected variable can be matched to any of the provided settings, select it from the list.

If you are working with a model which was created with a previous version of Host Integrator, the private key, public key, and passphrase variables will not be visible in the Map to Setting list. You will have to create them by clicking New.

Mapping a setting to a variable allows you to access Host Integrator session features at server runtime. For example, if a connector needs to specify a Telnet Terminal ID that may change from session to session, you can create a variable and map it to the Telnet Terminal ID setting. Another example is mapping the userID and password variables to the Host User Name and Host Password settings to access the AS/400 autosignon feature. If you are not going to utilize this autosignon feature, you should set the userID and password variables to (None).

- Encrypt value - Encrypts the value of the variable in the model file.
- Hidden - Makes the variable inaccessible to the Host Integrator connector after it has been deployed to the Host Integrator Server but still accessible from the model file when its opened in the Design Tool.
- Read - Allows the variable value to be readable from a Host Integrator connector.
- Write - Allows the variable value to be writable from a Host Integrator connector. A variable must be either read or write, or both read and write, or hidden.

Description

Type a description of the selected variable to be included in the documentation generated when using the options in the Export dialog box. In the sample models provided with the Design Tool, a description is provided for all pre-configured variables.

Initialization

The following options allow you to define when and how to initialize each variable after the model file has been copied to the Host Integrator Server:

- Initialize variable

Select one of the settings below to specify when a variable should be initialized. If a model variable is mapped to a setting, the Current Value displayed in Properties is the initialization default.

Initialize variable with default value— The variable will be initialized with the value specified in the Default value box during Host Integrator Server execution. In the Design Tool, the variable is initialized with the value specified in the Design default box.

Require initialization at client connect— The server will require that an initial value be provided at runtime. This value may be passed as part of the Connect() API function. Failure to provide a value will result in an initialization exception. When a user is connecting to a host from the Design Tool, the user is prompted with a dialog box to enter values for all variables marked with this option.

- Variable not initialized— Variables with this setting will remain uninitialized through session initialization. From within the Design Tool, right before an operation is performed, the user is prompted with a dialog box to enter values for all variables used in that operation that are marked with this option. The variable remains uninitialized until a value is written to the variable by an operation or a Host Integrator connector method.

If values are always supplied by model variable list entries or session pool model variable configuration, it is appropriate to select this option to avoid user prompting.

Note

If an uninitialized variable is accessed by a Host Integrator Server (for example, if a WriteVarToLocation command is issued), the server will issue an exception and halt the current activity.

- Default value

Type the default value that will represent the selected variable on the Host Integrator Server.

- Design default

Type the default value that will represent the selected variable in the Design Tool.

Host Events

To open the Host Event Edit dialog box, click Host Events on the Model menu or click the Edit button on the Event Expression Editor dialog box. Use this dialog box to add, edit, or delete global events to be used in operations throughout the modeling process.

Note

The events described here are used to synchronize the Host Integrator server with the state of the host application. These host synchronization events should be distinguished from the event handling that performs or extends actions in the model file itself.

Defining global host events enables you to rely on a specific order of events encapsulated into one WaitForMultipleEvents command in an operation. This feature enables a more reliable way to synchronize how data is being sent from a character mode host. Since Host Integrator handles only one command per packet of data, the WaitForMultipleEvents command is designed to encapsulate several global events into one unique command in an operation.

Enter a name for your event in the Event name box and choose from the following list of possible global events:

- Condition
- Cursor at attribute
- Cursor at recordset field
- Cursor enter terminal field
- Cursor enter position
- Cursor not in terminal field
- Cursor not in position
- Display string
- Keyboard enabled
- Host communication string
- Host communication silence
- Host connection
- Host disconnect
- Host update
- Milliseconds
- New host screen

CONDITION

WaitForCondition event

Events

Event name (WaitForCondition "Entity", "Condition")

Event Description

Waits until the condition is satisfied. The operation containing the event is paused until the condition is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Entity

Argument type: String

Specifies the entity on which the condition will be satisfied.

- Condition

Click the Edit button to open an editing dialog box for the command argument. Use this dialog box to create a condition that is specific to the entity selected above. For example, create a condition that waits for a certain attribute or attributes to be updated.

CURSOR AT ATTRIBUTE

WaitForCursorAtAttribute event

Events

Event name (WaitForCursorAtAttribute "Entity", "Attribute")

Event Description

Begins a wait that is satisfied when the cursor appears in the first coordinate of an attribute on a particular entity. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Entity

Select the entity on which the specified attribute exists.

- Attribute

Select the attribute at which the cursor should wait.

CURSOR AT RECORDSET FIELD

WaitForCursorAtRecordsetField event

Events

Event name (WaitForCursorAtRecordset "Entity", "Recordset", "Field")

Event Description

Begins a wait that is satisfied when the specified recordset field data is received from the host. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Entity

Select the entity on which the specified recordset exists.

- Recordset

Select the recordset that contains the recordset field.

- Field

Select the recordset field at which the cursor should wait. Use the Fields tab to add fields to recordsets.

CURSOR ENTER TERMINAL FIELD

WaitForCursorEnterField event

Events

Event name (WaitForCursorEnterField Row, Column)

Event Description

Begins a wait that is satisfied when the cursor enters the terminal field specified by the row and column coordinates. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Row

Argument type: Integer

The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

- Column

Argument type: Integer

The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

CURSOR ENTER POSITION

WaitForCursorEnterPosition event

Events

Event name (WaitForCursorEnterPosition Row, Column)

Event Description

Begins a wait that is satisfied when the cursor enters the position on the terminal screen specified by the row and column coordinates. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Row

Argument type: Integer

The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

- Column

Argument type: Integer

The minimum value is 1 (column 1 is the first column). The minimum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

CURSOR NOT IN TERMINAL FIELD

WaitForCursorExitField event

Events

```
Event name (WaitForCursorExitField Row, Column)
```

Event Description

Begins a wait that is satisfied when the cursor exits the terminal field specified by the row and column coordinates. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Row

Argument type: Integer

The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

- Column

Argument type: Integer

The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

CURSOR NOT IN POSITION

WaitForCursorExitPosition event

Events

Event name (WaitForCursorExitPosition Row, Column)

Event Description

Begins a wait that is satisfied when the cursor exits the position on the terminal screen specified by the row and column coordinates. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

- Row

Argument type: Integer

The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

- Column

Argument type: Integer

The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal model the Design Tool is emulating.

DISPLAY STRING

WaitForDisplayString event

Events

Event name (WaitForDisplayString "String", "Timeout", Row, Column, Relative)

Event Description

Begins a wait that is satisfied by the appearance of the specified string at the row and column coordinates on the terminal screen. The string can be received from the host or typed by the user. The operation is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

String -Argument type: String

The string that can terminate the wait.

Timeout -Argument type: String

Specifies an interval of time inHH:MM:SS format that can terminate the wait. If this method times out, an error occurs.

Row -Argument type: Integer

The row in which the string must be received.

Column -Argument type: Integer

The column in which the string must be received.

Relative

Select Yes or No to choose whether or not this event is relative to the other events in the Events list.

KEYBOARD ENABLED

WaitForKeyboardEnabled event

Events

Event name (WaitForKeyboardEnabled Duration)

Event Description

Begins a wait that is satisfied by enabling the keyboard. Use the Duration argument to specify how long to wait before an error occurs. The operation is paused until the wait expires in the specified duration or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

Duration -Argument type: Integer

Specifies an interval of time in seconds to wait for the keyboard to be enabled. The default is 1 second.

HOST COMMUNICATION STRING

WaitForCommString event

Events

Event name (WaitForCommString "Text", "Count")

Event Properties

Begins a wait that is satisfied by the reception of the specified string from the host. The operation containing this event is paused until the wait expires or is satisfied.

For VT and character mode HP hosts, you can wait for datastream sequences that include nonprintable characters using the Model Debug Messages dialog box. As the terminal processes characters, you can copy and paste text into the Text box to configure operations that include raw host communications.

- Often the best synchronization for VT and HP is in the datastream itself. Coupled with the `WaitForMultipleEvents` command, it will enable you to be very deterministic about synchronization.
- Use `WaitForCommString` only if you are connecting to your host using a host application that uses character mode rather than block mode. This applies to VT and HP character mode models only.
- If you are connecting directly to a host, use `WaitForDisplayString`.
- You can also record a login command list to determine which command is appropriate for your connection.

Event Parameters

- Text - Argument type: String
The string received from the host that can terminate the wait.
- Count -Argument type: Numeric
Specifies the number of times a string is received from the host. The default value is 1.

HOST COMMUNICATION SILENCE

WaitForCommSilence event

Event name (`WaitForCommSilence Duration`)

Event Description

Waits for the absence of data traveling to or from the host. Use the `Duration` argument to specify how long that period of silence must last before an error occurs.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

`Duration` -Specifies an interval of time to wait for the event to occur in milliseconds. The default is 1000 milliseconds.

HOST CONNECTION

WaitForHostConnection event

Event name (WaitForHostConnection)

Event Description

Waits for a host connection to occur.

Event Properties

Enter a name for your event in the Event name box.

HOST DISCONNECT

WaitForHostDisconnect event

Event name (WaitForHostDisconnect)

Event Description

Waits for the lack of a host connection.

Event Properties

Enter a name for your event in the Event name box.

HOST UPDATE

WaitHostUpdate event

Event name (WaitHostUpdate TopRow, LeftCol, BottomRow, RightCol, RegionType, Relative)

Event Description

Begins a wait that is satisfied when the specified region on the terminal screen is updated by the host. The operation containing the event is paused until the wait expires or is satisfied.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- TopRow -Argument type: Integer

The top row of the updated region. The minimum value is 1 (row 1 is the first row). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

- LeftCol - Argument type: Integer

The left column of the updated region. The minimum value is 1 (column 1 is the first column). The maximum value varies according to the number of rows supported by the terminal screen the Design Tool is emulating.

- BottomRow - Argument type: Integer

The bottom row of the updated region.

- RightCol - Argument type: Integer

The right column of the updated region.

- RegionType - Specifies what mode to use for the region selected on the terminal screen. Select either the Linear or Rectangular position. The default is Rectangular.

- Relative - Argument type: Enumeration

Select whether or not to make this region relative to the cursor's current location by selecting either Yes or No. The default is No.

MILLISECONDS

WaitMilliseconds event

Event name (WaitMilliseconds Duration)

Event Description

Begins a wait that is satisfied after the specified duration of time in milliseconds.

Event Properties

Enter a name for your event in the Event name box.

Event Parameters

- Duration - Argument type: Integer

Specifies an interval of time in milliseconds. The specified event must last this long to satisfy the wait.

NEW HOST SCREEN

WaitForNewHostScreen event

Event name (WaitForNewHostScreen)

Event Description

Waits for a new screen to be sent from the host.

Event Properties

Enter a name for your event in the Event name box.

TO ADD A GLOBAL HOST EVENT

Click the right arrow button and select global host event. To delete a host event, select it from the Events box and click the minus button.

Under Event properties, type a unique event name in the Event name box.

Under Event parameters, configure any of the options available.

When you have finished defining the event, click OK.

To create a WaitForMultipleEvents command using one or more global events:

Open the Operation Edit dialog box, click the right arrow button, then point to Events, and select WaitForMultipleEvents.

Click the Edit button to open the Host Event Expression Editor dialog box.

After you've created your expression using the operand buttons to string together any global events, click OK.

The new WaitForMultipleEvents command is now included in current operation.

Model Properties

On the Model menu, click Properties to open the Model Properties dialog box. Use the following options to configure advanced settings for the current model.

INITIAL LOCATIONS

Configure model location specifications with the following options:

Default home entity - Select a default home entity.

Default keep alive activity timeout - Specify in minutes the amount of time before a keep alive operation is executed on the host. You can define a keep alive operation for any given entity in the Advanced Entity Properties dialog box.

Host connection timeout - Specify in seconds how long a given session on the Session Server should wait for a connection to be made with the host.

Global navigation timeout - Specify in seconds how long navigation between two entities should take to execute before an error message appears.

ALTERNATE FONT CHARACTERS IN MODEL

Select from one of the following options if you are connecting to a host that uses fonts that are not part of the default character sets (such as VT line drawings):

Store default font equivalent—Default. Select this option to store an equivalent font character in the model.


Translate to spaces—Select this option to translate unfamiliar characters to spaces. This option is useful if storing a default equivalent is causing a conflict in screen recognition.

COMMAND LISTS

Create login and logout command lists that will execute after connecting to and before disconnecting from a host. To create a login or logout command list, click the Edit button to open the Command List Edit dialog box.

Login (after host connect)—Lists the commands that make up the login command list.

Logout (before host disconnect)—Lists the commands that make up the logout command list.

If you have attached a model event handler that implements either an Execute Login or Execute Logout event, a lightning bolt  appears to the left. This is a reminder that the event handler may include logic that overrides or extends the behavior of the command list.

To see the entire command list, click the Copy button and paste the contents into a text editor.

Since there are no commands that can define what entity appears after executing a login command list, the Host Emulator's screen tracing capabilities can be used in conjunction with server command lists.

Because the Host Emulator is sitting in place of the host and functions via screen traces, it can trace the entity to send to the Design Tool after a login command list has executed.


Before deploying a model to the Host Integrator Server, you need to define at least one entity in order for your server command lists to execute.

ADVANCED MODEL PROPERTIES

On the Model menu, click Properties, and click the Advanced button to open the Advanced Model Properties dialog box. This dialog box contains global settings to be used with hosts (like VT and character mode HP) that only recognize character mode cursor positioning. (If you are not working with a character-mode host, the controls in the Advanced Model Properties dialog box are unavailable.) To configure cursor movement using an operation defined on a specific entity, use the options available on the Cursor tab. See the Pine model example for more information about configuring cursor movement for character mode applications.

At runtime, Host Integrator uses the configuration hierarchy of entity, global command list, and then global default to decide which direction to move the cursor. For example, if you decide to configure cursor movement for a specific entity on the Cursor tab, this configuration will be used even if you have a global command list defined in the Advanced Model Properties dialog box.

Character Mode Cursor Positioning

The options below cannot be changed if you have attached a model event handler that implements a Move Cursor Forward or Move Cursor Backward event. In this case, a  appears to the left of the associated check box, and Use user-defined tabs to move cursor is automatically selected. The Execute button invokes the event handler (including the default callback to execute the command list, if present). The event handler may include logic that:

- Overrides the cursor movement behavior of the command list.

- Extends the cursor movement behavior of the command list.

- Makes use of user-defined tabs for cursor movement even if no command list is displayed here.

Use arrow keys to move cursor

This is the default. Select this option to use the arrow keys on your keyboard to move the cursor between attributes on an entity. The up and down arrow keys move the cursor one row up or down and the right and left arrow keys move the cursor one column to the right or left.

Use user-defined tabs to move cursor

Select this option to define your own command list for moving the cursor around attributes of an entity. To create a command list to move the cursor forward, select the Move forward check box and then click the Properties button to open the Command List Edit dialog box. After you've created your move forward command list, you can delete it with the Minus button or test it using the Execute button. To create a command list to move the cursor backward, select the Move backward check box and follow the same procedure.

To configure positioning of the cursor on a specified entity, open the Cursor tab in the Entity window.

To set a timeout for a move cursor command list, use the MoveCursor Command List Timeout setting.

4.15.9 Error Messages

This file contains brief explanations for all error messages that can appear as you're using Verastream Host Integrator. The messages are in numerical order.

LOGFATALEXCEPTION

[145] Fatal exception at [msg]

Contact [Technical Support](#).

LOGSEVEREEXCEPTION

[146] Severe exception at [msg]

Contact [Technical Support](#).

MCSABOVEINSTALLLIMIT

[189] The max concurrent session limit has been set above the value set by the installation. Current Limit: [n] Installation Limit: [n]

[Under Construction]

ERRAPPCANNOTCREATEMODELSFOLDER

[2503] The server could not create the models folder.

Check for file system hardware errors or file access settings problems. The application may have to be reinstalled.

ERRAPPOUTOFMEMORY

[2504] Out of memory.

There was not enough system memory to execute the application.

ERRAPPBADINSTALL

[2506] Application was not properly installed, please run setup to repair the installation.

The installation file has either been corrupted or could not be found.

ERRINVALIDEVENTHANDLERNAME

[2508] Event handler name is invalid : [reason]

Event handler names chosen by the user must conform to the rules of the respective programming language.

ERRAPPCMDLINECOMMANDSYNTAX

[2512] Invalid command line syntax.

The arguments supplied to the application on the command line were invalid.

ERRAPPCREATEOBJECTFAILED

[2516] Application object creation failed.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPCREATEWINDOWFAILED

[2518] Application window creation failed.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPINITFAILURE

[2520] Application initialization failed.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPINVALIDCMDLINEFILE

[2522] [filename] - Invalid command line file.

A file specified on the command line is either missing or invalid.

ERRAPPNCSDLLNOTFOUND

[2524] National character set library not found.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPSTARTROMFAILED

[2530] ROM start failed.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRBADLOCALIPADDR

[2532] The local IP address could not be established. Check your hosts file for accuracy.

Check your hosts file or DNS configuration for accuracy. The IP address may be incorrectly specified in your hosts file.

ERRAPPTIMEEXPIRED

[2534] Application time limit expired.

The application is a time-limited version that has expired. Contact [Technical Support](#).

ERRAPPUNRECOGNIZEDOPTION

[2536] Unrecognized option.

An unrecognized option was specified on the application command line.

ERRAPPAADSREGISTRATIONFAILED

[2542] Management server registration failed.

An internal error occurred during product initialization. Deprecated.

ERRAPPKERNELFAILURE

[2544] Application kernel initialization failed.

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPSERVERALREADYRUNNING

[2546] The server is already running.

Only one instance of the Host Integrator Server may be run on a given machine at one time.

ERRAPPINVALIDINSTALLFILE

[2548] [name] is not a valid installation file.

The installation file has been corrupted. Reinstall Host Integrator.

ERRAPPEXECFAILURE

[2550] Server exec failure: [n].

An internal error occurred during product initialization. Contact [Technical Support](#).

ERRAPPMAXINSTANCESACTIVE

[2551] The maximum number of instances [n] are already running.

The application you are attempting to start has an upper limit on the number of instances that can be run on a given machine at any one time.

ERRAPPLISTENINGPORTINUSE

[2553] Service [name] is unable to start, because the required TCP listening port [n] is in use.

The service you are attempting to start, requires a dedicated TCP listening port to be available. Currently this TCP port is in use by another program.

ERRMEMORYALLOCATION

[2554] Unable to allocate memory required for command.

If reducing the system load or adding RAM does not help, then contact [Technical Support](#).

ERRPARSINGMEMORYALLOCATION

[2555] Parsing error, unable to allocate memory required for command.

If reducing the system load or adding RAM does not help, then contact [Technical Support](#).

ERRSERVERCONFIGBUSY

[2556] The server is currently being configured by [name] on [location].

Deprecated in version 7.0. Another administrator is in configure mode. Only one webstation is allowed to configure the server at any one time.

ERRNOTCONFIGURINGSERVER

[2558] You are not currently configuring the server.

Deprecated in version 7.0. A communication error occurred between the Administrative Console and the Server. Log out and then log back into the Administrative Console.

ERRMODELHAVECHANGED

[2559] One or more models have been updated. Saving changes will activate the new model(s).

The version of one or more models stored on the Server is different than the version currently active on the Server.

ERRFILEERROR

[2560] [file name] - File component error.

A Server trace file could not be created or an unrecognized file system error has occurred.

ERRFILENOTFOUND

[2562] [file name] - File not found.

The specified file could not be found. Check the path and case of the file name.

ERRFILECONTENTSCORRUPT

[2563] [file name] - File contents corrupt.

The contents of the file cannot be read by Host Integrator. They may have been altered or corrupted.

ERRNOFILEACCESS

[2564] [file name] - Access to file denied.

The access permissions of the specified file do not allow the attempted operation.

ERRDUPLICATEFILENAME

[2566] [file name] - Duplicate file name.

You attempted to create a file that already exists.

ERRDISKFULL

[2568] [file name] - Disk is full.

Contact your system administrator.

ERRFILEREADONLY

[2570] [file name] - File is marked read only.

You attempted to write to a read-only file.

ERRCANTCREATE

[2572] [file name] - Unable to create specified file.

The specified file name is invalid, or the file exists and is protected. Check the spelling of the path, and make sure the path exists.

ERRFILENAMENOTALLOWED

[2574] [file name] - File name is not allowed by Host Integrator.

The file name contains invalid characters, refers to a directory, or is an invalid file.

ERRNOFILENAME SPECIFIED

[2576] No file name was specified in command.

You must specify a valid filename.

ERRNAMETOOLONG

[2578] [file name] - File name is too long.

Select a shorter filename.

ERRDOMAINTHREADSTARTFAILURE

[2579] Failed to start the domain server thread.

The Server failed to start the domain server. This error could be due to excessive network traffic or low system resources. Deprecated.

ERRPARSERSYNTAX

[2582] Parsing syntax error - [name].

The specified text was not recognized by the parser. Check the text for spelling, comma separation and string quoting accuracy.

ERRPARSERCMNDNAME

[2584] Unrecognized command - [name].

The specified text was not recognized as a valid Host Integrator command.

ERRPARSERNOTENOUGHARGS

[2590] Parsing syntax error - [syntax] - Not enough arguments for command.

The specified command requires more arguments than were supplied. Check the command syntax.

ERRPARSERINVALIDINT

[2592] Parsing syntax error - [syntax] - Integer argument expected.

A numeric argument was expected. Check the command syntax.

ERRPARSERINTRANGE

[2594] Parser integer [value] - argument out of range.

The specified numeric argument was not within the range of accepted values.

ERRPARSERINVALIDSTRING

[2596] Parser invalid string [value] - Quoted string argument expected.

A string delineated by double quotes is required for the command argument.

ERRPARSERINVALIDTIMESTRING

[2598] [string] - Quoted time string argument expected (hh:mm:ss).

A string delineated by double quotes with the specified format is required for the command argument.

ERRPARSERMISSINGQUOTE

[2600] [string] - Closing quote expected.

The closing quote for a string argument was not found.

ERRPARSERPDCNAME

[2602] [name] - Invalid predefined constant name.

The text supplied for the numeric argument does not identify a valid Host Integrator constant.

ERRPDCNAMENOTALLOWED

[2604] [name] - Predefined constant name not allowed in this command.

The Host Integrator constant supplied for the command argument is not valid for the command.

ERRERRORREADINGSETTINGS

[2606] An error occurred reading settings file [name].

The specified file has either been corrupted or does not contain Host Integrator settings.

ERRPARSERUNKNOWN

[2608] Unknown parser error.

An unknown parser error occurred. Contact [Technical Support](#).

ERRPARSERSTRINGTOOLONG

[2610] [string] - String argument too long.

The string supplied for the argument is longer than the command allows.

ERRPARSERCMNDNOTVALID

[2612] [name] - Command not valid for the current context.

The specified command is not valid within the context of the current command.

ERRPARSERCMNDINCOMPLETE

[2614] Command [name] was not completely constructed.

The specified command requires one or more arguments that were not supplied.

ERRCANTCREATEFOLDER

[2618] Could not create folder [name].

The folder specified could not be created. Check the folder for write permissions.

ERRWAITCMDCANCELLED

[2622] A command to wait for an event was cancelled.

The user cancelled a command to wait for an event.

ERREVENTEXPRESSIONFAILED

[2623] Multiple event expression [name] failed.

The specified multiple event expression was not satisfied within the timeout given in the `WaitForMultipleEvents` command.

ERRCOMMANDNOTIMPLEMENTED

[2624] The command [name] has not been implemented.

You might have a client/server version mismatch due to an incomplete product upgrade roll-out. Check your client and server versions. If the problem persists, contact [Technical Support](#).

ERREVENTTIMEOUTCONTEXT

[2625] The event timed out because [reason].

Some events have prerequisites that must be met before the event will be satisfied. This error message provides additional context for an event that timed out in the case that a prerequisite condition was not met.

ERREVENTTIMEOUT

[2626] A command to wait for an event timed out.

In order to keep Host Integrator in sync with the host application, operations and scripts often use various event commands to wait for the host to finish data transfer. Examples include waiting for a string to appear on the screen, the cursor to enter a specific location, or the host to update a particular region of the screen. This error message indicates that one of these commands failed. This usually points to a flawed model or an unexpected response from the host.

ERRWAITFORENTITYCHANGETIMEOUT

[2627] Request to wait for entity change timed out.

When a connector sends terminal commands to directly to Host Integrator instead of executing operations, there is often a need to queue a command to wait for the current entity to change to make sure Host Integrator remains in sync with the host. If that request to wait for a change times out, this error message is returned.

ERRCOMMANDDISABLEDBYMODE

[2628] You cannot execute the [name] command in the current mode.

The state of the Design Tool or Server prevents the specified command from being executed.

ERRSETTINGDISABLEDBYMODE

[2630] You cannot modify the [name] setting in the current mode.

The state of the Design Tool or Server prevents the specified setting from being modified.

ERRWAITFORENTITIESTIMEOUT

[2631] Request to wait for an entity timed out.

The request to wait for one of a list of entities to arrive waits for a period of time specified by the timeout parameter. This error means that the defined period of time elapsed without finding any of the desired entities.

ERRNOTASETTINGSFILE

[2632] The file specified is not a valid Host Integrator settings file.

The file has either been corrupted or does not contain Design Tool settings.

ERREMPYENTITYLIST

[2633] WaitForEntities requires at least one entity on which to wait.

This error occurs when WaitForEntities is called without any entities in the string array. At least one entity must be listed in a valid request.

ERRPOOLDISABLED

[2635] Session pool [name] is disabled.

The session pool is disabled due to a due to a problem with its model configuration. Contact your system administrator.

ERRUNRECOGNIZEDOPTION

[2636] Unrecognized command line switch - [option].

An invalid option was specified on the command line. Check the command line syntax for accuracy.

ERRPOOLSTOPPED

[2637] Session pool [name] is stopped.

The session pool is stopped and cannot allocate host sessions at this time. Contact your system administrator.

ERRSETTINGDISABLEDREADONLY

[2638] The [name] setting is a read-only setting.

The specified setting cannot be directly modified by the user.

ERRALLOCATIONFAILED

[2639] Could not allocate host session: [reason].

Out of system resources or out of system memory. Contact your system administrator.

ERRMODELNOTFOUND

[2640] Model [name] is not supported by any server in domain [name].

A client requested to attach to a model that is not supported by any Servers currently running in the domain. Verify that all Servers in the domain are running and that at least one of them supports the model requested.

ERRSESSIONPOOLNOTFOUND

[2641] Session pool [name] is not supported by any server in domain [name].

A client requested to attach to a session from a pool that is not supported by any Server currently running in the domain. Verify that all Servers in the domain are running and that at least one of them supports the session pool requested.

ERRNOSESSIONSAVAILABLE

[2643] User [name] at [location] was refused a connection to [name]. No servers in domain [name] are able to provide a session because they are either busy or offline.

The domain is operating at its maximum capacity. Contact your system administrator.

ERRREQUESTEDSESSIONNOTFOUND

[2645] Requested session not found.

A client request for a persistent session contained an invalid or outdated session ID.

ERRTISBBOUNDSERROR

[2646] Attempt to access display memory out of bounds.

This error means the Verastream Host Integrator session tried to access data beyond the bounds of the screen buffer. This can result from a request with invalid position parameters or from an internal error. If the request that generated this error contains any terminal screen position parameters, verify that they are valid. Otherwise, contact Technical Support.

ERRTISBINITIALIZEERROR

[2672] Display memory not initialized.

This error occurs when a session fails to initialize in a timely manner. Detail the exact circumstances that generated this error and contact Technical Support.

ERRINVALIDTERMINALMODEL

[2680] Invalid terminal model for the specified terminal type.

The model file may have been altered without using the Design Tool. On the Connection menu, click Session Setup to correct the terminal type and model ID.

ERRKEYNOTAKEYNAME

[2682] The keyboard key name entered [(name)] is not a PC key or scan code key.

An invalid keyboard key name was entered.

ERRRESERVEDKEYSTROKE

[2684] The keystroke entered [(name)] cannot be mapped.

The keystroke entered is a reserved key stroke and cannot be mapped.

ERRINVALIDKBDMODEL

[2686] The keyboard name entered [(name)] is not valid for a [(terminal type)] terminal.

The keyboard name that was entered cannot be used with the current terminal type.

ERRNOMOUSE

[2688] The Keyboard Map Setup dialog box requires that you have a mouse.

You need a mouse to use the options in this dialog box.

ERRKBDNOTFOUND

[2690] The keyboard name ([name]) cannot be found.

The keyboard name was not found. Contact [Technical Support](#).

ERRCANTINITIALIZER8KEYBDSDLL

[2692] Unable to initialize the ATKeybds keyboard DLL.

An essential DLL could not be loaded. Reinstall Host Integrator.

ERRNOR8KEYBDSDLL

[2694] The file atkeybds.dll was not found. This DLL is required for keyboard support.

An essential DLL is missing. Reinstall Host Integrator.

ERRCANTINITIALIZER8KEYBD2DLL

[2696] Unable to initialize the ATKeybd2 keyboard DLL.

An essential DLL could not be loaded. Reinstall Host Integrator.

ERRNOR8KEYBD2DLL

[2698] The file atkeybd2.dll was not found. This DLL is required for keyboard support.

An essential DLL is missing. Reinstall Host Integrator.

ERRCANTLOADTELNET

[2700] Unable to load the TELNET library.

If reducing the system load or adding RAM does not help, then contact [Technical Support](#).

ERRCANTLOADSSH

[2701] Unable to load the SSH library.

Contact [Technical Support](#).

ERRNOHOSTNAME

[2702] No host name has been set.

Please specify a valid host name or IP address for this session.

ERRSSHONLYSUPPORTSVT

[2703] SSH only supports VT terminals.

Please specify a VT Terminal.

ERRTRANSPORTERROR

[2704] An error occurred in communications - [error].

The specified error occurred during communication with the host. Contact your system administrator or [Technical Support](#) for further assistance.

ERRTRANSPORTWRITEERROR

[2706] A transport data write error has occurred.

Check your network settings. If this problem persists, then contact [Technical Support](#).

ERRTRANSPORTOBWRITEERROR

[2708] A transport out-of-band data write error has occurred.

Error writing out-of-band data (for example, Attn or SysRq). Contact [Technical Support](#).

ERRTRACEFILENODATA

[2710] The trace file [name] contains no data.

The trace file was not recorded properly or has been corrupted. Choose a different trace file or re-save the trace.

ERRNOTATRACEFILE

[2712] The file [name] is not a host datacomm trace file.

The file is not a host datacomm trace file, was not recorded properly, or has been corrupted. Choose a different trace file or re-save the trace.

ERRTRANSPORTTIMEDOUT

[2714] A timeout occurred while attempting to communicate with the host.

The keyboard was not reset properly or the host system is slow or unresponsive.

ERRTRACEFILEWRONGSETUP

[2716] Trace file is incompatible with current session setup. Session setup should be: Terminal Type=[type], Terminal Model=[model], Transport Type=[transport].

The current session setup differs from the session setup used to record the trace file. On the Connection menu, click Session Setup to correct the terminal type, model ID, and transport type.

ERRNOTRACEPLAYBACK

[2718] Trace playback of data from host is not allowed while connected.

Disconnect from the host before attempting to play back the trace file.

ERRRECORDLIMITISZERO

[2720] The record limit value must be non-zero.

To correct this error, change the record limit to a non-zero positive value, or elect to not limit the number of records.

ERRSAVEENTITYERROR

[2722] You must save or cancel current changes before proceeding.

Before leaving an entity, you must save or cancel any changes you have made.

ERRINVALIDFONTNAME

[2724] Invalid font name: [name], restoring previous name.

The font name specified is not valid for the Design Tool. Select a different font.

ERRTERMINALTOOTALLFORMODEL

[2736] The current font can't be displayed at the current terminal screen size. Changing to the default font.

The font size is too large for the display.

ERRCOMMANDSMUSTBESPECIFIED

[2738] The commands argument must be specified.

One or more commands are required for the item being configured.

ERRANYROWCOLNOTALLOWED

[2746] Row and column must be explicitly specified.

The use of Any Row or Any Column is not allowed in the current context.

ERRDESTENTITYINVALID

[2750] Destination entity [name] is invalid.

The name of the requested navigation destination could not be resolved. Check the destination's spelling and case.

ERRNOVALIDROUTE

[2752] No valid route between [name] and [name] was found.

No path could be found between the current location and the requested destination. This means there is no combination of operations marked as Use for Dynamic Traversal that can complete the requested traversal.

ERRNAVTIMEOUT

[2754] Navigation from [entity] to [entity] timed out.

The requested navigation took longer than the model's global timeout. This could indicate a faulty operation, a circular path, or that the requested traversal takes longer than the model allows.

ERRMODELTOONEW

[2756] Verastream [version] cannot load models or configurations from later versions of the product. Please use version [version] or newer to load this file.

This error occurs when an attempt is made to load a Verastream model that is newer than the version of Verastream being used to load it.

ERRDEPLOYUNABLETOCONTACTSERVER

[2758] No Verastream server was found at this location.

An unsuccessful attempt was made to contact the VHI server. This could mean that the server name or IP address is incorrect, or that the VHI server is not currently running.

ERRROGUESESSIONSUSPENDED

[2764] A Rogue session was suspended.

A pooled session was identified as a rogue session and suspended.

ERRDELETEREFERENCEDOBJECT

[2766] [Object name] cannot be deleted due to external references.

Before you delete this object, you must remove all its external references.

ERRENTITYREFERENCEDBYTABLE

[2770] The entity is the [name] table's home entity.

Before you delete this entity, you must use the Tables dialog box to choose a different home entity for the referencing table.

ERREntityREFERENCEDByPROCEDURE

[2772] The entity is referenced by the [name] procedure in the [name] table.

Before you delete this entity, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRRECORDSETREFERENCEDByPROCEDURE

[2774] The recordset is referenced by the [name] procedure in the [name] table.

Before you delete this recordset, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRRECORDSETREFERENCEDByOPERATIONCOMMANDLIST

[2775] The recordset is referenced by the command list for operation [name].

Before you delete this recordset, you must use the Operation Edit dialog box to remove it from the referencing command list.

ERRATTRIBUTEREFERENCEDByPROCEDURE

[2776] The attribute is referenced by the [name] procedure in the [name] table.

Before you delete this attribute, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERROPERATIONREFERENCEDByPROCEDURE

[2778] The operation is referenced by the [name] procedure in the [name] table.

Before you delete this operation, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRPROCEDUREREFERENCEDByPROCEDURE

[2780] The procedure is referenced by the [name] procedure in the [name] table.

Before you delete this procedure, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRCOLUMNREFERENCEDByPROCEDURE

[2782] The column is referenced by the [name] procedure in the [name] table.

Before you delete this column, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRFILTERPARAMREFERENCEDBYPROCEDURE

[2784] The filter parameter is referenced by the procedure.

Before you delete this filter parameter, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRDATAPARAMREFERENCEDBYPROCEDURE

[2786] The data parameter is referenced by the procedure.

Before you delete this data parameter, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERROUTPUTPARAMREFERENCEDBYPROCEDURE

[2788] The output parameter is referenced by the procedure.

Before you delete this output parameter, you must use the Procedure Editor dialog box to remove it from the referencing procedure.

ERRCANTSAVEASCURRENTMODEL

[2789] Overwriting the current model file is not allowed from the Save As dialog.

You may not Save As to overwrite the current model. Please use Save instead.

ERROPENSETTINGSWHILECONNECTED

[2790] The current session must be disconnected before opening a new model/settings file. Save work and select Disconnect from the Connection menu.

Log off and or disconnect from the host before attempting to open a new model or settings file.

ERRMOVECURSORTIMEOUT

[2793] The moveCursor command timed out moving to row [n], col [n].

Many types of requests require Host Integrator to move the terminal cursor into a specific location. These requests include navigation or operation execution that contain a MoveCursor command and a request to write attributes to the terminal screen. This error means that Host Integrator tried to move the cursor into the location specified, but the global move cursor timeout expired during a tab routine. If this error is seen intermittently, try increasing the global move cursor timeout setting from its default. A larger timeout interval will not detract from model performance unless the model relies on Verastream to tab backward and forward on the same terminal screen. This requirement can typically be avoided with careful model construction and/or use of the table interface.

ERRDEFAULTSWHILECONNECTED

[2794] The current session must be disconnected before restoring defaults.

Log off and or disconnect from the host before attempting to restore defaults.

ERRDEFAULTCURSORFAILURE

[2795] The terminal's default MoveCursor failed moving to row [n], col [n].

Many types of requests require Host Integrator to move the terminal cursor into a specific location. These requests include navigation or operation execution that contain a MoveCursor command and a request to write attributes to the terminal screen. This error means that Host Integrator tried to move the cursor into the location specified using the default mechanisms built into the product, but was unable to do so. This is because the use of arrow keys is not permitted or was insufficient. The model will need to be modified to customize how Verastream moves the cursor on the terminal screen in question. [See VHI help topic about details on advanced model properties.](#)

ERRMOVECURSORCOMMANDFAILURE

[2797] The moveCursor command list failed to execute.

Many character mode hosts do not allow you to use the error keys to move the cursor. On these hosts, you must override the cursor movement by specifying a different command, such as a tab key. If such a command fails, this error message is logged. Additional error messages from the failed command may also be logged. Many requests have implied requirements to move the cursor, such as writing attributes data to the terminal. This error message can appear even when you have not explicitly requested a cursor movement.

ERRCANTINITIALIZER8KEYBD3DLL

[2798] Unable to initialize the atkeybd3 keyboard DLL.

An essential DLL could not be loaded. Reinstall Host Integrator.

ERRMOVECURSORNOTAB

[2799] MoveCursor command: row [n], col [n] is not a tab stop.

Many types of requests require Host Integrator to move the terminal cursor into a specific location. These requests include navigation or operation execution that contain a MoveCursor command and a request to write attributes to the terminal screen. In some models, Host Integrator has to be given the valid locations for the cursor for each entity. This is done via tabstop definition. This error means that the current entity has tabstops defined, but Host Integrator was required to move the cursor to a place that was not defined as a tabstop. To resolve this problem, redefine an attribute's location, the supplemental tabstop definition, or both.

ERRNOR8KEYBD3DLL

[2800] The file atkeybd3.dll was not found. This DLL is required for keyboard support.

An essential DLL is missing. Reinstall Host Integrator.

ERRMOVECURSORFAILURE

[2801] Verastream was unable to move the cursor to row [n], col [n].

Many types of requests require Host Integrator to move the terminal cursor into a specific location. These requests include navigation or operation execution that contain a MoveCursor command and a request to write attributes to the terminal screen. This error means that Host Integrator tried to move the cursor into the location specified, but was unable to do so. This could be because the use of arrow keys is not permitted and no alternative was provided in the model, or because Host Integrator was already past the location given and could not reverse directions. [See VHI help topic about details on advanced model properties](#)

ERRCANTINITIALIZER8KEYBD4DLL

[2802] Unable to initialize the atkeybd4 keyboard DLL.

An essential DLL could not be loaded. Reinstall Host Integrator.

ERRNOR8KEYBD4DLL

[2804] The file atkeybd4.dll was not found. This DLL is required for keyboard support.

An essential DLL is missing. Reinstall Host Integrator.

ERRCHARACTERECHOTIMEOUT

[2805] A timeout occurred while waiting for the characters [string] to be echoed by the host.

The timeout may be due to a slow host system, network problems, or a connection failure.

ERRPROPERTYALREADYDEFINED

[2806] The [name] property for [property sheet] has already been defined.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRPARSINGPROPERTYALREADYDEFINED

[2807] Cannot parse the data, multiple properties not allowed.

An internal error occurred during parsing of a property sheet. Contact [Technical Support](#).

ERRPROPERTYNOTDEFINED

[2808] The [name] property is required for [property sheet] and has not been defined.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRPARSERPROPERTYNOTDEFINED

[2809] Parsing error, one of the required properties has not been defined.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRINVALIDPROPERTYSHEET

[2810] The requested property sheet is invalid.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRSERIALIZINGFROMSTREAM

[2816] Cannot unserialize from stream.

An error occurred while trying to unserialize data from stream, please contact [Technical Support](#) for assistance.

ERRSERIALIZINGTOSTREAM

[2817] Cannot serialize to stream.

Contact [Technical Support](#).

ERRDISPATCHING

[2818] Dispatching error.

Contact [Technical Support](#).

ERRPARSINGBADCOMMAND

[2819] Parsing error, bad command ID.

Contact [Technical Support](#).

ERRTIMEDOUT

[2820] Verastream client timed out waiting for a response.

The remote WCP object failed to send a response within specified time-out period. Check the remote object log for additional information. For example, the connector will wait 30 seconds (or different value specified with `SetMethodTimeout`) for a response from the VHI server. To determine the activity or errors, check the VHI server log.

ERRINCOMPATIBLEPRODUCTVERSION

[2821] Connection not allowed to this version of product.

Connection cannot be completed as the product version of this component is not allowed to connect to the remote component due to version limitation. Please contact Technical Support for assistance.

ERRSERVERNOTRUNNING

[2822] Connection refused.

Check your network configuration and firewall settings, and make sure that the server is running.

ERRBADSOCKETCLASS

[2824] Socket class not valid.

Host Integrator received an event for a deleted socket.

ERRSHAKETIMEOUT

[2826] Timed out while handshaking.

The connection failed because the remote object did not send a handshake reply within specified time-out period.

ERRCXREFUSED

[2828] Connection to the server cannot be established.

A connection to the remote service cannot be made. Please verify that the services are running on the machine you are connecting to.

ERROBJECTNOTREGISTERED

[2830] Requested session is not currently running.

The service you are connecting to is no longer running. It is likely that it was removed because it timed out while waiting for client connection.

ERRWCPHEADER

[2832] WCP Error - WCP header is not correct.

Contact [Technical Support](#).

ERRCANNOTCREATEWINDOW

[2834] Cannot create window.

Contact [Technical Support](#).

ERRDUPLICATEOBJECT

[2836] Object already exists in the running object table.

Contact [Technical Support](#).

ERRLOOPBACKADDRESS

[2837] Local machine name mapped to a loopback IP address.

Client applications on other machines cannot connect to the Host Integrator server if the machine name is mapped to the network loopback address (127.0.0.1). Usually this condition represents a network configuration error, but it may also occur when laptops or other DHCP clients are not connected to a network. The Host Integrator server will ignore this condition unless the environment variable VHI_IGNORELOOPBACKADDRESS is set to 0. For more information please contact [Technical Support](#).

ERRSOCKETCBKMISSING

[2838] Socket callback not valid or not registered.

Contact [Technical Support](#).

ERRALLOCATIONERROR

[2840] Could not allocate new resources.

Your system is running low on memory.

ERREXCEPTION

[2842] Exception caught.

Contact [Technical Support](#).

ERREXCEPTIONMSG

[2843] Exception at [msg].

Contact [Technical Support](#).

ERROBJECTSHUTTINGDOWN

[2844] Requested session is shutting down.

The service you are connecting to is shutting down. It is possible that the service was removed because it timed out while waiting for a connection.

ERRSOCKETCONNECTTIMEOUT

[2845] Socket Error - Connection timeout.

A connection to the remote service cannot be made. Verify that the services are running on the machine you are connecting to.

ERRSOCKETERROR

[2846] Socket error.

Got a socket error, please contact [Technical Support](#) for assistance.

ERRSOCKETNOBUFFERSPACE

[2848] Socket Error - No buffer space.

It is possible that your system is running low on memory.

ERRSOCKETNOTCONNECTED

[2850] Socket Error - Socket not connected.

The connection to the remote object has been broken.

ERRSOCKETTIMEDOUT

[2852] Socket Error - Socket timed out.

Check your hosts file or DNS configuration for accuracy. The IP address may be incorrectly specified in your hosts file or other name resolution. If the problem persists, contact [Technical Support](#).

ERRSOCKETSHUTDOWN

[2854] Socket Error - Socket has shutdown.

Contact [Technical Support](#).

ERRSOCKETWOULDBLOCK

[2856] Socket Error - Socket would block.

Cannot transmit data to the remote object. It has not finished processing data that was transmitted earlier.

ERRSOCKETINVALIDADDRESS

[2858] Socket Error - Invalid TCP address.

Verify that you have specified a valid IP address.

ERRSOCKETCONNREFUSED

[2860] Socket Error - Socket connection refused.

A connection to the remote service cannot be made. Verify that the services are running on the machine you are connecting to.

ERRSOCKETNETUNREACH

[2862] Socket Error - Network unreachable.

Contact [Technical Support](#).

ERRSOCKETADDINUSE

[2864] Socket Error - Socket address already in use.

Another service or application is listening on a port needed by VHI. Check your network settings and restart the computer. If the problem persists, contact [Technical Support](#).

ERRSOCKETMAXDESCRIPTOR

[2866] Socket Error - No more socket descriptors available.

The system cannot create any more connections. Try reducing the system load. If the problem persists, contact [Technical Support](#).

ERRSOCKETHOSTNOTFOUND

[2868] Socket Error - Invalid host name.

Verify that the name of the host or Server you are connecting to is correct.

ERRNULLPS

[2869] Cannot accept NULL Property Sheet.

Contact [Technical Support](#).

ERRPARSINGGRAMMAR

[2870] Parsing Error - Bad grammar.

Contact [Technical Support](#).

ERRBAD3270BINDRECEIVED

[2871] Error in 3270 Bind Data, received from host.

The 3270 screen size requested by the host is invalid or is not supported. Connect to a different host resource or contact your host system administrator.

ERRPROPERTYVALUEOUTOFRANGE

[2872] The property value for [name] must be between [min] and [max].

Enter a number within the range of allowed values.

ERRDEPLOYMENTPROTOCOL

[2873] The server's deployment protocol was interrupted, canceling deployment.

An internal sequencing error has occurred. If the problem persists after you retry the operation, contact [Technical Support](#).

ERRINVALIDMODELSPATH

[2874] Model path [path] is not valid.

Check the spelling and case of the path, and make sure the path exists on the Server.

ERRENCRYPTINGMODEL

[2875] The server encountered an error encrypting the model [name] during deployment.

An internal deployment error has occurred. If the problem persists after you retry the operation, please contact [Technical Support](#).

ERRMODELALREADYCONFIGURED

[2876] Model [name] has already been configured.

A model can be configured only once on a given Server.

ERRINVALIDMODELNAME

[2878] Model [name] is either not a valid model name or is not stored in a directory with the same name. Please check the name and the model path for accuracy.

Verify that the model file is stored in a directory which has the same name as the model. If the Server is running on a Linux system, verify both the model name and letter case.

ERRINVALIDVMRFILENAME

[2879] [filename] is not a valid VMR file name.

Specify a different VMR file name.

ERRSESSIONPOOLALREADYCONFIGURED

[2880] Session pool [name] has already been configured.

Specify a different name for the new session pool.

ERRINVALIDSESSIONPOOLNAME

[2882] Session pool [name] is not valid.

A client requested to attach to a session with an invalid session pool name.

ERRINVALIDSESSIONPOOLCOUNT

[2884] The number of sessions for session pool [name] must be between 0 and [n].

Enter a number within the range of allowed values.

ERRINVALIDSESSIONPOOLMAX

[2886] The maximum concurrent sessions for session pool [name] must be between 1 and the server max concurrent session limit ([n]).

Enter a number within the range of allowed values.

ERRINVALIDMODELNAMEFORPOOL

[2888] Model [name] is not valid for session pool [name].

Verify that the model path and model still exist on the Server.

ERRMODELFILECORRUPTED

[2890] The model file [file name] has been corrupted.

A change has been made to the specified model without using the Design Tool. Try reloading the model in the Design Tool, validate, and then save the model.

ERRMODELSRIPTSCORRUPTED

[2891] Model [modelname] has been disabled due to one or more missing model event handlers

One or more classes identified as event handlers for objects within the model were not found when loading the model.

ERRSESSIONPOOLLIMITREACHED

[2892] User [name] at [location] was refused a connection to session pool [name]. The maximum session count for the pool ([n]) is already active.

The session pool from which the session was requested has reached its maximum concurrent session count. Contact your system administrator.

ERRSESSIONLIMITREACHED

[2894] User [name] at [location] was refused a connection to model [name], session pool [name]. The maximum concurrent session count ([n]) is already active.

The Server from which the session was requested has reached its maximum concurrent session count. Contact your system administrator.

ERRMODELNOTVALIDATED

[2896] The model file [file name] has not been validated by the Design Tool.

Open the model with the Design Tool and use the model validator to determine the problem with the model. Once the problem is corrected, save the model.

ERRROUTOFMODELVARLISTENTRIES

[2898] User [name] at [location] was refused a connection to session pool [name]. Model variable list [name] has no available entries.

The model variable list assigned to the session pool from which the session was requested does not have any available entries to be allocated to a new session. Contact your system administrator.

ERRMODELVARLISTALREADYCONFIGURED

[2900] Model variable list [name] has already been configured.

Specify a different name for the new model variable list.

ERRUNIQUEVARIABLEREQUIRED

[2902] Model variable list [name] must have at least 1 unique variable.

If your host does not require unique values for the any of the model variables within a model variable list, you may not need a model variable list. If this problem persists, contact [Technical Support](#).

ERRDUPLICATEMODELVARINLIST

[2904] Model variable [name] is specified more than once in list [name].

Model variable lists cannot contain duplicate variables since variables can only be given one value.

ERRDUPLICATEMODELVARENTRY

[2906] Model variable value [name] is specified more than once for [name] in model variable list [name].

A duplicate value for a unique model variable in a model variable list was specified.

ERRDUPLICATEMODELVARINPOOL

[2908] Model variable [name] is specified more than once in session pool [name].

Only one value can be supplied for a model variable associated with a session pool.

ERRDUPLICATEMODELVARINPOOLLIST

[2910] Model variable [name] from the list [name] is already defined for session pool [name].

When more than one model variable list is used for a session pool, the variables in each list cannot overlap. A model variable can only be assigned one value).

ERRINVALIDMODELVARLISTNAME

[2912] [name] is not a valid model variable list name for session pool [name].

The model variable list specified for the given session pool does not exist. Either the name of the list has changed, or the list has been removed.

ERRNOTENOUGHMODELVARLISTENTRIES

[2914] Session pool configuration requires [n] entries for model variable list [name], but the list only has [n] entries.

Each session in the session pool must obtain a unique entry from the model variable list. Either decrease the count of sessions in the pool or add more entries to the model variable list.

ERRMODELVARIABLENOTFOUNDINPOOL

[2916] Model variable [name] does not exist in model [name] for session pool [name].

All model variables specified for a session pool must exist in the model associated with the pool. Check the variable's spelling and case.

ERRMODELVARIABLEFROMLISTNOTFOUND

[2918] Model variable [name] from model variable list [name] does not exist in the model for session pool [name].

All model variables in a model variable list specified for a session pool must exist in the model associated with the pool. Check the variable's spelling and case.

ERRINVALIDSTARTUPENTITYFORPOOL

[2920] Entity [name] is not a valid startup entity for model [name] in session pool [name].

Check the entity's spelling and case.

ERRMODELVARIABLENOTFOUND

[2922] Model variable [name] does not exist in model [name].

A client provided a value for a model variable that does not exist in the model for the session. Check the variable's name and case.

ERRMODELVARIABLELISTTOOSHORT

[2923] The number of entries in the model variable list ([n]) must be equal to or greater than the minimum number of session pools ([n]).

A client did not provide enough entries in model variable list to satisfy the minimum number of sessions in the session pool.

ERRTRACINGALREADYACTIVE

[2924] Tracing is already active on this server.

Stop the current trace before you start tracing to a different file.

ERRTRACESTARTFAILURE

[2925] Internal error while attempting to start tracing.

Check the trace file name and trace folder for write permissions and disk space.

ERRINVALIDTRACECONFIGFILENAME

[2929] [filename] is not a valid trace configuration file name.

The specified file could not be found or contains invalid trace configuration data.

ERRPROPERTYINVALID

[2930] The [name] [type] property in the [n] Property Sheet is invalid.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRPROPERTYVALUEINVALID

[2932] The [name] [type] property's value ([n]) in the [n] Property Sheet is invalid.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRPROPERTYLENGTHINVALID

[2934] The [name] [type] property's length ([n]) in the [n] Property Sheet is invalid.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRROUTOFRANGEPROPERTYVALUE

[2936] The [name] [type] property's value ([n]) in the [n] Property Sheet is out of range ([n]-[n]).

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRINVALIDDATATYPE

[2938] The [name] [type] property in the [n] Property Sheet is an invalid data type.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRINVALIDPROPERTYSHEETLIST

[2940] The creation of a property sheet list for the [name] [type] property failed.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRINVALIDDTDENTRY

[2942] The DTD entry for the [name] [type] property is invalid.

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRREQUIREDPROPERTYNOTPRESENT

[2944] The required [name] [type] property is not present in the Property Sheet: [name].

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRMORETHANONESINGLEPROPERTY

[2946] The 'ONLYONE' [name] [type] property appears more than once in the Property Sheet: [name].

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRUNDEFINEDPROPERTY

[2948] The [name] [type] property is not defined in the DTD for the Property Sheet: [name].

An internal error occurred during validation of a property sheet. Contact [Technical Support](#).

ERRNOTVIEWINGSESSION

[2956] You are not currently viewing the session.

Deprecated in version 7.0. A communication error occurred between the Administrative WebStation and the Server. Log out from the Administrative WebStation and then log back in.

ERRRECORDERFULL

[2958] The internal Host Integrator recorder is full.

Stop recording and either save or discard the operation.

ERRCONDITIONSYNTAX

[2988] [Error] - Condition syntax error ([offset], [length]).

There is a syntax error in the condition. See [VHI help topic about details on conditions and filter syntax](#).

ERRCONDITIONUNEXPECTEDCHAR

[2990] [Error] - Unexpected character(s) in condition ([offset], [length]).

There is a syntax error in the condition. See [VHI help topic about details on conditions and filter syntax](#).

ERRCONDITIONUNEXPECTEDEND

[2992] Unexpected end of condition statement.

There is a syntax error in the condition. See [VHI help topic about details on conditions and filter syntax](#).

ERRCONDITIONATTRIBUTENAME

[2994] [name] - Unrecognized attribute name ([offset], [length]).

The specified attribute does not exist on the current entity.

ERRCONDITIONATTRIBUTEID

[2996] [id number] - Unrecognized attribute ID ([offset], [length]).

Your model file is corrupted. You may be able to recover previous settings from install-dir\Attachmate\Verastream\HostIntegrator\etc\sesssrvr.config.backup. If the problem persists, contact [Technical Support](#).

ERRCONDITIONVARIABLENAME

[2998] [name] - Unrecognized variable name ([offset], [length]).

The specified variable does not exist.

ERRCONDITIONVARIABLEID

[3000] [id number] - Unrecognized variable ID ([offset], [length]).

Your model file is corrupted. You may be able to recover previous settings from install-dir\Attachmate\Verastream\HostIntegrator\etc\sesssrvr.config.backup. If the problem persists, contact [Technical Support](#).

ERRCONDITIONRECORDSETNAME

[3002] [Name] - Unrecognized recordset name in ([offset], [length]).

The specified recordset does not exist on the current entity.

ERRCONDITIONRECORDSETID

[3004] [id number] - Unrecognized recordset ID ([offset], [length]).

Your model file is corrupted. You may be able to recover previous settings from install-dir\Attachmate\Verastream\HostIntegrator\etc\sesssrvr.config.backup. If the problem persists, contact [Technical Support](#).

ERRCONDITIONFIELDNAME

[3006] [Name] - Unrecognized field name ([offset], [length]).

The specified field does not exist on the current entity.

ERRCONDITIONFIELDID

[3008] [id number] - Unrecognized field ID ([offset], [length]).

Your model file is corrupted. You may be able to recover previous settings from install-dir\Attachmate\Verastream\HostIntegrator\etc\sesssrvr.config.backup. If the problem persists, contact [Technical Support](#).

ERRCONDITIONCOMPTYPEMISMATCH

[3010] [Expression] - Comparison type mismatch ([offset], [length]).

The data types in the comparison do not match. Numbers cannot be compared to strings.

ERRCONDITIONNUMBEROUTOFRANGE

[3011] Number out of range ([offset], [length]) - [number].

The number is too large.

ERRCONDITIONNUMERICEXPEXPECTED

[3012] [Expression] - Numeric expression expected ([offset], [length]).

Only numeric expressions can be used with operators "-", "*", and "/".

ERRCONDITIONEVALUATION

[3013] An error occurred while evaluating the condition: [condition].

Check preceding errors for details.

ERRDSEXCEPTION

[3014] Directory Services Exception: [name].

The specified runtime exception (DSEXCEPTION) occurred in the management server code.
Deprecated.

ERRDSEXCEPTION1

[3016] Directory Services Exception: [name] with [name].

The specified runtime exception (DSEXCEPTION1) occurred in the management server code.
Deprecated.

ERRDSEXCEPTION2

[3018] Directory Services Exception: [name] with [name] and [name].

The specified runtime exception (DSEXCEPTION2) occurred in the management server code.
Deprecated.

ERRDSBADCXCATEGORY

[3020] Bad connection category: IP [name] category [name].

Deprecated

ERRDSCONFIGLOCKFAILURE

[3022] Unable to lock configuration: [name].

Deprecated in version 7.0. Another user is logged on to the Administrative WebStation in configure mode. Switch the other user to view mode and try again.

ERRDSDOMAINNOTFOUND

[3024] Domain [name] not found.

The specified domain name is not in the current configuration. Switch to configuration mode and add the domain name.

ERRDSSERVERNOTFOUND

[3026] Server [name] not found.

The specified Server does not appear in the list of Servers registered with this Directory Service. Reinstall the Server and register it with this Directory Service.

ERRDSNOSERVERRUNNING

[3028] No server running in domain [name].

None of the Servers in the specified domain are currently active.

ERRDSSERVERINTWODOMAINS

[3030] Server [name] assigned to domain [name] and domain [name].

You cannot add a Server to more than one domain.

ERRDSAADSPROPERTIESREJECTED

[3032] Management server properties rejected. Reason: [name].

Deprecated

ERRAADSINITIALIZATIONFAILURE

[3033] Management server initialization failure: [name].

Management server initialization encountered an error.

ERRDSADMINISTRATORPROFILEINCOMPLETE

[3035] No OS-groups assigned to Administrator profile.

When you enable security, you need to add at least one operating system group to the Administrator profile to enforce authentication on the next login.

ERRDSAADSSHUTDOWNREJECTED

[3036] Management server shutdown requested by [name] rejected.

Unauthorized attempt to shut down management server. Please use `atstart` to shut down management server.

ERRDSDOMAINDEFINEDMORETHANONCE

[3037] Domain [name] defined more than once.

You cannot add a domain that already exists.

ERRDSSERVERNOTINDOMAIN

[3038] Server [name] not in domain [name].

The specified Server is not a member of the domain. Switch to configure mode and add the Server to the domain.

ERRDSAADSSHUTTINGDOWN

[3039] Management server shutting down.

Authorized shutdown for management server in process.

ERRAAAUTHENTICATIONFAILURE

[3040] Authentication failure for specified userid and password; [name].

The operating system does not recognize the user ID or password. Verify the user ID and password and try again.

ERRAAAUTHORIZATIONFAILURE

[3041] Authorization failure for security profile [name].

The user ID is not a member of an operating system group associated with the specified security profile. Please add the user to the appropriate operating system group.

ERRSSLWANTWRITE

[3042] SSL Error - Not finished writing.

SSL state check failed. Check your network settings. If the problem persists, contact [Technical Support](#).

ERRSSLCERTIFICATE

[3043] SSL connection failed.

The SSL connection has been interrupted, or has timed out. It is also possible that the SSL certificates do not match.

ERRSSLWANTREAD

[3044] SSL Error - Not finished reading.

SSL state check failed. Check your network settings. If the problem persists, contact [Technical Support](#).

ERRSSLWANTX509LOOKUP

[3046] SSL Error - Needs X509 lookup.

SSL state check failed. Check your network settings. If the problem persists, contact [Technical Support](#).

ERRSSLSYSCALL

[3048] SSL Error - Syscall.

An I/O error occurred. Check your network settings. If the problem persists, contact [Technical Support](#).

ERRSSL

[3050] SSL Error - Could not complete the SSL connection.

There may be an SSL certificate mismatch.

ERRSSLFIPSMODE

[3051] SSL Error - Could not set FIPS mode.

FIPS capable SSL libraries may be missing.

ERRSSLZERORETURN

[3052] SSL Error - Zero return.

The TLS/SSL connection has been closed. Check your network settings. If the problem persists, contact [Technical Support](#).

ERROPENSSL

[3053] SSL Error: [message]

Detailed error message reported by OpenSSL library.

ERRSSLDIFFIEHELMAN

[3054] SSL Error - Cannot initialize Diffie-Helman parameters.

Check your network settings. If the problem persists, contact [Technical Support](#).

ERRSSLCIPHERSUITE

[3056] SSL Error - Cannot initialize cipher suite.

Check your certificates and network settings. If the problem persists, contact [Technical Support](#).

ERRSSLCTX

[3058] SSL Error - Cannot initialize SSL context.

Check your key manager and network settings. If the problem persists, contact [Technical Support](#).

ERRSSLNOCLIENTSUPPORT

[3060] SSL Error - No SSL support on client.

Could not negotiate a secure peer connection. Check your key manager and network settings. If the problem persists, contact [Technical Support](#).

ERRSSLNOSERVERSUPPORT

[3062] SSL Error - No SSL support on server.

Could not negotiate a secure peer connection. Check your network settings. If the problem persists, contact [Technical Support](#).

ERRWCPVERSION

[3064] WCP Versions are not compatible.

You might have a client/server version mismatch due to an incomplete product upgrade roll-out. Check your client and server versions. If the problem persists, contact [Technical Support](#).

ERRSSLNOSERVERSESSION

[3066] SSL handshake failure - Server session no longer in ROT.

The service you are connecting to is no longer running. It has probably timed out while waiting for a connection.

ERRPROPERTYSHEETPROCESSING

[3067] Property sheet processing error: [error]

If the problem persists, contact [Technical Support](#).

ERRGRAMMARVERSIONS

[3068] Grammar Error - WCP grammar is out of sync.

You might have a client/server version mismatch due to an incomplete product upgrade roll-out. Check your client and server versions. If the problem persists, contact [Technical Support](#).

ERRWCPPACKETCORRUPTION

[3069] WCP Packet corruption detected: [error]

If the problem persists, contact [Technical Support](#).

ERRNOCURRENTENTITY

[3070] No current entity.

The Host Integrator session has reached an undefined state. Request another session. This is usually an indication of an unexpected host response or a flawed model.

ERRINVALIDPOSITIONPARAMS

[3072] Invalid position parameters. They must be AtCursor and Length, Offset and Length, or TopRow, LeftColumn, NbrRows, and NbrColumns.

Position parameters must be AtCursor and Length, Offset and Length; or TopRow, LeftColumn, NbrRows, and NbrColumns.

ERRPOSITIONPARAMSRANGE

[3074] Position parameters not in valid range.

The position parameters do not fall within the confines of the current terminal screen.

ERRUNPROTECTEDFIELDNOTFOUND

[3076] Unprotected field not found at specified location for inserting string.

A request to write to a protected field has been received. This can happen directly or through an attribute or variable.

ERRENTITYHASNOATTRIBUTES

[3078] Entity [name] has no attributes.

A request was made to read or write attributes that have not been defined on the entity.

ERRDUPLICATEATTRIBUTENAME

[3080] Attribute [name] occurs more than once in the argument list.

Argument lists must contain unique names for each kind of model object. For example, you cannot pass two separate values for Attribute1 in the same SetAttributes() API call.

ERRENTITYNOTFOUND

[3081] The entity [name] was not found.

The name of a specified entity could not be resolved. Check the entity's spelling and case.

ERRATTRIBUTENOTFOUND

[3082] The attribute [name] was not found in entity [name].

The name of a specified attribute could not be resolved. Check the attribute's spelling and case.

ERRTERMOFFSETOUTOFRANGE

[3084] Terminal offset [n] is out of range.

An offset falls outside the confines of the current terminal screen.

ERRAPPLICATIONINVALID

[3086] Application is Invalid.

A valid model has not been loaded. Contact [Technical Support](#).

ERRGETENTITYOPTIONSINVALID

[3088] GetEntityOptions are invalid.

See the API help on meta-data requests for the valid options.

ERRGETVARIABLEOPTIONSINVALID

[3090] GetVariableOptions are invalid.

See the API help on meta-data requests for the valid options.

ERRCANNOTSUSPENDCORRUPTSESSION

[3091] The session or its model has been corrupted and cannot be suspended.

Either the session encountered a fatal exception or the model uses event handlers and the event handler engine has been disabled due to a communication error.

ERRGETTABLEOPTIONSINVALID

[3092] GetTableOptions are invalid.

See the API help on meta-data requests for the valid options.

ERRVARIABLENOTFOUND

[3094] The Variable [name] was not found in the current model.

The name of a specified variable could not be resolved. Check the variable's spelling and case.

ERRDUPLICATEVARIABLENAME

[3096] Variable [name] occurs more than once in the argument list.

Argument lists must contain unique names for each kind of model object. For example, you cannot pass two separate values for Variable1 in the same SetVariables() API call.

ERRDELAYEDINPUTCURRENTITY

[3098] An attempt to set delayed input to the current entity has failed.

Invalid data has been passed in earlier via a SetAttributesDelayed() method call on a connector. For example, the length of the data string may be too long.

ERRRECORDSETNOTFOUND

[3100] Recordset [name] not found in entity [name].

The name of a specified recordset could not be resolved. Check the recordset's spelling and case.

ERRNOCURRENTRECORDSET

[3102] No current recordset.

On entities with more than one recordset, you must explicitly set the current recordset. This error may also appear if you try to perform a recordset command on an entity with no recordsets.

ERRNOCURRENTRECORDINRECORDSET

[3103] No current record in recordset [name].

This error is usually generated when trying to perform an update or operation on a specific record in the recordset, but Host Integrator has not been given a record to treat as current. To resolve this problem, explicitly set the index to a valid value or fetch a record.

ERRINVALIDSCROLLOPERATION

[3104] Invalid Scroll Operation.

A model designer designates operations that implement the various scroll actions available on a recordset. However, not all recordsets can perform each kind of scroll. For example, some recordsets can move forward but not backwards. In that case, a request to PageUp will fail with this error.

ERRDUPLICATEFIELDNAME

[3106] Field [name] occurs more than once in the argument list.

Argument lists must contain unique names for each kind of model object. For example, you cannot pass two separate values for Field1 in the same UpdateRecord() API call.

ERRFIELDNOTFOUND

[3108] The field [name] was not found in recordset [name] of entity [name].

The name of a specified field could not be resolved. Check the field name's spelling and case.

ERRSETRECORDFIELDSBADPARAMS

[3110] SetRecordFields cannot have both a filter expression and a record index parameter.

A request to alter the current record can use a search filter or a specific index, but not both.

ERRPATTERNNOTFOUND

[3112] The pattern [name] was not found in entity [name].

The name of a specified pattern could not be resolved. Check the pattern name's spelling and case.

ERRIMPORTMODELFILEERROR

[3116] Importing Previous Version of Model File [file name] Failed.

Unable to import model file from a previous version due to one or more errors. Contact [Technical Support](#).

ERRLOCALEPROPERTIESFILENOTFOUND

[3118] Locale properties file not found.

Deprecated in version 7.0. File required by the Administrative WebStation was not properly installed. Please uninstall and then reinstall the Administrative WebStation.

ERRAASERVICESCONNECTIONFAILED

[3119] Authentication and authorization services connection failed.

The Server could not establish a connection to management server for user authorization. Check to see that management server is running. **Note:** This error can also appear when management server has been reinstalled.

ERRDIRECTORYSERVICESCONNECTIONFAILED

[3120] Directory services [name] connection failed.

The Server could not establish a connection to management server for registration. If this error occurs on startup, the Server will immediately shut down. Check to see if management server is running.

ERRDSREGISTRATIONFAILED

[3121] Directory services registration failed.

The Server failed to register with management server. Check to see if management server is running.

ERRSESSIONSERVERNOTRUNNING

[3122] Server not running ([name]) - connection failed.

The Server you selected is not running. Start the Server and try again.

ERRAUTHORIZATIONPACKAGEACCESSFAILED

[3123] Could not get authorization package.

Deprecated in version 7.0. Contact [Technical Support](#).

ERRSESSIONSERVERCONFIGMODENOTSET

[3124] Server ([name]) config mode could not be set.

The Server you have selected is in a bad state and cannot switch to configure mode. Stop and then restart the server, and then try again.

ERRSESSIONSERVERVIEWMODENOTSET

[3125] Server ([name]) view mode could not be set.

The Server you have selected is in a bad state and cannot switch to view mode. Stop and then restart the server, and then try again.

ERRRESELECTSERVERNODETOUPDATESTATUS

[3126] Reselect server node to update status.

Refresh the browser to get an updated Server status.

ERRSESSIONALREADYBEINGVIEWED

[3127] Session already being viewed.

A given session can only be viewed by one window at a time.

ERRPREVIOUSCOMMANDFAILEDTOCOMPLETE

[3128] Previous command ([command]) failed to complete, (current command([command])).

Deprecated in version 7.0. The Administrative WebStation failed to complete the specified action. Try again. If the problem persists, stop and then restart the Administrative WebStation.

ERRNOADDITIONALSELECTIONS

[3129] There are no [object name]'s to add.

There are no more possible additions to this list.

ERRDUPLICATENODENAME

[3130] Node name ([name]) already exists.

A duplicate node was found. Refresh the browser and submit your changes again.

ERRNOMODELSAVAILABLE

[3131] There are no models available for creating a session pool.

At least one model should be configured on the Server before creating a session pool.

ERRNEWUSERIDPASSWORD

[3132] Your userid and password will now be used for security. Please ensure that your OS group is in the Administrator profile, then save, logout and login for correct authorization.

Verify that your operating system group is in the Administrator profile, then save, logout, and log back in for correct authorization.

ERRSESSIONSERVERCONNECTIONFAILED

[3133] Server ([name]) connection failed.

The Server you selected is not running. Verify that the Server is running and then try again.

ERRDUPLICATEDIRECTORYSERVICENAME

[3134] Duplicate directory service name ([name]).

The Server Name that you entered already exists in the list of Directory Servers. Enter a different name or login with the existing Directory Server.

ERRINVALIDIRECTORYSERVICENAME

[3135] Invalid directory server name ([name]).

Enter a valid Directory Server name in the Server Name field.

ERRINVALIDINPUTFIELDSYNTAX

[3136] Invalid input field syntax, do not use <, > or &.

Do not use <, > or & in your input field. Use <, > or & instead.

ERRENDOFSESSION

[3137] End of session.

The session that you are viewing has ended on the Server. You cannot view this session anymore.

ERRUNRECOGNIZEDPROPERTY SHEET RETURN

[3138] Unrecognized property sheet returned from server, expected command [name] but received [name].

Deprecated in version 7.0. An internal error has occurred in the Administrative WebStation. Please document the steps leading to this problem and contact [Technical Support](#).

ERRNOMODELVARIABLESAVAILABLE

[3139] There are no model variables available.

All defined model variables are already within the session pool.

ERRBINDNOTRECEIVED

[3140] Data Object did not bind to host session.

Client allocated a session but never connected to it. Check network and NAT router settings. If this problem persists, contact [Technical Support](#).

ERRTERMINATINGREMOTESSESSION

[3141] Could not terminate a remote host session.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. Contact [Technical Support](#).

ERRSERVERSHUTTINGDOWN

[3142] The server is in the process of shutting down.

The Server is either being restarted or has shut down. Contact your system administrator.

ERRTERMINATINGLOCALSESSION

[3143] Could not terminate a local host session.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. Contact [Technical Support](#).

ERRCLIENTWCPCOMMUNICATIONCORRUPT

[3144] WCP Packet corruption detected in communication from the client. Client socket closed.

Check network settings. If this problem persists, contact [Technical Support](#).

ERRDXREMOTESSESSFROMHOST

[3145] Failure telling remote session to disconnect from the host.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRWCPCHANNELBUSY

[3146] WCP Channel is busy.

Try reducing the client load. If the problem persists, contact [Technical Support](#).

ERRDXLOCALSESSFROMHOST

[3147] Failure telling local session to disconnect from the host.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCANNOTSUSPENDPOOLSESSION

[3148] Pool sessions cannot be suspended.

Only model connections can be suspended.

ERRLOADINGREMOTESSESSION

[3149] Could not load a remote session.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCLIENTCONNECTIONDROPPED

[3150] The connection from the client was unexpectedly dropped.

The client was not able to gracefully close the connection.

ERRCLOSINGREMOTESSESSION

[3151] Could not close a remote session.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCLIENTCONNECTIONTIMEOUT

[3152] Allocated session timed out waiting for client connection.

When a client requests a connection to a model or session pool, the client has 60 seconds to bind to the host session. If the timeout elapses, the host session will terminate or return to the pool.

ERRRELEASELOCALSESSIONTOMS

[3153] Local session failed sending release to MS.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCLIENTINACTIVITYTIMEOUT

[3154] Connected session timed out waiting for client activity.

Verify the inactivity timeout configured on the Server or contact your system administrator.

ERRRELEASEREMOTESESSIONTOMS

[3155] Remote session failed sending release to MS.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRPERSISTENTTIMEOUT

[3156] Suspended persistent session timed out waiting for client reconnect.

When a model connection is suspended, a timeout must be provided by the client. This timeout determines how long the session will wait for a reconnect before self-terminating.

ERRDOMAINSERVERINIT

[3157] Domain server initialization failed.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRSECONDARYPROCESSFAILURE

[3158] The server failed to launch a secondary process. All other configuration changes have been activated and saved.

The Server could be running low on system resources. Try restarting the Server or contact [Technical Support](#).

ERROPENINGLOCALCHANNELTOAPMS

[3159] Could not open a local channel to the APMS.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCONSOLETOKENMISSING

[3160] Authorization token missing from the console for config mode.

Deprecated in version 7.0. The Server is configured for Administrative WebStation security, but the webstation did not provide a certificate. Log out and then log back into the Administrative WebStation.

ERROPENINGREMOTECCHANNELTOAPMS

[3161] Could not open a remote channel to the APMS.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRPASSWORDMISSING

[3162] A required password was not supplied from the client.

Server security is enabled, but the client did not provide a password for authorization.

ERRSTOPPINGDOMAINSERVER

[3163] Could not shut down the domain server.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRUSERAUTHORIZATION

[3164] User could not be authorized to use the session.

The user ID and password supplied by the client is invalid or the user does not have permission to connect to a host session.

ERRUPDATINGSERVERLOAD

[3165] Could not get server load for domain.

An internal communication error occurred. This error could be due to excessive network traffic, low system resources, or one of the Servers in the domain shutting down unexpectedly. If the problem persists, contact [Technical Support](#).

ERRVERIFICATIONINITFAILED

[3166] The server failed to initialize the token verifier.

An internal error occurred during security initialization. If the problem persists, contact [Technical Support](#).

ERRUPDATINGSERVERLOADSETUP

[3167] Could not get server load setup for domain.

An internal communication error occurred. This error could be due to excessive network traffic, low system resources, or one of the Servers in the domain shutting down unexpectedly. If the problem persists, contact [Technical Support](#).

ERRTOKENVERIFICATIONFAILED

[3168] The server failed to verify the token from the client.

Deprecated in version 7.0. The token provided by the webstation or management server was not valid. You will get this error if you configured security profiles and turned on security for your server just now. You must first log out and log back in to get a new security token that is authorized to configured this server. If the problem persists even after you logout and log back in, contact [Technical Support](#).

Note: This error can occur if the system time does not match on the machines running management server and the Server.

ERRDOMAINLOADSESSIONFROMSERVER

[3169] Could not load a session from server [name].

An internal communication error occurred. This error could be due to excessive network traffic, low system resources, or one of the Servers in the domain shutting down unexpectedly. If the problem persists, contact [Technical Support](#).

ERRSHUTTINGDOWNSERVERPROCESS

[3170] Could not shut down a server process.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRSETTINGSERVERPROCESSTRACING

[3171] Could not set server process tracing.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRSETTINGSERVERPROCESSLOGGING

[3172] Could not set server process logging.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRSENDINGCFGTOREMOTEPROCESS

[3173] Could not send configuration to remote server process.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRSENDINGVARINFOTOREMOTEPROCESS

[3174] Could not send var info to remote server process.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCLEARINGREMOTEVARINFO

[3175] Could not clear var info on remote server process.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRPREPARINGREMOTESSESSION

[3176] Could not prepare a remote host session.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRGETTINGAUTHORIZATIONFROMAA

[3177] Failed to get authorization info from AA.

The user ID and password from a client could not be authorized due to communication problems with management server. Check to see that management server is running.

ERRRUNTIMEEXCEPTION

[3178] A runtime exception has been caught. Exception reference is [name].

A host session encountered a fatal error. The Server will gracefully shut down and restart. Make a note of the circumstances that led to this error and contact [Technical Support](#).

ERRTIMEOUTWAITINGFORAACHANNEL

[3179] A timeout occurred waiting for an available AA channel.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. Contact [Technical Support](#).

ERRTHREADSTARTFAILURE

[3181] Failed to start a thread for a new session.

The Server failed to start a new host session. This error could be due to excessive network traffic or low system resources. Contact [Technical Support](#).

ERRSECURITYVIOLATION

[3182] Security violation.

An unauthorized connection to the server was attempted. You will get this error if you configured security profiles and turned on security for your server just now. You must first log out and log back in to get a new security token that is authorized to configured this server. If the problem persists even after you logout and log back in, contact [Technical Support](#).

ERRRECOGNIZEDBUTNOTVALIDATED

[3183] At operation timeout, entity [name] had been recognized but could not be validated.

Entities can have validation parameters such as cursor position, conditions, or patterns in addition to a signature. When an operation times out, if an entity's signature was found but it could not be validated, this error is generated in addition to the timeout error.

ERREXCEEDEDMAXIMUMOPERATIONNESTING

[3184] System detected operation nesting exceeded limit of 16.

Operations can call other operations through the Navigate command, up to a maximum of 16 concurrent calls. This error indicates that an unexpected circular loop has occurred in operation execution.

ERRNOOPERATIONSURINGCANCEL

[3185] Operation [name] aborted because system command list cancel is on.

When an error has occurred during an operation, no other operations can be started until the current operation is completed.

ERRUSERCANCELED

[3186] Operation [name] from entity [name] was cancelled.

An operation was cancelled. Unless cancelled by the user in the Verastream Design Tool, this error should be accompanied by another message indicating source of cancellation and/or the command being executed when cancel was pressed.

ERRNAVNOLOCATION

[3187] Navigation could not start; current location is undefined.

Starting a dynamic traversal requires that the session be on a defined entity.

ERRNAVOPERATIONFAILED

[3188] Navigation from [name] to [name] aborted: [name] operation failed.

All navigation requests are filled by executing one or more operations marked as Use for Dynamic Traversal. If one of those operations fails, then the entire navigation request will fail.

ERRNAVINTERMEDIATEFAILURE

[3189] Navigation from [name] to [name] aborted: intermediate navigation error.

A request to navigate between two entities by any available path often requires traversing through several intermediate entities. One of these intermediate navigations failed, causing an overall failure of the navigation request.

ERRNAVINTERMEDIATETIMEOUT

[3190] Intermediate navigation from [entity name] to [entity name] aborted due to parent timeout.

A request to navigate between two entities by any available path often requires traversing through several intermediate entities. This error indicates that an intermediate navigation was aborted because the overall navigation request timed out.

ERROPCONDITIONERROR

[3191] Operation [name] from entity [name] aborted due to condition failure: [expression].

A developer can designate several kinds of conditions to check during an operation. These include required attribute inputs and a string condition expression. This error indicates that a CheckOperationConditions command failed. The error message should detail the type of condition failure. See the operation definition for a list of conditions. If the message seems incorrect, make sure the command is not positioned too early in the command list. It should be positioned after all attributes have been updated and immediately before the command(s) that cause the session to move off the starting entity.

ERROPCOMMANDERROR

[3192] Operation [name] from entity [name] aborted due to command error.

All operations are composed primarily of commands that manipulate the terminal session. This error indicates that the operation failed because one of these constituent commands resulted in an error. In most cases, there should be additional error message(s) giving more specific information about the failed command.

ERROPINTERMEDIATECOMMANDERROR

[3193] Operation [name] from entity [name] aborted; intermediate destination [name] command failed.

Operations can designate intermediate destinations, or screens that may or may not be seen during operation execution. When such an entity is recognized, Host Integrator can be instructed by the model designer to execute additional commands to bypass the screen and reach the ultimate destination. This error indicates one of the commands associated with the listed intermediate destination failed. In most cases, there should be additional error message(s) giving more specific information about the command that failed.

ERROPTIMEOUT

[3194] Operation [name] from entity [name] timed out.

An operation is given a specific amount of time to complete by the model designer. This error indicates that an operation took longer than the allowed amount of time, and the operation was therefore aborted.

ERROPBADLOCATION

[3195] Entity [name] recognized. This is not defined as a valid location during operation [name] from entity [name].

An operation has a set of valid locations that can be recognized during execution. These are the primary destination, the set of alternate destinations, and the set of intermediate destinations. If any other defined location is found during the operation, this error will result. If the model designer wants to have Host Integrator ignore the entity and wait for another, then this entity should be added as an intermediate destination. If the entity constitutes a valid outcome of the operation, then it should be added as an alternate destination. The same rules apply to the origin as any other entity; if the first host update does not result in departure from the origin, the origin may need to be added as an intermediate destination.

ERROPUNDEFINED

[3196] Operation [name] is undefined for entity [name].

The operation named in the request could not be found on the current entity. This could be because the operation is defined on a different entity, or the name could be misspelled or not have the correct case.

ERROPUNKNOWNLLOCATION

[3197] Operation [name] cannot be executed - current location is undefined.

All operations are properties of an entity. In order to execute an operation, the session's current location must be on the operation's owning entity. This error indicates that the session is not currently on any entity.

ERRNAVARENTTIMEOUT

[3198] Navigation from [entity name] to [entity name] aborted due to parent timeout.

The navigation was aborted because another operation that queued the request to navigate timed out. Check the error stack for a failed operation.

ERROPARENTTIMEOUT

[3199] Operation [name] from entity [name] aborted due to parent command timeout.

An operation can be called from another operation by use of the Navigate command. Under these conditions, if the calling operation times out or otherwise fails, then all child operations will report this error.

ERROPREQUIREDATATTRIBUTE

[3200] Attribute [name] is a required input and was not set.

A model designer can designate a set of attributes on an entity as required inputs for an operation. This error means that Host Integrator detected a required attribute that was not updated by the time the CheckOperationConditions command was executed. Check the operation definition for a list of required attributes. If this message seems incorrect, the model designer needs to check that the CheckOperationConditions command is properly positioned after all attribute updates but before the command(s) that cause Host Integrator to leave the origin entity.

ERROPARENTUSER

[3201] Operation [name] from entity [name] aborted. A parent operation detected a user-defined model error.

A model designer can define certain host conditions during an operation as errors. These include arrival at a certain entity or detection of a certain pattern on the screen. If a navigation or operation spawns a child operation via a Navigate command, and Host Integrator detects one of these model-defined error conditions in the parent operation, the child operation will abort and log this error message. There should be additional error messages from the parent operation indicating the exact error.

ERROPUSERPTN

[3202] User-defined error ([name] pattern) detected during operation [name] from entity [name]: [error string].

A model designer can define certain host conditions during an operation to be errors. One type of model-defined error is the detection of a pattern on the terminal screen. This message indicates that Host Integrator found an error pattern on the screen. The error message should include some model-defined text specifying the nature of the problem.

ERROPUSERDEST

[3203] User-defined error ([name] entity) detected during operation [name] from entity [name]: [error string].

A model designer can define certain host conditions during an operation to be errors. One type of model-defined error is the recognition of a particular entity that is defined as an error state. This message indicates that Host Integrator detected such an error entity. The error message should include some model-defined text specifying the nature of the problem.

ERRDUPLICATETABLENAMES

[3204] Duplicate table names: [names].

Two tables with the same name are not allowed. Change one of the table names in the Tables dialog box to correct the problem.

ERRUNIXSHAREDMEMORYFAILURE

[3205] Failed to create a shared memory resource on Linux.

An error occurred while trying to obtain a shared memory resource on a Linux host. If the problem persists, contact [Technical Support](#).

ERRDUPLICATECOLUMNNAMES

[3206] Duplicate column names: [names].

Two columns with the same name in a table are not allowed. Change one of the column names in the Tables dialog box to correct the problem.

ERROPSRIPTUNKNOWNLOCATION

[3207] Script completed operation [name] from entity [name] at an unknown screen.

The model assumes any valid state left by an operation is intended. However, this error will be logged when a script returns from an operation and the terminal session is on an unknown screen because that is an invalid state. This can occur when an operation script completes while the terminal still has unprocessed host data waiting. If this error occurs intermittently or it otherwise appears this could be the problem, use `WaitForEntities` or `CheckTerminal` methods in a loop at the end of the script to let the terminal process the host data and arrive at a defined location.

ERRDUPLICATEPROCEDURENAMES

[3208] Duplicate procedure names: [names].

Two procedures with the same name in a procedure are not allowed. Change one of the procedure names in the Tables dialog box to correct the problem.

ERROFFAILEDSCRIPT

[3209] Operation [name] from entity [name] aborted due to script error.

If a script invoked to execute an operation returns an error, the operation will be aborted. See prior messages for details on the script error.

ERRMINCOLUMNVALUEGREATERTHANMAX

[3210] The min value for the [name] column is greater than the max value.

The specified column's minimum value is greater than its maximum value. Change either the min or max value in the Tables dialog box to correct the problem.

ERROPNOHOSTRESPONSE

[3211] Operation [name] from entity [name] did not receive a host response.

An operation is given a specific amount of time to complete by the model designer. If the operation is configured to expect a host response, it will wait up to this period of time for host data. This error indicates that an operation waited for the allotted amount of time for a host response but did not received one. If this operation is not expecting a host response, uncheck this option in the operation's configuration and use one or more wait commands to synchronize completion of the operation as needed.

ERRMINCOLUMNLENGTHGREATERTHANMAX

[3212] The min length for the [name] column is greater than the max value.

The specified column's minimum length is greater than its maximum length. Change either the min or max length in the Tables dialog box to correct the problem.

ERREXTERNALTIMEOUT

[3213] Operation [name] from entity [name] was halted due to an external timeout.

Operation execution was halted because another model object, such as an event handler script, had its timeout expire. This error should be accompanied by another message indicating the source of the timeout.

ERRFILTERTOOUTPUTTYPEMISMATCH

[3214] Type mismatch between the [name] filter and the [name] output.

The filter parameter is mapped to an output parameter of a different data type. This may cause data to be lost in the translation when the procedure is run.

ERROPABORTSYSTEMERROR

[3215] Operation [name] from entity [name] aborted due to a reported system error.

This message indicates that some object in the model other than the operation itself reported an error. For example, if an entity arrival event handler throws an exception during an operation, this message is a likely result. See other error messages for details about what caused the operation to abort.

ERRPROCFILTERPARAMNOTMAPPED

[3216] The [name] filter parameter is not used in the procedure.

The filter parameter is marked as an input to the procedure but it is not used in the procedure. See [VHI help topic for details on creating procedures](#).

ERROPUSERDESTENTITYCHANGE

[3217] User-defined error ([name] entity) detected during operation [name] from entity [name]: [error string].

A model designer can define certain host conditions during an operation to be errors. One type of model-defined error is the recognition of a particular entity that is defined as an error state. This message indicates that Host Integrator detected such an error entity but finds yet another entity has become current when the error message was constructed. This is usually the result of an entity arrival script on the error entity causing the host to move away from the error entity before Host Integrator can process the error.

ERRPROCDATAPARAMNOTMAPPED

[3218] The [name] data parameter is not used in the procedure.

The data parameter is marked as an input to the procedure but it is not used in the procedure. See [VHI help topic for details on creating procedures](#).

ERROPNOENTITYUPDATE

[3219] Operation [name] from entity [name] did not receive the expected entity changes from the host.

An operation is given a specific amount of time to complete by the model designer. If the operation is configured to expect a host response and none of its destinations are also the origin, it will wait up to this period of time for host data that causes a departure from the origin entity. This error indicates that an operation waited for the allotted amount of time but did not received the expected data from the host. If this operation is not expecting a host response, uncheck this option in the operation's configuration and use one or more wait commands to synchronize completion of the operation as needed. If this operation can end on the origin entity, add that as a destination to the operation definition.

ERRPROCOUTPUTPARAMNOTMAPPED

[3220] The [name] output parameter is not used in the procedure.

The output parameter is marked as an output of the procedure but it is not used in the procedure. See VHI help topic for details on creating procedures.

ERRUSERCANCELEDVARINIT

[3221] The model variable initialization dialog was cancelled.

The model variable initialization dialog was cancelled, canceling the connection attempt.

ERRPROCNOVALIDROUTE

[3222] No route between [entity name] and [entity name] was found.

No route was found between the two specified entities in the procedure. To create a route between the two entities either change an existing operation to default or add a new default operation.

ERRPROCNOROUTEFROMEND

[3223] No route between the last entity [name] and the model's home entity [name] was found.

No route was found between one of the procedure's last entities and the home entity of the model. A navigation route must exist for the procedure to execute on the server.

ERRPRIMARYDESTNOTPROCVALIDDEST

[3224] The primary destination [entity name] of operation [name] is not a destination in the procedure.

The primary destination of the operation is not a branch or error entity in the procedure. Add the entity as a branch or error entity in the Procedure Editor to correct the problem.

ERRPROCNOROUTETOSTART

[3225] No route between the model's home entity [name] and the starting entity [name] was found.

No route was found between the model's home entity and the procedure's starting entity. A navigation route must exist for the procedure to execute on the server.

ERRALTDESTNOTPROCDEST

[3226] The alternate destination [entity name] of operation [name] is not a valid or error destination in the procedure.

The alternate destination of the operation is not a branch or error entity in the procedure. Add the entity as a branch or error entity in the Procedure Editor to correct the problem.

ERRINVALIDSUBPROC

[3227] The sub procedure [name] cannot be executed due to missing parameters.

A compound procedure uses a procedure that cannot be executed due to unsatisfied input parameters.

ERRPROCDESTNOTPRIMARYORALTDDEST

[3228] The entity [name] is not a destination of operation [name].

The branch entity in the procedure is not a primary or alternate destination in the operation. Remove this branch entity in the Procedure Editor to correct this problem.

ERRPROCBRANCHONDYNAMICPATH

[3229] Procedure branching is not allowed on a dynamic path.

A procedure can only branch on a static path. Either delete the branches or change to a static path in the Procedure Editor to correct this problem.

ERRPROCOPREQUIRESATTRIBUTE

[3230] The operation [name] requires attribute [name] to be mapped.

The operation along the path requires the specified attributes to have values. Map input parameters to these attributes in the Procedure Editor or mark the attributes as not required in the operation.

ERRSUPERFLUOUSUBPROC

[3231] The sub procedure [name] is not needed.

A compound procedure uses a procedure that provides no additional information.

ERRPROCENTITYUNREACHABLE

[3232] The entity [name] is unreachable.

The Procedure Wizard could not find a route to this entity. To create a route to the entity, either change an existing operation to default or add a new default operation.

ERRPARAMREADFROMWRITEONLYFIELD

[3233] The parameter [name] is read from the field [name], but the field is write-only.

The output parameter is mapped to a field that is write-only. This will cause an error during execution of the procedure.

ERRCOLUMNLENGREATERTHANATTRIBUTE

[3234] The maximum length of column [name] is greater than the maximum length of attribute [name].

The maximum length of the column is greater than the length an attribute to which it is mapped. This could cause an error during execution of the procedure.

ERRPARAMCOMPAREDTOWRITEONLYFIELD

[3235] The parameter [name] is compared to the field [name], but the field is write-only.

The input parameter is compared to a field that is write-only. This will cause an error during execution of the procedure.

ERRATTRIBUTELENGGREATERTHANCOLUMN

[3236] The maximum length of attribute [name] is greater than the maximum length of column [name].

The maximum length of the column is less than the length an attribute to which it is mapped. This could cause an error during execution of the procedure.

ERRPARAMWRITTENTOREADONLYFIELD

[3237] The parameter [name] is written to the field [name], but the field is read only.

The input parameter is mapped to a field that is read-only. This will cause an error during execution of the procedure.

ERRPARAMWRITTENTOREADONLYATTRIBUTE

[3238] The parameter [name] is written to the attribute [name], but the attribute is read only.

The input parameter is mapped to an attribute that is read-only. This will cause an error during execution of the procedure.

ERRPROCRECORDSETDOESNOTSUPPORTUPDATE

[3239] The recordset [name] on entity [name] does not support direct record update.

You cannot update a recordset in a procedure if the recordset does not support direct record update.

ERRPARAMREADFROMWRITEONLYATTRIBUTE

[3240] The parameter [name] is read from the attribute [name], but the attribute is write-only.

The output parameter is mapped to an attribute that is write-only. This will cause an error during execution of the procedure.

ERRPARAMCOMPAREDTOTYPEONLYATTRIBUTE

[3241] The parameter [name] is compared to the attribute [name], but the attribute is write-only.

The input parameter is compared to an attribute that is write-only. This will cause an error during execution of the procedure.

ERRPROCRECORDSETDOESNOTSUPPORTINSERT

[3242] The recordset [name] on entity [name] does not support record insertion.

You cannot insert into a recordset in a procedure if the recordset does not support insertion.

ERRPROCOPERATIONREFERENCESRECORDSET

[3243] The operation [names] references one or more fields in a recordset.

References to fields are not allowed in this operation. There are only two situations in which fields may be referenced in an operation used in a procedure. The first situation is for operations that lead away from an entity with a recordset marked as First Record Only. The second situation is for operations that lead away from a recordset. In all other circumstances there will be no current record in the recordset.

ERRREFERENCEDOBJECTDOESNOTEXIST

[3244] Internal error - object referenced by [property] does not exist: [object id number].

Your model file is corrupted. Contact Technical Support for further assistance.

ERRPARAMTYPEINVALIDFORPROCTYPE

[3246] Internal error - [parameter type] parameters not valid for [procedure type] procedures.

Your model file is corrupted. Contact [Technical Support](#).

ERRPROPNOTVALIDFORCOLUMNTYPE

[3248] Internal error - [name] property not valid for [name] columns.

Your model file is corrupted. Contact [Technical Support](#).

ERREXPECTEDPROPERTY

[3250] Internal error - [name] property expected in [property sheet].

Your model file is corrupted. Contact [Technical Support](#).

ERRUNEXPECTEDPROPERTY

[3252] Internal error - [name] property unexpected in [property sheet].

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONREFERENCEDBYOPERATIONONENTITY

[3253] The operation is referenced by the [name] operation on the [name] entity.

Before you delete this operation, you must go to the Destinations dialog on the Operation tab of the specified entity and remove it from the referencing operation.

ERRUNEXPECTEDDUPLICATEPROPERTY

[3254] Internal error - multiple [name] properties unexpected in [property sheet].

Your model file is corrupted. Contact [Technical Support](#).

ERRPROCDOESNTSTARTATHOME

[3256] Internal error - the procedure does not start at the table's home entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPROCDOESNTENDATHOME

[3258] Internal error - the procedure does not end at the table's home entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRFILTERPARAMREQINSUBPROC

[3260] Internal error - the [name] filter parameter is not required in the compound procedure, but it is required in the sub procedure [name].

Your model file is corrupted. Contact [Technical Support](#).

ERRDATAPARAMREQINSUBPROC

[3262] Internal error - the [name] data parameter is not required in the compound procedure, but it is required in the sub procedure [name].

Your model file is corrupted. Contact [Technical Support](#).

ERRSQLSYNTAX

[3290] SQL syntax error ([offset], [length]) - [SQL fragment].

There is a syntax error in the SQL statement. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNEXPECTEDCHAR

[3292] Unexpected character(s) in SQL ([offset], [length]) - [SQL fragment].

There is a syntax error in the SQL statement. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNEXPECTEDEND

[3294] Unexpected end of SQL statement.

There is a syntax error in the SQL statement. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLNOTSUPPORTED

[3296] SQL not supported ([offset], [length]) - [SQL fragment].

The SQL keyword(s) specified in the error are not supported. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNSUPPORTEDINSELECTSTMT

[3298] Unsupported SQL in SELECT statement ([offset], [length]) - [SQL fragment].

GROUP BY and HAVING clauses are not supported. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNSUPPORTEDINSETCLAUSE

[3300] Unsupported SQL in SET clause ([offset], [length]) - [SQL fragment].

DEFAULT and NULL cannot be used for values in a SET clause. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNSUPPORTEDINCOMP

[3302] Unsupported SQL in comparison ([offset], [length]) - [SQL fragment].

The quantifiers ALL, SOME and ANY are not supported in comparisons. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLUNSUPPORTEDINROWCTOR

[3304] Unsupported SQL in row constructor ([offset], [length]) - [SQL fragment].

DEFAULT and NULL cannot be used for values in a VALUES clause. See [VHI help topic for details on SQL-92 syntax in Host Integrator](#).

ERRSQLJOINSUNSUPPORTED

[3306] SQL joins not supported ([offset], [length]) - [SQL fragment].

Joins are not supported. See [VHI help topic for details on SQL-92 syntax](#) in Host Integrator.

ERRSQLPARAMETERSUNSUPPORTED

[3308] SQL parameters not supported ([offset], [length]) - [SQL fragment].

Parameters are not supported. See [VHI help topic for details on SQL-92 syntax](#) in Host Integrator.

ERRSQLSUBQUERIESUNSUPPORTED

[3310] SQL subqueries not supported ([offset], [length]) - [SQL fragment].

Sub queries are not supported. See [VHI help topic for details on SQL-92 syntax](#) in Host Integrator.

ERRSQLTABLECORRELATIONSUNSUPPORTED

[3312] SQL table correlations unsupported ([offset], [length]) - [SQL fragment].

Table correlations unsupported. See [VHI help topic for details on SQL-92 syntax](#) in Host Integrator.

ERRSQLTABLENAME

[3314] Unrecognized table name ([offset], [length]) - [SQL fragment].

The specified table does not exist.

ERRSQLCOLUMNNAME

[3316] Unrecognized column name ([offset], [length]) - [SQL fragment].

The specified column in the table does not exist.

ERRSQLORDERBYCOLUMNNAME

[3318] SQL column not specified in SELECT clause ([offset], [length]) - [SQL fragment].

The ORDER BY clause can only be used to sort by columns selected in the SELECT clause.

ERRSQLORDERBYCOLUMNNUM

[3320] SQL column number out of range ([offset], [length]) - [SQL fragment].

The column number in the ORDER BY clause is larger than the number of columns in the SELECT clause.

ERRSQLAMBIGUOUSCOLUMNREF

[3322] SQL ambiguous column reference ([offset], [length]) - [SQL fragment].

A column with the name exists in more than one table. Prefix the table name to resolve the ambiguity.

ERRSQLCOMPTYPEMISMATCH

[3324] SQL comparison type mismatch ([offset], [length]) - [SQL fragment].

The data types in the comparison cannot be compared. Strings and numbers cannot be compared.

ERRSQLCOMPDEGREEMISMATCH

[3326] SQL comparison degree mismatch ([offset], [length]) - [SQL fragment].

There is a mismatch between the number of elements being compared.

ERRSQLLITERALEXPECTED

[3327] SQL literal expression expected ([offset], [length]) - [SQL fragment].

Only literal expressions are allowed.

ERRSQLBOOLEANEXPECTED

[3328] SQL boolean expression expected ([offset], [length]) - [SQL fragment].

Only boolean expressions are allowed.

ERRSQLNUMERICEXPECTED

[3330] SQL numeric expression expected ([offset], [length]) - [SQL fragment].

Only numeric expressions are allowed.

ERRSQLSTRINGEXPECTED

[3332] SQL string expression expected ([offset], [length]) - [SQL fragment].

Only string expressions are allowed.

ERRSQLSCALAREXPECTED

[3334] SQL scalar value expected ([offset], [length]) - [SQL fragment].

Only scalar values are allowed.

ERRSQLNUMBEROUTOFRANGE

[3336] SQL number out of range ([offset], [length]) - [SQL fragment].

The number is too large.

ERRSQLASSIGNTYPEMISMATCH

[3340] SQL assignment type mismatch ([offset], [length]) - [SQL fragment].

Data types in the assignment mismatch. Strings cannot be assigned to numeric columns and numbers cannot be assigned to string columns.

ERRSQLROWCTORDEGREEMISMATCH

[3342] SQL row constructors degree mismatch ([offset], [length]) - [SQL fragment].

The number of columns in the row constructor does not match either the number of columns in the INSERT INTO clause or the number of columns in the table.

ERRSQLMULTIPLESTATEMENTS

[3344] SQL multiple statements not supported.

Multiple SQL statements are not supported.

ERRSQLMULTIPLETABLES

[3346] Multiple table references in an SQL statement are not supported.

Multiple tables are not allowed in the FROM clause.

ERRSQLCOLLATINGSEQUENCENAME

[3348] Unrecognized collating sequence ([offset], [length]) - [SQL fragment].

The specified collating sequence is not recognized. The only recognized collating sequences are CASE_INSENSITIVE and CASE_SENSITIVE.

ERRSQLCOLLATEDATATYPE

[3350] COLLATE can only be used on string data types ([offset], [length]) - [SQL fragment].

The COLLATE operator cannot be used on numeric values.

ERRSQLCOMPCOLLATINGMISMATCH

[3352] SQL collating type mismatch ([offset], [length]) - [SQL fragment].

The strings in the comparison have conflicting collating sequences. Use COLLATE CASE_INSENSITIVE or COLLATE CASE_SENSITIVE to resolve the mismatch.

ERRSQLEXPPLICITCOLLATEMISMATCH

[3354] SQL explicit collating type mismatch ([offset], [length]) - [SQL fragment].

The strings cannot be concatenated because their explicit collating sequences conflict. Use COLLATE CASE_INSENSITIVE or COLLATE CASE_SENSITIVE to resolve the mismatch.

ERRSQLRESOLUTION

[3356] Unable to resolve SQL statement into a set of procedures.

The SQL statement could not be resolved into a set of procedures. This means that no procedure(s) could be found that match the request SQL operation. Subsequent error messages will list the closest procedures, if any.

ERRTABLENAME

[3358] [table] is an unrecognized table.

The specified table for the procedure does not exist.

ERRCOLUMNNAME

[3360] [column] is an unrecognized column in table [table].

The specified column for the procedure does not exist in the table.

ERRPROCNAME

[3362] [procedure] is an unrecognized procedure in table [table].

The specified procedure does not exist in the table.

ERRREQUIREDFILTERMISSING

[3364] The procedure [name] requires [name] as a filter parameter.

The specified filter parameter is a required input of the procedure but is not supplied. All required filter parameters must be supplied.

ERRREQUIREDDATAMISSING

[3366] The procedure [name] requires [name] as a data parameter.

The specified data parameter is a required input of the procedure but is not supplied. All required data parameters must be supplied.

ERRCOLUMNNOTAFILTERINPUT

[3368] The column [name] is not a filter parameter to procedure [name].

The specified column is not a filter input to the procedure but it was supplied.

ERRCOLUMNNOTADATAINPUT

[3370] The column [name] is not a data parameter to procedure [name].

The specified column is not a data input to the procedure but it was supplied.

ERRCOLUMNNOTANOUTPUT

[3372] The column [name] is not an output parameter of procedure [name].

The specified column is not an output parameter but it was requested.

ERRINTEGEROUTOFRANGEFORCOLUMN

[3374] [Value] - Integer out of range. Values for column [name] must be between [min] and [max].

The specified integer is out of range for the column.

ERRINTEGERTOOSMALLFORCOLUMN

[3376] [Value] - Integer out of range. Values for column [name] must be greater than or equal to [n].

The specified integer is too small for the column.

ERRINTEGERTOOLARGEFORCOLUMN

[3378] [Value] - Integer out of range. Values for column [name] must be less than or equal to [n].

The specified integer is too large for the column.

ERRSTRINGTOOLONGFORCOLUMN

[3380] [String] - String too long. Strings for column [name] must be less than or equal to [max] characters long.

The specified string is too long for the column.

ERRSTRINGTOOSHORTFORCOLUMN

[3382] [String] - String too short. Strings for column [name] must be at least [min] character(s) long.

The specified string is too short for the column.

ERRSTRINGEXPECTEDFORCOLUMN

[3384] [Value] - String value expected. The column [name] only allows string values.

The column only allows string values.

ERRINTEGEREXPECTEDFORCOLUMN

[3386] [Value] - Integer value expected. The column [name] only allows integer values.

The column only allows integer values.

ERRFLOATEXPECTEDFORCOLUMN

[3388] [Value] - Float value expected. The column [name] only allows float values.

The column only allows float values.

ERRTABLEDOESNTSUPPORTSELECT

[3390] The table [name] does not support SELECT queries.

No Select procedures have been defined for the table.

ERRTABLEDOESNTSUPPORTUPDATE

[3392] The table [name] does not support UPDATE queries.

No Update procedures have been defined for the table.

ERRTABLEDOESNTSUPPORTINSERT

[3394] The table [name] does not support INSERT queries.

No Insert procedures have been defined for the table.

ERRTABLEDOESNTSUPPORTDELETE

[3396] The table [name] does not support DELETE queries.

No Delete procedures have been defined for the table.

ERRPROCLACKSFILTERS

[3398] The procedure [name] does not match because it lacks the following filter parameters: [filters].

This procedure matches the SQL statement's type, but it cannot be used because it lacks filter parameters that appear in the statement's WHERE clause.

ERRPROCQUIRESFILTERS

[3400] The procedure [name] does not match because it requires the following filters: [filters].

This procedure matches the SQL statement's type, but it cannot be used because it requires filter parameters that do not appear in the statement's WHERE clause.

ERRPROC DUPLICATE FILTERS

[3401] The procedure [name] does not match because of multiple unequal values for the following filters: [filters].

This procedure matches the SQL statement's type, but it cannot be used since multiple unequal values for the same filter parameter appear in the statement's WHERE clause.

ERRPROC LACKS DATA

[3402] The procedure [name] does not match because it lacks the following input data: [data names].

This procedure matches the SQL statement's type, but it cannot be used because it lacks data parameters that appear in the statement's SET (UPDATE statements) or VALUES (INSERT statements) clauses.

ERRPROC REQUIRES DATA

[3404] The procedure [name] does not match because it requires the following input data: [data names].

This procedure matches the SQL statement's type, but it cannot be used because it requires data parameters that do not appear in the statement's SET (UPDATE statements) or INSERT INTO (INSERT statements) clauses.

ERRPROC LACKS OUTPUTS

[3406] The procedure [name] does not match because it lacks the following outputs: [output names].

This procedure matches the SQL statement's type, but it cannot be used because it does not have output parameters that appear in the statement's SELECT clause. If you are using "SELECT * FROM ..." then enable "Allow SQL SELECT statements to return a subset of columns when all columns are requested" on the table in the model.

ERRSQL DUPLICATE ROWS POSSIBLE

[3408] The SQL statement resolves in a set of procedures that may return duplicate rows. Either use SELECT DISTINCT or remove one or more OR's in the WHERE clause.

Host Integrator cannot guarantee that duplicate records will not be returned by the set of procedures chosen. Either use SELECT DISTINCT or remove one or more OR's in the WHERE clause to correct this problem.

ERRPROC FAILED

[3410] Procedure [name] on table [name] failed.

The specified procedure failed. Check preceding errors for details.

ERRPROCANCELED

[3411] Procedure cancelled.

The procedure was cancelled before completing.

ERRPROCUNEXPECTEDENTITY

[3412] Procedure failed due to an unexpected arrival at entity [name].

The procedure failed due to arrival at entity not specified in the procedure. To prevent this error, use the Procedure Editor to add the entity as a branch or error entity.

ERRPROCOPERATIONFAILED

[3413] Procedure failed due to operation %0 on entity %1 failing.

The procedure failed due to an operation failing. To prevent this error, correct the operation that failed.

ERRPROCUNRECOGNIZEDENTITY

[3414] Procedure failed due to an unrecognized entity.

The procedure failed due to arrival at an undefined entity. To prevent this error, create an entity and then use the Procedure Editor to add it as a branch or error entity.

ERRPROCREADATTRIBUTE

[3416] Procedure failed due to an error reading from attribute [name] on entity [name].

The procedure failed due to an error reading from an attribute. Check preceding errors for details.

ERRPROCREADFIELD

[3417] Procedure failed due to an error reading from field [recordset].[field] on entity [name].

The procedure failed due to an error reading from a field. Check preceding errors for details.

ERRPROCWRITEATTRIBUTE

[3418] Procedure failed due to an error writing to attribute [name] on entity [name].

The procedure failed due to an error writing to an attribute. Check preceding errors for details.

ERRPROCWRITEFIELD

[3419] Procedure failed due to an error writing to field [recordset].[field] on entity [name].

The procedure failed due to an error writing to a field. Check preceding errors for details.

ERRPROCUSERDEFINED

[3420] User-defined error detected: [error string]

This is a user-defined error. Check with the model designer for details.

ERRRECORDSETSYNCFAILED

[3422] Synchronization of recordset [name] failed. Unable to find the synchronization record: [n].

Host Integrator was unable to find the synchronization record upon returning the entity with the recordset. Try using Previous Record as the synchronization option for the recordset in the procedure. The synchronization options for the recordset can be changed in the Procedure Editor.

ERRPROCFINDNEXTRECORDFAILED

[3423] No record matching record found in recordset [name] on entity [name].

No record matching could be found in the recordset. One record must be found when using the First Record Only option for the recordset in the procedure.

ERRPROCTIMEDOUT

[3424] Procedure timed out.

The procedure timed out.

ERRPROCINITFAILED

[3426] Procedure initialization failed. Unable to navigate to the procedure starting entity [name].

The procedure failed because Host Integrator was unable to navigate to the procedure's starting entity.

ERRPROCRESETFAILED

[3428] Procedure reset failed. Unable to navigate back to the home entity [name].

Host Integrator failed to navigate back to the procedure's starting entity after a failed run.

ERRATTRIBUTEREFERENCEDBYOPERATIONCOMMANDLIST

[3430] The attribute is referenced by the command list for operation [name].

Before you delete this attribute, you must remove it from the referencing command list.

ERRATTRIBUTEREFERENCEDBYENTITYVALIDATION

[3431] The [name] attribute is referenced by entity validation.

Before you delete this attribute, you must use the Advanced Entity Properties dialog box to remove it from the entity validation.

ERRATTRIBUTEREFERENCEDBYOPERATION

[3432] The attribute is referenced by the [name] operation.

Before you delete this attribute, you must go to the Operation tab and remove it from the referencing operation.

ERRATTRIBUTEREFERENCEDBYRECORDSET

[3433] The attribute is referenced by the [name] recordset.

Before you delete this attribute, you must go to the Recordset tab and remove it from the referencing recordset.

ERRATTRIBUTEREFERENCEDBYOPERATIONONENTITY

[3434] The attribute is referenced by the [name] operation on the [name] entity.

Before you delete this attribute, you must go to the Operation tab of the specified entity and remove it from the referencing operation.

ERRATTRIBUTEREFERENCEDBYRECORDSETSCROLLDOWNTERMINATION

[3435] The attribute is referenced by the scroll down termination in the [name] recordset.

Before you delete this attribute, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRATTRIBUTEREFERENCEDBYRECORDSETSCROLLUPTERMINATION

[3436] The attribute is referenced by the scroll up termination in the [name] recordset.

Before you delete this attribute, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRATTRIBUTEREFERENCEDBYRECORDSETFETCHTERMINATION

[3437] The attribute is referenced by the fetch termination in the [name] recordset.

Before you delete this attribute, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERREVENTREFERENCEDBYINTERMEDIATECOMMANDLIST

[3441] The event is referenced by a `WaitForMultipleEvents` command in the intermediate command list for entity `[name]` in operation `[name]`.

Before you delete this event, you must go to the Operation tab, select the entity in the intermediate list of the Destinations dialog, and use the Command List Editor dialog box to remove it from the referencing command.

ERREVENTREFERENCEDBYOPERATIONCOMMANDLIST

[3442] The event is referenced by a `WaitForMultipleEvents` command for entity `[name]` in operation `[name]`.

Before you delete this event, you must go to the Operation tab and use the Command List Editor dialog box to remove it from the referencing command.

ERREVENTREFERENCEDBYGLOBALCOMMANDLIST

[3443] The event is referenced by the command list for the `[name]` model property.

Before you delete this event, you must remove the event from the command list in this model property.

ERRENTITYREFERENCEDBYOPERATIONCOMMANDLIST

[3444] The entity is referenced by the command list for operation `[name]`.

Before you delete this entity, you must remove it from the referencing command list.

ERRENTITYREFERENCEDBYINTERMEDIATECOMMANDLIST

[3446] The entity is referenced by the intermediate destination command list for entity `[name]` in operation `[name]`.

Before you delete this entity, you must go to the Operation tab and use the Operation Destinations dialog box to remove it from the referencing operation.

ERRENTITYREFERENCEDBYHOMEENTITY

[3447] The entity is the home entity.

Before you delete this entity, you must use the Model Properties dialog box to choose a different home entity.

ERRENTITYREFERENCEDBYHESTARTENTITY

[3448] The entity is the Host Emulator Start entity.

Before you delete this entity, you must use the Model Properties dialog box to choose a different Host Emulator Start entity.

ERRENTITYREFERENCEDBYGLOBALCOMMANDLIST

[3449] The entity is referenced by the [name] command list.

Before you delete this entity, you must remove it from the referencing command list.

ERRENTITYREFERENCEDBYOPERATION

[3450] The entity is referenced by the [name] operation on the [name] entity.

Before you delete this entity, you must go to the Operation tab and remove it from the referencing operation.

ERROPERATIONREFERENCEDBYCURSORMOVEMENT

[3451] The operation is mapped for moving the cursor.

Before you delete this operation, you must use the Cursor tab to select a different operation to move the cursor.

ERROPERATIONREFERENCEDBYKEEPALIVEOPERATION

[3452] The operation is the keep-alive operation.

Before you delete this operation, you must use the Advanced Entity Properties dialog box to choose a different keep-alive operation.

ERROPERATIONREFERENCEDBYRECORDSET

[3454] The operation is referenced by the [name] recordset.

Before you delete this operation, you must go to the Recordset tab and remove it from the referencing recordset.

ERROPERATIONREFERENCEDBYATTRIBUTE

[3455] The operation is referenced by the [name] recordset.

Before you delete this operation, you must go to the Attribute Operations tab and remove it from the referencing attribute.

ERRPATTERNREFERENCEDBYATTRIBUTE

[3456] The pattern is referenced by the [name] attribute.

Before you delete this pattern, you must go to the Attribute tab and remove it from the referencing attribute.

ERRPATTERNREFERENCEDBYENTITYVALIDATION

[3458] The pattern is referenced by the entity validation.

Before you delete this pattern, you must use the Advanced Entity Properties dialog box to remove it from the entity validation.

ERRPATTERNREFERENCEDBYOPERATION

[3460] The pattern is referenced by the [name] operation.

Before you delete this pattern, you must go to the Operation tab and remove it from the referencing operation.

ERRPATTERNREFERENCEDBYRECORDSET

[3462] The pattern is referenced by the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and remove it from the referencing recordset.

ERRPATTERNREFERENCEDBYRECORDSETCOLUMN

[3464] The pattern is referenced by the columns in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and remove it from the referencing recordset columns.

ERRPATTERNREFERENCEDBYRECORDSETFIELD

[3466] The pattern is referenced by the [name] field in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and remove it from the referencing recordset field.

ERRPATTERNREFERENCEDBYRECORDSETRECORD

[3468] The pattern is referenced by the records in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and remove it from the referencing recordset records.

ERRPATTERNREFERENCEDBYRECORDSETSCROLLDOWNTERMINATION

[3470] The pattern is referenced by scroll down termination in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRPATTERNREFERENCEDBYRECORDSETSCROLLUPTERMINATION

[3472] The pattern is referenced by scroll up termination in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRPATTERNREFERENCEDBYRECORDSETFETCHTERMINATION

[3474] The pattern is referenced by fetch termination in the [name] recordset.

Before you delete this pattern, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRVARIABLEREFERENCEDBYATTRIBUTE

[3476] The variable is referenced by the [name] Attribute in the [name] entity.

Before you delete this variable, you must go to the Attribute tab and remove it from the referencing attribute.

ERRVARIABLEREFERENCEDBYGLOBALCOMMANDLIST

[3477] The variable is referenced by the [name] command list.

Before you delete this variable, you must remove it from the referencing command list.

ERRVARIABLEREFERENCEDBYOPERATIONCOMMANDLIST

[3478] The variable is referenced by the command list for operation [name].

Before you delete this variable, you must remove it from the referencing command list.

ERRVARIABLEREFERENCEDBYINTERMEDIATECOMMANDLIST

[3479] The variable is referenced by the intermediate destination command list for entity [name] in operation [name].

Before you delete this variable, you must remove it from the referencing command list.

ERRVARIABLEREFERENCEDBYENTITYVALIDATION

[3480] The variable is referenced by the entity validation.

Before you delete this variable, you must use the Advanced Entity Properties dialog box to remove it from the entity validation.

ERRVARIABLEREFERENCEDBYOPERATION

[3482] The variable is referenced by the [name] operation in the [name] entity.

Before you delete this variable, you must go to the Operation tab and use the Operation Edit dialog box to remove it from the referencing operation.

ERRVARIABLEREFERENCEDBYRECORDSET

[3484] The variable is referenced by the [name] recordset in the [name] entity.

Before you delete this variable, you must go to the Recordset tab and remove it from the referencing recordset.

ERRVARIABLEREFERENCEDBYRECORDSETFIELD

[3486] The variable is referenced by the [name] field in the [name] recordset in the [name] entity.

Before you delete this variable, you must go to the Recordset tab and remove it from the referencing recordset field.

ERRVARIABLEREFERENCEDBYRECORDSETSCROLLDOWNTERMINATION

[3488] The variable is referenced by the scroll down termination in the [name] recordset in the [name] entity.

Before you delete this variable, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRVARIABLEREFERENCEDBYRECORDSETSCROLLUPTERMINATION

[3490] The variable is referenced by the scroll up termination in the [name] recordset in the [name] entity.

Before you delete this variable, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRVARIABLEREFERENCEDBYRECORDSETFETCHTERMINATION

[3492] The variable is referenced by the fetch termination in the [name] recordset in the [name] entity.

Before you delete this variable, you must go to the Recordset tab and use the Recordset Termination Options dialog box to remove it from the referencing recordset.

ERRMISSINGPROPERTY SHEET

[3494] An object exists with an invalid [name] Property Sheet in the [name] Property Sheet.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRNULLPROPERTYSHEETLIST

[3496] An invalid Property Sheet list was created trying to process the [name] Property Sheet.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRINVALIDATTRIBUTESPROPERTY

[3498] The ATTRIBUTES_ property ([name]) in the [name] Property Sheet ([name]) is invalid.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRINVALIDTYPEPROPERTY

[3500] The TYPE_ property ([name]) in the [name] Property Sheet ([name]) is invalid.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRINVALIDLOCATION

[3502] The Location specified by the [name] Property Sheet ([name]) is invalid.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRINVALIDRELATIVEIDPROPERTY

[3504] The relative Pattern specified in the [name] Property Sheet in the [name] Property Sheet ([name]) is invalid.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONSPSVALIDATION

[3506] The validation of the Applications property sheet failed.

An invalid Property Sheet was found. Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONSPSHSTARTENTITY

[3508] The Host Emulator start entity specified in the model is invalid.

Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONSPSNOHSTARTENTITY

[3510] No Host Emulator start entity has been specified in the model.

Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONSPSHHOMEENTITY

[3512] *The Home Entity specified in the model is invalid.*

Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONSPSNOHOMEENTITY

[3514] *No Home Entity has been specified in the model.*

Your model file is corrupted. Contact [Technical Support](#).

ERRAPPLICATIONPSNOENTITY

[3516] *No Entities have been specified in the model.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSAPPENDTEXT

[3518] *The APPENDTEXT_ property has been improperly specified in the [name] attribute in the [name] entity.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSOVERRIDE

[3520] *The override attribute has been improperly specified in the [name] attribute in the [name] entity.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSPREPENDTEXT

[3522] *The PREPENDTEXT_ property has been improperly specified in the [name] attribute in the [name] entity.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSREADDEFAULT

[3524] *The READDEFAULT_ property has been improperly specified in the [name] attribute in the [name] entity.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSREADVARIABLEID

[3526] *The READVARIABLEID_ property has been improperly specified in the [name] attribute in the [name] entity.*

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSYMBOLS

[3528] The SYMBOLS_ property has been improperly specified in the [name] attribute in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSWRITEECHO

[3529] The write-echo attribute has been improperly specified in the [name] attribute in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSWRITEVARIABLEID

[3530] The WRITEVARIABLEID_ property has been improperly specified in the [name] attribute in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRATTRIBUTEPSAUTOTABONUNDERFILL

[3531] The autotab-on-underfill attribute has been improperly specified in the [name] attribute in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRENTITYPSKEEPALIVEOPERATION

[3532] The KEEPALIVEOPERA_ property has been improperly specified in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRERRORENTITYPSPROPERTIES

[3534] The ERRORENTITYID_ property ([name]) has been improperly specified in the ERRORENTITY_ property sheet in the [name] operation in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRENTITYVALIDATIONFAILED

[3535] The validation of entity ([name]) failed.

Make sure that the validation criteria are proper and correctly specified.

ERRINTERMEDIATEPSDESTENTITYID

[3536] The DESTENTITYID_ property has been improperly specified in the INTERMEDIATE_ property sheet in the [name] operation in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRINTERMEDIATEPSDUPDESTENTITY

[3538] The DESTENTITYID_ property ([name]) specified in the INTERMEDIATE_ property sheet in the [name] operation in the [name] entity is a duplicate.

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONPSALTDESTID

[3540] The ALTDESTID_ property ([name]) has been improperly specified in the [name] operation in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONPSDESTINATIONENTITY

[3542] The DESTENTITYID_ property ([name]) has been improperly specified in the [name] operation in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONPSDUPALTDEST

[3544] The ALTDESTID_ property ([name]) specified in the [name] operation in the [name] entity is a duplicate.

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONPSDUPDESTENTITY

[3546] The DESTENTITYID_ property ([name]) specified in the INTERMEDIATE_ property sheet in the [name] operation in the [name] entity is a duplicate.

Your model file is corrupted. Contact [Technical Support](#).

ERROPERATIONPSDUPERRORENTITYS

[3548] The ERRORENTITYID_ property ([name]) specified in the ERRORENTITY_ property sheet in the [name] operation in the [name] entity is a duplicate.

Your model file is corrupted. Contact [Technical Support](#).

ERRVALIDATIONNOPATTERNS

[3551] An entity was found with no patterns.

An Entity requires at least one screen pattern so Host Integrator can recognize it.

ERRPATTERNPSAUTOMATIC

[3552] The Pattern type has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSDELIMITER

[3553] The Pattern type has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSCASESENSITIVITY

[3554] The case sensitivity has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSCOLOR

[3556] The COLOR_ property has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSMODIFIER

[3558] The MODIFIER_ property has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSOPERATOR

[3560] The OPERATOR_ property has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSSIGNATURE

[3562] The signature pattern has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRPATTERNPSSYMBOLS

[3564] One of the symbols properties has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRENTITYDOESNOTMATCHSNAPSHOT

[3565] The signature patterns on entity [name] do not match the snapshot.

Either the snapshot is out of date or the entity signature needs to be changed.

ERRPATTERNPSTEXT

[3566] One or more of the text properties has been improperly specified in the [name] pattern in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRENTITYMATCHESWRONGSNAPSHOT

[3567] The signature patterns on entity [name] match the snapshot for entity [name].

Either the snapshot is out of date or patterns need to be added to make the entity signature unique.

ERRRECORDSETPSAFTERINSERTOPERA

[3568] The AFTERINSERTOPERA_ property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRRECORDSETPSAFTERUPDATECMDLIST

[3569] The AFTERUPDATECMDLIST_ property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRRECORDSETPSBEFOREINSERTOPERA

[3570] The BEFOREINSERTOPERA_ property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRRECORDSETPSBEFOREUPDATECMDLIST

[3571] The *BEFOREUPDATECMDLIST_* property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRRECORDSETPSCONDITION

[3572] The *CONDITION_* property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRRECORDSETBADREGEXFILTERCOND

[3573] The filter condition in the [name] recordset in the [name] entity is incorrect. Edit the recordset options for detailed error information.

The filter condition in the recordset is incorrect. Edit the recordset options for detailed error information.

ERRRECORDSETPSFETCHTYPE

[3574] The *FETCHTYPE_* property has been improperly specified in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRSELECTIONPSHOSTRESET

[3576] The *Host Reset* attribute has been improperly specified in the *SELECTION_* property sheet in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRSELECTIONPSITERATIVE

[3578] The *Iteration* attribute has been improperly specified in the *SELECTION_* property sheet in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRSELECTIONPSNOTSUPPORTED

[3580] The *selection not supported* attribute has been improperly specified in the *SELECTION_* property sheet in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRSELECTIONPSSUPPORTED

[3582] The selection supported attribute has been improperly specified in the SELECTION_ property sheet in the [name] recordset in the [name] entity.

Your model file is corrupted. Contact [Technical Support](#).

ERRSIZEPSFIRSTLASTFIELD

[3584] The First/Last Field attribute has been improperly specified in the [name] property sheet in the [name] property sheet.

Your model file is corrupted. Contact [Technical Support](#).

ERRSIZEPSROWCOLWISE

[3586] The Row/Column Wise attribute has been improperly specified in the [name] property sheet in the [name] property sheet.

Your model file is corrupted. Contact [Technical Support](#).

ERRNODEFAULTPATHTOHOMEENTITY

[3587] Warning: this entity does not have a default path to navigate to the home entity.

If this entity is reached on the Server, the model must disconnect and reconnect to reach the home entity. This will degrade the Server's performance.

ERRINVALIDNAME

[3588] The [object] name you provided is not valid.

This design tool message is always accompanied by additional error messages given the reasons the name was rejected.

ERRRECORDSETCOLUMNWIDTHSVARY

[3589] Warning: Multi-column recordset has columns of varying width.

There is a small possibility that a multi-column recordset with column of differing widths has been incorrectly defined. If so, fetched data would be misaligned.

ERREMPYNAME

[3590] An object name cannot be an empty string.

Object names must be at least one character long.

ERRFIRSTCHARNOTALPHAUNDER

[3592] [Char] invalid. The first character must be alphabetic or underscore.

The specified character is not permitted to be the first character. In particular, a number as the first character can be misinterpreted, lead to the string being treated as a number instead of a string identifier.

ERRNOEXTENDEDCHARS

[3594] [Char] invalid. Only ASCII characters are permitted.

No extended characters are permitted in object names.

ERRCHARNOTALPHANUMUNDER

[3596] [Char] invalid. Only alphanumeric and underscore characters are permitted.

Only alphanumeric characters, as defined by the system locale, and the underscore character are considered valid for use in an object name.

ERROBJNAMETOOLONG

[3598] Name too long. Maximum length is 128 characters.

Object names must be between 1 and 128 characters in length, inclusive.

ERRRESERVEDWORD

[3600] [Name] is a reserved word.

Verastream Host Integrator has only one reserved word: variables, in any letter case. Objects cannot be named with the word variables.

ERRBADX509CERTIFICATE

[3602] Supplied certificate is not a valid X509 certificate.

The security token supplied has a bad format (zero length). Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRNULLTOKENVERIFIER

[3604] Token verifier was not instantiated.

Security token parsing error (NULLTOKENVERIFIER): Contact [Technical Support](#).

ERRTOKENPARSING

[3606] Invalid security token.

Security token parsing error (TOKENPARSING). You will get this error if you configured security profiles and turned on security for your server just now. You must first log out and log back in to get a new security token that is authorized to configured this server. If the problem persists even after you logout and log back in, contact [Technical Support](#).

ERRUNSUPPORTEDSUBJECTTYPE

[3608] Token subject is not supported.

Security token parsing error (UNSUPPORTEDSUBJECTTYPE): Contact [Technical Support](#).

ERRSIGNATUREHASH

[3610] Signature hash does not match token hash.

Security token parsing error (SIGNATUREHASH): Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRINVALIDSIGNATUREISSUER

[3612] Signing principal does not match token issuer.

Security token parsing error (INVALIDSIGNATUREISSUER): Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRUNSUPPORTEDSIGNATURETYPE

[3614] Signature type is not a supported algorithm.

Security token parsing error (UNSUPPORTEDSIGNATURETYPE): Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRINVALIDTOKEN

[3616] Token is invalid.

Security token parsing error (INVALIDTOKEN): Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRSECCTXREADONLY

[3618] Security context cannot be modified in read only mode.

Security token parsing error (SECCTXREADONLY): Contact [Technical Support](#).

ERRTOKENDATEINVALID

[3620] Token current time is outside of validity range.

Security token parsing error (TOKENDATEINVALID): This error can occur if the system time does not match on the machines running the management server and the session server. Check time synchronization. If the problem persists, contact [Technical Support](#).

ERRNOSUBJECTINFO

[3622] Subject information not available.

Security token parsing error (NOSUBJECTINFO): Check your certificate store. If the problem persists, contact [Technical Support](#).

ERRBADBINDING

[3624] Token binding is invalid. This error could be due to multiple ip addresses on the client machine.

Security token parsing error (BADBINDING): Check your network settings. If the problem persists, contact [Technical Support](#).

ERRARGUMENTOBJECTDOESNOTEXIST

[3626] Internal error - Object does not exist ([object id number]) for argument [n] in command [n] in COMMANDLIST_ [object id number].

Your model file is corrupted. Contact [Technical Support](#).

UINAMEHASIMESUFFIXNOTIMPORTED

[3627] [Object] [name] was created only to resolve a reference from an object that you imported. Please remove or rename the object.

During Model Elements import, you imported an object that had a reference to an object that you did not import. To resolve that reference, a placeholder was created. Please remove or rename this placeholder.

UINAMEHASIMESUFFIXNOTREMOVED

[3628] [Object] [name] was created only to resolve an existing reference in the model. Please remove or rename the object.

During Model Elements import, an existing object could not be removed because a reference to it still existed. To resolve that reference, a placeholder was created. Please remove or rename this placeholder.

IMPORTMODELHASDIFFERENTSESSIONTYPE

[3629] Cannot import from model [name] because it has a different Session Type.

You can only import model elements from models with the same Session Type (5250, 3720, HP, VT) as you are currently editing.

ERRNOHOSTSCREEN

[3650] Host did not return any data.

The host system may be unresponsive or Session Setup may be incorrect. On the Connection menu, click Session Setup and verify the session settings.

ERRNOENTITYFOUND

[3651] Initial screen or post-login screen was not recognized.

If a login command list is defined, the screen present at completion must be recognized as a defined entity. Otherwise, the initial host screen must be recognized. This error means that Host Integrator failed to recognize the screen. Possible explanations are the host application has changed, the login sequence was not completed successfully or differs based on the user, or the screen is just not defined in the model.

ERRLOGINSCRIPTFAILED

[3652] Login command list failed.

The login command list is executed on session connect in the Server or upon request in the Design Tool. This error is typically accompanied by another error specifying the exact command that failed. If the error is not reproducible using the Design Tool command debugger, Host Integrator might be getting ahead of the host. You may need to add additional wait commands prior to sending data.

ERRLOGOUTSCRIPTFAILED

[3653] Logout command list failed.

The logout command list is executed when a Server session is being shut down, or upon request in the Design Tool. This error is typically accompanied by another error specifying the command that failed. If the error is not reproducible using the Design Tool command debugger, Host Integrator might be getting ahead of the host. You may need to add additional wait commands prior to sending data. Other problems could include failure to reach the start entity or the start entity being altered in a session pool to a screen not supported by the logout script.

ERRTABSTOPGENDUPLICATE

[3660] Auto generation of tab stops failed due to a duplicate tab stop.

The auto generation of tab stops failed because a duplicate tab stop was encountered. Define the tab stops manually instead.

ERRTABSTOPGENVERIFYFAILED

[3661] Tab stops were generated, but they failed to pass a validation test.

During tab stop generation Host Integrator makes two passes and compares the results. This error occurs when the results of the second pass do not match the first pass. The results of the first pass are still used, but they may not be correct.

ERRMOVECURSORINVALIDLOCATION

[3662] An attempt was made to move the cursor to an invalid location: row [n], col[n].

A request to move the cursor must always supply row and column values that are on the display screen. This error indicates the row and column pair do not constitute a valid screen location.

ERRMOVECURSORRECURSION

[3663] A move cursor command invocation resulted in a recursive move cursor command request. This is not permitted.

The most likely cause of this error is a move cursor command or API call in a cursor forward or backward override. This would result in infinite recursion and is therefore not permitted.

ERRMOVECURSORDEFAULTFAILED

[3664] The cursor default callback moved the cursor to row [n], column [n]. This is not a valid tabstop.

When using a cursor tab override, calling the default callback to move the cursor forward or backward must result in the cursor moving to a valid tabstop. This message is the result when a move cursor forward or backward default callback moves the cursor but the resulting location is not defined as a tabstop.

ERRMOVECURSORDEFAULTCANCELED

[3665] The cursor default callback was cancelled due a parent request error, timeout, or cancellation.

Any cursor default callback will stop if a system interrupt for a timeout, user cancellation in the Design Tool, or an error. Other error messages reported subsequent to this should provide information about what caused the cursor callback to halt.

ERRPROPGRINIT

[3670] Error while trying to initialize Property Manager.

Host Integrator is unable to initialize the Property Manager. Close the current Administrative WebStation browser session, then restart the Administrative WebStation on the machine where it is installed. Start a new Administrative WebStation browser session, and try again to logon to the Administrative WebStation.

ERRPROPMSGREXCEPTION

[3671] Exception while trying to initialize Property Manager.

Host Integrator is unable to initialize the Property Manager. Close the current Administrative WebStation browser session, then restart the Administrative WebStation on the machine where it is installed. Start a new Administrative WebStation browser session, and try again to logon to the Administrative WebStation.

ERRCONSOLESESSIONTIMEOUT

[3672] Console session timed out.

The Administrative WebStation session has timed out because it has been inactive for 30 minutes. When the Administrative WebStation times out, it returns to the login panel. The timeout is configured on the web server running the Administrative WebStation.

ERRCONSOLESESSIONEXISTS

[3673] Console session exists.

You have started more than one instance of the Administrative WebStation from a single browser session or have ended an Administrative WebStation session without first logging out. Each Administrative WebStation session must run in its own browser session; log out from all additional sessions. If you want to run multiple instances of the Administrative WebStation, start each session in its own browser window. If you continue to experience problems, close all browser sessions and then try to start an Administrative WebStation browser session.

ERRNOAADS

[3674] Valid Directory Server required for login.

A valid directory server has not been specified for this installation of the Administrative WebStation. On the Administrative WebStation Login panel, click the Directory Serv button to specify a directory server.

ERRNOFINGERPRINT

[3675] Unable to extract fingerprint from Directory Server.

There is a problem with the currently selected directory server. Most likely the specified name is invalid or the directory server is not functioning. On the Administrative WebStation login page, click Directory Serv and verify that the specified directory server name is correct. If you continue to experience a problem, try stopping and then restarting the directory server.

ERRNOCURSORFORWARDCOMMANDLIST

[3680] *There is no command list defined for move cursor forward.*

Your model file is corrupted. Contact [Technical Support](#).

ERRNOCURSORBACKWARDCOMMANDLIST

[3681] *There is no command list defined for move cursor backward.*

Your model file is corrupted. Contact [Technical Support](#).

ERRHETRACEMISSING

[3690] *Obsolete*

ERRHETRACENEEDSRERECORD

[3691] *Obsolete*

ERRINCORRECTCHECKCONDITIONENTITY

[3695] *The CheckOperationConditions command can only be executed on start entity [name]; current entity is [name].*

The CheckOperationConditions command checks the status of the terminal screen for various model-defined properties. If the screen has already changed from the origin entity, Host Integrator will issue this error because it cannot check the conditions.

ERRCHECKCONDITIONNOTALLOWED

[3696] *The CheckOperationConditions command is only valid in an operation. It is not permitted in other command lists.*

The CheckOperationConditions command was executed in the context of a command list such as login or logout. Remove the command from the command list.

ERRNOCURRENTOPERATION

[3697] *The CheckOperationConditions command is invalid - there is no current operation.*

The CheckOperationConditions command was executed in a context other than operations. Remove the command removed from the command list.

ERRINTERNALERROR

[3700] *Internal error ([file] line [line]).*

An internal error occurred. If the problem persists, contact [Technical Support](#).

ERRCANTAUTOGENFORVT

[3701] Patterns/attributes cannot be generated automatically for a VT host.

To add a pattern/attribute to an entity, select a section of the host screen and then click New Pattern/Attribute on the Pattern/Attribute tab.

ERRAUTOGENNOATTRIBUTES

[3702] No attributes are automatically generated.

To add an attribute to an entity, select a text field on the host screen and click New Attribute on the Attribute tab.

ERROPERATIONCOMMANDERROR

[3703] Command could not be saved. Command: [name], [argument] could not be parsed.

Use the Operation Edit dialog box to edit the invalid command. In the Operation Edit dialog box, select a command and press F1 or click Command Help for detailed online help for that command.

ERRNOCONDITION

[3704] You must enter a condition.

Use the Condition Edit dialog box to create a condition.

ERRNOVALIDATEPATTERNS

[3705] You must select at least one unused pattern.

To move a pattern from the Unused box to the Check box, use the right arrow button. The only patterns that will appear in the Unused box will be patterns that are not considered to be a part of the entity signature.

ERRTABSTOPTESTFAILED

[3706] Tabstop test failed.

A tabstop in the list does not accurately reflect an attribute on the entity. Click Generate to identify the tabstop positions automatically.

Note: You must first configure a move cursor command list in the Advanced Model Properties dialog box.

ERRINVALIDDIR

[3707] Can't create the logging directory.

Choose a different name for the logging directory.

ERRCANTCLOSEBOXWHENRUNNINGSCRIPT

[3708] The Model Properties dialog cannot be closed while a command list is being executed. Please try again later.

To cancel the execution of a command list, click Cancel button in the Command List in Progress dialog box.

ERRNOMOVEFORWARDBACKWARDSSCRIPTS

[3709] You must select at least one operation: Move forward or Move backward.

To select an operation to move the cursor forward, select the Move forward check box and then choose an operation. To select an operation to move the cursor backward, select the Move backward check box and follow the same procedure.

ERRCANTNAVIGATE

[3710] You must define an operation to navigate to the selected entity.

To add an operation to an entity, click New Operation on the Operation tab, choose an operation destination, and define a command list. Alternatively, if the Design Tool is set to generate operations automatically, an operation will be generated as you manually traverse to the selected entity.

ERRCANTMATCHSNAPSHOT

[3711] Warning: the defined patterns on this entity do not match the associated snapshot.

Use the Signature Analyzer dialog box to identify pattern mismatches, then edit or delete the bad patterns on the Pattern tab. Alternatively, you can connect to the host, navigate to this entity, and click Update in the General tab of the Advanced Entity Properties dialog box to update the snapshot.

ERRCANTFINDSNAPSHOT

[3712] Snapshot not found for entity [name].

To create a snapshot for an entity, connect to the host, navigate to that entity, and click the Update button on the General tab of the Advanced Entity Properties dialog box.

ERRCANTUNIQUELYIDENTIFYENTITY

[3713] Error: the defined patterns do not constitute a unique signature.

Use the Signature Analyzer to compare the patterns of this entity with other entities in your host application model. Add one or more patterns on the Pattern tab to uniquely identify this entity.

ERRCANTUPDATEENTITY

[3714] The current entity cannot be updated.

An error occurred while attempting to update the current entity. Please document the sequence of events that led to this error, click Cancel on the entity window, save your work to a new file, and contact [Technical Support](#).

ERRCANTDUPLICATENAME

[3715] The name [name] is already used.

Choose a different name.

ERRNOOPERATIONDESTINATION

[3716] You must select an operation destination.

Choose an operation destination.

ERRNOOPERATIONCOMMAND

[3717] Operation command list cannot be empty. Use the editing buttons to add and edit commands.

Use the Operation Edit dialog box to create an operation command list.

ERRNONAME

[3718] A name is required.

Enter a name.

ERRAUTOGENNOPATTERNS

[3719] No patterns are automatically generated.

To add a pattern to an entity, select a static section of the host screen and click New Pattern on the Pattern tab.

ERRPATTERNALREADYUSEDFORVALIDATION

[3720] The current pattern is already used for validation. To use it in entity signature, please go to the Advanced Entity Properties dialog and remove the pattern from the 'wait for patterns' list first.

A pattern cannot be used for both entity signature and validation. To remove a pattern from the validation list, open the Advanced Entity Properties dialog box, go to the Validation tab, select that pattern, and click Remove.

ERRNOPATTERNS

[3721] This Entity type requires at least one screen pattern so Host Integrator can recognize it.

To add a pattern to an entity, use the cursor to select a static section of the host screen and click the New Pattern button on the Pattern tab.

ERRTISBOUTOFBOUNDS

[3722] A write to display memory was out of bounds. Please document the sequence of events that led to this error, exit Host Integrator, and contact Technical Support.

An error occurred while attempting to display a host screen. Please document the sequence of events that led to this error, exit Host Integrator, and contact [Technical Support](#).

ERRRECORDSETSELECTFAILED

[3723] Recordset select test failed.

Use the Test Recordset dialog box to test a data fetch of a recordset.

Note: Testing recordsets will only function while you are connected to a host.

ERRNOENTITIES

[3724] Error: The signature analyzer cannot be run since no entities have been defined.

To add an entity, traverse to a host screen and click New Entity on the entity window.

ERRCANTCREATELOG

[3725] Error: Can't create the log file.

Choose a different name for the log file.

ERRNORECORDSETTERMINATIONPROPS

[3726] You must select at least one recordset termination criterion.

Choose one or more recordset termination criteria.

ERRNORECORDSETRECORDDELIMITERSTRING

[3727] You must enter a string for recordset record.

Select a section of recordset text in the host screen and click the Update button.

ERRCANTADDRECORDSETFIELD

[3728] The current screen selection cannot be added as a recordset field. Please select a region within the recordset.

Select a section of the recordset in the host screen and click New Field.

ERRNOREADWRITEPROPS

[3729] You must select Read and/or Write.

Select the Read check box to make this field readable. Select the Write check box to make this field writable.

ERRNORECORDSETOPERATION

[3730] You must define an operation before enabling this option.

To add an operation to an entity, click New Operation on the Operation tab, choose an operation destination, and then define a command list.

ERRNOPATTERNPROPS

[3731] You must select at least one pattern property.

Choose one or more pattern properties, for example text.

ERRNOPATTERNTEXT

[3732] This pattern requires text for its definition. You must either type or select some text from the terminal screen, or uncheck the Text checkbox in the Definition tab.

Enter text that appears on that area of the host screen.

ERRPATTERNALREADYUSEDFOROPERATIONERROR

[3733] The current pattern is already used as an error pattern in operation [name]. To use it in entity signature, please remove the pattern as an error pattern from the operation first.

A pattern cannot be used for both entity signature and operation error. To remove a pattern from the operation pattern list, go to the Conditions dialog for the operation.

ERRNOATTRIBUTEERRORSTRING

[3734] You must enter an error string.

Enter text to be returned on the host screen if unable to locate a relative attribute during fetch.

ERRCANTCHANGEPROCEDURETYPE

[3735] This procedure's type cannot be changed because it is used in a compound procedure.

As long as a procedure is used in a compound procedure its type cannot be changed. If you want to change it you must first remove it from the compound procedure.

ERRCANTVALIDATEVARIABLES

[3736] The variables cannot be updated.

An error occurred while attempting to save the variables. Please document the sequence of events that led to this error, cancel the dialog box, save your work to a new file, and contact [Technical Support](#).

ERRRECORDSETSELECTFAILURE

[3737] The selection operation did not reach an expected destination.

The recordset selection test failed. The selection operation did not reach an expected destination. Click the Operation tab and use the Operation Edit dialog box to edit the selection operation.

ERRVARREADWRITEWARNING

[3738] A variable must be either read or write, or both read and write, or hidden.

Select the Read check box to allow the variable to be readable. Select the Write check box to allow an API developer to change the value of this variable after it's been deployed to the Host Integrator Server. Select the Hidden check box to make the variable inaccessible to a developer via the model file after it's been deployed to the Host Integrator Server.

ERRINVALIDMINLENGTH

[3739] Invalid minimum length.

Enter a different number.

ERRINVALIDMAXLENGTH

[3740] Invalid maximum length.

Enter a different number.

ERRMINLENGTHLARGER THANMAX

[3741] The minimum length cannot be larger than the maximum.

The minimum length must be smaller than the maximum. Enter a different number.

ERRINVALIDMINVALUE

[3742] Invalid minimum value.

Enter a different number.

ERRINVALIDMAXVALUE

[3743] *Invalid maximum value.*

Enter a different number.

ERRMINVALUELARGER THANMAX

[3744] *The minimum value cannot be larger than the maximum.*

The minimum value must be smaller than the maximum. Enter a different number.

ERRAUTOGENATTRIBUTES

[3745] *Are you sure you want to generate attributes automatically?*

Click Yes to generate attributes automatically, No to cancel.

ERRCONFIRMDISCARDOPERATION

[3746] *Are you sure you want to discard this operation?*

Click Yes to discard this operation, No to cancel.

ERRTABSTOPTESTSUCCEEDED

[3747] *Tabstop test succeeded.*

The list of tabstops matches the list of tab keystrokes that stop at each attribute on the selected entity.

ERRCONFIRMDELETEENTITY

[3748] *Are you sure you want to permanently delete the current entity?*

Click Yes to permanently delete the current entity, No to cancel.

ERRGOOFFLINE

[3749] *You are not currently connected to a host. Would you like to use offline mode?*

Click Yes to use offline mode, No to cancel.

ERRDISCONNECTNOW

[3750] *The current session must be disconnected now. Continue?*

Click Yes to disconnect from the current session, No to cancel.

ERRDUPLICATEENTITYNAME

[3751] *This entity name is already used. Do you want the Design Tool to generate a unique name?*

Click Yes to generate a unique name, No to cancel.

ERRCHANGEDEFAULTOPERATION

[3752] Operation [name] is currently being used for navigation commands to this destination. Do you want to use the current operation instead?

Click Yes to use the current operation when a dynamic traversal is requested and that traversal needs to jump from the current entity to an adjacent entity. Click No to cancel.

ERRAUTOGENPATTERNS

[3753] Are you sure you want to generate patterns automatically? Note: Some patterns may require removal later for better performance, or because they contain non-static data such as time and date.

Click Yes to generate patterns automatically, No to cancel.

ERRPATTERNMULTIPLELENGTHTEXT

[3754] A pattern with a multi-line, rectangular text property must have all lines equal. Press OK to have Host Integrator make each line a separate pattern. Press Cancel to return to the dialog.

Click OK to have Host Integrator make each line a separate pattern. Click Cancel to return to the dialog box and adjust the pattern text to make all lines equal.

ERRREPLACESCRIPT

[3755] Do you want to replace the existing command list?

Click Yes to replace the existing command list, No to cancel.

ERRPATTERNTEXTSIZEMISMATCH

[3756] The pattern text must be the same width and height as the pattern region.

Redefine the pattern so that the text and search region are the same size, or enable the "Search in extended region" option.

ERRENCRYPTVARIABLE

[3757] Unchecking the encrypted checkbox will erase encrypted values for this variable. Do you wish to continue?

Click Yes to erase encrypted values for this variable, No to cancel.

ERRVALIDATEDCONDITION

[3758] The condition was validated successfully.

Performed syntax checking of the expression. No errors found.

ERRPROCEDURESUCCEEDED

[3759] Procedure completed successfully.

The procedure ran to completion successfully.

ERRVALIDATEDPROCEDURE

[3760] [name]: Procedure validated successfully.

The procedure has no validation errors or warnings.

ERRCONFIRMCHANGEPROCEDURETYPE

[3761] Changing the procedure type will delete its implementation. Do you want to change the procedure type?

You cannot change a procedure's type without losing the entity and parameter mappings. Click Yes to change the type, No to cancel.

ERRCONFIRMDELETEPROCEDUREPARAMETER

[3762] This parameter is currently used in the procedure. Do you want to remove this parameter anyway?

The parameter is mapped to an attribute, field, or another parameter. Removing the parameter from the procedure will also remove these mappings. Click Yes to remove the parameter, No to cancel.

ERRCONFIRMCHANGETABLEHOMEENTITY

[3763] Changing the procedure's home entity may cause it to break. Additional editing may be needed in order to repair it. Do you want to change the procedure's home entity?

Changing a procedure's home entity may break the procedure until it is re-edited. If you change a procedure's home entity you should check that each procedure's path is still correct in the Procedure Editor. Click Yes to change the home entity, No to cancel.

ERRCANTMATCHPATTERNS

[3764] Warning: the defined patterns on this entity do not match the current screen. Do you still want to Apply changes?

Click Yes to apply changes, No to cancel. Go to offline mode, use the Signature Analyzer dialog box to identify pattern mismatches, and then edit or delete the bad patterns on the Pattern tab.

ERRRESETKEYBOARDMAPS

[3765] The new local keyboard is substantially different. Reset keyboard maps to defaults?

The new local keyboard is substantially different. Click Yes to reset keyboard maps to defaults, No to cancel.

ERRCONFIRMDELETESCRIPT

[3766] Are you sure you want to delete the entire command list?

Click Yes to delete the entire command list, No to cancel.

ERRRECORDSETFIELDOUTOFRANGE

[3767] Recordset field [name] either starts or is beyond the range of a record in recordset [name] on entity [name].

A recordset field must be within a record of the recordset. To add a field to a recordset, select an area within the recordset on the terminal screen that contains specific data and click the New Field button. By default, a name and the start and end offsets of the field are provided by the Design Tool.

ERRNOOBJECTFORARGUMENT

[3768] Command requires an object selection for argument [name], but none was made. If none are available, the command must be removed.

Use the Operation Edit dialog box to edit the invalid command. In the Operation Edit dialog box, select a command and press F1 or click Command Help for detailed online help for that command.

ERRBUILDCOMMANDFAILED

[3769] Host Integrator could not build this command for an unspecified reason. Please document the current command and argument values, cancel the dialog, save work to a new file, and contact Technical Support.

An error occurred while attempting to save a command. Please document the current command and argument values, cancel the dialog, save work to a new file, and contact [Technical Support](#).

ERRNORECORDSETFETCHTERMPROPS

[3770] You must select at least one filter option for recordset fetch termination.

Choose one or more filter options for recordset fetch termination.

ERRRECORDSETFIELDOUTOFRANGEEX

[3771] Recordset field [name] either starts or is beyond the range of a record in recordset [name] on entity %2. Note: Records on recordset column [name] have [n] characters only.

A recordset field must be within a record of the recordset. To add a field to a recordset, select an area within the recordset on the terminal screen that contains specific data and click the New Field button. By default, a name and the start and end offsets of the field are provided by the Design Tool.

ERRCONFIRMUPDATEINITIALCURSORPOS

[3772] Are you sure you want to change the initial cursor position to the current cursor position?

Click Yes to update the initial cursor position, No to cancel. Note: If you have configured any entity properties relative to a cursor position, these property coordinates will be automatically updated. Click Apply to save your changes or click Cancel to revert your initial cursor position back to its original state.

ERRNOERRORATTRIBUTES

[3773] You must select at least one unused attribute.

To move an attribute from the Unused box to the Return box, use the right arrow button.

ERRCANTUPDATESNAPSHOT

[3774] The snapshot cannot be updated because the host screen has changed. Please update it later.

To update the snapshot for an entity, connect to the host, navigate to that entity, and click the Update button on the General tab of the Advanced Entity Properties dialog box.

ERRCANTUSEENDOFSCREENALWAYSDELIMITER

[3775] 'End of screen is always a delimiter' cannot be used because the recordset is set to overlap records across screens.

To use the 'End of screen is always a delimiter' option, you must go to the Recordset Options dialog and deselect the 'Host overlaps records across screens' option.

ERRCANTUSEOVERLAPRECORDSACROSSSCREENS

[3776] 'Host overlaps records across screens' cannot be used because the recordset is set to use end of screen as a delimiter.

To use the 'Host overlaps records across screens' option, you must go to the Recordset/Layout tab and uncheck the 'End of screen is always a delimiter' box.

ERRTEMPLATENOOBJECTFORARGUMENT

[3777] Command [name] requires an object selection for argument [name], but none are available. The command was removed.

Use the Operation Edit dialog box to edit the command list. In the Operation Edit dialog box, select a command and press F1 or click Command Help for detailed online help for that command.

ERRTEMPLATEBUILDCOMMANDFAILED

[3778] Host Integrator could not build command [name] for an unspecified reason. The command was removed.

An error occurred while attempting to save a command. Please document the current command and argument values, cancel the dialog, save work to a new file, and contact [Technical Support](#).

ERRTEMPLATEOPERATIONCOMMANDERROR

[3779] Command [name], [argument] could not be parsed. The command was removed.

Use the Operation Edit dialog box to edit the command list. In the Operation Edit dialog box, select a command and press F1 or click Command Help for detailed online help for that command.

ERRTEMPLATEOPERATIONCREATEERROR

[3780] Some problems were encountered while creating a command list using the selected template.

Use the Operation Edit dialog box to edit the command list. In the Operation Edit dialog box, select a command and press F1 or click Command Help for detailed online help for that command.

ERRVALIDATEEVENT

[3781] The event was validated successfully.

Performed syntax checking of the expression. No errors found.

ERRINVALIDOPERATIONCOMMAND

[3782] An operation command could not be parsed.

The model loaded into the Design Tool contained an invalid command in an operation. Please document the text that could not be parsed, and contact [Technical Support](#).

ERRINVALIDROWCOL

[3783] Non-relative absolute rows and columns must be positive integers.

If the specified location is not relative to the cursor position, row and column parameters must be from 1 to the current screen height and width of the terminal display.

ERRLOGOUTSCRIPTNOTFROMHOMEENTITY

[3784] Execution of the logout command list will always start at the home entity. Since recording started at a different entity, we will add a 'Navigate to [name] entity' command to the start of your command list. At run-time, this may cause unnecessary host navigation. Do you want to continue?

Click Yes to add a 'Navigate to the starting entity' command to the start of the logout command list, No to cancel.

ERRENTITYCHANGEDCANTGENERATETABSTOP

[3785] Failed to generate the next tabstop. The terminal screen has changed and does not match the current entity.

The terminal screen has changed and does not match the current entity. Check the pattern definitions. Temporarily clear the Wait for cursor check box on the Validation tab of the Advanced Entity Properties dialog box. If you do not clear this check box, the screen snapshot will fail because the entity is set to match attributes according to cursor positioning and generating a tabstop changes the cursor position.

ERRLOGOUTSCRIPTNOTFROMANYENTITY

[3786] Logout command list must start at the home entity. If you experience any problems, try recording another command list from the home entity.

Logout command list must start at the home entity. If you experience any problems, press the Record button to record another command list from the home entity.

ERRNOSIGNATUREPATTERNS

[3787] This entity requires at least one 'Use in entity signature' pattern so Host Integrator can recognize it.

To add a pattern to an entity, use the cursor to select a static section of the host screen and click the New Pattern button on the Pattern tab.

ERRNOPATTERNPROPVALUE

[3788] You must select a pattern property value.

Choose one from the pattern property combo box.

ERREMPYCONDITION

[3789] The Event Expression or Condition cannot be an empty string.

The Event Expression or Condition cannot be an empty string.

ERRPATTERNABANDONED

[3790] Line [number] has no text, and this pattern requires text for its definition. This line is being discarded from the multi-line pattern.

This line is discarded because it has no text.

ERRCONFIRMOVERWRITELIBRARY

[3791] Are you sure you want to overwrite existing library [name]?

Click Yes to overwrite existing library, No to cancel.

ERRPATTERNTEXTTOOWIDE

[3792] The pattern text is wider than the pattern region. Redefine the pattern by either decreasing the width of the pattern text or increasing the width of the pattern region.

Each line of the pattern may not be wider than the pattern region. Redefine the pattern by either decreasing the width of the pattern text or increasing the width of the pattern region.

ERRUPDATEINITIALCURSORPOSBEFORESAVE

[3793] The current cursor position for this entity has changed from the initial cursor position. Patterns that are relative to the cursor may need to be updated. Would you like to change the initial cursor position to the current cursor position?

Click Yes to update the initial cursor position, No to cancel. Note: If you have configured any entity properties relative to a cursor position, these property coordinates will be automatically updated.

ERRPATTERNSMATCHOTHERENTITIES

[3794] Warning: the defined patterns on this entity also match the snapshot(s) for %0. Do you still want to Apply changes?

Click Yes to apply changes, No to cancel. More patterns should be added to make the screen signature unique.

ERRINVALIDTAGNAME

[3795] Invalid character [character] entered in new tag field.

The specified character is not allowed in tags.

ERRTAGSMODIFIED

[3796] The modified tags have not been saved. Click Apply before selecting new tags.

The modified tags have not been saved. Click Apply before selecting new tags.

ERRALLCAPSDIRECTORY

[3800] To preserve case between files and folders, files may not be created in all caps under Windows NT.

In Windows NT, directories created with all capital letters are translated to mixed case. This can result in a mismatch between model folder names and their corresponding model file names. If you transfer model files to a Host Integrator Server running on a Linux or Windows system, this situation could result in failure to find models.

ERRINVALIDCHARINMODELNAMEFIELD

[3801] Invalid character [characters] entered in model name field.

The specified character is not allowed in model names.

CANNOTOPENCONFIGURATIONFILE

[3802] Error opening configuration file [file]

Error opening the file containing the Java configuration values.

CONFIGURATIONVALUEMISSING

[3803] Configuration value [value] is missing from configuration file [file], or has an invalid value

A value is missing from the Java configuration file, or has an invalid value.

DIRECTORYORFILEDOESNOTEXIST

[3804] [Directory_or_file] [name] does not exist

The specified directory or file does not exist.

WEFOUNDAREGSERVERANDAREDONE

[3805] We found a /REGSERVER or /UNREGSERVER and we're al done.

We found a /REGSERVER or /UNREGSERVER and we're al done.

ERRNOCLOSEWHILEEXECUTING

[3806] The Test Recordset dialog cannot be closed because a recordset operation is executing.

In the Design Tool, you cannot close the Test Recordset dialog box while an operation is still executing. When you need to stop the operation, click Break.

ERRSETCURRENTRECORDFAILED

[3807] Unable to set the current record index.

Host Integrator was unable to set the requested index. The most likely reason is that the index is greater than the total number of records. This message will also be displayed when Host Integrator has no means to reach the index. For example, if the recordset has scrolled past the requested index and there is no PageUp or Home operation defined, Host Integrator cannot fulfill this request and will return this error.

ERRENDOFRECORDSEMPYFETCH

[3808] End of recordset reached - no records were fetched.

No records were returned because the end of the recordset has been reached.

ERRHENAVERAGE

[3820] Obsolete

ERRHETRACEFILEMISSING

[3821] Obsolete

ERRHETRACEMISMATCH

[3822] Obsolete

ERRVARIABLEREAD

[3830] Cannot read from variable [name].

This error usually results from a variable being marked as write-only. This may have been done to protect passwords or other sensitive data.

ERRVARIABLEWRITE

[3832] Cannot write to variable [name].

The variable is probably marked as read-only.

ERRVARIABLEVALUENOTWELLFORMED

[3833] Cannot write to variable [name] because the variable value is not compatible with the setting it is mapped to.

Some Host Integrator system settings, such as cryptographic certificates and keys, require that the data values assigned have a specific format.

ERRVARIABLEWRITEDUETOSETTING

[3834] Cannot write to variable [name] because the setting it is mapped to is currently disabled.

Some Host Integrator system settings, such as UserID and Password, Device Name, and Port, are disabled after the session is connected to the host. This reflects the fact that the state at host connect is the only one of significance for these settings. If a variable is mapped to one of these system settings, writes to that variable will also be disabled.

ERRVARVALUENOTINITIALIZED

[3836] The value of variable [name] is uninitialized.

During model execution Host Integrator was asked to write data from a variable to the terminal, but the specified variable has never been initialized with a valid value. This most often happens in the context of an operation.

ERRVARVALUENOTINITIALIZEDATCONNECT

[3838] The value of variable [name] must be initialized at connect.

If a variable is marked as needing initialization at host connect, the variable must be provided with a value either from the connector in the connect() method or from a model variable list created in the console. Host Integrator detected a failure to initialize such a variable.

ERRMODELVARIABLELISTDUPLICATE

[3842] Unique variable [name] has a duplicate value [name] in row [n] and row [n].

Change the value of the variable to a unique value, or turn off the Unique checkbox for this variable.

ERRMODELMULTIPLEVARIABLELISTDUPLICATE

[3843] Some of the unique variables have duplicate values, see details.

Change the value of the variable to a unique value, or turn off the Unique checkbox for this variable.

ERRMODELVARIABLELISTBADVARIABLE

[3844] Variable [name] referenced in the deployment options no longer exists in current model, and will be removed from the variables tab of the deployment options dialog.

A variable in the previously saved Model Variable List does not exist in the current model. The variable and its values have been discarded. If you wish to retain the entries for this model variable, you may cancel from the deployment options dialog after dismissing this error message, then redefine the model variable.

ERRDEPLOYBADHOMEENTITY

[3847] Entity [name] no longer exists in current model, defaulting to model home entity.

The home entity designated for the session pool which was previously saved in the deployment file no longer exists. The model home entity is being used as default. When you dismiss this error you can press Cancel to make no changes to your Deployment Package data, Clear to start fresh with no Deployment Package data, or edit and save your Deployment Package data.

ERRDEPLOYBADHOMEENTITYDETAILS1

[3848] The entity [name] which was designated as the home entity for the session pool in a previously saved in the deployment file no longer exists. The model home entity is being used as default.

The home entity designated for the session pool which was previously saved in the deployment file no longer exists. The model home entity is being used as default. When you dismiss this error you can press Cancel to make no changes to your Deployment Package data, Clear to start fresh with no Deployment Package data, or edit and save your Deployment Package data.

ERRDEPLOYBADHOMEENTITYDETAILS2

[3849] When you dismiss this error you can press Cancel to make no changes to your Deployment Package data, Clear to start fresh with no Deployment Package data, or edit and save your Deployment Package data.

The home entity designated for the session pool which was previously saved in the deployment file no longer exists. The model home entity is being used as default. When you dismiss this error you can press Cancel to make no changes to your Deployment Package data, Clear to start fresh with no Deployment Package data, or edit and save your Deployment Package data.

ERRMODEL CANNOT BE DEPLOYED

[3850] The model file was saved but it cannot be deployed to a Host Integrator server due to one or more validation errors. Please see the validator for details.

For help with the validator, [see VHI help topic about details](#).

ERRMODEL CANNOT BE ACTIVATED LOCALLY

[3851] The current model could not be deployed to the Verastream Host Integrator server [name].

Make sure the Host Integrator server is running and that the model has been successfully validated.

ERRDEPLOYMENT FAILED

[3852] Deployment of model failed: [message text]

One or more error messages have been returned by the deployment tool, as the result of an attempt to deploy a model to the session server.

ERRMODELMULTIPLEVARIABLELISTDUPLICATE10

[3853] Some of the unique variables have duplicate values. Stopped after 10 errors were found. See details.

Change the value of the variable to a unique value, or turn off the Unique checkbox for this variable.

ERRPLACEMENTOUTOFBOUNDS

[3855] The [name] object on entity [name] could not be located.

An object's location is calculated to be off the terminal screen.

ERRDEPLOYMENTFAILEDUNREGISTERED

[3857] Deployment of model failed: The server [server] must be registered with a management server before the model can be deployed.

The model could not be deployed to the session server. The session server must be registered with the management server, which normally is accomplished by the setup.exe installation program. To manually register your session server with the management server, run Administrative Console, connect to the management server (as administrator), and add the session server.

ERRRECORDSETOPERATIONFAILED

[3860] The [name] operation ([name]) for the recordset [name] on entity [name] failed.

A recordset operation failed.

ERRSYNCHOSTPAGETORECORDSETFAILED

[3861] Synchronization of the current host page with the current page in recordset [name] on entity [name] failed.

Host Integrator tries to keep the current record in the recordset on the screen at all times using the Page Up and Page Down operations. This error indicates that Host Integrator failed to do so because either a recordset operation failed or a recordset operation is not defined.

ERRRECORDSETOPERATIONNOTSUPPORTED

[3862] The [name] operation ([name]) for the recordset [name] on entity [name] is not supported.

An internal error has occurred. Contact [Technical Support](#).

ERRSYNCHOSTRECORDTORECORDSETFAILED

[3863] Synchronization of the current host record with the current record in recordset [name] on entity [name] failed.

Using the Line Up and Line Down operations, Host Integrator tried to set the host's current record to match the current record in the recordset. This error indicates that Host Integrator failed to do so because either a recordset operation failed or a recordset operation is not defined.

ERRRECORDSETOPERATIONNOTDEFINED

[3864] The [name] operation for the recordset [name] on entity [name] is not defined.

In order to perform a specific recordset operation, it must be defined under Recordset Operations in the Design Tool.

ERRTIMEOUTWAITINGFORHOSTRECORD

[3865] Timeout waiting for record [n] on the screen to be selected by the host. Instead record [n] is selected by the host.

Host Integrator executed either the Line Up or Line Down operation expecting the host to change its current record, but the host's current record either did not change or did not change to the expected record.

ERRRECORDSETOPERATIONNOTALLOWED

[3866] The [name] operation ([name]) for the recordset [name] on entity [name] is not allowed.

This recordset operation cannot be performed in the current context. Typically this means a Page Up operation was attempted on the first page or a Page Down operation was attempted on the last page.

ERRDYNAMICHOSTRECORDFAILED

[3867] Unable to determine the host's current record in recordset [name] on entity [name].

Host Integrator is unable to determine the host's current record using the model specified current record preferences.

ERRRECORDSETINDEXOUTOFRANGE

[3868] The Recordset index ([n]) is beyond the end of the recordset [name] on entity [name].

The Recordset index cannot be set beyond the end of the recordset.

ERRDYNAMICHOSTRECORDNOTVALID

[3869] The host's current record is not a valid record in recordset [name] on entity [name].

The current host record has been filtered out of the fetch. Thus the current record index for the recordset cannot be set at this record.

ERRSELECTRECORDNOTSUPPORTED

[3870] SelectRecord is not supported by the recordset [name].

A recordset can support navigation by record selection. However, the model designer must map an operation on the entity to the Selection operation property of the recordset. An attempt to execute the selection operation without such a mapping will result in this error.

ERRINVALIDRECORDSETINDEX

[3871] Invalid recordset index.

The recordset index is invalid.

ERRINSERTRECORDNOTSUPPORTED

[3876] InsertRecord is not supported by the recordset [name].

Direct insertion of new records into a recordset is a feature that can be enabled and specified by the model designer as appropriate. If that feature is not enabled, a request to perform an insertion will be rejected with this error.

ERRUPDATERECORDNOTSUPPORTED

[3877] UpdateRecord is not supported by the recordset [name].

Direct updates of records in a recordset is a feature that can be enabled and specified by the model designer as appropriate. If that feature is not enabled, a request to perform an update will be rejected with this error.

ERRSELECTRECORDFAILED

[3878] SelectRecord on the recordset [name] failed.

The operation mapped as the Selection operation on the specified recordset failed. There should be information regarding the nature of the failure in additional error messages generated as part of the exception.

ERRNORECORDMATCHINGCONDITION

[3880] No record was found matching the condition: [expression].

When searching for a record by filter, if no matching records were found between the starting point and the end of the recordset, Verastream Host Integrator will return this error.

ERRINSERTRECORDFAILED

[3882] InsertRecord on the recordset [name] failed.

This message error should be accompanied by additional error messages providing more details on the exact nature of the problem encountered.

ERRUPDATERECORDFAILED

[3883] UpdateRecord on the recordset [name] failed.

This message error should be accompanied by additional error messages providing more details on the exact nature of the problem encountered.

ERRBLANKINSERTPOINTNOTFOUND

[3884] Unable to insert record because no blank records were found.

One property of the direct record insertion feature is where to insert the new record. One option is to find the first blank record. If that option is selected and Verastream Host Integrator is unable to locate a blank record, this error will result.

ERRCONDITIONINSERTPOINTNOTFOUND

[3886] Unable to insert record because no records matching the insert condition were found: [expression].

One property of the direct record insertion feature is where to insert the new record. One option for the model designer is to specify a condition that matches the insertion record. If this option is selected and Verastream Host Integrator cannot find a matching record, this error will result.

ERRRECORDSETEMPTY

[3888] The [name] Recordset in the [name] entity is empty.

The Recordset is empty.

ERRRSFIELDNOTONCURRENTENTITY

[3889] The recordset field [entity].[recordset].[field] is not on the current entity; [name].

The requested action assumes the context of the current entity. However, no recordset field by this name was found. Be sure the name is spelled correctly and with correct letter case.

ERRRSFIELDBEFOREWRITEOPERATIONFAILED

[3890] The before write operation [operation] failed in the [entity].[recordset].[field] recordset field.

A recordset field before write operation failed.

ERRRSFIELDAUTOTABOPERATIONFAILED

[3891] The auto-tab operation [operation] failed in the [entity].[recordset].[field] recordset field.

A recordset field auto-tab operation failed.

ERRRSFIELDAFTERWRITEOPERATIONFAILED

[3892] The after write operation [operation] failed in the [entity].[recordset].[field] recordset field.

A recordset field after write operation failed.

ERRRSFIELDNOTONCURRENTRECORDSET

[3893] The recordset field [entity].[recordset].[field] is not on the current recordset; [entity].[recordset].

The requested action assumes the context of the current entity. However, no recordset field by this name was found. Be sure the name is spelled correctly and with correct letter case.

ERRRSNOTCURRENTRECORDSETONENTITY

[3894] The recordset [recordset] is not the current recordset on entity [entity].

The requested action assumes that the named recordset is the current recordset on the given entity.

ERRRECORDSETMAXPAGEDOWN

[3895] The maximum number of page down operations ([n]) for recordset [name] has been reached.

The maximum number of page down operations for this recordset has been reached.

ERRSESSIONNOTBOUND

[3900] Session not bound to the Log Manager.

A Host Integrator session attempted to send an event to the Log Manager without first initializing its connection to this subsystem.

ERRACCESSDENIED

[3901] Insufficient privileges to perform the requested operation.

You have not been granted permission to perform the requested operation.

ERRINVALIDPARAM

[3902] Invalid [name] command, parameter: [name].

A request sent to the Host Integrator Log Manager was incorrectly formed. Contact [Technical Support](#).

ERRLOGGERNOTFOUND

[3904] The [logger type] logger is not currently active.

A Host Integrator process attempted to bind to a server's logger that is not currently active.

ERRCORRUPTSTORE

[3906] Log message store [name] is corrupt.

The currently configured logging directory contains corrupted message files. Try restarting the Log Manager or specifying a new directory for log messages.

ERRACTIVATESTORE

[3908] Cannot activate data store [name].

The Host Integrator Log Manager is unable to use the currently configured directory to store log messages. Try specifying a new directory for log messages.

ERRACCESSSTORE

[3910] Can't access data store [name].

The Host Integrator Log Manager cannot access the files in the currently configured logging directory. Check the permissions on the directory and the files it contains.

ERRSMTPCONNECT

[3912] Cannot connect to SMTP server [hostname].

The Host Integrator Log Manager is unable to establish a connection with the configured email server. Verify that your email notification settings are correct, the hostname can be resolved, and the email server is running on the specified port.

ERRSMTPREFUSED

[3914] SMTP server [hostname] refused our email message.

The currently configured email server refused to accept an email message from the Host Integrator Log Manager. Verify that the configured email server allows email messages to originate from this server's address.

ERRBADPSINTOLOGMANAGER

[3916] Unknown command received by the Host Integrator Log Manager.

A request was made to the Host Integrator Log Manager, but it does not recognize the request.

ERRLOGGERNOTINITIALIZED

[3918] The logger has not been initialized.

Contact [Technical Support](#).

ERRCANTCONNECTTOLOCALLOGMANAGER

[3920] Cannot connect to the Host Integrator Log Manager. Error code = [n].

The Host Integrator Server cannot connect to the Log Manager. Ensure the Host Integrator Log Manager is running on this host. The server will periodically retry connection attempts. If this error continues to occur, contact [Technical Support](#).

ERRCANTCONNECTTOLOGMANAGER

[3922] Cannot connect to the Host Integrator Log Manager.

The Host Integrator log viewer cannot establish a connection to the requested Host Integrator Log Manager. Verify that the requested Log Manager is running.

ERRINTERNALLOGVIEWER

[3924] Error within the Host Integrator Log Viewer.

Contact [Technical Support](#).

ERRINVALIDLOGGINGDIR

[3926] Invalid logging directory: [path].

The specified directory cannot be used by the Host Integrator Log Manager. Verify that the directory exists on the server, and that the Log Manager has read and write permissions to access it.

ERRLOGGERFAILEDCONSTRUCTION

[3928] A failure occurred during the construction of the logger.

Contact [Technical Support](#).

ERRNOLOGSAVAILABLETOVIEW

[3930] There are no messages available for viewing.

The Log Manager does not currently contain any entries to view for the selected server.

ERRINVALIDMSGSTORETYPE

[3931] Invalid message store type: [type].

The type of message store requested is not known by the log viewer and cannot be opened on the server selected.

ERRCURSORNAMEINUSE

[3932] Cursor [name] is already in use on this connection.

The cursor name specified during the creation of a new cursor is already in use on this connection. Try a different name or close the previous cursor.

ERRQUERYDIALECTUNKNOWN

[3933] Cannot recognize query dialect.

The Host Integrator Log Manager does not recognize the query dialect used by this log viewer. Verify that you are using compatible versions.

ERRQUERYSYNTAX

[3934] Query syntax error at position [position].

A syntax error has been detected in the query string at the indicated position.

ERRUNKNOWNCOLUMN

[3935] Column [name] does not exist in table [name].

The Host Integrator Log Manager does not maintain a column with the indicated name.

ERRCURSORNOTOPEN

[3936] Cursor [name] is not open.

The cursor specified is not currently open.

ERRTYPEMISMATCH

[3937] Type mismatch: [description].

An attempt was made to perform an operation using two different SQL data types.

ERRUNKNOWNTABLE

[3938] Table [name] does not exist.

The Host Integrator Log Manager does not maintain a table with the indicated name.

ERRMSGROUTERNOTFOUND

[3939] The requested message store is not present on this server.

A client attempted to access a message store that is not present on the selected server.

ERRCOMMANDFAILED

[3950] Command failed: [command].

The specified command failed. There is usually some additional information the form of prior error messages or in the descriptive text to explain the failure. If not, consider the state of the terminal screen and the context of the command when trying to determine a cause.

ERRINVALIDOBJECTID

[3952] Internal error - invalid object identifier [id number] found.

Save the work to a new file and contact [Technical Support](#). If the model file has been edited outside the Design Tool, it may have been corrupted.

ERRINVALIDOBJECTIDTYPE

[3954] Internal error - invalid object identifier [id number] found. The object identifier is not of the expected type [type].

Save the work to a new file and contact [Technical Support](#). If the model file was edited outside the Design Tool, it may have been corrupted.

ERRCOMMANDLISTFAILED

[3955] Command list failed at index [n]: [command].

The specified command failed at the specified index of the command list. There should be some additional information the form of subsequent error messages to provide the context of the command list failure. There may also be a prior message indicating a more specific reason for failure.

ERRCOMMANDLISTCANCELED

[3956] Command list cancelled at index [n]: [command].

When a script or operation is cancelled while a command is executing, Host Integrator will report the current command for informational purposes. If the system seems stalled during a command sequence, it is usually because the executing command is waiting for something the host is never going to do.

ERRNOCMDLISTSDURINGCANCEL

[3958] Command list aborted because system cancel is on.

Once you have clicked cancel, you cannot queue any additional operations or scripts until the cancelled command set has been fully stopped. Wait for the prior operation or script to finish.

ERRATTRSREADINVALATTRID

[3960] While reading attributes, an invalid attribute id [id number] was encountered.

All objects in Verastream Host Integrator, including attributes, are referenced internally by a numeric identifier. If an attempt is made to access an attribute on the current entity with an invalid identifier, this error will be returned. If all requested attributes exist on the current entity, then the next most likely problem is a corrupt model. You will probably need to contact Technical Support for assistance.

ERRATTRSWRITEINVALATTRID

[3962] While writing attributes, an invalid attribute id [id number] was encountered.

All objects in Verastream Host Integrator, including attributes, are referenced internally by a numeric identifier. If an attempt is made to access an attribute on the current entity with an invalid identifier, this error will be returned. If all requested attributes exist on the current entity, then the next most likely problem is a corrupt model. You will probably need to contact Technical Support for assistance.

ERRATTRSWRITEDIFFPARENT

[3964] Attempt to write two attributes [name] and [name] that have different parents.

Each request to modify attributes must contain attributes from a single entity. Any request that contains attributes from more than one entity will cause this error to be generated.

ERRATTRSWRITETOOLONG

[3966] Attempted to write the value [value] to attribute [name] which has a maximum length of [n].

Each attribute has a maximum length governed by the model and based on the properties of the terminal screen. This error indicates that the value string is too long for the attribute.

ERRATTRSWRITEMOVECURSORFAILED

[3968] Unable to write to attribute [name] because the cursor could not be moved to row [n], col [n].

Verastream Host Integrator must move the cursor to the start of an attribute in order to write data into it. If the attempt to move the cursor into position fails, then Verastream Host Integrator will return this error. Possible problems include the model may need to have cursor movement commands redefined, or the entity requiring additional tabstop definitions, or the terminal state preventing the cursor from being moved.

ERRATTRWRITELOCATIONPROTECTED

[3970] The location of the attribute or recordset field [name] is protected.

In IBM blockmode terminals and HP format mode screens, the user is only allowed to modify certain areas of the screen. If an attribute or recordset field is marked as writable but corresponds to a read-only (protected) area of the screen, Verastream Host Integrator will disallow access and return this error.

ERRATTRWRITELOCATIONNUMERIC

[3972] The location of the attribute [name] is numeric only.

Some terminal screens can define fields to be numeric only. This error indicates an attempt to Write non-numeric text to one these fields.

ERRATTRIBUTENOTONCURRENTENTITY

[3974] The attribute [entity].[attribute] is not on the current entity; [name].

The requested action assumes the context of the current entity. However, no attribute by this name was found. Be sure the name is spelled correctly and with correct letter case.

ERRATTRIBUTEISREADONLY

[3976] The attribute [name] is read only.

The model designer has marked this attribute as read only. No updates from operations or from a connector are permitted.

ERRFIELDISREADONLY

[3977] The field [name] in recordset [name] is read only.

The model designer has marked this field as read only. No updates from operations or from a connector are permitted.

ERRATTRIBUTEISWRITEONLY

[3978] The attribute [name] is write only.

The model designer has marked this attribute as write only. No access from operations or from a connector are permitted. This is often done to prevent access to data such as passwords.

ERRFIELDISWRITEONLY

[3979] The field [name] in recordset [name] is write only.

The model designer has marked this field as write only. No access from operations or from a connector are permitted. This is often done to prevent access to data such as passwords.

ERRATTRSWRITENOTATTABSTOP

[3980] Unable to write attribute [name] because it is not at a tab stop.

If the default cursor movement has been overridden and tabstops defined on the entity, each writable attribute must correspond to a tabstop. Otherwise, Verastream Host Integrator will be unable to position the cursor correctly and the write will fail. This error occurs when an attempt is made to write to an attribute that does not have a tabstop. The model designer should either add the appropriate tabstop or mark the attribute as read-only.

ERRATTRBEFOREWRITEOPERATIONFAILED

[3981] The before write operation [operation] failed in the [entity].[attribute] attribute.

An attribute before write operation failed.

ERRATTRAUTOTABOPERATIONFAILED

[3982] The auto-tab operation [operation] failed in the [entity].[attribute] attribute.

An attribute auto-tab operation failed.

ERRATTRAFTERWRITEOPERATIONFAILED

[3983] The after write operation [operation] failed in the [entity].[attribute] attribute.

An attribute after write operation failed.

ERRDELAYEDINPUTDISALLOWED

[3984] Delay attribute inputs are not permitted on entity [name] due to its configuration.

This error indicates that a client or script tried to set delayed inputs on an entity which has the immediate arrival processing flag turned on. Use of this entity property is not compatible with delayed inputs. If dynamic data input is needed on arrival, try setting variable values and writing them to the terminal in an entity arrival event handler. In that event handler, be sure that the terminal prepared for input by using any needed wait commands (such as `WaitForKeyboardUnlocked` or `WaitForDisplayScreen`) in the arrival event) or the host session may stall or report errors due to illegal input.

ERRNOENTITY

[3989] The request cannot be completed because there is no current entity.

A request was made on the Verastream model which requires the current terminal screen to be defined as an entity. If a client application receives this error, define and map an unrecognizedScreen event and save the contents of the terminal screen for analysis. If an event handler receives this error, capture the contents of the terminal screen in the exception processing code.

ERROBJLOCNOCURRENTENTITY

[3990] No current Entity exists for the GETOBJECTLOCATIONS_ command.

The GETOBJECTLOCATIONS_ command can only be successfully processed when a current Entity exists.

ERROBJLOCNOCURRENTRECORDSET

[3992] No current RecordSet exists for the GETOBJECTLOCATIONS_ command.

The GETOBJECTLOCATIONS_ command can only be successfully processed when a current Entity exists.

ERROBJLOCCANTCREATEPROPERTY SHEET

[3994] Can't create the OBJECTLOCATION_ Property Sheet for the GETOBJECTLOCATIONS_ command.

The OBJECTLOCATION_ Property Sheet is a vital element of the GETOBJECTLOCATIONS_ command. Without it, there is not reason for being.

ERROBJLOCINVALIDOBJECTTYPE

[3998] The object type [type] parameter passed to the GETOBJECTLOCATIONS_ command is invalid.

The GETOBJECTLOCATIONS_ command can only be successfully processed on the following objects: Attributes, RecordSet Fields, Patterns, and RecordSets.

ERRSCROLLLOCKSTOPPINGTERMINALREAD

[4000] Scroll Lock is stopping terminal display - turn off Scroll Lock to continue.

The Scroll Lock keyboard mode is used to stop terminal input under some conditions. This error message indicates that there is text on hold until Scroll Lock is turned off.

ERRKEYBOARDLOCKTIMEOUT

[4001] Timed out waiting for host keyboard to unlock.

The host keyboard was not reset properly or the host system is slow or unresponsive.

ERRDSHOSTNAMEANDIPADDRESSMISMATCH

[4040] [name] name resolution addresses [name] do not match wcp peer address [name].

The server name does not resolve to the ip address.

ERRDSDIRECTORYWRITINGEXCEPTION

[4041] Exception [name] encountered writing to directory at [name].

Check file system permissions. If the problem persists, contact [Technical Support](#).

ERRDSNAMINGEXCEPTION

[4042] Naming exception [name] writing to directory at [name].

Check file system permissions. If the problem persists, contact [Technical Support](#).

ERRAASERVEREXCEPTION

[4043] AAServer exception [name] at [name].

Contact [Technical Support](#).

ERRDSCONFIGLOCKNOTAUTHORIZED

[4044] User has not been authorized to lock configuration: [name].

User has not been authorized to lock configuration.

ERRDSCONFIGLOCKALREADYLOCKED

[4045] Configuration is already locked: [name].

Deprecated in version 7.0. Another user is logged on to the Administrative WebStation in configure mode. Switch the other user to view mode and try again.

ERRDSCONFIGLOCKUNABLETOUNLOCK

[4046] Unable to unlock configuration: [name].

Unable to unlock configuration.

ERRSDNSFAILURE

[4047] DirServer: DNS failure looking up [name].

Unable to find the server name in DNS.

ERRDSPEERUNRECOGNIZED

[4048] DirServer: unrecognized peer [name] in context [name].

The peer AADS initiating contact has not been registered with this AADS.

ERRDSREJECTINGPEERCONFIGURATION

[4049] DirServer: rejecting peer configuration: [name].

DirService rejecting configuration update from peer DirService.

ERRDSCANTAUTHENTICATEPEER

[4050] DirServer: unable to authenticate peer [name] in context [name].

DirService unable to authenticate peer AADS.

ERRDSCANTCONNECTTOPEER

[4051] DirServer: [name].

DirService unable to connect to or authenticate to peer AADS.

ERRDSCANTUNREGISTERPRIMARYAADS

[4052] DirServer: attempt to unregister primary AADS reported from [name].

DirService unable to unregister primary AADS.

ERRDSCANTSTOREPROPERTIES

[4053] DirServer: failure to store properties reported from [name].

DirService unable to store properties file.

ERRDSCANTLOADPROPERTIES

[4054] DirServer: failure to load properties reported from [name].

DirService unable to load properties file.

ERRDSLAPPARAMETERERROR

[4055] DirServer: error in ldap parameters [name].

DirService detects error in ldap parameters.

ERRDSUNABLETOGETGROUPLIST

[4056] DirServer: unable to get group list from [name].

DirService unable to obtain group information.

ERRDSUNABLETOAUTHENTICATELDAPADMINISTRATOR

[4057] DirServer: unable to authenticate Ldap Administrator [name] to Ldap directory.

DirService: incorrect userid or password for Ldap administrator.

ERRDSUNABLETOGETINITIALCONFIGURATION

[4058] DirServer: unable to contact management server for initial configuration.

DirService: unable to contact peer management server for initial configuration.

ERRDSUNABLETOREVERTCONFIGURATION

[4059] DirServer: unable to restore configuration to prior state.

DirService: unable to restore configuration to prior state.

ERRDSUNABLETOSWITCHAUTHENTICATIONMODE

[4060] DirServer: switching authentication modes requires turning off product security and clearing security profile groups.

DirService: switching authentication modes requires turning off product security and clearing security profile groups.

ERRDSUNABLETOSAVECONFIGURATION

[4061] DirServer: configuration changes rejected by management server: [name].

DirService: Configuration changes rejected by management server. Check log messages for rejection reasons.

ERRDSUNABLETOPROCESSCERTIFICATE

[4062] DirServer: unable to process X509 certificate: [name].

DirService: unable to process X509 certificate.

ERRDSRELEASINGCONFIGLOCK

[4063] DirServer releasing configuration lock: [name].

DirService releasing configuration lock.

ERRDSCONFIGURATIONNOTLOCKED

[4064] DirServer: configuration not locked.

DirService: configuration not locked.

ERREVENTEXPRSYNTAX

[4070] [Error] - Event expression syntax error ([offset], [length]).

There is a syntax error in the event expression.

ERREVENTEXPRUNEXPECTEDCHAR

[4072] [Error] - Unexpected character(s) in event expression ([offset], [length]).

There is a syntax error in the event expression.

ERREVENTEXPRUNEXPECTEDEND

[4074] Unexpected end of event expression statement.

There is a syntax error in the event expression.

ERREVENTEXPREVENTNAME

[4076] [name] - Unrecognized event name ([offset], [length]).

The specified event does not exist on the current entity or in the global events.

ERREVENTEXPREVENTID

[4078] [id number] - Unrecognized event ID ([offset], [length]).

Your model file is corrupted. Contact [Technical Support](#).

ERREVENTCOULDNOTBECREATED

[4080] The event [name] could not be created.

One of the events specified as part of a WaitForMultipleEvents command could not be created.

ERREVENTCOULDNOTBEEVALUATED

[4082] An error occurred while evaluating the event [name].

One of the events specified as part of a WaitForMultipleEvents command had an error occur during its evaluation.

ERREVENTINVALIDSCREENREGION

[4090] The region specified as part of the [name] event is not a valid screen region.

The region on the screen specified as part of a wait event is not a valid region.

ERREVENTINVALIDCURRENTENTITY

[4092] The current entity is invalid in the definition of the [name] event.

The current entity used in the definition of a wait event is not a valid entity.

ERREVENTINVALIDLOCATION

[4093] The location is invalid in the definition of the [name] event.

The location in the definition of a wait event is not a valid location.

ERRXMLLOAD

[4100] XML Import failed while loading. [name]Line [n]:[name]

The XML file you are trying to import has a syntax error.

ERRXMLIOREQUIREDATTRIBUTE

[4101] XML node missing required attribute.

See Documentation for required values for this XML node.

ERRXMLIOREQUIREDCHILDNODE

[4102] XML node missing required child node.

See Documentation for required values for this XML node.

ERRXMLIOINVALIDSYMBOL

[4103] The value of XML node does not match a valid symbol.

See Documentation for valid values for this XML node.

ERRXMLIOWRONGNODESFORMODE

[4104] XML node has improper number of child nodes for particular mode (rectangular or linear).

Node has different child nodes depending whether its mode is rectangular or linear.

ERRXMLIOPSACCESS

[4105] While reading XML node, failed to access placement property sheet.

An internal error occurred while importing XML. Contact [Technical Support](#).

ERRXMLIOPSCREATE

[4106] While reading XML node, failed to create a property sheet.

An internal error occurred while importing XML. Contact [Technical Support](#).

ERRXMLIOGETTEXT

[4108] Failed to get text from node.

An internal error occurred while importing XML. Contact [Technical Support](#).

ERRXMLIOINVALIDID

[4109] Node contains Invalid Id.

Node Id must take the format 'NodeName.#'.

ERRXMLIOINVALIDNUMBER

[4110] Node contains Invalid Number.

A valid number is either a decimal integer or of the form 0X### for hex or 0### for octal.

ERRXMLIOSYMBOLLOOKUPERROR

[4111] The value of XML node is not valid in its context.

See Documentation for valid values for this XML node.

ERRXMLIONODENAMELOOKUPERROR

[4112] Internal error occurred while looking up a node name.

An internal error occurred while importing XML. Contact [Technical Support](#).

ERRXMLIOSTARTENDCOLUMNREQUIRED

[4114] RSRecordSize node with this configuration requires StartColumn and EndColumn nodes.

RSRecordSize node with a PatternMarker containing PMNormal, PMNumeric, or PMAlpha requires StartColumn and EndColumn nodes.

ERRXMLIOSTARTENDCOLUMNINVALID

[4115] StartColumn and/or EndColumn values are invalid.

StartColumn and EndColumn values must be between 1 and the record set width and StartColumn cannot be greater than EndColumn.

ERRXMLIOSTARTENDCOLUMNNOTREQUIRED

[4116] RSRecordSize node with this configuration should not have StartColumn and EndColumn nodes.

RSRecordSize node with a PatternMarker containing PMNormal, PMNumeric, or PMAAlpha requires StartColumn and EndColumn nodes.

ERRXMLIOINTRANGE

[4120] Node contains number that is out of range.

The specified numeric argument was not within the range of accepted values.

ERRXMLIOINVALIDNAME

[4121] The UIName is not valid.

This design tool message is always accompanied by additional error messages given the reasons the name was rejected.

ERRXMLIODUPLICATEOBJECTNAME

[4122] The UIName is a duplicate.

Each object in a list must have a unique name.

ERRXMLIOINVALIDOID

[4123] The node has an invalid object id.

The id of an object should correspond to its node type.

ERRXMLIOINVALIDOIDREF

[4124] The node contains an invalid object idref.

Object idref's can only be of certain types.

ERRXMLIOVALIDATIONPATNOTFOUND

[4125] The validation node contains a pattern not found in the entity.

Patterns used in validation must belong to the same entity as the validation node.

ERRXMLIOVALIDATIONPATUSEDFORVALIDATION

[4126] The validation node contains a pattern which is already used for validation.

A pattern cannot be used for both entity signature and validation.

ERRXMLIOOVERLAPDELIMITER

[4127] EndOfScreenIsDelimiter and overlapsRecordsAcrossScreens cannot both be used in the same Recordset.

EndOfScreenIsDelimiter is in RSRecordSize and overlapsRecordsAcrossScreens is in RSScrollingBehavior.

ERRXMLIOCONTENTREQUIRED

[4128] Node requires text. It cannot be empty.

Enter some text in this node.

ERRXMLIOIDREFNOTINCORRECTNODE

[4129] Node contains an idref to an object that does not belong to the correct node.

Certain objects referred to by the idref must belong to the same root node as this node.

ERRXMLIOIDREFINTERNALERROR

[4130] Internal error occurred while validating idref.

An internal error occurred while importing XML. Contact [Technical Support](#).

ERRXMLIOBADDOCTYPE

[4131] The XML file must contain a DOCTYPE of VHIModel.

Bad Doctype.

ERRXMLIOINVALIDMODELCONTEXT

[4132] Please open an associated Model before importing the XML file

Cannot import an XML model file without a model context. Please open an associated Model before importing the XML file

ERRXMLWRITENULLID

[4180] Property Sheet is invalid. It has a Null ID.

Property Sheets cannot have a property ID_ of 0x0.

ERRXMLWRITEMULTIPLEEVENTS

[4181] Command List has Command Item: WaitForMultipleEvents with empty parameters.

WaitForMultipleEvents Command Item must contain one or more commands.

ERRSVOUTOFMEMORYPROPERTYSHEET

[4200] Allocation of a Session View Property Sheet failed.

An internal error occurred while creating a Session View packet. Contact [Technical Support](#).

ERRSVOUTOFMEMORYPROPERTY

[4202] Allocation of a Session View Property failed.

An internal error occurred while creating a Session View packet. Contact [Technical Support](#).

ERRENTITYREFERENCEDBYEVENT

[4221] The entity is referenced by the [name] global event.

Before deleting the entity, you must go to the Events dialog and remove this event or change its parameters to no longer reference this entity.

ERRATTRIBUTEREFERENCEDBYEVENT

[4222] The attribute is referenced by the [name] global event.

Before deleting the attribute, you must go to the Events dialog and remove this event or change its parameters to no longer reference this object.

ERRRECORDSETREFERENCEDBYEVENT

[4223] The recordset is referenced by the [name] global event.

Before deleting the recordset, you must go to the Events dialog and remove this event or change its parameters to no longer reference this object.

ERRFIELDREFERENCEDBYEVENT

[4224] The recordset field is referenced by the [name] global event.

Before deleting the recordset field, you must go to the Events dialog and remove this event or change its parameters to no longer reference this object.

ERRGLOBALEVENTPARAMETERMISSING

[4231] Parameter number [nbr] of the [event] global event must be specified.

Before accepting a global event definition, all of the event parameters must be properly specified.

ERRCHARACTERECHOAUTOTABTIMEOUT

[4252] A timeout occurred while waiting for the cursor to arrive at the next tabstop after transmitting [string].

The timeout may be due to a slow host system, network problems, or a connection failure.

ERRCHARACTERECHOSAMENUMBERTIMEOUT

[4253] A timeout occurred while waiting for the same number of characters as [string] to be echoed by the host.

The timeout may be due to a slow host system, network problems, or a connection failure.

ERRCHARACTERECHOTABORBEHINDCURSORTIMEOUT

[4254] A timeout occurred while waiting for either the characters [string] to be echoed by the host, or for the cursor to arrive at the next tabstop.

The timeout may be due to a slow host system, network problems, or a connection failure.

ERRSCRIPTMANAGERFAILEDSTART

[4300] The [type] scripting manager failed to initialize.

See the help on resetting the script manager for troubleshooting information.

ERRSCRIPTINGDISABLED

[4301] The Host Integrator server's script manager encountered a fatal error and the Host Integrator server must be restarted.

Please consult the Host Integrator server log for more information on the error that caused the script manager to fail.

ERRSCRIPTTCPINUSE

[4302] Error connecting to the script manager. It is likely that the script manager's TCP Listening port (number) is in use by a third-party application.

Prior to initializing the script manager, a test connection is made to ensure that there is no script manager already running and that the TCP listening port for the server is not in use. The information obtained from the connection attempt suggests that there is a third-party application listening on the TCP port used by the script manager.

ERRUSERCOMMANDFAILED

[4303] The user-supplied command [command] failed. Use the Event Handler Settings dialog to update the command.

The Design Tool invokes Windows CreateProcess method with user-defined command lines. Check that any escape code substitution gave the desired result and that any applications referenced are on the system path.

ERRSCRIPTMANAGERCONNECTIONFAILED

[4304] Failed to establish a connection to the script manager.

An attempt to connect to the script manager failed.

ERRSCRIPTMANAGERCOMM

[4305] An error occurred in communicating with the script manager.

An attempt to communication with script manager failed.

ERRSCRIPTMANAGEROFFLINE

[4306] The script manager is offline.

An attempt to communicate with the script manager failed because the script manager is offline. In the design tool the script manager can be restarted with the 'Reload Handlers' button on the toolbar. In the server the script manager will be restarted upon server restart.

ERRMANAGERLASTCHANCETRAP

[4307] An internal error occurred in the scripting runtime while processing a request. Cause: [fault cause].

An internal error occurred in the scripting runtime while processing a request. More information surrounding the fault can be found in via the tracing system.

ERREVENTABORTED

[4308] The event in progress was aborted.

The event in progress was aborted by the user or a failure to recover after a timeout.

ERRSCRIPTLASTCHANCETRAP

[4309] An unhandled [exception] exception occurred in the [class].[method] () event handler: [cause]

An unhandled exception occurred in an event handler. See message contents for a description of the error.

ERRSCRIPTEVENTRESULTINVALID

[4310] Invalid return value from event handler [class].[method]: [value].

An invalid value was returned from the specified event handler method.

ERRSCRIPTEVENTFAILED

[4311] The [class].[method] event handler failed.

A call to event resulted in an exception. See additional error messages for details.

ERRSCRIPTCALLBACKPARAMETERINVALID

[4312] Invalid [parameter] parameter passed to event handler callback [class].[method]: [value].

An invalid parameter was passed to the specified event handler callback method.

ERRSCRIPTCALLBACKFAILED

[4313] The [class].[method] event callback failed.

A callback to the Host Integrator server from an event failed. See additional error messages for details.

ERRSCRIPTEVENTRECURSION

[4314] Recursion detected handling the [event name] event

An event handler was invoked recursively. An event handler for a given event object can not be invoked while is it already in progress.

ERRSCRIPTCALLBACKNOTALLOWED

[4315] Event handler callback not allowed in current state.

An event callback method was invoked at an inappropriate time. If you have the Host Integrator Development Kit Online Reference installed, [see VHI help topic about details for event handler guidelines and limitations.](#)

ERRHANDLERTIMEOUT

[4316] The event handler [class name].[method name] timed out.

An event handler failed to return its response before its configured timeout.

ERRMAXSCRIPTMANAGERCONNECTIONS

[4317] Unable to obtain a connection to the script manager because the maximum number of concurrent connections has been reached ([max]).

An attempt to communicate with the script manager timed out waiting for a connection.

ERRENVIRONMENTREADONLY

[4318] One or more files or folders required for model development in [model path] is read-only. The model will be opened in read-only mode.

The Design Tool requires the ability to write to files and folders within the model folder during model development. In the event that read-only access is encountered for any of these files or folders, the model will be opened in read-only mode.

ERRMODELSRIPTSDISABLED

[4319] Model [modelname] has been disabled due to a 'Model Load' event handler failure

The 'Model Load' event handler returned an error when activating the model. This event handler must succeed (if supplied) for the model to be available on the Host Integrator server.

ERRSCRIPTMANAGERDISABLED

[4320] The script manager is disabled.

An event handler has encountered a fatal error that has left the script manager in an unstable condition. The Host Integrator server will be shut down and restarted.

ERRSCRIPTENTITYCHANGED

[4321] The current entity changed from [original] to [new] during execution of the [event] Event Handler.

Changing the current entity while executing this event handler is not allowed.

ERRSCRIPTCURRENTRECORDSETCHANGED

[4322] The current recordset on entity [entity] changed from [original] to [new] during execution of the [event] Event Handler.

Changing the current recordset while executing this event handler is not allowed.

ERRHANDLERINTERRUPTED

[4323] The event handler [class name].[method name] was interrupted due to a parent timeout.

An operation or procedure responsible for initiating an event has timed out, causing the need to interrupt the event handler.

ERRSCRIPTMANAGERDIEDUNEXPECTEDLY

[4324] The script manager terminated unexpectedly with exit code [code].

The script manager process terminated unexpectedly. The Host Integrator server will be shut down and restarted.

ERRATTRREADSCRIPTERROR

[4325] Request to read attribute [entity].[attribute] failed - call to event handler resulted in an error.

This message should always be preceded by one or more other error messages with more specific information. The additional information could be derived from an exception thrown by the user's event handler code. Alternatively, the additional messages could be internal system errors such as the event handler could not be found or the handler time out expired.

ERRATTRWRITESCRIPTERROR

[4326] Request to write attribute [entity].[attribute] failed - call to event handler resulted in an error.

This message should always be preceded by one or more other error messages with more specific information. The additional information could be derived from an exception thrown by the user's event handler code. Alternatively, the additional messages could be internal system errors such as the event handler could not be found or the handler time out expired.

ERRINITIALIZINGCLR

[4327] An error occurred initializing the Common Language Runtime for the .NET event handler engine: [error]

Contact [Technical Support](#).

ERRDOTNETSSCOMM

[4328] An error occurred marshaling data within the .NET event handler engine.

Contact [Technical Support](#).

ERRDETERMININGSPORT

[4329] An error occurred determining the communications port for the event handler engine.

Contact [Technical Support](#).

ERRFIELDREADSCRIPTERROR

[4330] Request to read recordset field [recordset].[field] on entity [entity] failed - call to event handler resulted in an error.

This message should always be preceded by one or more other error messages with more specific information. The additional information could be derived from an exception thrown by the user's event handler code. Alternatively, the additional messages could be internal system errors such as the event handler could not be found or the handler time out expired.

ERRFIELDWRITESCRIPTERROR

[4331] Request to write recordset field [recordset].[field] on entity [entity] failed - call to event handler resulted in an error.

This message should always be preceded by one or more other error messages with more specific information. The additional information could be derived from an exception thrown by the user's event handler code. Alternatively, the additional messages could be internal system errors such as the event handler could not be found or the handler time out expired.

ERRFIELDLOCATIONSRIPTERROR

[4332] Event handler returned an invalid location for recordset field [recordset].[field] on entity [entity] - the location either begins or extends beyond the range of the parent record.

Field locations returned from the ParseRecord event must fall within the bounds of the parent record. Check your event handler to ensure that it is returning a valid location for this field.

ERRRECORDLOCATIONSRIPTERROR

[4333] Event handler returned an invalid location for a record in recordset [recordset] on entity [entity] - the location either begins or extends beyond the range of the parent recordset.

Record locations returned from the ParseScreen event must fall within the bounds of the parent recordset. Check your event handler to ensure that it is returning a valid location for each record.

ERRGETCURRENTHOSTINDEXSCRIPTERROR

[4334] Event handler returned an invalid current host record index for the recordset [recordset] on entity [entity].

Check your event handler to ensure that it is returning a valid current host record index.

ERRRECORDSIZESCRIPTERROR

[4335] Event handler returned an invalid location for a record in recordset [recordset] on entity [entity] - one or more fields defined for the record begins beyond the end of the returned record location.

Record locations returned from the ParseScreen event must fall within the bounds of the parent recordset. Check your event handler to ensure that it is returning a valid location for each record.

ERRRECORDSETBUSY

[4336] Command not allowed while a fetch, insert or update is in progress for the recordset [recordset] on entity [entity].

A command or event handler callback was invoked at an inappropriate time. See the documentation on the command or event handler callback for details about when it can be called.

ERRSCRIPTMANAGERCOMMABORTED

[4337] Communication with the script manager was aborted.

An attempt to communicate with the script manager was aborted.

ERRDOTNETSCRIPTENGINE NOTPRESENT

[4338] The .net script engine is not available on this platform.

The model being deployed contains .net scripting, but the .net script engine is not available on this platform.

ERREVENTHANDLER NOTFOUND

[4340] Assigned event handler [classname] was not found.

The named event handler assigned to this object was not found in any of the event handler libraries.

ERREVENTHANDLERWRONGTYPE

[4341] Assigned event handler [classname] was of the wrong type.

The event handler assigned to this object implements the wrong type of events for that object.

ERREVENTHANDLERNOEVENTS

[4342] Assigned event handler [classname] has no defined events.

The assigned event handler does not implement any events.

ERRUNRECOGNIZEDSCREENEVENTEXCEPTION

[4345] The unrecognized screen event reported an exception: [error message].

The unrecognized screen event is triggered at defined points when Verastream expects to see a defined entity but does not find one. Verastream will report an error if this condition is not corrected. This message indicates that, in addition to the error generated by the lack of a current entity, the unrecognized screen event also returned an exception of its own. The contents of the exception are shown at the end of this message.

ERRENTITYARRIVALEVENTEXCEPTION

[4346] *The entity arrival event reported an exception: [error message].*

The entity arrival event is triggered when Verastream initially detects a defined entity. There is no event that can be triggered during a system timeout, but entity arrival and departure events will be triggered as appropriate regardless of system error state. This message indicates that, in addition to the possible additional errors present, the entity arrival event returned an exception of its own. The contents of the exception are shown at the end of this message.

ERRENTITYDEPARTUREEVENTEXCEPTION

[4347] *The entity departure event reported an exception: [error message].*

The entity departure event is triggered when Verastream initially detects a defined entity is no longer present. There is no event that can be triggered during a system timeout, but entity arrival and departure events will be triggered as appropriate regardless of system error state. This message indicates that, in addition to the possible additional errors present, the entity departure event returned an exception of its own. The contents of the exception are shown at the end of this message.

ERRWAITFORARRIVALTIMEOUT

[4350] *Request to wait for arrival on the [entity].[operation] operation timed out.*

As part of an operation implemented by an event handler, a callback was made to wait for arrival. That request includes a timeout, and this error means that timeout expired before arrival was detected.

ERRWAITFORARRIVALCANCELED

[4351] *WaitForArrival callback aborted due to user cancellation.*

A scripted operation either explicitly called the WaitForArrival callback or it implicitly called that functionality by returning at a location that was not a valid destination. While WaitForArrival was processing, the user pressed cancel or some other object the system reported an error. Either condition results in WaitForArrival to be cancelled.

ERRWAITFORARRIVALEXTERNALTIMEOUT

[4352] *WaitForArrival callback aborted due to external timeout.*

A scripted operation either explicitly called the WaitForArrival callback or it implicitly called that functionality by returning at a location that was not a valid destination. While WaitForArrival was processing, another object timed out or logged an error to cause an abort.

ERRSCRIPTCLASSLOADER

[4359] *An error occurred while loading the script classes. [fault cause].*

An error occurred while loading the script classes. One possible cause is that your model is saved in a location that is not fully trusted. Either use the .NET security tools to fully trust the location, or move the model to a trusted location such as a local disk.

ERRSCRIPTINVALIDDEBUGPORT

[4360] Script manager debug port is invalid or out of range.

The script manager TCP port, used for debugging, must be a numeric value between 0 and 65535.

ERRDEBUGPASSWORDSDONTMATCH

[4361] Client password and confirmation strings do not match.

The password and confirm password strings must match.

ERRSCRIPTPROXYNOTFOUND

[4362] A session proxy with instance identifier [Id] does not exist.

An attempt to manipulate a scripted session has failed do to an invalid identifier.

ERRINVALIDENTITYOVERRIDE

[4370] An event handler overrides the current entity to be [name], but that entity is not defined in this model.

Under some circumstances, the UnrecognizedScreen and EntityDeparture events can request an entity to be current even though Verastream does not recognize its signature patterns. However, the name provided does not match any defined entity. Check spelling and letter case.

ERRSCRIPTMETHODPARAMETERINVALID

[4381] Invalid [parameter] parameter passed to method [class].[method]: [value].

An invalid parameter was passed to the specified method.

ERRSCRIPTMETHODFAILED

[4382] The [class].[method] method failed.

A call to the Host Integrator server from an event failed. See additional error messages for details.

ERRTRACEPLAYERFAILEDSTART

[4390] The Host Emulator failed to initialize : [reason]

The Host Emulator failed to initialize. Please check file destool/logs/server.log

ERRTRACEPLAYERFAILEDSTOSTOP

[4391] The Host Emulator failed to stop : [reason]

The Host Emulator failed to stop. Please check file destool/logs/server.log

ERRSTOPRECORDINGNOW

[4392] The recording of the current session will be stopped now. Do you want to continue?

Click Yes to stop recording the current session, No to cancel.

ERRATTRIBUTEPLACEMENTOUTOFBOUNDS

[4400] The [name] attribute on entity [name] could not be located.

An attribute's location is calculated to be off the terminal screen.

ERRRSFIELDPLACEMENTOUTOFBOUNDS

[4401] The [name] recordset field in recordset [name] on entity [name] could not be located.

A recordset field's location is calculated to be off the terminal screen.

ERRRSRECORDPLACEMENTOUTOFBOUNDS

[4402] A recordset record in recordset [name] on entity [name] could not be located.

A recordset record's location is calculated to be off the terminal screen.

ERRRSCOLUMNPLACEMENTOUTOFBOUNDS

[4403] A recordset column in recordset [name] on entity [name] could not be located.

A recordset column's location is calculated to be off the terminal screen.

ERRCANNOTLOCATERELATIVEOBJECT

[4404] The [name] object on entity [name] could not be located, since the relative-to object [name] could not be located.

An object cannot be located on the screen, since the object to which it is relative could not be located.

ERRATTRIBUTECANNOTLOCATERELATIVEOBJECT

[4405] The [name] attribute on entity [name] could not be located, since the relative-to object [name] could not be located.

An attribute could not be located on the screen, since the object to which it is relative could not be located.

ERRRSFIELDCANNOTLOCATERELATIVEOBJECT

[4406] The [name] recordset field in recordset [name] on entity [name] could not be located, since the relative-to object [name] could not be located.

A recordset field could not be located on the screen, since the object to which it is relative could not be located.

ERRRSRECORDCANNOTLOCATERELATIVEOBJECT

[4407] A recordset record in recordset [name] on entity [name] could not be located, since the relative-to object [name] could not be located.

A recordset record could not be located on the screen, since the object to which it is relative could not be located.

ERRRSCOLUMNCANNOTLOCATERELATIVEOBJECT

[4408] A recordset column in recordset [name] on entity [name] could not be located, since the relative-to object [name] could not be located.

A recordset column could not be located on the screen, since the object to which it is relative could not be located.

ERRCXREMOTESSESSTOHOST

[4420] Failure telling remote session to connect to the host.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRCXLOCALSESSTOHOST

[4421] Failure telling local session to connect to the host.

An internal communication error occurred. This error could be due to excessive network traffic or low system resources. If the problem persists, contact [Technical Support](#).

ERRREJUVENATIONONLY

[4422] The model [name] is designed for rejuvenation only and must be accessed using a rejuvenation session.

In order to connect to a model from a non-rejuvenation session, the model must have at least one entity defined.

ERRDEPLOYOPTIONSLOAD

[4500] The deployment options for this model cannot be reloaded. Reverting to defaults.

A problem has occurred accessing previously stored deployment options for this model. Default settings for these options will be used.

ERRXMLVALIDATION

[4501] File [filename] failed schema validation at line [linenumber], column [column].

The portion of the XML file referenced by this error does not match the schema definition.

ERRXMLERRORDESC

[4502] XML Error: [description]

Provides details of the XML error encountered.

INVALIDIPADDRESS

[4503] Invalid IP Address [address]

The IP Address is badly formed

INVALIDHOSTNAME

[4504] Invalid Host Name [name]

The Host Name has an illegal character

HOSTPORTNUMBEROUTOFRANGE

[4505] The Host Port Number [number] is out of range. It should be between 1 and 65535.

The Host Port Number should be between 1 and 65535

ERRXMLPARSE

[4507] Cannot parse XML file '[filename]'.

The XML file referenced by this error is either not present or not well formed.

ERRXMLPARSEDETAIL

[4508] [reason] Line [line number], Column [column number]: [text].

The portion of the XML file referenced by this error is not well formed.

ERRCOMGENERIC

[4550] A COM error has occurred. Code: [code], Code Meaning: [reason], Source: [source], Description: [description]

This error describes an internal fault. Please contact Technical Support for assistance.

ERRCOMOBJECTCREATION

[4551] Cannot create COM object of type [object type].

The system cannot create a required COM object. This error typically occurs due to a corrupt installation.

ERRBADPACKAGENAME

[4600] The package or namespace contains invalid characters or reserved words

The package or namespace contains invalid characters or reserved words

ERRBADCLASSNAME

[4601] The class name contains invalid characters or is a reserved word

The class name contains invalid characters or is a reserved word

ERRRESERVEDCLASSNAME

[4602] The class name is reserved and cannot be used for an event handler

The class name is reserved and cannot be used for an event handler

ERRMODELNAMEBLANK

[4651] The Model name must not be blank.

Enter a valid model name

ERRMODELEXISTS

[4652] The model already exists.

The model already exists

ERRSETTINGSFILEBLANK

[4653] The settings file name must not be blank.

Enter a valid settings file name

ERRMODELNAMERESERVED

[4654] You are not allowed to use model name [model name].

You are using a model name that is not allowed.

ERRWAITFORHOSTDATATIMEOUT

[4660] Timed out waiting for host data.

Either the amount of time to wait was not long enough or the host was never going to send more data.

ERRAPIGENERIC

[4700] [message text]

An error occurred in the API library. Specifics are detailed in the error text.

ERRUSERGENERIC

[4701] [message text]

A user defined error. Specifics are detailed in the error text.

ERRPROCESSSTRINGNOTSUPPORTED

[4710] Process string is not supported by the model.

The process string event handler is not defined in the model. The process string API can only be used if this event handler is defined.

ERRSTARTMODELDEBUGRECORDINGFAILED

[4720] Unable to start model debug recording for session [Id].

An error occurred while attempting to create a model debug report. This error only affects the generation of a model debug report for the session. Client requests will remain unaffected.

ERRMODELDEBUGRECORDINGFAILED

[4721] Model debug recording failed for session [Id].

An error occurred while recording a model debug report. This error only affects the generation of a model debug report for the session. Client requests will remain unaffected.

ERRSAVEMODELDEBUGRECORDINGFAILED

[4722] Unable to save the model debug recording for session [Id].

Host Integrator was unable to move the generated model debug report to its storage location. This error only affects the generation of a model debug report for the session. Client requests will remain unaffected.

ERRMESSAGESTORECHANGED

[4723] The file [file] has unexpectedly changed during model debug recording.

The temporary model debug file used by Host Integrator has been changed or deleted by a third party. This error only affects the generation of a model debug report for the session. Client requests will remain unaffected.

ERRERRORREADINGMESSAGESTOREFILE

[4724] An error occurred reading debug report file [name].

The specified file has either been corrupted or does not contain a Host Integrator debug report.

ERRINVALIDMESSAGESTOREVERSION

[4725] The debug report file [name] was created with a more recent version of Host Integrator and cannot be opened.

Use a more recent version of Host Integrator to open the specified file.

ERROPENMESSAGESTORECANCELED

[4726] Open model debug report cancelled.

The opening of the model debug report was cancelled by the user.

ERRSAVEMESSAGESTORECANCELED

[4727] Save model debug report cancelled.

The saving of the model debug report was cancelled by the user.

ERRTRUNCATEDMESSAGESTORE

[4728] An error occurred reading debug report file [name]. Only the first [number] of records will be shown.

The specified file has either been corrupted or does not contain a Host Integrator debug report. Only part of the information will be shown.

ERRMAXMODELDEBUGREPORTSOPENED

[4750] The maximum number of model debug reports is open.

Close one or more model debug reports before attempting to open another report.

ERRNOMESSAGEFOUND

[4751] No message found.

No message was found matching the search criteria.

ERROBJECTNOTFOUND

[4752] Object [name] not found in the current model.

The current model does not contain the specified object. Load the model that was used to generate the message store to avoid this error.

ERRFINDMESSAGECANCELED

[4753] Find message cancelled.

Find message was cancelled before a matching message was found.

ERRNOTAILINGESCAPESEQUENCE

[4754] The data received does not end with an escape sequence.

A unique trailing escape sequence can not be found because the data does not end with an escape sequence.

ERRUNRECOGNIZEDREGEXOPERATOR

[4770] [operator] - Unrecognized regular expression operator.

The regular expression operator is unrecognized. The only valid operators are "m" and "s".

ERRDUPLICATEREGEXOPERATOR

[4771] [operator] - Duplicate regular expression operator.

The regular expression operator is specified more than once.

ERRUNRECOGNIZEDREGEXMODIFIER

[4772] [modifier] - Unrecognized regular expression modifier.

The regular expression modifier is unrecognized. The only valid modifier is "i".

ERRDUPLICATEREGEXMODIFIER

[4773] [modifier] - Duplicate regular expression modifier.

The regular expression modifier is specified more than once.

ERRREGEXCOLLATE

[4774] Invalid collating element name in regular expression: [expression]

The regular expression contains an invalid collating element name.

ERRREGEXCTYPE

[4775] Invalid character class name in regular expression: [expression]

The regular expression contains an invalid character class name.

ERRREGEXESCAPE

[4776] Invalid escaped character or trailing escape in regular expression: [expression]

The regular expression contains an invalid escaped character or trailing escape.

ERRREGEXSUBREG

[4777] Invalid back-reference in expression: [expression]

The expression contains an invalid back-reference.

ERRREGEXBRACK

[4778] Mismatched [and] in regular expression: [expression]

The regular expression contains mismatched [and].

ERRREGEXPAREN

[4779] Mismatched (and) in regular expression: [expression]

The regular expression contains mismatched (and).

ERRREGEXBRACE

[4780] Mismatched { and } in regular expression: [expression]

The regular expression contains mismatched { and }.

ERRREGEXBADBRACE

[4781] Invalid range in a {} in regular expression: [expression]

The regular expression contains an invalid range in a {}.

ERRREGEXRANGE

[4782] Invalid character range in regular expression: [expression]

The regular expression contains an invalid character range. For example [b-a].

ERRREGEXBADREPEAT

*[4783] One of *?+{ was not preceded by a valid regular expression: [expression] **

One of *?+{ was not preceded by a valid regular expression.

ERRREGEXCOMPLEXITY

[4784] The regular expression is too complex: [expression]

The regular expression is too complex.

ERRREGEXSTACK

[4785] Insufficient memory to determine whether the regular expression could match: [expression]

Insufficient memory to determine whether the regular expression could match.

ERRREGEXBADREF

[4786] A nested regular expression is uninitialized: [expression]

A nested regular expression is uninitialized.

ERRREGEXBADLOOKBEHIND

[4787] An attempt to create a variable-width look-behind assertion was detected in regular expression: [expression]

An attempt to create a variable-width look-behind assertion was detected.

ERRHOSTSSLPRIVATEKEYENCRYPTED

[4800] SSL Error - Private key is encrypted.

VHI does not support encrypted private keys for use in secure host communications.

ERRHOSTSSLPRIVATEKEYDOESNOTMATCH

[4801] SSL Error - Private key does not match the certificate used in a secure host connection.

The provide private key does not match the certificate.

MODELINREADONLYFOLDER

[4820] The model [name] cannot be opened from its current read-only folder. Do you want to copy this model to a different location and open it from there?

This model cannot be opened because you do not have write permissions for the folder in which it is stored. This often happens when you try to open an example model. The Design Tool will offer to copy the model to a different location and open it.

CANNOTCOPYMODELFOLDER

[4821] An error occurred while copying model folder [name]: [detail]

The Design Tool encountered an error while copying the model folder. Check the error message for details. To fix this problem, copy the folder containing the model to a different location and open the model.

MODELXMLERRORSPARSINGFILE

[4822] Errors in model XML file [filename]

One or more errors have occurred while reading a certain model definition file.

MODELXMLWARNING

[4823] Warning [line],[col]: [message]

A warning occurred while reading a model XML file.

MODELXMLERROR

[4824] Error [line],[col]: [message]

An error occurred while reading a model XML file.

MODELXMLFATALERROR

[4825] Fatal Error [line],[col]: [message]

An Error occurred while reading a model XML file; processing the XML file cannot continue.

MODELXMLXERCESINIT

[4826] Error initializing the XML parser: [message]

An error has occurred while initializing the model XML parser.

MODELXMLERRORSWRITINGFILE

[4827] Errors during writing of model XML file [filename]: [message]

One or more errors have occurred while writing a model definition file.

MODELXMLENTITYFILENAMEMISMATCH

[4828] Entity [name] should be defined in a file called [expectedname]

The name of the entity does not correspond with the filename in which it was defined. Rename the file to match the name of the entity.

MODELXMLTABLEFILENAMEMISMATCH

[4829] Table [name] should be defined in a file called [expectedname]

The name of the table does not correspond with the filename in which it was defined. Rename the file to match the name of the table.

MODELXMLUNRESOLVEDNAME

[4830] Unresolved name: [type] [name]

A reference name in the model XML file could not be resolved. The referenced object may have been renamed, or the file that contains the object is missing from the model folder.

MODELXMLUNRESOLVEDNAMES

[4831] The model contains one or more names that could not be resolved

This message error should be accompanied by additional error messages providing more details on the exact nature of the problem encountered.

MODELXMLINTERNALERRORDUPLICATENAME

[4832] Internal error: unexpected duplicate name [name] while reading model xml

An internal error has been detected. A duplicate name was found while reading a model xml file. Please contact [Technical Support](#) for assistance.

ERROPENINGCSSFILE

[4833] Unable to open CSS file vmr.css.

Unable to open CSS file vmr.css.

HELPNOTENABLED

[4834] Help is not currently available. To enable, reinstall product with the Help feature selected.

Help is not currently available. To enable, reinstall product with the Help feature selected.

MODELXMLTOOMANYOBJECTS

[4835] This model contains too many elements of type [type]

The model is too large; the maximum number of elements of a particular type has been exceeded.

WSERROR

[4836] Web Service error: [message]

An error has occurred in the web service subsystem.

WSWARNING

[4837] Web Service warning: [message]

A warning has occurred in the web service subsystem.

WSINFO

[4838] Web Service info: [message]

An info message has occurred in the web service subsystem.

WSDEBUG

[4839] Web Service debug: [message]

A debug message has occurred in the web service subsystem.

WSVERBOSE

[4840] Web Service verbose: [message]

A verbose message has occurred in the web service subsystem.

CFGXMLWARNING

[4841] Warning [file] [line],[col]: [message]

A warning occurred while reading a deployment descriptor XML file.

CFGXMLERROR

[4842] Error [file] [line],[col]: [message]

An error occurred while reading a deployment descriptor XML file.

CFGXMLFATALERROR

[4843] Fatal Error [file] [line],[col]: [message]

An Error occurred while reading a deployment descriptor XML file; processing the XML file cannot continue.

4.16 Connectors and APIs

4.16.1 Connectors and APIs

VHI connectors are collections of runtime objects, APIs, and libraries that will help you develop efficient client/server and Web applications that integrate host data, by means of the Host Integrator server, into various development environments.

Client/server and Web applications access data from, and input data to, a host application, by making calls to Host Integrator Server using a Host Integrator API. Upon receiving a client request, Host Integrator server instantiates a session with the host system using the logic stored in the model. Host Integrator Server navigates through the host application, fetches the requested data, and returns it to the client in a form native to the client development environment.

There are connectors available for a variety of programming languages and environments:

- [Microsoft .NET](#)

The .NET Connector creates Microsoft .NET applications that integrate host data into client/server and Web applications. Microsoft .NET is an XML Web services platform that developers can use to create programs that transcend device boundaries and fully harness the connectivity of the Internet.

- [Java](#)

The Java connector is compatible with EJB and CORBA frameworks.

- [JDBC](#)

The JDBC connector provides an industry standard Structured Query Language (SQL) interface to Host Integrator servers. Although the server is not a relational base system, the Table feature provides access to the host application in a way that simulates traditional sets of relational database tables. The JDBC connector works only with the table layer of the modeling process; access to model layer or terminal layer interfaces is not provided.

- [COM](#)

Use the COM connector to create ASP, Visual Basic, VBScript, and C++ applications that integrate host data into Web or client/server applications.

- [C](#)

Use the C connector to create applications that integrate host data into Web applications or client/server applications on Linux and Windows platforms.

Examples and Documentation

Along with a complete description of the interface, example programs are available. In the online help under the Java, COM, C, and COM Connector nodes, click Examples. You can use the Web Builder tool to create Java projects, based on the "demo" models provided with Verastream Host Integrator, and then examine the Java code in your projects to see how the Java connector works.

There are also event handler examples for both .NET and Java; open `<install directory>\Verastream\HostIntegrator\examples`. This folder also contains an example of Verastream Host Integrator C API.

API documentation is provided for the Java, C, JDBC, and .NET connectors. A Visual Basic API reference is provided for the COM connector, though COM supports additional programming tools, including C++, JScript, and VBScript.

See [Accessing Host Data](#).

Additional Host Integrator APIs

There are references for two additional Host Integrator APIs available:

- Event Handling API

Event handling extends the capabilities of models by letting you define specific events that suspend the interpretation of a model and turn control over to user-supplied procedural code.

- Model Variable Management API

The `com.wrq.vhi.sconfig` package contains methods for managing model variable lists on Host Integrator servers. See [Working with Model Variable Lists](#), in the Host Integrator Administrative Console, under [How to Use Host Integrator](#) for information on creating model variables and model variable lists.

4.16.2 Accessing Host Data

Creating client applications that interact with host applications involves:

First, modeling your host application using the Design Tool

Second, connecting to a Host Integrator server

Third, interacting with the data in the host application

You can also use filter expressions to fetch records from a recordset.

Modeling your host application using the Design Tool

With the Design Tool, you create a model of the host application. The model consists of a main model file (.modelx) and several supporting files that are located in the `\<VHI install directory>\models\<your model>` folder. See the Design Tool Reference for detailed information on how to create models.

Connecting to a Host Integrator Server

To connect to a Host Integrator server, you must decide whether you will connect to a model or session pools. The key difference is that host application models do not connect to and log onto a host until you request the model; sessions are typically already connected to the host and are queued to a particular screen in a host application, thus have superior performance results and is generally recommended. However, your Host Integrator implementation may not use session pools.

There are four methods available to you:

ConnectToModel

ConnectToModelViaDomain

ConnectToSession

ConnectToSessionViaDomain

Secure connections

You can make sure the connectors communicate with the Host Integrator servers using a secure connection by configuring the server to require a secure connection or having the client request a secure connection by calling the `RequireSecureConnection` method before establishing a connection.

Note

`RequireSecureConnection` is deprecated. Connectors now always connect securely.

Interacting with data in the host application

Note

An application that interacts with VHI models should not mix model-level and table-level methods. This is because the operation definitions in these two model layers make certain assumptions about entity navigation paths and cursor positions. For example, if you issue a model-level method in a table-based model, it can break the table operations because the model subsequently can't locate the table's home entity. Unless you are very familiar with a model, you should avoid mixing procedures with direct model access.

Accessing host data

You can access host data using one of the following approaches:

Procedures

Models

At the terminal level

Using filter expressions

PROCEDURES

This approach is possible when you're working with host models that contain tables and procedures. This is the most efficient way to access host applications, and is also the technique that requires the least knowledge about the host application and the model created from that host application.

While using model-level methods requires some knowledge of the host application model, using procedure-level methods to access table definitions of data requires no direct knowledge of the host application or the modeling process. The person who created the host application model will have abstracted the host application data into tables that can respond to SQL queries just like an actual relational database. The table interface supports a subset of the SQL 92 standard for SELECT, UPDATE, INSERT, and DELETE statements.

There are two key methods in each of the connectors for interacting with the host application at the procedure level:

The `PerformTableProcedure` method specifically references procedures created in the Design Tool. It can take input and filter values and can return a recordset with a collection of records.

The `ExecuteSQLStatement` method executes an SQL statement and return a recordset with a collection of records.

You can use either one of these methods depending on your preference and experience. If you have SQL or database experience, you might prefer to use `ExecuteSQLStatement`. But the two methods are largely interchangeable. The concept is the same in either case: working at the procedure level turns your host application into a virtual database.

Some of the filtering and sorting options available in SQL are not available when you access host data; regardless of the method you choose.

All of the host navigation and interpretation is already contained in the procedure definitions in the model file. To see a list of procedures and columns stored in the model file, you can generate an XML- or HTML-based model file using the Export dialog box of the Design Tool.

Your target API has documentation on the data structures you use to call `ExecuteSQLStatement` or `PerformTableProcedure`, or to manipulate the returned data.

MODELS

You may encounter host applications that you cannot effectively mine using procedures (although event handlers can help you with even the most unorthodox host applications.) For such applications, you can create clients that interact directly with the model's entities, attributes, operations, and recordsets.

If you choose not to interact with the host application using procedures, use model-level methods to access host data or to navigate through a host application. Working at the model layer requires knowledge of the Design Tool's modeling process, and of the specific model you are interacting with. Information on the model's entities, operations, recordsets, and attributes can be accessed directly from the model, or from the XML- or HTML-based model file created in the Design Tool's Export dialog box.

Navigating Host Applications

You can use several different model-level methods to navigate through a host application. Use the `SetCurrentEntity` method to navigate directly to a host application screen, or the `PerformEntityOperation` method to navigate through a host application by executing an operation directly.

You can also navigate by means of a recordset that supports selection, using a record in the recordset to navigate to another entity. You can set the recordset index using `SetCurrentRecordIndex` to set the index to an absolute value, and `MoveCurrentRecordIndex` to set the index by a relative movement method (for example, `ScrollHome`, `ScrollEnd`, `ScrollLineUp`, `ScrollLineDown`, `ScrollPageUp`, and `ScrollPageDown`).

You can select a record to navigate to another entity using the following methods:

`SelectCurrentRecord` – Selects the record at the current record index.

`SelectRecordByIndex` – Selects the record at an absolute index.

`SelectRecordByFilter` – Selects the first record which meets the filter conditions.

`SetCurrentRecordSetByName` – Sets which recordset to use for entities that have more than one recordset defined.

AT THE TERMINAL LEVEL

This approach bypasses the model—and the abstraction and optimization offered by Host Integrator—completely.

Unlike the model-level methods, using terminal methods requires an intimate knowledge of the host application since the entity definitions created in the model file are not used with terminal-level methods. Terminal-level access is the most laborious and inefficient way to access host data, and can usually be avoided with good modelling practices.

You can set host data directly using `InsertStringAtOffset` and `InsertStringAtRowColumn`, or access host functionality with `PerformAidKey`. You can access host data with `GetStringAtOffset` and `GetStringAtRowColumn`. You should only use terminal methods if you do not want to use the entity definitions that have been created in the model file.

Terminal methods also contain synchronization methods that wait for an event on the host application before executing. For example, `WaitForCursor` waits for the cursor to be in placed in a given row or column, `WaitForString` waits for a string to appear at the given row and column, and `WaitForStringRelCursor` waits for a string to appear at a screen position relative to the host application cursor.

Retrieving Terminal Attributes

Terminal attributes are host-generated properties that affect how an attribute displays in the terminal window. Terminal attributes are used to denote the importance of a field or its availability. Before you can retrieve terminal attributes from a model, you must enable them in the model. If you are unsure whether terminal attributes are enabled in your model, check with the model designer. For more information, see the [Enable terminal attributes](#) setting.

Note

After you enable terminal attributes, they remain enabled for the duration of the session, until you explicitly disable them. Leaving terminal attributes enabled will have an impact on performance, so after you have retrieved attributes, disable the setting until you need to retrieve attributes again.

To retrieve terminal attributes from a host application model:

Enable the retrieval of terminal attributes by calling the `EnableTerminalAttributes` method and setting the `enable` parameter to `True`.

Use `getAttributes` or `fetchRecords` to obtain a `ModelRecord` or `RecordSet` object to be used by the `getTerminalAttributes` method.

3. Retrieve the terminal attributes using the `getTerminalAttributes` method. For example, here is a section of Java code that demonstrates this process:

3. `java mySession.enableTerminalAttributes(enable);`

```
JOptionPane.showMessageDialog(null, "Current Entity: " + mySession.getCurrentEntity());
entityName = JOptionPane.showInputDialog("Goto entity: ");

mySession.setCurrentEntity(entityName);
JOptionPane.showMessageDialog(null, "Current Entity: " + mySession.getCurrentEntity());

myModelRecord = mySession.getAttributes(null); //null specifies all attributes
JOptionPane.showMessageDialog(null, myModelRecord.toString());
System.out.println(myModelRecord.toString());

attributeName = JOptionPane.showInputDialog("Attribute name: ");
myTerminalAttributes = myModelRecord.getTerminalAttributes(attributeName);

if (myTerminalAttributes.isReverse())
{
    JOptionPane.showMessageDialog(null, "isReverse = True");
} else
{
    JOptionPane.showMessageDialog(null, "isReverse = False");
}
...
```

USING FILTER EXPRESSIONS

Many methods let you specify filter expressions when fetching records from a recordset. The following table illustrates the format for syntax expressions:

Condition expressions

You can use expressions on both the left and right sides of the condition statement. Be sure to enclose string data in quotation marks.

Expression	Example
AND	condition_expression AND condition_expression
OR	condition_expression OR condition_expression
NOT	NOT condition_expression
Equal to	value_expression = value_expression
Equal to (case insensitive)	value_expression =* value_expression
Not equal to	value_expression <> value_expression

Expression	Example
Less than	value_expression < value_expression
Greater than	value_expression > value_expression
Less than or equal to	value_expression <= value_expression
Greater than or equal to	value_expression >= value_expression
()	(multiple condition_expressions)

Value expressions

A value expression can take any of the following formats:

Variable (variables.variablename)

Attribute (attribute name)

Field (recordset.recordsetfield)

String

Integer

Floating point number

Filter expression examples

This example returns all recordset fields in the patients recordset that are not "Smith":

```
patients.lastname <> "Smith"
```

Here's an example that is exactly equivalent to the first example: NOT (patients.lastname = "Smith")

This example returns all records in the SearchResults recordset with the field "last name" equal to "Smith" and the field "first name" equal to "Steven":

```
(SearchResults.LastName = "Smith") and (SearchResults.FirstName = "Steven")
```

This last example returns all fields in the AccountNumbers recordset greater than or equal to 10000 and less than or equal to 20000:

```
(AccountNumbers.Accounts >= 10000) and (AccountNumbers.Accounts <= 20000)
```

4.16.3 .NET Connector

Requirements

Method reference

Use the Verastream .NET Connector to create WS-I compliant Microsoft .NET applications that integrate host data into Web applications or client/server applications. Microsoft .NET is an XML Web services platform that developers can use to create programs that transcend device boundaries and fully harness the connectivity of the Internet.

This reference is intended for developers who are familiar with object-oriented development environments and who have a thorough knowledge of Visual Basic, C#, or ASP.NET Web programming with scripting languages like JScript or VBScript. You should also be familiar with the Host Integrator Design Tool, VHI model files, and the Host Integrator server.

This section describes Verastream's .NET connector and provides you with tips on how to build powerful and robust applications that take advantage of the connector's features and options:

Overloaded Methods

Other Verastream connectors support the use of optional parameters or null arguments to accommodate the different arguments you might or might not want to include. For example, the `ConnectToModel` method in the COM connector has two required arguments and three optional arguments, as indicated by this syntax:

```
object.ConnectToModel Server, ModelName, [UserID], [Password], [ModelVariables]
```

In the .NET connector help, separate versions of a method, called *overloads*, are presented for likely or common combinations of arguments. For `ConnectToMethod`, there are four such overloads:

```
ConnectToModel(server, modelName)
ConnectToModel(server, modelName, userID, password)
ConnectToModel(server, modelName, userVariables)
ConnectToModel(server, modelName, userID, password, modelVariables)
```

This style of presentation makes explicit exactly which combinations of arguments are legal. It may seem obvious that the password argument cannot be used without the userID argument, the overloaded syntax statements in the .NET connector make this explicit, while the optional argument syntax used in the COM connector does not.

The namespace for the .NET Connector is `WRQ.Verastream.HostIntegrator`. A .NET namespace is analogous to a package in Java.

Help Available in Multiple Formats

The classes, methods, and properties that comprise the .NET connector are documented in a MSDN-style compiled help system that is included with Verastream Host Integrator.

Context-sensitive help on the various elements of the .NET connector is also available with Visual Studio.

Requirements

The .NET Connector is an install option, which copies the necessary files to your computer.

To use the .NET connector to develop an application that accesses the host application model file on your Host Integrator Server, you need:

- .NET development tool—Microsoft Visual Studio .NET., version 7.0 or higher.

- .NET Framework SDK, version 2.0 or higher, to link to information on .NET data types from the Verastream MSDN-style compiled help system. The .NET Framework extends Microsoft's .NET implementation, providing improvements to existing features and enhancements to the documentation. You can use the .NET Framework to build, deploy, and run XML Web services and .NET-connected applications.

To create .NET projects that can interact with Host Integrator through the .NET connector, you must add a reference to the connector library in Visual Studio .NET which can be found here:

```
<VHI install folder>\lib\dotnet\WRQ.Verastream.HostIntegrator.dll
```

Method Reference

The [reference for the Host Integrator .NET connector](#) is available here.

IMPORTANT NOTE

Depending on how you are accessing this file, it may be necessary for you to save the file to a local machine and open the file from that location. To open the file from the local machine, right-click and choose Properties. On the General tab, click Unblock, and then OK.

4.16.4 Java Connector

Use the Java Connector to create Java applications that integrate host data into web applications or client/server applications. This section describes the Java connector and provides tips on how to build powerful and robust applications that take advantage of the connector's features and options.

You can use the Web Builder tool to create Java projects, based on the "demo" models provided with Verastream Host Integrator, and then examine the Java code in your projects to see how the Java connector works.

To be able to use the Verastream Java Connector on your computer, you must choose the "Java Connector" when you're installing Verastream Host Integrator. Installing with this option selected copies all necessary files to your computer including the Java Development Kit.

To use the Java connector to develop an application that accesses the host application model file on your Host Integrator Server, you need the provided Java Development Kit as well as a Java IDE of your choice. You can use the JDK Framework to build, deploy, and run XML Web services and Java-connected applications.

[Java class library](#)

[Java interfaces](#)

[Importing Java classes](#)

[Method reference](#)

Java Class Library

The AppConn Class Library

The file `apptrieve.jar` contains the class libraries for the Java connector. The AppConn Java methods are documented in a set of Javadoc HTML files installed with the SDK. To view these files, you need a supported browser. View the [methods reference](#) in a new browser window.

The AppConn Java interface is implemented through the following classes:

- `com.wrq.apptrieve.appconn.AppConnSession`

This class contains methods that manage model- and terminal-level tasks like connecting and disconnecting from the Host Integrator Server, navigating to and fetching entities and attributes, performing operations, and managing host application model variables. It also contains methods that perform table-level tasks, such as executing SQL statements and performing procedures.

- `com.wrq.apptrieve.appconn.AppConnRecord`

This contains methods that access the data fields returned by access methods in the `AppConnSession` class.

- `com.wrq.apptrieve.appconn.AppConnRecordSet`

This class contains methods that access data records returned by the fetch methods in the `AppConnSession` class. It always contains a collection of either `AppConnRecord` or `AppConnModelRecord` objects.

- `com.wrq.apptrieve.agent.AppConnException`

These libraries contain methods that help you manage exceptions, errors and security.

IMPORTING THE APPCONN JAVA CLASSES

To use the `AppConn` interfaces, add the following statement into your classes:

```
import com.wrq.apptrieve.appconn.*;
import com.wrq.apptrieve.agent.*;
import org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider;
import org.bouncycastle.jsse.provider.BouncyCastleJsseProvider;
import java.security.Security;
```

ADDING THE BOUNCY CASTLE SECURITY PROVIDER

```
security.addProvider(new BouncyCastleFipsProvider());
security.addProvider(new BouncyCastleJsseProvider());
```

VHI uses Bouncy Castle as the default SSL engine for HTTPS connections. To facilitate this the Bouncy Castle provider must be the default provider and the default SSL provider. Instead of

importing the Bouncy Castle security statements, you can add the the Bouncy Castle security provider by modifying your `java.security` file in the JRE you are using to run the Java application:

Edit your `<jre>/lib/security/java.security` file and add the Bouncy Castle providers to the top of the list. Remember to re-number the elements below the Bouncy Castle providers.

Java Interfaces

The AppConn Java methods are implemented through the interfaces listed below. Click the interface name to view the Java documentation for it.

- `com.wrq.apptrieve.appconn.AppConnChannel`

The `AppConnChannel` interface contains communication methods for connecting to the Host Integrator Server and establishing sessions with models and session pools.

- `com.wrq.apptrieve.appconn.AppConnTable`

Use the `AppConnTable` interface to interact with host application data that is based in tables. `AppConnTable` extends `AppConnChannel`, so all required connect or disconnect methods are automatically included.

- `com.wrq.apptrieve.appconn.AppConnModel`

Use the `AppConnModel` interface to interact with host application data that is based in attribute and recordset field definitions. `AppConnModel` extends `AppConnChannel`, so all required connect or disconnect methods are automatically included.

- `com.wrq.apptrieve.appconn.AppConnTerm`

The `AppConnTerm` interface contains methods for managing terminal sessions. You can issue any available terminal key as well as query the host for strings at the given the cursor position or screen coordinates. `AppConnTerm` extends `AppConnChannel`, so all required connect or disconnect methods are automatically included.

STRUCTURING YOUR APPLICATIONS

The `AppConnChannel`, `AppConnTable`, `AppConnModel`, and `AppConnTerm` interfaces are all implemented by the `AppConnSession` class. It's unlikely that you will work with a host application model that requires you to interact on the model, table, and terminal levels at the same time. For this reason, we recommend that when you begin new applications, you use the more specific interface definitions to keep the number of available methods within the relevant framework. For example,

```
AppConnTable mySession=new AppConnSession();
```

The statement above creates a new object that supports only the `AppConnTable` interface class.

IMPORTING JAVA CLASSES

To use the AppConn interfaces, add the following statement into your classes:

```
import com.wrq.aptrieve.appconn.*; //appconn classes
import com.wrq.aptrieve.agent.*; //exception classes
import org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider;
import org.bouncycastle.jsse.provider.BouncyCastleJsseProvider;
import java.security.Security;
```

Method Reference

The method reference for the Host Integrator java connector is available [here](#).

FIELD ATTRIBUTES AND COLOR CONSTANTS

The method `getXMLText` (`public java.lang.String getXMLText()`) creates an XML document that specifies the text, colors, fields, and field attributes of a terminal screen. These attributes are also valuable if you want to understand or modify the XSLT used to create the green terminal screen.

The XML document references these attributes:

Field attributes

Attribute	Value
BL	Blinking text
CO	Column separated (AS/400)
Hi	Hidden
Hn	High intensity
In	Input (unprotected)
Nu	Numeric
Pe	Pen detectable (3270)
Re	Reverse video
Un	Underscore
CLR	color
Ofs	Offset
Len	Length

Attribute	Value
MDT	Field-modified

Color constants

Color constant	Value
Red	Host red
Blu	Host blue
Pnk	Host pink
Grn	Host green
Tur	Host turquoise (cyan)
Ylw	Host yellow
Wht	Host white
SL	Status line (AS/400)
PNA	Protected/Normal/Alpha
PNN	Protected/Normal/Numeric
PHA	Protected/Highlight/Alpha
PHN	Protected/Highlight/Numeric
UNA	Unprotected/Normal/Alpha
UNN	Unprotected/Normal/Numeric
UHA	Unprotected/Highlight/Alpha
UHN	Unprotected/Highlight/Numeric
ML	Message line (AS/400)
EL	Error line (AS/400)

Color constant	Value
SR	Sys request (AS/400)

EXAMPLES

You can use the Web Builder tool to create Java projects, based on the "demo" models provided with Verastream Host Integrator, and then examine the Java code in your projects to see how the Java connector works.

Pacific Department Stores Demo

The Pacific Department Stores web front-end provides users with a simple, easy-to-use interface to a mainframe application. The mainframe application's business processing is accessed through a Verastream model.

The demo is deployed to the local server and uses the Host Emulator to simulate the mainframe application. Before you run the demo, make sure that the localhost server is running in the Host Emulator.

To run the Pacific Department Stores demo:

Deploy the PurchasesDemo and CICSAcctsDemo models to localhost. You can find these models in the `Micro Focus\Verastream\HostIntegrator\examples\ModelSamples` directory.

Type or paste this URL into your browser: <http://localhost:8081/PacificDepartmentStores>.

Login as a CSR (Customer Services Representative).

Display, modify, or add customer accounts.

Because the mainframe application is just a simulation in this case, available data is limited to customers with names beginning with K or W, and to account numbers 20000 through 20004.

4.16.5 JDBC Connector

The JDBC connector provides an industry standard Structured Query Language (SQL) interface to Host Integrator servers. Although the server is not a relational database system, the Design Tool's Table and Procedure feature provides access to the host application in a way that simulates traditional relational database tables.

The JDBC connector works only if you're accessing host data using procedures; accessing host data at the model or terminal level is not an option.

This document assumes that you are familiar with using JDBC to access a standard relational database and also that you understand the basics of creating procedures and tables using the Design Tool.

Connecting to a Host Integrator Server using the `ApptrieveDriver` Class

Use of the Host Integrator JDBC connector is always within the context of the standard `java.sql.*` classes. Connecting to a Host Integrator session using JDBC is a four step process:

1. Import the `java.sql.*` class. Include this line at the top of your Java file:
 1. `import java.sql.*;`
2. Insert the Bouncy Castle security providers. Add the following method and invoke it in the beginning of your program:

```
static void registerProviders()
{
    // initialize a few properties used by BCFIPS and BCJSSE
    Security.setProperty("ssl.KeyManagerFactory.algorithm", "PKIX");
    System.setProperty("org.bouncycastle.ec.disable_mqv", "true");
    System.setProperty("org.bouncycastle.jsse.ec.disableChar2", "true");

    // the next line shows how to place BCFIPS in approved-only mode.
    // CryptoServicesRegistrar.setApprovedOnlyMode(true);
    Security.insertProviderAt(new BouncyCastleFipsProvider(), 1);
    Security.insertProviderAt(new BouncyCastleJsseProvider(), 2);
}
```

1. Register the Host Integrator driver with the JDBC Driver Manager. The Host Integrator driver is implemented in `com.wrq.apptrieve.jdbc.ApptrieveDriver.class`;
 1. Include the following Java statement before using the JDBC connector:
 1. `Class.forName("com.wrq.apptrieve.jdbc.ApptrieveDriver");`
 2. Connect to the Host Integrator JDBC driver using the standard `DriverManager` methods.

The `java.sql.DriverManager` class contains a `getConnection` method which establishes a connection to a database using the desired driver. There are three forms of this method:

- one that takes only a URL string which specifies the connection to make,
- one that takes a URL and a `java.util.Properties` object, and
- one that takes a URL and user name and password String parameters.

The key parameter to each of these three variants is the URL string. The syntax for Host Integrator is:

```
jdbc:apptrieve:table[;managementserver=managementserver_name][;domain=domain_name][;serverid=vhi_server]
[;database=model_name][;session=sessionpool_name]
[;user=user_name][;password=xxx][;variables.varname=value]
[;debug=out|err]
```

The first part of the URL, `jdbc:apptrieve:table` indicates that you want to make a connection using the Host Integrator JDBC connector (formerly called Apptrieve).

A connection can be made either directly with the Host Integrator Server or, to use the dynamic load balancing and failover features of the product, indirectly using a domain. To specify a serverID, use the server keyword and include either the hostname of the server or the IP address for the desired server. To specify a domain, include the management server keyword and the management server hostname or IP address and the domain keyword, along with the name of the Host Integrator domain that you wish to use. Never include both the server keyword and the management server/domain keywords because they are mutually exclusive.

A Host Integrator session can either load a model when the connection is established, or it can use an existing session from a session pool. To specify a model-based session, use the database keyword and include the name of the model that you wish to use.

Models and session pools are configured with the Host Integrator Administrative Console. Never include both a database keyword and a session keyword in a URL – they are mutually exclusive.

If security is enabled for the domain or Host Integrator Server, you must specify the user name and password for the connection using the user and password keywords. Alternatively, you can use the `getConnection()` method that includes user and password String parameters – these do exactly the same thing as specifying the user and password in the URL itself. These keywords refer to the user name and password for the Host Integrator Server – not the host that the Host Integrator Server connects to for executing the model or the session pool. If these need to be passed into the session, use the 'variables' part of the URL to set the built-in userid and password model variables. The variables keyword can set any model variable that is included in the model and has write permission as established in the model definition.

Connections to a Host Integrator Server are always encrypted, independent of the security setting.

The debug keyword enables the JDBC connector debug messages, which are sent to either standard 'out' or standard 'err' output stream. The messages may be useful in debugging your application.

EXAMPLE URL STRINGS

The following are example URLs to illustrate the various alternatives of establishing a connection to a Host Integrator session:

- Simple connection to a model, no security or model variables:

```
String url = jdbc:apptrieve:table;serverid=vhi_server;database=CCSDemo
```

- Connection to a domain, security enabled:

```
String url =
jdbc:apptrieve:table;aads=vhi_aads;domain=myDomain;database=CCSDemo;
user=vhi_user;password=login1
```

- Connection to a session pool, specifying host user name and password:

```
String url =
jdbc:apptrieve:table;serverid=vhi_server;session=CCSDemoSessionPool;
variables.userid=bjones;variables.password=bjones
```

EXAMPLES OF USING DRIVERMANAGER.GETCONNECTION()

Once you have a suitable URL string, include it in one of the three `getConnection` methods to establish a connection with the Host Integrator Server. For example, to specify everything using a URL:

```
Connection myConnection = DriverManager.getConnection(url);
```

To specify a Host Integrator user name and password in the method call rather than in the URL, use the three parameter form:

```
Connection myConnection = DriverManager.getConnection(url, "vhi_user", "login1");
```

Finally, any of the keywords that can be used in a URL string can alternatively be put into a `java.util.Properties` object and sent to the JDBC connector using the two parameter form of `getConnection()`:

```
java.util.Properties myProps = new java.util.Properties();
myProps.put("serverid", "vhi_server");
myProps.put("database", "CCSDemo");
Connection myConnection = DriverManager.getConnection("jdbc:apptrieve:table", myProps);
```

Using Standard SQL Statements

The primary way of interacting with the Host Integrator Server using the JDBC connector is SQL statements. Host Integrator supports a subset of the SQL language that allows retrieving information from the server, adding new information, deleting information, and updating information. See [Creating SQL-Based Queries](#) for a full description of the allowed SQL subset and how it relates to tables constructed in the Design Tool.

To execute an SQL statement using the JDBC connector, use one of the execute methods in the `java.sql.Statement` class. There are three forms of execute: one for any SQL statement, one for queries, and one for updates, selects, inserts, and deletes. The most common form, `executeQuery`, is for retrieving information using a SELECT SQL statement, (queries). An example of using `executeQuery()` is:

```
Statement stmt = myConnection.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM Accounts where AcctNum=20000");
```

This query asks the Host Integrator Server to fetch all of the columns from the Accounts table for the account number (20000) specified. The JDBC ResultSet object, rs, will hold the record that the Host Integrator returns. (There is a single record since AcctNum selects a unique record.)

To update, add, or delete information in a table, use the executeUpdate() JDBC method. For example, the following shows how to insert a record:

```
Statement stmt = myConnection.createStatement();
String acctCols = "AcctNum, LastName, FirstName, MiddleInitial, Title, Phone, Address1, Address2,";
acctCols += "NumCardsIssued, DayIssued, MonthIssued, YearIssued, Reason, CardCode, ApprovedBy";
String acctVals = "20005, 'Smith', 'Steven', 'W', 'exec', '2065551234', '1342 15th Street E', 'Seattle, WA,'";
acctVals += "2, '06', '01', '99', 'M', 'C', 'GWB'";
int recordsUpdated = stmt.executeUpdate("INSERT INTO Accounts (" + acctCols + ") VALUES (" + acctVals + ")");
```

Finally, an SQL statement can be executed with the java.sql.statement execute() method. For a query statement, the resulting recordset object can be retrieved by calling the getResultSet() method on the statement object.

Handling SQL Exceptions

Many of the methods in the java.sql classes throw SQLException upon failure. Host Integrator takes advantage of SQLException to chain multiple exceptions together. Use the method getNextException to get the next exception in the chain, stopping when this method returns null.

For example, to print the full exception chain starting with the top level SQLException, use the following Java method:

```
void printException(Exception e) {
    Class cl= e.getClass();
    if(cl.getName().compareTo("java.sql.SQLException") == 0)
    {
        SQLException SQLEx = (SQLException) e;
        System.out.println("Exception chain:");
        while(SQLEx != null) {
            System.out.println("Code:" + SQLEx.getErrorCode() + " -- " + SQLEx.getMessage());
            SQLEx = SQLEx.getNextException();
        }
    }
    else {
        System.out.println("Exception: " + e.getMessage());
    }
}
```

The error code number that Host Integrator supplies with a SQLException signifies the class of Host Integrator exception that occurred.

You can retrieve the error code for SQLException using the getErrorCode() method.

Error code	Description
001	ApptrieveException - the top level exception indicating the JDBC method that failed.
101	ChannelException - provides information on errors that occur in the network interactions with the Host Integrator Server.

Error code	Description
201	DeadSessionException - class provides information on errors that occur at the Host Integrator Server resulting from fatal errors that occur between the Host Integrator Server and the host's terminal session. The condition is not recoverable.
301	MarshallerException - provides information on errors that occur in the network type marshaller.
401	ModelDataException - class provides information on errors that occur at the Host Integrator Server resulting from bad arguments passed in method calls.
501	ModelDefException - provides information on errors that occur at the Host Integrator Server resulting from bad arguments passed in method calls.
601	ServerException - provides information on errors that occur at the Host Integrator Server.
701	TerminalException - provides information on errors that occur at the Host Integrator Server resulting from errors that originate on the host's terminal session.
801	TimeoutException - relays method call timeouts to the user.

Error code	Description
901	UserException - provides information on errors that occur at the Host Integrator Server dealing with error conditions defined by the model author.

Using Prepared Statements

The primary advantage of using prepared statements with the Host Integrator JDBC connector is to allow parameterized SQL statements. Host Integrator does not support compiling SQL statements, which is the traditional reason for using prepared statements; therefore there is no gain in efficiency when executing a prepared statement with Host Integrator JDBC.

However, it is often convenient to create a prepared statement that includes parameters to provide variable aspects of an SQL statement. For example, in a Web application, to insert a new record into the host application, parameters can be associated with input items in an HTML form. When the user enters new data into the form, each input item is transferred into the corresponding prepared statement parameter and then the prepared statement is executed. This results in the insertion of the desired record into the host application via the Host Integrator table layer.

The following Java example illustrates how a prepared statement might appear for an insert operation (written for the CICSAccts model).

```
String acctCols = "AcctNum, LastName, FirstName, MiddleInitial, Title, Phone, Address1, Address2,";
acctCols += "NumCardsIssued, DayIssued, MonthIssued, YearIssued, Reason, CardCode, ApprovedBy";

String sql = "INSERT INTO Accounts (" + acctCols + ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
PreparedStatement pstmt = myConnection.prepareStatement(sql);

pstmt.setInt(1, 20005);
pstmt.setString(2, "WILSON");
pstmt.setString(3, "MARTIN");
pstmt.setString(4, "B");
pstmt.setString(5, "MR");
pstmt.setString(6, "9876543210");
pstmt.setString(7, "1234 56TH ST.");
pstmt.setString(8, "HOUSTON, TX 98765");
pstmt.setInt(9, 1);
pstmt.setString(10, "01");
pstmt.setString(11, "01");
pstmt.setString(12, "2001");
pstmt.setString(13, "N");
pstmt.setString(14, "1");
pstmt.setString(15, "1");
pstmt.executeUpdate();
```

When setting parameters, note that the first parameter is 1, and that Host Integrator always sets the parameter value to the equivalent string. For example, the PreparedStatement method setDate(int

parameterIndex, java.sql.Date x) converts the supplied date object into a string and sets the parameter specified by parameterIndex to the resulting value.

Executing Table Procedures

The Host Integrator JDBC connector directly accesses the table layer of a given model. When processing an SQL statement, the Host Integrator Server analyzes the statement and decides the most appropriate table procedure to perform. It is possible to perform a procedure directly using the [CallableStatement JDBC](#) class. In the Host Integrator driver, the only purpose for the CallableStatement class is to provide access to the predefined procedure that exists in the connected model.

The format for preparing a callable statement is always the same for Host Integrator:

```
CallableStatement cstmt = myConnection.prepareCall("{call  
<procedureName>(?, ?, ?, ?, ?, ?)}");
```

Where `<procedureName>` is the name of the procedure in the Host Integrator model. The parameters always have a fixed meaning, as follows:

Parameter	Fixed meaning
table name	a java.lang.String that contains the Host Integrator table which defines the procedure (required and may not be null).
data input values	a java.util.Map object that contains any data input name-value pairs for the procedure (not required and may be null).
filter values	a java.util.Map object that contains any filter name-value pairs for the procedure (not required and may be null).
filter is case sensitive flag	a Boolean object that sets whether or not the filter should be case sensitive (not required and may be null).
output columns	a java.util.List object that contains a list of java.lang.String objects, each of which is a column to return in the output result. The default is to return all output columns defined for the procedure (not required and may be null).

Parameter	Fixed meaning
maximum rows	an integer that set the maximum number of rows to return in the result The default is to return all available rows, which is the same as setting this parameter to 0 (not required).

For String parameters, use `setString()`; for Map, List and Boolean parameters, use `setObject`; and for the integer parameter (maximum rows), use `setInt()`.

EXAMPLE

To call the `CompoundNameSearch` procedure of the `Accounts` table in the `CICSAccts` model, the Java implementation would look like this:

```
CallableStatement cstmt = myConnection.prepareCall("{call CompoundNameSearch(?,?,?,?,?,?)");

cstmt.setString(1, "Accounts"); // table name

HashMap filter = new HashMap();
filter.put("LastName", "W");
cstmt.setObject(3, filter); // filter

cstmt.setObject(4, new Boolean(true)); // filter is case sensitive flag

List outputCols = new Vector();
outputCols.add("AcctNum");
outputCols.add("LastName");
outputCols.add("FirstName");
outputCols.add("MiddleInitial");
cstmt.setObject(5, outputCols); // output columns

cstmt.setInt(6, 5); // maximum rows

ResultSet rs = cstmt.executeQuery();
```

Accessing Table Metadata

Information about the Host Integrator 'database' (that is, model that you are using) is available in two forms: `ResultSet` metadata and Database metadata.

ResultSet Metadata

After performing a SQL `SELECT` statement, you can retrieve metadata from the returned `ResultSet` object. By calling the `getMetaData` method of `ResultSet` to obtain a `ResultSetMetaData` object for that `ResultSet`, you can then query several items of metadata. Some of the most important items for Host Integrator are:

ResultSet MetaData Method Name	Description
<code>getColumnCount</code>	Returns the number of columns in a row of the <code>ResultSet</code> .
<code>getColumnName</code>	Gets the name of a column in the <code>ResultSet</code>

ResultSet MetaData Method Name	Description
getColumnType	Gets the data type of a column in the ResultSet (from java.sql.Types).
getTableName	Gets the name of the Host Integrator table being used with this connection.

Database Metadata

For information about the database being used in the current connection, use the DatabaseMetaData object. For Host Integrator, the term 'database' refers to the model being used for the session. You can obtain the DatabaseMetaData object for a current connection by executing the getMetaData method on the current JDBC connection object. for example:

```
DatabaseMetaData DBMetaData = myConnection.getMetaData();
```

With the DatabaseMetaData object, you can find out many things about the Host Integrator model being used. The most important of these are:

Database MetaData Method Name	Description
getColumns	Returns information about table columns available for a table in the Host Integrator model.
getConnection	Gets the connection object that produced the DatabaseMetaData object.
getPrimaryKeys	Returns information about the primary key columns for a table in the Host Integrator model (as designated by the model designer).
getProcedures	Returns information about stored (table) procedures available in the Host Integrator model.
getProcedureColumns	Returns information about the input, output, and results associated with stored (table) procedures available in the Host Integrator model.

Database MetaData Method Name	Description
getTables	Returns information about tables available in the Host Integrator model.

Using ResultSets

Data returned by many of the `java.sql` classes are contained within a `ResultSet` object. In Host Integrator, all `ResultSet` objects contain static data. That is, the data that is fetched from the Host Integrator Server is stored in a `ResultSet` object, but once fetched there is no connection maintained to the server from within the `ResultSet`. This means that the only methods that are implemented in the `ResultSet` are concerned with accessing the data within the `ResultSet`. Conversely, methods that update or modify the `ResultSet` object are not implemented. From a JDBC terminology perspective, these facts mean that Host Integrator `ResultSets` are `CONCUR_READ_ONLY` rather than `CONCUR_UPDATEABLE`.

A `ResultSet` object in Host Integrator is always of type `TYPE_SCROLL_INSENSITIVE`. This means that the result set is scrollable: for instance, its cursor can move forward or backward and can be moved to a particular row or to a row whose position is relative to the current cursor position. As described above, however, the result set does not reflect changes made to the underlying host application. That is, the data in a `ResultSet` object are static - fixed at the time the `ResultSet` object was created.

The following example is a Java method that sequences through a Host Integrator result set returned from a SQL `SELECT` statement and prints each row in the `ResultSet` to the standard output stream.

```

void processRS(ResultSet rs) throws Exception
{
    // First, get the list of columns from ResultSetMetaData
    ResultSetMetaData rsMd = rs.getMetaData();
    ArrayList outputColumns = new ArrayList();
    int cols = rsMd.getColumnCount();

    for(int i = 1; i <= cols; i++) {
        String colName = rsMd.getColumnName(i);
        outputColumns.add(colName);
    }

    // Now sequence through all of the rows in the ResultSet
    int rowIndex=1;
    while(rs.next()) {
        String rowData = new String("Row "+rowIndex+": ");
        Iterator it = outputColumns.iterator();
        while(it.hasNext()) {
            String col = (String)it.next();
            rowData += col+"="+rs.getString(col)+" ";
        }

        // Print the row string
        System.out.println(rowData);
        rowIndex++;
    }
}

```

4.16.6 COM Connector

COM Connector

Use the Host Integrator COM connector to create Visual Basic and C++ applications that integrate host data into web applications or client/server applications. This section describes the COM API methods and provides you with tips on how to build powerful and robust applications by taking advantage of the COM connector's features and options.

The COM connector is installed as a component selection during the basic Host Integrator install program.

Intended Audience This reference is meant for COM and C developers who are familiar with object-oriented development environments and who have a thorough knowledge of Visual Basic, C++, or ASP web programming with scripting languages like JScript or VBScript. In addition, you should become familiar with the Design Tool and Host Integrator Server and the model files associated with them. You can use Web Builder to create projects based on the demo models provided with Verastream Host Integrator and then examine the code in your projects to see how the connector works. To view examples of individual methods, see the Visual Basic methods help file.

[Understanding the AppConn Object Model](#)

[VB Method Reference](#)

UNDERSTANDING THE APPCONN OBJECT MODEL

The Host Integrator COM connector connects to the Host Integrator Server and implements four main interfaces. The AppConnSessionEx object implements these interfaces. Each of the following interfaces contains methods accessible from any programming environment that supports COM:

- AppConnModel interface: Use model-level methods to access host data from the host application model. To use model level methods, you must have knowledge of the Design Tool's modeling process, though an intimate knowledge of the host application is not required.
- AppConnChannel interface: The AppConn COM channel interface contains methods for connecting to and obtaining information about sessions and models running on Verastream Host Integrator Servers.
- AppConnTable interface: Use table/procedure methods to access table definitions of data created during the Design Tool's modeling process and stored in the host application model. Using the methods in this interface requires no knowledge of the host application or the modeling process.
- AppConnTerm interface: Use terminal methods to access data directly from the host. Unlike model-level methods, using terminal methods requires an intimate knowledge of the host application since the entity definitions created in the model file are not used with terminal level methods.

See the [Accessing host data section](#) of Using Host Integrator Connector APIs for more information on these interfaces.

Each of the main interfaces implements certain objects differently and may include additional objects derived from one of the basic objects. The following list describes each basic object and provides links to the corresponding sections of the Visual Basic Reference.

Each of the main interfaces implements certain objects differently and may include additional objects derived from one of the basic objects. The following list describes each basic object. See the [Visual Basic Reference](#).

Object	Description
AppConnRecord	Provides storage for a collection of elements returned from the Host Integrator Server. It may also contain an index number and terminal attributes collection for data returned from the model interface (for example, FetchRecords and GetAttributes). Read-only
AppConnModelRecord	A record that also implements index number and terminal attributes accessor functions using a derived class. The index number and terminal attributes collections are read-only.
AppConnRecordSet	Provides storage for a collection of records and metadata for the elements of the records returned from the Host Integrator Server. Empty recordsets can be created and filled in by the FromXMLString method. Once you execute an insert on a recordset, you need to reset the recordset by either navigating away from it and returning, or executing the home operation on the recordset before executing any other method on it. Read-only.
AppConnStringList	Provides dynamic storage for a list of strings that can be used to send data to, and retrieve data from, an AppConn session.
AppConnStringMap	Provides dynamic storage for a keyed map of strings that can be used to send data to or retrieve data from, an AppConn session. You can also get data from a record.
AttributeMetaData	Contains the metadata defined in the model for an entity attribute. This object is read only.
ColumnMetaData	Contains the metadata defined in the model for a table column. Read-only.
FieldMetaData	Contains the metadata defined in the model for a recordset field. Read-only.

Object	Description
OperationMetaData	Contains the metadata defined in the model for a procedure. Read-only.
ProcedureMetaData	Contains the metadata defined in the model for a procedure. Read-only
RecordSetMetaData	Contains the metadata defined in the model for a recordset. Read-only.
TerminalAttributes	Contains the screen attributes on the host terminal for an attribute or field. Read-only.
VariableMetaData	Contains the metadata defined in the model for a model variable. Read-only
ElementLocationList	Provides storage for a list of element locations. Read-only
TerminalField	Contains the location information for the element. Read-only

Object	Description
StringMapSet	Provides dynamic storage for a list of string maps that can be used to send data to an AppConn session.

USING APPCONN COM FROM VISUAL BASIC

To use AppConn in a Visual Basic project, select References from the Project menu and select AppConn from the list of available references. If AppConn does not appear, use the Browse button to select the appconn.dll from the Windows\System32 folder.

Once AppConn is included in the project, all of the AppConn objects will appear in the Visual Basic IntelliSense. If AppConnLib is selected when using the IntelliSense, a list of the AppConn objects will appear. Objects may be declared with or without the AppConnLib prefix. The following declarations are the same:

```
Dim Session As AppConnTable
Dim Session As AppConnLib.AppConnTable
...

To create an AppConn object, use the new operator: `Set Session = new AppConnTable`

Visual Basic IntelliSense provides a list of properties and methods available for the object. IntelliSense also provides a list of the parameters for methods. The IntelliSense also supplies the parameter type. Optional parameters are enclosed in square brackets (e.g. [varUserID] ).

IntelliSense will also show the return type for methods that have parameters. Variables can be set to object properties and method return values. If the property return type is a COM object, Visual Basic requires using Set; otherwise a simple assignment statement can be used.

``java
Dim Rec As AppConnRecordSet
Dim Index As Long
Dim Connected As Boolean

' Requires Set because method returns a COM object
Set Rec = Session.FetchRecords

' Does not require Set because method returns a long
Long Index = Session.GetCurrentRecordIndex

'Does not require Set because property is a Boolean
Connected = Session.IsConnected
```

Visual Basic requires that properties and methods that return a value use parentheses when parameters are included in the call.

```

' Does not require parentheses because all of the parameters are optional
' and none are included
Set Recs = Session. FetchRecords

' Requires parentheses because a parameter is included
Set Recs = Session. FetchRecords (1)

'Requires parentheses because parameters are included and a return value is used
Set Recs = PerformTableProcedure("Pets", "Get")

' Does not require parentheses because return value is not used
PerformTableProcedure "Pets", "Delete"
...

**Examples**

You can use Web Builder to create projects based on the "demo" models provided with Host Integrator and then examine the code in your
projects to see how the connector works. Examples for individual methods are included with the [Visual Basic Methods Reference](com-vb-
method-reference.md).

**Handling errors**

Errors can be handled by declaring an On Error handler. A handler can be either Resume Next, GoTo 0, or GoTo '<label>'.

Resume Next will ignore the error and continue executing the next statement: `On Error Resume Next`

GoTo 0 will use the default error handler: `On Error GoTo 0`

GoTo '<label>' will cause the program to jump to the label and execute the next statement.

```java
On Error GoTo HandleError
 Session.ConnectToModel "localhost", "CCSDemo"
Exit Sub

ErrorHandler:
 MsgBox Err.Description, vbOKOnly + vbCritical, "Error"
...
```

**Using Variable Type Parameters**

Visual Basic allows variable type parameters. For example, the Item method of the AppConnStringMap can take either a number (the index of an
element) or a string (the key of an element). The method will determine the type of the parameter and return the correct element for that
type.

```java
Dim Attributes As AppConnStringMap
Dim Attribute as String

Set Attributes = new AppConnStringMap 'Create the string map
Attributes.Add("LastName", "Culver") 'Populate the string map
Attribute = Attributes.Item(1) 'Get the first element (VisualBasic indexes start at 1)
Attribute = Attributes.Item("LastName") 'Get the element named "LastName"
```

```

Using Default Methods

Default methods can be defined for COM objects. When the method name is omitted Visual Basic will use the default method. For the AppConnRecordSet, AppConnRecord, AppConnModelRecord, AppConnStringList, and AppConnStringMap the Item method is defined as the default method. The calls to the Item method can be written as follows:

```

Attribute = Attributes(1)
Attribute = Attributes("LastName")

```

When the parameter is a string, the default method can be invoked using the exclamation point (!) and the parameter string without quotes: `Attribute = Attributes!LastName`

The AppConnRecordSet, AppConnRecord, AppConnModelRecord, AppConnStringList, and AppConnStringMap objects are defined as COM collections, which means that VisualBasic can iterate over the collection using the For Each Next construct. In the following example a For Each statement is imbedded inside a For Each statement to iterate over each record in a record set and each field in the record:

```

Dim Rec
Dim Field
Dim Line As String

For Each Rec In Recs
    Line = "Fields:"
    For Each Field In Rec
        Line = Line + " " + Field
    Next
Next

```

C++ WITH AN IMPORT STATEMENT

To begin using AppConn in your C++ project, the AppConn COM control must be imported into the project using the VC++ compiler. To import the appConn.dll, installed to the `\Winnt\System32` folder by default, place the following import statement at the beginning of the source file that will be using the AppConn component:

```
#import "appconn.dll"
```

The import command will generate two new files, appconn.tlh and appconn.tli, which are automatically included in the source file. The appconn.tlh file defines wrapper classes for each of the AppConn interfaces, smart pointers for the wrapper classes, and all of the AppConn constants inside of the namespace AppConnLib. To use the AppConn definitions, the AppConn definition must be prefaced with AppConnLib:: or by including a using namespace statement: `using namespace AppConnLib;`

Accessing AppConn Using the Smart Pointer and Wrapper Class

AppConn objects are accessed using the smart pointer to the wrapper class. The smart pointer will perform all of the life time management for the object. When an object is assigned to the smart pointer, the smart pointer will call AddRef on the object. When the pointer is reassigned, the smart pointer will call Release on the object, and when the smart pointer is destroyed it will call Release on the object. The smart pointer's name is the name of the interface with the suffix "Ptr". For example, declare an interface pointer with the following statement:

```
IAppConnTablePtr pTable;
```

To create an AppConn object, use the smart pointer `CreateInstance` method passing the class Id, which can be obtained using the `__uuidof` keyword with the class name. `CreateInstance` will return an HRESULT, which can be checked for success or failure.

```
hr = pSession.CreateInstance(__uuidof(AppConnTable));
```

The wrapper class creates a wrapper method for each object method. The methods do not return an HRESULT the way COM method normally do, but will throw the exception *com_error* if the call fails, so all calls should be placed inside a try/catch block. The *_com_error* exception contains the *ErrorInfo* for the error that occurred. Optional parameters which are declared as *_variant_t* have a default parameter, so they may be omitted. The wrapper class replaces BSTR parameters with the wrapper class *_bstr_t* and VARIANT parameters with the wrapper class *_variant_t*. *_bstr_t* contains a BSTR and will automatically convert from *char*, *wchar_t*, BSTR, and *_variant_t*. *_bstr_t* will call *SysAllocString* to create the internal string, and will call *SysFreeString* to destroy the string when a new value is assigned or when it is destroyed. *_variant_t* contains a VARIANT and will automatically convert values that can be stored as a VARIANT. *_variant_t* will call *VariantClear* to destroy the variant when a new value is assigned or when it is destroyed. By changing BSTR and VARIANT to the wrapper classes, parameters can be passed to the method as primitive COM pointer or printf type and the compiler will handle conversion and life type management. Methods will return the COM return value directly instead of as the last parameter of the parameter list as COM methods normally do.

```
try {
    pSession->ConnectToModel("localhost", "CCSDemo", "bjones", "bjones");
}
catch (_com_error &err) {
    printf("Error: %s\n", OLE2A((LPOLESTR) err.Description()));
}
```

Return values from the wrapper class method are returned directly to the caller. BSTR values are returned as *_bstr_t*, VARIANT values are returned as *_variant_t*, and pointers to COM interfaces are returned as smart pointers, so life management will be handled automatically by the returning object.

```
_bstr_t bstrString;
try {
    bstrString = pSession->GetStringAtOffset(10, 5);
}
catch (_com_error &err) {
    printf("Error: %s\n", OLE2A((LPOLESTR) err.Description()));
}
```

The wrapper class also declares properties which the compiler treats as data members by changing their references into function calls. The properties can be treated like data members.

```
VARIANT_BOOL bConnect = pSession->IsConnected
```

The wrapper class also declares properties with parameters. In the following example, *Item* is a property of *AppConnStringMap* which handles both get and put with an index that can be either a string or an number. Put with a string will add the key/value if the key does not exist and replace the value if the key does exist. Put with a number will replace the value if the index exists and throw an error if it does not.

```
IAppConnStringMapPtr pAttributes;
pAttributes.CreateInstance(__uuidof(AppConnStringMap));
pAttributes->Item["LastName"] = "S";
bstr_t bstrLastName = pAttributes->Item[1];
```

The following example demonstrates the different ways the wrapper classes can be used to simplify using the AppConn objects. The first line inside the try block creates an instance of AppConnStringList assigned to pSelectFields. Because pSelectFields is declared as a smart pointer life time management will be done automatically and the object will be destroyed when pSelectFields goes out of scope. The second line inside the try block adds an entry to pFilter with the key "LastName" and value "Brown". Because the parameters to Add are declared as _bstr_t in the wrapper class the parameters will automatically be converted from char * to BSTR and cleaned up. The third line inside the try block executes the PerformTableProcedure method which will return an AppConnRecordSet. The value is assigned to pRecords which is a smart pointer.

If the return value is not assigned to a variable the AppConnRecordSet will be destroyed automatically. The first two parameters of PerformTableProcedure are declared as _bstr_t, so AppConn will automatically convert the parameters from char * to BSTR and clean up the values after the method call. The next five parameters are optional so they could be omitted and are declared as _variant_t in the wrapper class. The fourth parameter is included in the method call, so all the parameters up to the fourth parameter must be included.

The third parameter is declared inline as an instance of _variant_t, which will create an empty parameter and be handled as if it is omitted. The fourth parameter is passed as a smart pointer which _variant_t will automatically convert to a VARIANT and clean up the value when it is done. The for loop will iterate over each of the records in the recordset using the Count property declared in the AppConnRecordSet wrapper class to get number of records. The Item property declared in the AppConnRecordSet is used to assign each record to pRecord. The Detach method must be called on the smart pointer before assigning a value to pRecord; otherwise, it will cause an assert.

```

IAppConnRecordSetPtr pRecords;
IAppConnStringMapPtr pFilter;
IAppConnModelRecordPtr pRecord;
try {
    pFilter.CreateInstance(_uuidof(AppConnStringMap));
    pFilter->Add("LastName", "Brown");
    pRecords = pSession->PerformTableProcedure("Person", "GetPerson",
        _variant_t(), pFilter);
    for (long i = 0; i < pRecords->Count; i++) {
        pRecord = pRecords->Item[i];
        pRecord.Detach();
    }
}
catch (_com_error &err) {
    printf("Error: %s\n", OLE2A((LPOLESTR) err.Description()));
}

```

C++ WITHOUT AN IMPORT STATEMENT

Using AppConn COM from C++ without using the import statement provides an efficient way to use the AppConn COM connector without having to use the VC++ compiler. The following topics explain how to begin a C++ project without using an import statement.

Overview

The following files contain the definitions needed to create and use the AppConn controls: appconn_i.c and appconn.h (located in the <VHI install folder>\include folder). The appconn_i.c file contains the AppConn class and interface Ids. This file should be compiled and linked with the program that uses AppConn. The appconn.h file contains the interface definitions for all of the AppConn objects, all of the AppConn constants, and external references to the AppConn class and interface Ids defined in appconn_i.c. This file should be included in source files that use AppConn objects using the following statement:

```
#include "appconn.h"
```

AppConn objects are accessed using an interface pointer to the object. To declare an interface pointer, use the following statement:

```
IAppConnModel *pSession;
```

Creating an AppConn Object

To create an AppConn object, use the CoCreateInstance method, passing the class Id, the aggregate object's IUnknown interface, the class context, the interface Id, and a pointer to the interface pointer. CoCreateInstance will return an HRESULT, which can be checked for success or failure.

```
HRESULT hr = CoCreateInstance(CLSID_AppConnModel,  
    NULL,  
    CLSCTX_INPROC_SERVER,  
    IID_IAppConnModel,  
    (LPVOID *)&pSession);
```

Object lifetimes must be managed explicitly. When an object is assigned to another interface pointer AddRef must be called on the interface, and when the interface is no longer used Release must be called. When the reference count goes to 0 the object will be deleted.

```
IAppConnModel *pModel = pSession;  
pModel->AddRef();  
pSession->Release();  
pSession = NULL;
```

Converting C++ Types Into COM Types

Calling AppConn methods may require converting C or C++ types into COM types. Strings must be converted to BSTR using SysAllocString to create the string and SysFreeString to destroy it. bool must be converted to VARIANT_BOOL and set to either VARIANT_FALSE (0) or VARIANT_TRUE (-1). Variant parameters must be initialized using VariantInit and destroyed using VariantClear. Optional parameters are declared as VARIANT and may be set as VT_EMPTY or VT_NULL to indicate that the parameter is not being included. Return parameters are returned to the caller as a pointer to an object in the last parameter of the method. All methods return an HRESULT.


```

BSTR bstrServer;
BSTR bstrModel;
VARIANT varUserID;
VARIANT varPassword;
VARIANT varModelVariables;

bstrServer = SysAllocString(L"localhost");
bstrModel = SysAllocString(L"CCSDemo");
VariantInit(&varUserID); //Will set VARIANT to VT_EMPTY
VariantInit(&varPassword); //Will set VARIANT to VT_EMPTY
VariantInit(&varModelVariables); //Will set VARIANT to VT_EMPTY
hr = pSession->ConnectToModel(
    bstrServer,
    bstrModel,
    varUserID,
    varPassword,
    varModelVariables);
SysFreeString(bstrServer);
SysFreeString(bstrModel);
VariantClear(&varUserID);
VariantClear(&varPassword);
VariantClear(&varModelVariables);
);
...

```

When a method call fails, the HRESULT may contain a standard Windows error code or a custom AppConn error code. The AppConn error codes are as follows:

```

...java
APPTRIEVE_ERROR = 0x801E0001;
CHANNEL_ERROR = 0x801E0002;
SERVER_ERROR = 0x801E0003;
MARSHALLER_ERROR = 0x801E0004;
TIMEOUT_ERROR = 0x801E0005;
MODEL_DATA_ERROR = 0x801E0006;
MODEL_DEF_ERROR = 0x801E0007;
TERMINAL_ERROR = 0x801E0008;
DEAD_SESSION_ERROR = 0x801E0009;
USER_EXCEPTION_ERROR = 0x801E000A;
LOCKING_ERROR = 0x801E000B;
ALREADY_CONNECTED_ERROR = 0x801E000C;
NOT_CONNECTED_ERROR = 0x801E000D;
INVALID_ARG_ERROR = 0x801E000E;
INVALID_POINTER_ERROR = 0x801E000F;
BAD_VAR_TYPE_ERROR = 0x801E0010;
OUT_OF_MEMORY_ERROR = 0x801E0011;
ELEMENT_DOES_NOT_EXIST_ERROR = 0x801E0012;
ELEMENT_ALREADY_EXISTS_ERROR = 0x801E0013;
UNKNOWN_ERROR = 0x801E0014;

```

Error Handling

To get more specific error information, retrieve the `ErrorInfo` object. The `ErrorInfo` object is a standard COM object that can be retrieved by calling `GetErrorInfo`. `ErrorInfo` is stored in thread local storage, so there can only be one `ErrorInfo` object. `AppConn` will set the `ErrorInfo` when an error occurs and clear the `ErrorInfo` on the next call to an `AppConn` object. The `ErrorInfo` object contains a textual description of the error that can be accessed with the `GetDescription` method.

```

IErrorInfo *pErrorInfo;
LPOLESTR strDescription;
if (SUCCEEDED(GetErrorInfo(0, &pErrorInfo)))
{
    pErrorInfo->GetDescription(&strDescription);
    printf("Error: %s\n", OLE2A((LPOLESTR) strDescription));
    SysFreeString(strDescription);
    pErrorInfo->Release();
}

```

USING ASP WITH VBSCRIPT

VBScript works the same as Visual Basic except for some notable exceptions:

Types cannot be declared in VBScript with the `As` keyword.

Objects are not created with the `new` operator, but with `CreateObject` method.

```

Set Session = Server.CreateObject ("VeraStream.AppConnTable") 'Create an instance in ASP
Set Session = Wscript.CreateObject ("VeraStream.AppConnTable") 'Create an instance in vbs engine

```

Error Handling

On Error Goto Label does not work in VBScript, so error handling is done inline by checking the Err object:

```
Session.ConnectToModel "localhost", "CCSDemo"  
If Err.Number <> 0 Then  
    'Perform error handling  
End If
```

VB Method Reference

AppConn COM Visual Basic Methods Reference

Click an object below to see the methods it implements. For an overview, see [Appconn COM object model](#).

Objects

[AttributeMetaData](#)

[ColumnMetaData](#)

[ElementLocation](#)

[ElementLocationList](#)

[ErrorInfo](#)

[FieldMetaData](#)

[MetaData](#)

[ModelRecord](#)

[OperationMetaData](#)

[ProcedureMetaData](#)

[Record](#)

[RecordSet](#)

[RecordSetMetaData](#)

[SessionEx](#)

[StringList](#)

[StringMap](#)

[StringMapSet](#)

[TerminalAttributes](#)

[Terminal Field](#)

[VariableMetaData](#)

ATTRIBUTEMETADATA

Use the AttributeMetaData object to manage attribute metadata.

Click a method to see more information on its use, syntax, and parameters:

Description
IsReadable
IsWriteable
Length
MetaDataType
Name
ReadVariable
TerminalAttributesEnabled
WriteVariable

DESCRIPTION

Used to get the readable flag of an attribute.

Use the Description property to get the description of an attribute defined within the model file.

Syntax

```
object.Description
```

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("Description " & objMetaData.Description)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

ISREADABLE

Used to get the readable flag of an attribute. Use the IsReadable property to get the readable flag of an attribute

Syntax

```
object.IsReadable
```

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("IsReadable " & objMetaData.IsReadable)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing

```

ISWRITEABLE PROPERTY

Used to get the writeable of an attribute. Use the IsWriteable property to get the writeable flag of an attribute.

Syntax

```
object.IsWriteable
```

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("IsWriteable " & objMetaData.IsWriteable)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing

```

LENGTH PROPERTY

Used to get the length of an attribute.

Syntax

```
object.Length
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remarks

Use the Length property to get the length of an attribute.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("Length " & objMetaData.Length)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing

```

METADATATYPE PROPERTY

Used to get the metadata type of an attribute.

Syntax

```
object.MetaDataType
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remark

Use the MetaDataType property to get the type of metadata within a Verastream table. The enumeration values for AppConnMetaData are defined in the Type Library.

| Metadata Types |
|----------------|
| AttributeMeta |
| OperationMeta |
| RecordSetMeta |
| FieldMeta |
| VariableMeta |
| TableMeta |
| ColumnMeta |
| |

Metadata Types

ProcedureMeta

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("MetaDataType " & objMetaData.MetaDataType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

NAME PROPERTY

Used to get the name of an attribute.

Syntax

```
object.Name
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remarks

Use the Name property to get the name of an attribute.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("Name " & objMetaData.Name)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

READVARIABLE PROPERTY

Used to get the read variable of an attribute.

Syntax

```
object.ReadVariable
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remarks

Use the ReadVariable property to get the read variable of an attribute. The ReadVariable is the model variable that gets updated when executing a ReadFromMappedAttr command in an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("ReadVariable " & objMetaData.ReadVariable)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

TERMINALATTRIBUTESENABLED PROPERTY

Used to get the terminal attributes enabled flag of an attribute.

Syntax

```
object.TerminalAttributesEnabled
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remarks

Use the TerminalAttributesEnabled property to get the terminal attributes enabled flag of an attribute. The terminal attributes of an attribute are things like color, IsBlinking etc.

Example


```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaDat

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("TerminalAttributesEnabled " & objMetaData.TerminalAttributesEnabled)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing

```

WRITEVARIABLE PROPERTY

Used to the write variable of an attribute.

Syntax

```
object.WriteVariable
```

| Part | Description |
|--------|-----------------------------|
| object | An AttributeMetaData object |

Remarks

Use the WriteVariable property to get the write variable of an attribute. The WriteVariable is the model variable that a value is gotten from when the WriteToMappedAttrf command is called within an operation.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("WriteVariable " & objMetaData.WriteVariable)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing

```

COLUMNMETADATA OBJECT

Use the AppConnMetaData object to manage data based in table columns.

Method use, syntax, and parameters:

ColumnType
Description
IsKey
MetaDataType
Max
Min
Name

COLUMNTYPE PROPERTY

Used to get the type of a column within a Verastream table.

Syntax

```
object.ColumnType
```

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the ColumnType property to get the type of a column. The enumeration values for AppConnColumnType are defined in the Type Library.

| Column Type | ColumnType Return Value |
|----------------|-------------------------|
| Integer Column | 0 |
| Text Column | 1 |

| Column Type | ColumnType Return Value |
|--------------|-------------------------|
| Float Column | 2 |

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column type = " & ColumnMeta.ColumnType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

DESCRIPTION PROPERTY

Used to get the description of a column within a Verastream table.

Syntax

```
object.Description
```

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the Description property to get the description of a column within a Verastream table.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column description = " & ColumnMeta.Description)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

ISKEY PROPERTY

Used to get the key flag of a column within a Verastream table.

Syntax

```
object.IsKey
```

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the IsKey property to get the key flag of a column within a Verastream table. It returns a boolean value about whether the column is a key field or not.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Is the column a key for the table? " & ColumnMeta.IsKey)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

MAX PROPERTY

Used to get the maximum value of a column within a Verastream table.

Syntax

```
object.Max
```

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the Max property to get the maximum value of a column within a Verastream table. The meaning of the max value depends on the type of the column. For integer and float columns, it represents the maximum value for the column. For string columns, it represents the maximum number of characters in the string.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column maximum = " & ColumnMeta.Max)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing

```

METADATATYPE PROPERTY

Used to get the metadata type of a column within a Verastream table.

Syntax

```
object.MetadataType
```

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the MetadataType property to get the type of metadata within a Verastream table. The enumeration values for AppConnMetadataType are defined in the Type Library.

| Metadata Types |
|----------------|
| AttributeMeta |
| OperationMeta |
| RecordSetMeta |
| FieldMeta |
| VariableMeta |
| TableMeta |
| ColumnMeta |
| |

Metadata Types

ProcedureMeta

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Metadata type = " & ColumnMeta.MetaDataType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

MIN PROPERTY

Used to get the minimum value of a column within a Verastream table.

Syntax

object.Min

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the Min property to get the minimum value of a column within a Verastream table. The meaning of the min value depends on the type of the column. For integer and float columns, it represents the minimum value for the column. For string columns, it represents the minimum number of characters in the string.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column minimum = " & ColumnMeta.Min)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

NAME PROPERTY

Used to get the name of a column within a Verastream table.

Syntax

object.Name

| Part | Description |
|--------|-------------------------|
| object | A ColumnMetaData object |

Remarks

Use the Min property to get the minimum value of a column within a Verastream table. The meaning of the min value depends on the type of the column. For integer and float columns, it represents the minimum value for the column. For string columns, it represents the minimum number of characters in the string.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ColumnMeta As ColumnMetaData

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column name = " & ColumnMeta.Name)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ColumnMeta = Nothing
```

ELEMENTLOCATION OBJECT

ElementLocation Object

Use the ElementLocation object to obtain information from element location.

Click a method to see more information on its use, syntax, and parameters:

[ElementType](#)

[LeftColumn](#)

[Length](#)

[Name](#)

[NumColumns](#)

[NumRows](#)

[Offset](#)

[RegionType](#)

[TopRow](#)

ELEMENTTYPE PROPERTY

Used to get the type of the element.

Syntax

```
object.ElementType
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the ElementType property to get the type of a element. The enumeration values for AppConnElementType are defined in the Type Library.

| Element Type |
|--------------|
| Attribute |
| Field |
| Pattern |
| |

Element Type

RecordSet

LEFTCOLUMN PROPERTY

Used to get the location of the left column of the element.

Syntax

```
object.LeftColumn
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the LeftColumn property to get the location of the left column of an element that is defined as a rectangular region. If the element is defined as a linear region the value will be -1.

LENGTH PROPERTY

Used to get the length of the element.

Syntax

```
object.Length
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the Length property to get the length for an element that is defined as a linear region. If the element is defined as a rectangular region the value will be -1.

NAME PROPERTY

Used to get the name of the element.

Syntax

```
object.Name
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the Name property to get the name of an element.

NUMCOLUMNS PROPERTY

Used to get the number of columns of the element.

Syntax

```
object.NumColumns
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the NumColumns property to get the number of columns for an element that is defined as a rectangular region. If the element is defined as a linear region the value will be -1.

NUMROWS PROPERTY

Used to get the number of rows for the element.

Syntax

```
object.NumRows
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the NumRows property to get the number of rows for an element that is defined as a rectangular region. If the element is defined as a linear region the value will be -1.

OFFSET PROPERTY

Used to get the offset of the element.

Syntax

`object.Offset`

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the Offset property to get the offset for an element that is defined as a linear region. If the element is defined as a rectangular region the value will be -1.

REGIONTYPE PROPERTY

Used to get the type of region for the element.

Syntax

`object.RegionType`

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the RegionType property to get the type of region. The enumeration values for AppConnRegionType are defined in the Type Library.

| Region Type |
|-------------|
| Linear |
| |

Region Type

Rectangular

TOPROW PROPERTY

Used to get the location of the top row flag of the element.

Syntax

```
object.TopRow
```

| Part | Description |
|--------|----------------------------|
| object | An element location object |

Remarks

Use the TopRow property to get the location of the top row of an element that is defined as a rectangular region. If the element is defined as a linear region the value will be -1.

ELEMENTLOCATIONLIST OBJECT

ElementLocationList Object

Click a method to see more information on its use, syntax, and parameters:

[Count](#)

[Item](#)

COUNT PROPERTY

Used to get the number of elements in the element location list.

Syntax

```
object.Count
```

| Part | Description |
|--------|--------------------------------|
| object | A element location list object |

Remarks

Use the Count property to get the number of elements in a element locationlist.

ITEM PROPERTY

Used to get an element in a element locationlist.

Syntax

```
object.Item(Index)
```

```
object.(Index)
```

| Part | Description |
|--------|--------------------------------|
| object | A element location list object |
| | |

| Part | Description |
|-------|---|
| Index | The name or index (starting at 1) of an element in the list |

Remarks

Use the Item property to get an element in a element location list.

ERRORINFO OBJECT

VSErrorInfo Object

Use the VSErrorInfo object to retrieve error information from the Appconn connector.

Click any of the following methods or properties to see information on its use, syntax, and parameters:

[ErrorCodeArray](#)

[ErrorMessageArray](#)

[ErrorParameterArray Method](#)

ERRORCODEARRAY

Returns an array of error message ID's (RTE numbers.)

Syntax

```
object.ErrorCodeArray
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

ERRORMESSAGEARRAY

Returns an array of error message strings

Syntax

```
object.ErrorMessageArray
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

ERRORPARAMETERARRAY METHOD

Syntax

```
object.ErrorParameterArray ErrorMessageIndex
```

Syntax

```
object.ErrorMessageArray
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| | |

| Part | Description |
|-------------------|---|
| ErrorMessageIndex | 1-based index of the error message for which parameters are to be obtained. |

Return value: Array of parameter strings.

Data Type: Integer

FIELDMETADATA OBJECT

FieldMetaData Object

Use the IFieldMetaData object to manage fields.

Click a method to see more information on its use, syntax, and parameters:

[IsKey](#)

[IsReadable](#)

[IsWriteable](#)

[Length](#)

[TerminalAttributesEnabled](#)

ISKEY PROPERTY

Used to get the key flag of a field.

Syntax

```
object.IsKey
```

| Part | Description |
|--------|-------------------------|
| object | A field metadata object |

Remarks

Use the IsKey property to get the key flag of a field. A key value is unique for the associated field in the recordset—no other records will have the same value for that field.

ISREADABLE PROPERTY

Used to get the readable flag of a field.

Syntax

```
object.IsReadable
```


| Part | Description |
|--------|-------------------------|
| object | A field metadata object |

Remarks

Use the `IsReadable` property to get the readable flag of a field, which indicates whether the field can be read by a client.

ISWRITEABLE PROPERTY

Used to get the writeable flag for the field.

Syntax

```
object.IsWriteable
```

| Part | Description |
|--------|-------------------------|
| object | A field metadata object |

Remarks

Use the `IsWriteable` property to get the writeable flag of a field, which indicates whether Host Integrator allows this field to have data written to it.

LENGTH PROPERTY

Used to get the length of a field.

Syntax

```
object.Length
```

| Part | Description |
|--------|-------------------------|
| object | A field metadata object |

Remarks

A return value of -1 indicates that the field length is variable.

TERMINALATTRIBUTESENABLED PROPERTY

Used to get the terminal attributes enabled flag of a field.

Syntax

`object.TerminalAttributesEnabled`

| Part | Description |
|--------|-------------------------|
| object | A field metadata object |

Remarks

Gets the flag for the field that specifies whether terminal attributes are enabled. The terminal attributes enabled flag indicates whether Host Integrator can return the terminal attributes of the field. Terminal attributes specify various properties the text can have, including color, whether the text is blinking, and whether the text is displayed in reverse video.

METADATA OBJECT

AppConnMetaData Interface

Use the AppConnMetaData interface to manage model metadata.

Click a method to see more information on its use, syntax, and parameters:

[Description](#)

[MetaDataType](#)

[Name](#)

DESCRIPTION PROPERTY

Used to get the description of a metadata object.

Syntax

`object.Description`

| Part | Description |
|--------|-------------------|
| object | A metadata object |

Remarks

Use the Description property to get the description of a metadata object.

METADATATYPE PROPERTY

Used to get the type of the metadata.

Syntax

`object.RecordType`

| Part | Description |
|-------------|--------------------|
| object | A metadata object |

Remarks

Use the `MetaDataType` property to get type of a metadata object. The enumeration values for `AppConnMetaData` are defined in the Type Library.

| Metadata Types |
|-----------------------|
| AttributeMeta |
| OperationMeta |
| RecordSetMeta |
| FieldMeta |
| VariableMeta |
| TableMeta |
| ColumnMeta |
| |

Metadata Types

ProcedureMeta

NAME PROPERTY

Used to get the name of a metadata object.

Syntax

```
object.Name
```

| Part | Description |
|--------|-------------------|
| object | A metadata object |

Remarks

Use the Name property to get the name of a metadata object.

MODELRECORD OBJECT

AppConnModelRecord Object

Use the AppConnModelRecord object to manage model records.

Click a method to see more information on its use, syntax, and parameters:

[Count](#)

[ElementNames](#)

[FromXMLString](#)

[GetTerminalAttributes](#)

[Index](#)

COUNT METHOD

Used to get the numbers of records in the AppConnModelRecord object.

Syntax

```
object.Count
```

| Part | Description |
|--------|------------------------------|
| object | An AppConnModelRecord object |

| Part | Description |
|-------|--|
| Count | The number of records in the AppConnModelRecord object |

Remarks

Use the Count method to get the numbers of records in the AppConnModelRecord object.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeRecord As AppConnModelRecord
Dim strModelName, strServerName, strEntity, strMessage As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.SetCurrentEntity (strEntity)

Set AttributeRecord = New AppConnModelRecord
Set AttributeRecord = Verastream_Session.GetAttributes

strMessage = "The number of records is " & AttributeRecord.Count
MsgBox strMessage, vbExclamation, "Verastream Message"

Set AttributeRecord = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

ELEMENTNAMES PROPERTY

Used to get an AppConnStringList object with the element names on the entity.

Syntax

```
Set AttributeList = object.ElementNames
```

| Part | Description |
|---------------|-----------------------------|
| AttributeList | An AppConnStringList object |

| Part | Description |
|--------|------------------------------|
| object | An AppConnModelRecord object |

Remarks

Used to get an AppConnStringList object with the element names on the entity so you can further manipulate them with the AppConnStringList methods.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeRecord As AppConnModelRecord
Dim strModelName, strServerName, strEntity, strMessage As String
Dim AttributeList As AppConnStringList
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "MainMenu"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.SetCurrentEntity (strEntity)

Set AttributeRecord = New AppConnModelRecord
Set AttributeRecord = Verastream_Session.GetAttributes

Set AttributeList = AttributeRecord.ElementNames

For I = 1 To AttributeRecord.Count
MsgBox ("Attribute " & AttributeList(I) & " = " & AttributeRecord(I))
Next

Set AttributeRecord = Nothing
Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

FROMXMLSTRING METHOD

Used to set a record from an XML representation.

Syntax

```
object.FromXMLString(Record)
```

| Part | Description |
|--------|------------------------------|
| object | An AppConnModelRecord object |

| Part | Description |
|--------|-------------------------------------|
| Record | An XML representation of the record |

Remarks

Use the FromXMLString method to set a record from an XML representation.

GETTERMINALATTRIBUTES METHOD

Used to get the host terminal attributes from a record.

Syntax

```
Set TerminalAttributes = object.GetTerminalAttributes(Index)
```

| Part | Description |
|--------|--|
| object | An AppConnModelRecord object |
| Index | The name or index(starting at 1) of an element in the list |

| Part | Description |
|--------------------|---|
| TerminalAttributes | The terminal attributes for the given element |

Remarks

Use the GetTerminalAttributes method to get the terminal attributes of an element in a record.

INDEX PROPERTY

Used to get the Host Integration recordset index of record.

Syntax

object.Index

| Part | Description |
|--------|-----------------|
| object | A record object |

Remarks

Use the Index property to get the index of the record.

OPERATIONMETADATA OBJECT

OperationMetaData Object

Use the OperationMetaData object to manage operation metadata.

Click a method to see more information on its use, syntax, and parameters:

[AltDestinations](#)

[AttributesUsed](#)

[Description](#)

[Destination](#)

[IsDefault](#)

[MetaDataType](#)

[Name](#)

[Timeout](#)

[VariablesUsed](#)

ALTDESTINATIONS PROPERTY

Used to get an AppConnStringList of the alternate destinations of an operation.

Syntax

```
Set Destinations = object.AltDestinations
```

| Part | Description |
|--------------|--|
| object | An operation metadata object |
| Destinations | An AppConnStringList of alternate destinations |

Remarks

Use the AltDestination property to get an AppConnStringList of alternate destinations of an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData
Dim Destinations As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

Set Destinations = objMetaData.AltDestinations
For I = 1 To Destinations.Count
MsgBox ("Alternate destination: " & Destinations(I))
Next
Set Destinations = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

ATTRIBUTESUSED PROPERTY

Used to get an AppConnStringList of the attributes used for an operation.

Syntax

```
Set Attributes = object.AttributesUsed
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|------------|--|
| Attributes | An AppConnStringList of attributes used for an operation |

Remarks

Use the AttributesUsed property to get an AppConnStringList of attributes used for an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData
Dim AttributesUsed As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)
Set AttributesUsed = objMetaData.AttributesUsed
For I = 1 To AttributesUsed.Count
MsgBox ("Attributes used: " & AttributesUsed(I))
Next
Set AttributesUsed = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

DESCRIPTION PROPERTY

Used to get the description of an operation.

Syntax

```
object.Description
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the Description property to get the description of an operation.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)
MsgBox ("Operation metadata Description is " & objMetaData.Description)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing

```

DESTINATION PROPERTY

Used to get the destination of an operation.

Syntax

```
object.Destination
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the Destination property to get the primary destination of an operation.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

MsgBox ("Operation metadata Destination is " & objMetaData.Destination)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing

```

ISDEFAULT PROPERTY

Method used to get the default flag of the operation.

Syntax

```
object.IsDefault
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the IsDefault property to get the default flag of an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
strOperationName)

MsgBox ("Operation metadata IsDefault is " & objMetaData.IsDefault)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

METADATATYPE PROPERTY

Method used to get the type of metadata the operation is.

Syntax

```
object.MetadataType
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the MetadataType property to get the type of metadata within a Verastream table. The enumeration values for AppConnMetadataType are defined in the Type Library.

| Metadata Types |
|----------------|
| AttributeMeta |
| OperationMeta |
| RecordSetMeta |
| FieldMeta |
| VariableMeta |

| |
|-----------------------|
| Metadata Types |
| TableMeta |
| ColumnMeta |
| |

Metadata Types

ProcedureMeta

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData
Dim Destinations As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

MsgBox (objMetaData.MetaDataType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

NAME PROPERTY

Method used to get the name of the operation.

Syntax

```
object.Name
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the Name property to get the name an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)
MsgBox ("Operation metadata name is " & objMetaData.Name)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

TIMEOUT PROPERTY

Used to get the operation timeout.

Syntax

```
object.Timeout
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

Remarks

Use the Timeout property to get the timeout duration (seconds) for an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

MsgBox ("Operation metadata Timeout is " & objMetaData.Timeout)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

VARIABLESUSED PROPERTY

Used to get an AppConnStringList of the variables used for an operation.

Syntax

```
Set Variables = object.VariablesUsed
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|----------|---|
| Variable | An AppConnStringList of variables used for an operation |

Remarks

Use the VariablesUsed property to get an AppConnStringList of variables used for an operation.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData
Dim VariablesUsed As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

Set VariablesUsed = objMetaData.VariablesUsed
For I = 1 To VariablesUsed.Count
    MsgBox ("Variables used: " & VariablesUsed(I))
Next
Set VariablesUsed = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
Set VariablesUsed = Nothing
```

PROCEDUREMETADATA OBJECT

ProcedureMetaData Object

Use the ProcedureMetaData object to manage procedure metadata.

Click a method to see more information on its use, syntax, and parameters:

[FilterColumns](#)

[InputColumns](#)

[IsRequiredFilter](#)

[IsRequiredInput](#)

[MetaDataType](#)

[OutputColumns](#)

[ProcedureType](#)

[UsedForSQL](#)

FILTERCOLUMNS PROPERTY

Used to get an AppConnStringList of the filter columns of a procedure.

Syntax

```
Set FilterColumns = object.FilterColumns
```

| Part | Description |
|---------------|--|
| object | A ProcedureMetaData object |
| FilterColumns | An AppConnStringList of filter columns for a procedure |

Remarks

Use the FilterColumns property to get an AppConnStringList of filter columns for a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
"GetTransactions")

Set ProcStringList = ProcMetaData.FilterColumns
For I = 1 To ProcStringList.Count
MsgBox ("Filter column " & I & " is " & ProcStringList(I))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

INPUTCOLUMNS PROPERTY

Used to get an AppConnStringList of the input columns of a procedure.

Syntax

```
Set InputColumns = object.InputColumns
```

| Part | Description |
|--------|----------------------------|
| object | A ProcedureMetaData object |

| Part | Description |
|--------------|--|
| InputColumns | An AppConnStringList of input columns for a procedure. |

Remarks

Use the InputColumns property to get an AppConnStringList of input columns for a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
"GetTransactions")

Set ProcStringList = ProcMetaData.InputColumns
For I = 1 To ProcStringList.Count
MsgBox ("Input column " & I & " is " & ProcStringList(I))
MsgBox ("Is " & ProcStringList(I) & " required?" & _
ProcMetaData.IsRequiredInput(ProcStringList(I)))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

ISREQUIREDFILTER METHOD

Used to determine if a column is required filter for a procedure.

Syntax

```
IsRequiredFilter = object.IsRequiredFilter(ColumnName)
```

| Part | Description |
|------------|-----------------------------|
| object | A ProcedureMetaData object |
| ColumnName | The string name of a column |

| Part | Description |
|------------------|--|
| IsRequiredFilter | Boolean indicator whether the column is required for filter or not |

Remarks

Use the IsRequiredFilter method to determine if a column is required filter for a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

Set ProcStringList = ProcMetaData.FilterColumns
For I = 1 To ProcStringList.Count
MsgBox ("Filter column " & I & " is " & ProcStringList(I))
MsgBox ("Is " & ProcStringList(I) & " required?" & _
    ProcMetaData.IsRequiredFilter(ProcStringList(I)))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

ISREQUIREDINPUT METHOD

Used to determine if a column is required input for a procedure.

Syntax

```
IsRequiredInput = object.IsRequiredInput(ColumnName)
```

| Part | Description |
|------------|-----------------------------|
| object | A ProcedureMetaData object |
| ColumnName | The string name of a column |

| Part | Description |
|-----------------|---|
| IsRequiredInput | Boolean indicator whether the column is required for input or not |

Remarks

Use the IsRequiredInput method to determine if a column is required input for a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

Set ProcStringList = ProcMetaData.InputColumns
For I = 1 To ProcStringList.Count
    MsgBox ("Input column " & I & " is " & ProcStringList(I))
    MsgBox ("Is " & ProcStringList(I) & " required? " & _
        ProcMetaData.IsRequiredInput(ProcStringList(I)))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

METADATATYPE PROPERTY

Used to get the type of metadata for the procedure.

Syntax

```
object.MetadataType
```

| Part | Description |
|--------|----------------------------|
| object | A ProcedureMetaData object |

Remarks

Use the MetadataType property to get the type of metadata. The enumeration values for AppConnProcedureType are defined in the Type Library.

| Metadata Types |
|----------------|
| AttributeMeta |
| OperationMeta |
| RecordSetMeta |

| Metadata Types |
|----------------|
| FieldMeta |
| VariableMeta |
| TableMeta |
| ColumnMeta |
| ProcedureMeta |

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

MsgBox ("Metadata type is " & ProcMetaData.MetaDataType)

Verastream_Session.Disconnect
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

OUTPUTCOLUMNS PROPERTY

Used to get an AppConnStringList of the output columns of a procedure.

Syntax

```
Set OutputColumns = object.OutputColumns
```

| Part | Description |
|--------|----------------------------|
| object | A ProcedureMetaData object |

| Part | Description |
|---------------|--|
| OutputColumns | An AppConnStringList of output columns for a procedure |

Remarks

Use the OutputColumns property to get an AppConnStringList of output columns for a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

Set ProcStringList = ProcMetaData.OutputColumns
For I = 1 To ProcStringList.Count
    MsgBox ("Output column " & I & " is " & ProcStringList(I))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

PROCEDURETYPE PROPERTY

Used to get the type of a procedure.

Syntax

```
object.ProcedureType
```

| Part | Description |
|--------|----------------------------|
| object | A ProcedureMetaData object |

Remarks

Use the Procedure property to get the type of a procedure. The enumeration values for AppConnProcedureType are defined in the Type Library.

| Return Values | Procedure Types |
|---------------|-----------------|
| 0 | DeleteProcedure |
| 1 | UpdateProcedure |
| 2 | SelectProcedure |

| Return Values | Procedure Types |
|---------------|-----------------|
| 3 | InsertProcedure |

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

MsgBox ("Procedure type is " & ProcMetaData.ProcedureType)

Verastream_Session.Disconnect
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

USEDFORSQL PROPERTY

Used to get the used for SQL flag of a procedure.

Syntax

```
object.UsedForSQL
```

| Part | Description |
|--------|----------------------------|
| object | A ProcedureMetaData object |

Remarks

Use the UsedForSQL property to get the used for SQL flag of a procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

MsgBox ("Is the procedure used for SQL? " & ProcMetaData.UsedForSQL)

Verastream_Session.Disconnect
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

RECORD OBJECT

AppConnRecord Object

Use the AppConnRecord object to manage model records.

Click a method to see more information on its use, syntax, and parameters:

[Count](#)

[ElementNames](#)

[FromXMLString](#)

[GetElements](#)

[Item](#)

[RecordType](#)

[ToXMLString](#)

COUNT PROPERTY

Used to get the number of elements in the record.

Syntax

```
object.Count
```

| Part | Description |
|--------|-------------------------|
| object | An AppConnRecord object |

Remarks

Use the Count property to get the number of elements in a record.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Set modRecordSet = Verastream_Session.FetchRecords
Set modRecord = modRecordSet.Item(1)
MsgBox ("Record count = " & modRecord.Count)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing
```

ELEMENTNAMES PROPERTY

Method used to get a list of the names of the elements in the record.

Syntax

```
object.ElementNames
```

| Part | Description |
|--------|-------------------------|
| object | An AppConnRecord object |

Remarks

Use the ElementNames property to get a list of the names of the elements in a record.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Attributes.Clear

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Set modRecordSet = Verastream_Session.FetchRecords
Set modRecord = modRecordSet.Item(1)
For I = 1 To modRecord.Count
    MsgBox ("Record element names are " & modRecord.ElementNames(I))
Next

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing
```

FROMXMLSTRING METHOD

Used to set a record from an XML representation.

Syntax

```
object.FromXMLString(Record)
```

| Part | Description |
|--------|-------------------------|
| object | An AppConnRecord object |

| Part | Description |
|--------|-------------------------------------|
| Record | An XML representation of the record |

Remarks

Use the FromXMLString method to set a record from an XML representation.

GETELEMENTS METHOD

Used to get a stringmap of the elements in the record.

Syntax

```
Set StringMap = object.GetElements
```

| Part | Description |
|--------|-------------------------|
| object | An AppConnRecord object |

| Part | Description |
|-----------|--|
| StringMap | A AppConnStringMap of elements in name/value pairs |

Remarks

Use the GetElements method to get a string map of the elements in a record. This function can be used when updating a record by getting the elements of the record, updating one or more fields, and using the elements in an update record function.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName`

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Attributes.Clear

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Set modRecordSet = Verastream_Session.FetchRecords
Set modRecord = modRecordSet.Item(1)
Set Attributes = modRecord.GetElements
MsgBox ("Record element 'Date' value is = " & Attributes.Item("Date"))
MsgBox ("Record element 'Date' value is = " & Attributes("Date"))
MsgBox ("Record element 'Date' value is = " & Attributes!Date)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing
```

ITEM PROPERTY

Used to get an element in a record.

Syntax

object.Item(Index)

object.Item("Name")

object(Index)

object("Name")

object!Name

| Part | Description |
|-------------|---|
| object | An AppConnRecord object |
| Index | The index (starting at 1) of an element within the record. The index is the location in the record of the field relative to the other fields. |
| | |

| Part | Description |
|------|--------------------------------------|
| Name | The name of an element in the record |

Remarks

Use the Item property to get an element in a record.

Example

```
Dim Verastream_Session As AppConnModel
Dim I As Integer
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Dim resultsStr As String
resultsStr = ""
Set modRecordSet = Verastream_Session.FetchRecords
For I = 1 To modRecordSet.Count
    Set modRecord = modRecordSet.Item(I)
    resultsStr = resultsStr & vbNewLine & modRecord.Item("Date")
Next
MsgBox resultsStr, vbOKOnly, "Dates"

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing
```

RECORDTYPE PROPERTY

Used to get the type of the record.

Syntax

```
object.RecordType
```

| Part | Description |
|--------|-------------------------|
| object | An AppConnRecord object |

Remarks

Use the RecordType property to get type of a record. The enumeration values for AppConnRecordType are defined in the Type Library.

| Return Value | Record Types |
|--------------|--------------|
| 0 | ModelRecord |
| 1 | TableRecord |

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Attributes.Clear

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Set modRecordSet = Verastream_Session.FetchRecords
Set modRecord = modRecordSet(1)
MsgBox (" The record type is " & modRecord.RecordType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing

```

TOXMLSTRING METHOD

Used to get an XML representation of the record.

Syntax

```
XMLString = object.ToXMLString([URLDTD])
```

| Part | Description |
|--------|---|
| object | An AppConnRecord object |
| URLDTD | [optional] The URL of a DTD that will be used to validate the XML |

| Part | Description |
|-----------|--------------------------------------|
| XMLString | An XML representation of the record. |

Remarks

Use the ToXMLString method to get an XML representation of the record.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim modRecord As AppConnRecord
Dim modRecordSet As AppConnRecordSet
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Attributes.Clear

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

Set modRecordSet = Verastream_Session.FetchRecords
Set modRecord = modRecordSet(1)

MsgBox (" The record in XML format is = " & modRecord.ToXMLString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set modRecord = Nothing
Set modRecordSet = Nothing
Set Attributes = Nothing
```

RECORDSET OBJECT

AppConnRecordSet Object

Use the AppConnRecordSet object to manage recordsets.

Click a method to see more information on its use, syntax, and parameters:

[Count](#)

[FromXMLString](#)

[GetElementMetaData](#)

[Item](#)

[NumElements](#)

[ToXMLString](#)

COUNT PROPERTY

Used to get the number of records in the recordset.

Syntax

`object.Count`

| Part | Description |
|--------|----------------------------|
| object | An AppConnRecordSet object |

Remarks

Use the Count property to get the number of records in a recordset.

Example

```
strModelName = "CCSDemo"  
strServerName = "localhost"  
  
Set Verastream_Session = New AppConnModel  
Verastream_Session.ConnectToModel strServerName, strModelName  
  
strEntityName = "CustInquiryPanel"  
Verastream_Session.SetCurrentEntity strEntityName  
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then  
    MsgBox ("Set Current Entity Error")  
End If  
  
strAttrName = "AcctNumber"  
strAttrValue = "167439459"  
Set Attributes = New AppConnStringMap  
Attributes.Add strAttrName, strAttrValue  
Verastream_Session.SetAttributes Attributes  
  
strEntityName = "AcctTransactions"  
Verastream_Session.SetCurrentEntity strEntityName  
Set Records = Verastream_Session.FetchRecords  
  
MsgBox ("There are " & Records.Count & " records in the recordset")  
  
Verastream_Session.Disconnect  
Set Verastream_Session = Nothing
```

FROMXMLSTRING METHOD

Used to set a recordset from an XML representation.

Syntax

`object.FromXMLString(Record)`

| Part | Description |
|--------|--------------------|
| object | A recordset object |

| Part | Description |
|--------|--|
| Record | An XML representation of the recordset |

Remarks

Use the FromXMLString method to set a recordset from an XML representation.

GETELEMENTMETADATA METHOD

Used to get the metadata for an element of the records in a recordset.

Syntax

```
Set Metadata = object.GetElementMetaData(Index)
```

| Part | Description |
|----------|--|
| object | A recordset object |
| Index | The name or index(starting at 1) of an element in the list |
| MetaData | The RecordSetMetaData object for the given record |

Remarks

Use the GetElementMetaData method to get the metadata of an element in the records of a recordset.

ITEM PROPERTY

Used to get an AppConnRecord in an AppConnRecordSet.

Syntax

```
Set Record = object.Item(Index)
```

```
Set Record = object(Index)
```

| Part | Description |
|--------|---|
| object | A AppConnRecordSet object |
| Index | An index (starting at 1) of a record in the recordset |

| Part | Description |
|--------|------------------|
| Record | An AppConnRecord |

Remarks

Use the Item property to get an AppConnRecord in an AppConnRecordSet.

Example

```

Dim Verastream_Session As AppConnModel
Dim Records As AppConnRecordSet
Dim Record As AppConnRecord
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Set Records = Verastream_Session.FetchRecords

Set Record = Records(1)
Set Record = Records.Item(1)

Verastream_Session.Disconnect
Set Records = Nothing
Set Record = Nothing
Set Verastream_Session = Nothing

```

NUMELEMENTS PROPERTY

Used to get the number of elements in the records of the records.

Syntax

```
object.NumElements
```

| Part | Description |
|--------|--------------------|
| object | A recordset object |

Remarks

Use the NumElements property to get the number of elements in the records of a recordset.

Example

```

Dim Verastream_Session As AppConnModel
Dim Records As AppConnRecordSet
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Set Records = Verastream_Session.FetchRecords
MsgBox ("There are " & Records.NumElements & " elements in a record in the recordset")

Verastream_Session.Disconnect
Set Records = Nothing
Set Verastream_Session = Nothing

```

TOXMLSTRING METHOD

Used to get an XML representation of the recordset.

Syntax

```
XMLString = object.ToXMLString([URLDTD])
```

| Part | Description |
|--------|---|
| object | A recordset object |
| URLDTD | [optional] The URL of a DTD that will be used to validate the XML |

| Part | Description |
|-----------|--|
| XMLString | An XML representation of the recordset |

Remarks

Use the ToXMLString method to get an XML representation of the recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim Records As AppConnRecordSet
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Set Records = Verastream_Session.FetchRecords

MsgBox (Records.ToXMLString)

Verastream_Session.Disconnect
Set Records = Nothing
Set Verastream_Session = Nothing
```

RECORDSETMETADATA OBJECT

RecordSetMetaData Object

Use the AppConnRecordSetMetaData object to manage recordset metadata.

Click a method to see more information on its use, syntax, and parameters:

[Description](#)

[FieldNames](#)

[GetScrollOperation](#)

[MetaDataType](#)

[Name](#)

[ScrollMovements](#)

[ScrollMovementArray](#)

[SupportsDirectInserts](#)

[SupportsSelect](#)

DESCRIPTION PROPERTY

Used to get a description of a recordset within the Verastream model.

Syntax

```
object.Description
```

| Part | Description |
|-------------|--------------------------------|
| object | An operation metadata object |
| Description | A description of the recordset |

Remarks

Use the Description property to get the description of a recordset in the Verastream model.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)
MsgBox ("Recordset description is " & objMetaData.Description)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

FIELDNAMES PROPERTY

Used to get a list of the fields of the recordset.

Syntax

```
Set FieldNames = object.FieldNames
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|-----------|---------------------------------------|
| FieldName | A list of field names for a recordset |

Remarks

Use the FieldNames property to get a list of field names for a recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

Set appStringList = objMetaData.FieldNames
For I = 1 To appStringList.Count
    MsgBox ("Field name: " & appStringList(I))
Next
appStringList.Clear

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
Set appStringList = Nothing
```

GETSCROLLOPERATION METHOD

Used to get the scroll operation for the given scroll movements of the recordset.

Syntax

```
OperationName = object.GetScrollOperation(ScrollMovement)
```

| Part | Description |
|----------------|-----------------------------|
| object | A recordset metadata object |
| ScrollMovement | A scroll movement |

| Part | Description |
|---------------|----------------------------------|
| OperationName | The name of the scroll operation |

Remarks

Use the GetScrollOperation method to get the name of a scroll operation for a scroll movement of a recordset.

The enumeration values for AppConnScrollMovement are defined in the Type Library.

| Scroll Movement Names |
|------------------------------|
| ScrollHome |
| ScrollEnd |
| ScrollLineUp |
| ScrollLineDown |
| ScrollPageUp |
| |

Scroll Movement Names

ScrollPageDown

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData
Dim scrollMovements As Variant
Dim count, scrollMovement As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

scrollMovements = objMetaData.scrollMovements
count = UBound(objMetaData.scrollMovements) - LBound(objMetaData.scrollMovements)
For I = 0 To count
    Select Case objMetaData.scrollMovements(I)
        Case Is = AppConnScrollMovement.ScrollPageDown
            MsgBox ("Page Down operation: " & objMetaData.GetScrollOperation(ScrollPageDown))
        Case Is = AppConnScrollMovement.ScrollPageUp
            MsgBox ("Page Up operation: " & objMetaData.GetScrollOperation(ScrollPageUp))
        Case Is = AppConnScrollMovement.ScrollEnd
            MsgBox ("End operation: " & objMetaData.GetScrollOperation(ScrollEnd))
        Case Is = AppConnScrollMovement.ScrollHome
            MsgBox ("Home operation: " & objMetaData.GetScrollOperation(ScrollHome))
        Case Is = AppConnScrollMovement.ScrollLineDown
            MsgBox ("Line Down operation: " & objMetaData.GetScrollOperation(ScrollLineDown))
        Case Is = AppConnScrollMovement.ScrollLineUp
            MsgBox ("Line Up operation: " & objMetaData.GetScrollOperation(ScrollLineUp))
    End Select
Next

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

METADATATYPE

Used to get the metadata type of the recordset.

Syntax

```
object.MetaDataType
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|--------------|-------------------------------------|
| MetaDataType | The metadata type for the recordset |

Remarks

Use the MetaDataType property to get the metadata type for the recordset. The enumeration values for AppConnScrollMovement are defined in the Type Library.

| Return Value | Definition |
|--------------|------------|
| 0 | |
| 1 | |

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)
MsgBox ("Recordset type is " & objMetaData.MetaDataType)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set objMetaData = Nothing
```

NAME PROPERTY

Used to get the name of the recordset.

Syntax

`object.Name`

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|------|---------------------------|
| Name | The name of the recordset |

Remarks

Use the Name property to get the name of the recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScroll0per As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

MsgBox ("Recordset name is " & objMetaData.Name)
Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SCROLLMOVEMENTARRAY PROPERTY

Used to get an array of scroll movements of the recordset.

Syntax

```
Movements = object.ScrollMovementArray
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |

| Part | Description |
|-----------|--|
| Movements | A list of scroll movements for a recordset |

Remarks

Identical in functionality to the ScrollMovements property but modified to work in VBScript programs.

SCROLLMOVEMENTS PROPERTY

Used to get an array of scroll movements of the recordset.

Syntax

```
Movements = object.ScrollMovements
```

| Part | Description |
|--------|------------------------------|
| object | An operation metadata object |
| | |

| Part | Description |
|-----------|--|
| Movements | A list of scroll movements for a recordset |

Remarks

Use the ScrollMovements property to get a list of scroll movements for a recordset. The enumeration values for AppConnScrollMovement are defined in the Type Library.

[Scroll Movement Names] |:-----| ScrollHome ScrollEnd ScrollLineUp ScrollLineDown ScrollPageUp ScrollPageDown

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData
Dim scrollMovements As Variant
Dim count, scrollMovement As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

scrollMovements = objMetaData.scrollMovements
count = UBound(objMetaData.scrollMovements) - LBound(objMetaData.scrollMovements)
For I = 0 To count
    Select Case objMetaData.scrollMovements(I)
        Case Is = AppConnScrollMovement.ScrollPageDown
            MsgBox ("Page Down operation: " & objMetaData.GetScrollOperation(ScrollPageDown))
        Case Is = AppConnScrollMovement.ScrollPageUp
            MsgBox ("Page Up operation: " & objMetaData.GetScrollOperation(ScrollPageUp))
        Case Is = AppConnScrollMovement.ScrollEnd
            MsgBox ("End operation: " & objMetaData.GetScrollOperation(ScrollEnd))
        Case Is = AppConnScrollMovement.ScrollHome
            MsgBox ("Home operation: " & objMetaData.GetScrollOperation(ScrollHome))
        Case Is = AppConnScrollMovement.ScrollLineDown
            MsgBox ("Line Down operation: " & objMetaData.GetScrollOperation(ScrollLineDown))
        Case Is = AppConnScrollMovement.ScrollLineUp
            MsgBox ("Line Up operation: " & objMetaData.GetScrollOperation(ScrollLineUp))
    End Select
Next

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SUPPORTSDIRECTINSERTS PROPERTY

Used to get the supports direct inserts flag of the recordset.

Syntax

```
object.SupportsDirectInserts
```

| Part | Description |
|--------|-----------------------------|
| object | A recordset metadata object |

Remarks

Use the SupportsDirectInserts property to get the supports direct inserts flag for a recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName`

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

MsgBox ("Recordset name is " & objMetaData.Name)
MsgBox ("Recordset description is " & objMetaData.Description)
MsgBox ("Recordset type is " & objMetaData.MetaDataType)
MsgBox ("Recordset SupportsDirectInserts is " & objMetaData.SupportsDirectInserts)

Set appStringList = objMetaData.FieldNames
For I = 1 To appStringList.Count
    MsgBox ("Field name: " & appStringList(I))
Next
appStringList.Clear

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SUPPORTSSELECT PROPERTY

Used to get the supports select flag of the recordset.

Syntax

```
object.SupportsSelect
```

| Part | Description |
|--------|-----------------------------|
| object | A recordset metadata object |

Remarks

Use the SupportsSelect property to get the supports select flag for a recordset.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

MsgBox ("Recordset name is " & objMetaData.Name)
MsgBox ("Recordset description is " & objMetaData.Description)
MsgBox ("Recordset type is " & objMetaData.MetaDataType)
MsgBox ("Recordset SupportsSelect is " & objMetaData.SupportsSelect)

Set appStringList = objMetaData.FieldNames
For I = 1 To appStringList.Count
    MsgBox ("Field name: " & appStringList(I))
Next
appStringList.Clear

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing

```

SESSIONEX OBJECT

AppConnSessionEx Object

Use the AppConnSessionEx object to issue terminal commands and obtain information about terminal, model, and table metadata.

Click any of the following methods or properties to see information on its use, syntax, and parameters:

ConnectionTimeout
ConnectToModel
ConnectToModelViaDomain
ConnectToSession
ConnectToSessionViaDomain
Disconnect
EnableTerminalAttributes
ErrorCount
ExecuteSQLStatement
ExecuteSQLStatementWithMaxRows
FetchRecords
GetAttributeAtCursor
GetAttributeLocations
GetAttributes
GetColumnMetaData
GetCurrentEntity
GetCurrentRecord
GetCurrentRecordIndex
GetCurrentRecordSetName
GetEntityAttributes
GetEntityDescription
GetEntityOperations
GetEntityRecordSets
GetFieldLocations
GetHomeEntityName
GetFieldMetaData
GetHomeEntityName
GetLastRequestID
GetLocale
GetLoggingLevel
GetMajorVersion
GetMethodTimeout
GetMinorVersion

GetModelEntities
GetModelVariableNames
GetModelVariables
GetOperationMetaData
GetPatternLocations
GetProcedureMetaData
GetRecordSetLocations
GetRecordSetMetaData
GetSessionID
GetStringAtOffset
GetStringAtRowColumn
GetTableColumns
GetTableDescription
GetTableNames
GetTableProcedures
GetTerminalFieldAtCursor
GetVariableMetaData
GetVersionString
IsSecureConnection
InsertRecord
InsertRecords
InsertStringAtCursor
InsertStringAtOffset
InsertStringAtRowColumn
IsConnected
LastErrorMessageList
MetaDataOnly
ModelName
MoveCurrentRecordIndex
NextRecord
PerformAidKey
PerformEntityOperation
PerformTableProcedure

ProcessString
RequireSecureConnection
ResumeConnection
SelectCurrentRecord
SelectRecordByFilter
SelectRecordByIndex
SessionType
SetAttributes
SetAttributesDelayed
SetCurrentEntity
SetCurrentRecordIndex
SetCurrentRecordSetByName
SetLocale
SetLoggingLevel
SetMethodTimeout
SetModelVariables
SuspendConnection
UpdateCurrentRecord
UpdateRecordByFilter
UpdateRecordByIndex
UpdateRecords
WaitForCondition
WaitForCursor
WaitForEntityChange
WaitForString
WaitForStringRelCursor

CONNECTTOMODEL METHOD

Establishes a connection to a Verastream server on the specified server with the specified model name. User ID, password, and model variables to initialize may also be specified.

Syntax

```
object.ConnectToModel Server, ModelName, [UserID], [Password], [ModelVariables]
```

| Part | Description |
|-------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |
| Server | The Verastream server DNS name or IP address |
| ModelName | The name of the Verastream model |
| User ID | [optional] The user ID for authorization on the Verastreamserver |
| Password | [optional] The password for authorization the Verastreamserver |
| | |

| Part | Description |
|----------------|---|
| ModelVariables | [optional] An AppConnStringMap with name/values pairs of variables to set during connection |

Remarks

The parameters userID and password are used by VerastreamServer if the security option is ON.

Reasons for failure include:

Server is not running

Invalid server address

Invalid model name

Invalid userID and/or password.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strModelName, strUserID, strPassword As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[ConnectToModelViaDomain](#)

[ConnectToSession](#)

[ConnectToSessionViaDomain](#)

[SuspendConnection](#)

[ResumeConnection](#)

[Disconnect](#)

CONNECTTOMODELVIADOMAIN METHOD

Establishes a connection to a Verastream server and create a host session with the specified model name in the specified Verastream Verastream domain. User ID, password, and model variables to initialize may also be specified.

Syntax

```
object.ConnectToModelViaDomain DirectoryServer, Domain, ModelName, [UserID],  
[Password], [ModelVariables]
```

| Part | Description |
|-----------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |
| DirectoryServer | The Verastream directory server DNS name or IP address |
| Domain | The name of the Verastream domain |
| ModelName | The name of the Verastream model |
| User ID | [optional] The user ID for authorization on the Verastreamserver |
| Password | [optional] The password for authorization the Verastreamserver |
| | |

| Part | Description |
|----------------|---|
| ModelVariables | [optional] An AppConnStringMap with name/values pairs of variables to set during connection |

Remarks

Reasons for failure include:

Server is not running

Invalid server address

Invalid model name

Invalid userID and/or password.

The parameters userID and password are used by Verastream if the security option is ON.

Example

```
Dim Verastream_Session As AppConnModel
Dim strDirectoryServerName, strDomainName, strModelName, strUserID, _
    strPassword As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strDirectoryServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModelViaDomain strDirectoryServerName, _
    strDomainName, strModelName, strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[ConnectToModel](#)

[ConnectToSession](#)

[ConnectToSessionViaDomain](#)

[SuspendConnection](#)

[ResumeConnection](#)

[Disconnect](#)

CONNECTTOSESSION METHOD

Establishes a connection to a Verastream server on the specified server for the specified session. User ID, password, and model variables to initialize may also be specified.

Syntax

```
object.ConnectToSession Server, Session, [UserID], [Password],[ModelVariables]
```

| Part | Description |
|----------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| Server | The Verastream server DNS name or IP address |
| Session | The name of the Verastream session |
| User ID | [optional] The user ID for authorization on the Verastream server |
| Password | [optional] The password for authorization the Verastream server |
| | |

| Part | Description |
|----------------|---|
| ModelVariables | [optional] An AppConnStringMap with name/values pairs of variables to set during connection |

Remarks

The parameters userID and password are used by Verastream server if the security option is ON.

Reasons for failure include:

Server is not running

Invalid server address

Invalid model name

Invalid userID and/or password.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strSessionPoolName, strUserID, strPassword As String
Dim ModelVars As AppConnStringMap

strSessionPoolName = "CCSDemo"
strServerName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToSession strServerName, strSessionPoolName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[ConnectToModel](#)

[ConnectToSession](#)

[ConnectToSessionViaDomain](#)

[SuspendConnection](#)

[ResumeConnection](#)

[Disconnect](#)

CONNECTTOSESSIONVIADOMAIN METHOD

Establishes a connection to a Verastream server and creates a host session for the specified session in the specified Verastream Verastream domain. User ID, password, and model variables to initialize may also be specified.

Syntax

```
object.ConnectToSessionViaDomain DirectoryServer, Domain, Session, [UserID],  
[Password], [ModelVariables]
```

| Part | Description |
|-----------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| DirectoryServer | The Verastream directory server DNS name or IP address |
| Domain | The name of the Verastream domain |
| Session | The name of the Verastream session |
| User ID | [optional] The user ID for authorization on the Verastreamserver |
| Password | [optional] The password for authorization the Verastreamserver |
| | |

| Part | Description |
|----------------|---|
| ModelVariables | [optional] An AppConnStringMap with name/values pairs of variables to set during connection |

Remarks

Reasons for failure include:

Server is not running

Invalid server address

Invalid model name

Invalid userID and/or password.

The parameters userID and password are used by VerastreamServer if the security option is ON.

Example

```
Dim Verastream_Session As AppConnModel
Dim strDirectoryServerName, strDomainName, strSessionPoolName, strUserID, _
    strPassword As String
Dim ModelVars As AppConnStringMap

strSessionPoolName = "CCSDemo"
strDirectoryServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToSessionViaDomain strDirectoryServerName, _
    strDomainName, strSessionPoolName, strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[ConnectToModel](#)

[ConnectToSession](#)

[ConnectToSessionViaDomain](#)

[SuspendConnection](#)

[ResumeConnection](#)

[Disconnect](#)

DISCONNECT METHOD

Disconnects from a Verastream Server session.

Syntax

object.Disconnect([Reserved])

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| | |

| Part | Description |
|----------|-------------------------------|
| Reserved | [optional] Reserved parameter |

Remarks

Use the Disconnect method to end the connection to a Verastream session.

Reasons for failure include:

Server session has not been established

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strModelName, strUserID, strPassword As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

ERRORCOUNT

Use this property to specify the error count.

Syntax

```
object.ErrorCount
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, or AppConnTable) |

GETATTRIBUTEATCURSOR METHOD

Retrieves the name of the attribute at the current cursor position.

Syntax

```
AttributeName = object.GetAttributeAtCursor
```

| Part | Description |
|---------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| AttributeName | The name of the attribute |

Remarks

Use the `GetAttributeAtCursor` method to get the name of the attribute at the current cursor position, when a host application positions the cursor on an attribute as part of its processing.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strAttributeName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strAttributeName = Verastream_Session.GetAttributeAtCursor
MsgBox ("The attribute name at the current cursor position is" strAttributeName)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETATTRIBUTELOCATIONS METHOD

Used to get the locations for the given attributes.

Syntax

```
Locations = object.GetAttributeLocations(AttributeNames)
```

| Part | Description |
|----------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |
| AttributeNames | The names of the attributes |

| Part | Description |
|-------------|---------------------------------|
| Locations | The locations of the attributes |

Remarks

Use the GetAttributeLocations method to get locations of the given attributes.

GETATTRIBUTE METADATA METHOD

Retrieves the metadata for an attribute of an entity.

Syntax

```
Set AttributeMetadata = object.GetAttributeMetadata(EntityName, AttributeName)
```

| Part | Description |
|---------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |
| AttributeName | The name of the attribute |
| | |

| Part | Description |
|-------------------|--------------------------------|
| AttributeMetaData | The metadata for the attribute |

Remarks

Use the GetAttributeMetaData method to get the metadata for an attribute of an entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttribute As String
Dim objMetaData As AttributeMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"
strAttribute = "AcctNumber"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set objMetaData = Verastream_Session.GetAttributeMetaData(strEntity, strAttribute)

MsgBox ("Description " & objMetaData.Description)
MsgBox ("IsReadable " & objMetaData.IsReadable)
MsgBox ("IsWritable " & objMetaData.IsWritable)
MsgBox ("Length " & objMetaData.Length)
MsgBox ("MetaDataType " & objMetaData.MetaDataType)
MsgBox ("Name " & objMetaData.Name)
MsgBox ("ReadVariable " & objMetaData.ReadVariable)
MsgBox ("TerminalAttributesEnabled " & objMetaData.TerminalAttributesEnabled)
MsgBox ("WriteVariable " & objMetaData.WriteVariable)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETATTRIBUTES METHOD

Retrieves the attributes of the current entity.

Syntax

```
Set Attributes = object.GetAttributes([AttributeNames])
```

| Part | Description |
|----------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| AttributeNames | [optional] An AppConnStringList with the names of the attributes. If no AttributeNames are specified, GetAttributes returns all attributes of the entity. |

| Part | Description |
|------------|---|
| Attributes | An AppConnRecord with the Attributes names and values |

Example

```

Dim Verastream_Session As AppConnModel
Dim AttributeRecord As AppConnModelRecord
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.SetCurrentEntity (strEntity)
Set AttributeRecord = Verastream_Session.GetAttributes
Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)

For I = 1 To AttributeRecord.Count
    MsgBox ("Attribute " & AttributeList(I) & " = " & AttributeRecord(I))
Next

Set AttributeRecord = Nothing
Set AttributeList = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing

```

ENABLETERMINALATTRIBUTES METHOD

Sets the AppConn object to return terminal attribute information when retrieving a recordset.

Syntax

```
object.EnableTerminalAttributes enable
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|---------------|
| enable | True or False |

Remarks

Use the EnableTerminalAttributes method to set the AppConn connector so that subsequent fetches of recordsets will contain terminal attribute information.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.EnableTerminalAttributes True

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

EXECUTESQLSTATEMENT METHOD

Used to execute an SQL statement to return a recordset.

Syntax

```
Set RecordSet = object.ExecuteSQLStatement(SQLStatement)
```

| Part | Description |
|--------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| SQLStatement | An SQL statement that will be executed by the server |

| Part | Description |
|-----------|---|
| RecordSet | The AppConnRecordset returned for the SQL statement |

Remarks

Use the ExecuteSQLStatement method to execute an SQL statement.

Example

```
Dim Verastream_Session As AppConnTable
Dim Verastream_Recordset As AppConnRecordSet
Dim Verastream_Record As AppConnRecord
Dim strModelName, strServerName, strSQLStatement As String

strModelName = "CCSDemo"
strServerName = "localhost"
strSQLStatement = "select * from transactions where AcctNumber = '167439459'"

Set Verastream_Session = New AppConnTable
Verastream_Session.ConnectToModel strServerName, strModelName

Set Verastream_Recordset = Verastream_Session.ExecuteSQLStatement(strSQLStatement)

For Each Verastream_Record In Verastream_Recordset
    MsgBox ("The date field within the recordset is " & Verastream_Record!Date)
Next

Verastream_Session.Disconnect
Set Verastream_Record = Nothing
Set Verastream_Recordset = Nothing
Set Verastream_Session = Nothing
```

EXECUTESQLSTATEMENTWITHMAXROWSMETHOD

Used to execute an SQL statement to return a recordset, specifying maximumrows.

Syntax

```
Set RecordSet = object.ExecuteSQLStatementWithMaxRows(SQLStatement, maxRows)
```

| Part | Description |
|--------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| SQLStatement | An SQL statement that will be executed by the server |
| maxRows | Maximum number of rows of data to be returned |

| Part | Description |
|-------------|---|
| RecordSet | The AppConnRecordset returned for the SQL statement |

Remarks

Use the ExecuteSQLStatement method to execute an SQL statement while specifying a maximum number of rows to be returned.

FETCHRECORDS METHOD

Fetches data from a recordset on the Verastream server. If the number of rows to fetch is not specified or is 0 then the number of rows that will be returned is unlimited. If the field names is not specified or specified as an empty list then all fields will be returned.

Syntax

```
Set RecordSet = object.FetchRecords([MaxRows], [FieldNames], [FilterExpression])
```

| Part | Description |
|------------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| MaxRows | [optional] The maximum number of rows that will be fetched |
| FieldNames | [optional] An AppConnStringList with the names of the fields to be returned |
| FilterExpression | [optional] An expression used to filter which records will be fetched |

| Part | Description |
|-----------|--|
| RecordSet | The set of records returned from the fetch |

Remarks

Use the FetchRecords method to when retrieving information from a recordset defined in Verastream.

Example

```
Dim Verastream_Session As AppConnModel
Dim Records As AppConnRecordSet
Dim record As Variant
Dim CustomerAccounts As Collection
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) = 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) = 0 Then
    Set Records = Verastream_Session.FetchRecords
    For Each record In Records
        MsgBox ("Amount = " record!Amount)
    Next record
Else
    MsgBox ("Set Current Entity Error")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

CONNECTIONTIMEOUT METHOD

Gets or sets how long Host Integrator continues attempting to establish a connection if for any reason the connection cannot be established on the first try. The value is in seconds.

Syntax

For getting: Timeout = object.ConnectionTimeout

For setting: object.ConnectionTimeout Timeout

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-------------|--|
| Timeout | The amount of time (seconds) Host Integrator continues attempting to establish a connection if for any reason the connection cannot be established on the first try. |

Remarks

Getting or setting a connection timeout is useful, for example, if the server is temporarily unable to allow any more sessions, or if the domain load has been reached. Connection attempt information, include the number of connection attempts and the time of those attempts, is written to the log.

The default value is 30 (seconds).

GETCOLUMNMETADATA METHOD

Retrieves the metadata for the named column in the named table.

Syntax

```
Set ColumnMetaData = object.GetColumnMetaData(TableName, ColumnName)
```

| Part | Description |
|-------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |
| TableName | The name of the table defined in Verastream as a string |
| ColumnName | The name of the column defined in the table as a string |

| Part | Description |
|----------------|--|
| ColumnMetaData | The ColumnMetaData object for the column |

Remarks

Use the GetColumnMetaData method to get the metadata for the given column.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strRecordset, strRecordsetOut As String
Dim ColumnMeta As ColumnMetaData
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

strRecordset = "AcctTransData"
Verastream_Session.SetCurrentRecordSetByName strRecordset

Set ColumnMeta = Verastream_Session.GetColumnMetaData("Transactions", "AcctNumber")
MsgBox ("Column type = " & ColumnMeta.ColumnType)
MsgBox ("Column description = " & ColumnMeta.Description)
MsgBox ("Column is key field = " & ColumnMeta.IsKey)
MsgBox ("Column meta data type = " & ColumnMeta.MetaDataType)
MsgBox ("Column maximum = " & ColumnMeta.Max)
MsgBox ("Column minimum = " & ColumnMeta.Min)
MsgBox ("Column name = " & ColumnMeta.Name)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
Set ColumnMeta = Nothing
```

GETCURRENTENTITY METHOD

Retrieves the current entity (for example, screen) of the legacy application.

Syntax

```
CurrentEntity = object.GetCurrentEntity
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------------|------------------------|
| CurrentEntity | The name of the entity |

Remarks

Use the GetCurrentEntity method to get the name of the current entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strCurrentEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strCurrentEntity = Verastream_Session.GetCurrentEntity
MsgBox ("Current entity name = " & strCurrentEntity)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETCURRENTRECORD METHOD

Retrieves the record at the current index for a Verastream recordset.

Syntax

```
Set Record = object.GetCurrentRecord
```

|Part|Description| |object|An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |Record|A record with the fields names and values

Remarks

Use the GetCurrentRecord method to retrieve the record at the current record index in a Verastream recordset.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName As String
Dim strAttrName, strAttrValue, strIndex As String
Dim modRecord As AppConnModelRecord
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Set Attributes = Nothing

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Verastream_Session.SetCurrentRecordIndex (2)
Set modRecord = Verastream_Session.GetCurrentRecord

For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next
Set modRecord = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing

```

GETCURRENTRECORDINDEX METHOD

Retrieves the current record index of the current recordset.

Syntax

```
RecordIndex = object.GetCurrentRecordIndex
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-------------|---------------------|
| RecordIndex | The recordset index |

Remarks

Use the `GetCurrentRecordIndex` method to get the index of the current record in the current recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strIndex As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Verastream_Session.SetCurrentRecordIndex (1)
strIndex = Verastream_Session.GetCurrentRecordIndex
MsgBox ("Current record index = " & strIndex)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETCURRENTRECORDSETNAME METHOD

Method used to retrieve the name of the current recordset.

Syntax

```
RecordSetName = object.GetCurrentRecordSetName
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------------|-----------------------------------|
| RecordSetName | The name of the current recordset |

Remarks

Use the `GetCurrentRecordSetName` method to get the name of the current recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strRecordset, strRecordsetOut As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

strRecordset = "AcctTransData"
Verastream_Session.SetCurrentRecordSetByName strRecordset

strRecordsetOut = Verastream_Session.GetCurrentRecordSetName
MsgBox ("Current record set is = " & strRecordsetOut)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETENTITYATTRIBUTES METHOD

Retrieves the attribute names for the named entity.

Syntax

```
Set EntityAttributes = object.GetEntityAttributes(EntityName)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |

| Part | Description |
|------------------|---|
| EntityAttributes | An AppConnStringList with the names of the attributes |

Remarks

Use the `GetEntityAttributes` method to get the name of the attributes for the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity, strAttribute As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)

For I = 1 To (AttributeList.Count)
    strAttribute = AttributeList.Item(I)
    MsgBox ("Attribute = " & strAttribute)
Next

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

GETENTITYDESCRIPTION METHOD

Retrieves the description for the named entity.

Syntax

```
Description = object.GetEntityDescription(EntityName)
```

[Part|Description|] object|An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) EntityName|The name of the entity Description|The description of the entity

Remarks

Use the `GetEntityDescription` method to get the description for the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityDescr, strEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
strEntityDescr = Verastream_Session.GetEntityDescription(strEntity)
MsgBox ("Entity description = " & strEntityDescr)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETENTITYOPERATIONS METHOD

Method used to retrieve the names of the operations for the named entity.

Syntax

```
Set Operations = object.GetEntityOperations(EntityName)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |
| Operations | An AppConnStringList with the names of the operations |

Remarks

Use the GetEntityOperations method to get a list of the operations for the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim EntityOperations As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set EntityOperations = Verastream_Session.GetEntityOperations(strEntity)
For I = 1 To EntityOperations.Count
    MsgBox ("Entities operations = " & EntityOperations(I))
Next

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETENTITYRECORDSETS METHOD

Retrieves the recordset names for the named entity.

Syntax

```
Set RecordSets = object.GetEntityRecordSets(EntityName)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |

| Part | Description |
|------------|---|
| RecordSets | An AppConnStringList with the names of the recordsets |

Remarks

Use the GetEntityRecordSets method to get a list of the recordsets for the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim EntityRecordsets As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "AcctTransactions"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set EntityRecordsets = Verastream_Session.GetEntityRecordSets(strEntity)
For I = 1 To EntityRecordsets.Count
    MsgBox ("Entities record sets = " & EntityRecordsets(I))
Next

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETFIELDLOCATIONS METHOD

Used to get the locations for the given fields.

Syntax

```
Locations = object.GetFieldLocations(FieldNames)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |
| FieldNames | The names of the fields |

| Part | Description |
|-----------|-------------------------------|
| Locations | The locations of the patterns |

Remarks

Use the `GetFieldLocations` method to get locations of the given fields.

Recordset fields are always linear regions.

This method is guaranteed to work correctly only when a record in the recordset is selected. If no record is selected:

For recordsets containing fixed records, the field locations returned are those for the first record in the recordset, based upon the information stored in the model.

For recordsets containing variable-length records, the information returned is not meaningful.

GETFIELDMETADATA METHOD

Retrieves the metadata for the named field from the named entity on the named recordset.

Syntax

```
Set FieldMetaData = object.GetFieldMetaData(EntityName, RecordSetName,FieldName)
```

| Part | Description |
|---------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |
| RecordSetName | The name of the recordset |
| FieldName | The name of the field |

| Part | Description |
|---------------|----------------------------|
| FieldMetaData | The metadata for the field |

Remarks

Use the GetFieldMetaData method to get the metadata for the given field of the recordset on the entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim objMetaData As FieldMetaData
Dim strEntityName, strRecordsetName, strFieldName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"
strFieldName = "Date"
Set objMetaData = Verastream_Session.GetFieldMetaData(strEntityName, _
    strRecordsetName, strFieldName)

MsgBox ("Field metadata name is " & objMetaData.Name)
MsgBox ("Field metadata description is " & objMetaData.Description)
MsgBox ("Field metadata IsKey is " & objMetaData.IsKey)
MsgBox ("Field metadata IsReadable is " & objMetaData.IsReadable)
MsgBox ("Field metadata IsWritable is " & objMetaData.IsWritable)
MsgBox ("Field metadata length is " & objMetaData.Length)
MsgBox ("Field metadata type is " & objMetaData.MetaDataType)
MsgBox ("Field metadata TerminalAttributesEnabled is " & objMetaData.TerminalAttributesEnabled)

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETHOMEENTITYNAME METHOD

Used to get the name of the home entity.

Syntax

```
EntityName = object.GetHomeEntityName
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|------------|------------------------------|
| EntityName | The name of the home entity. |

Remarks

Use the `GetHomeEntityName` method to get the name of the home entity defined in the model for the host system. If there is no home entity (for example, when the model contains no entities), this method returns "None".

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
strEntityName = Verastream_Session.GetHomeEntityName
MsgBox ("The home entity for the model is " & strEntityName)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETLASTREQUESTID METHOD

Gets an integer identifier for the most recent request performed against the current Host Integrator Server session.

Syntax

```
RequestID = object.GetLastRequestID
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-----------|---|
| RequestID | The unique identification number of the last request that was performed |

Remarks

Use the `GetLastRequestID` method to get the last request id on the current session, which can be used in referencing logging information.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName, strEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

MsgBox (Verastream_Session.GetLastRequestID)
Verastream_Session.SetCurrentEntity ("SignonPanel")
MsgBox (Verastream_Session.GetLastRequestID)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

See also

- [ConnectToModel](#)
- [ConnectToSession](#)
- [ConnectToSessionViaDomain](#)
- [SuspendConnection](#)
- [ResumeConnection](#)
- [Disconnect](#)

GETLOCALE METHOD

Retrieves the Locale for the AppConn object.

Syntax

```
Locale = object.GetLocale
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the GetLocale method to get the locale of the AppConn object, which may be set with the SetLocale method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, strLocale, strTempString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

strLocale = "en_us"
Verastream_Session.SetLocale (strLocale)

strTempString = Verastream_Session.GetLocale
MsgBox ("The Verastream server locale is " & strTempString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[SetLocale](#)

GETLOGGINGLEVEL METHOD

Used to get the logging level for the session.

Syntax

```
LoggingLevel = object.GetLoggingLevel
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|--------------|---|
| LoggingLevel | A numeric logging level (i.e. Errors, ErrorsAndWarnings,or All) |

Remarks

Use the GetLoggingLevel method to get the logging level for a session.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName, strEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

MsgBox ("The logging level for the Verastream Server is " & Verastream_Session.GetLoggingLevel)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

See also

[GetLoggingLevel](#)

GETMAJORVERSION METHOD

Gets the major version for the AppConn object.

Syntax

```
object.GetMajorVersion
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |

Remarks

Use the GetMajorVersion method to get the major version number of the AppConn connector.

Example

```

Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, tmpString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If
tmpString = Verastream_Session.GetMajorVersion
MsgBox ("Major version = " & tmpString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

See also

[GetMinorVersion](#)

METADATAONLY METHOD

Use this property to specify that the current session is "metadata only" or to find out if the current session is a metadata-only session.

Syntax

```
object.MetadataOnly
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

This property is read/write if the session is not connected, and read-only if the session is connected.

A metadata only session does not require a host connection and allows only methods and properties that interact with metadata. You must set MetadataOnly to true before you use a Connect method.

Once a connect method has been called, and before the Disconnect method has been called, any attempt to change the value of the MetadataOnly property will generate an error with this text: "The MetadataOnly property cannot be changed while a connection is active."

When a client connects with MetadataOnly set to true, the server will not report the connection as a session, and will not allocate a new session or a session from a pool.

Example

```
Set session = CreateObject("VeraStream.AppConnSessionEx")
session.MetadataOnly = True
session.ConnectToModel "MyServer", "MyModel"
...
session.Disconnect
```

GETMETHODTIMEOUT METHOD

Retrieves the timeout value set on the AppConn connection.

Syntax

```
Timeout = object.GetMethodTimeout
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| | |

| Part | Description |
|---------|--|
| Timeout | The amount of time (milliseconds) methods will process before timing out |

Remarks

Use the `GetMethodTimeout` method to get the timeout for methods of the `AppConn` object, which may be set with the `SetMethodTimeout` method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, strMethodTimeout, strTempString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

strMethodTimeout = "20000"
Verastream_Session.SetMethodTimeout (strMethodTimeout)
strTempString = Verastream_Session.GetMethodTimeout
MsgBox ("The Verastream server method timeout is " & strTempString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[SetMethodTimeout](#)

GETMINORVERSION METHOD

Gets the minor version for the `AppConn` object.

Syntax

```
object.GetMinorVersion
```

| Part | Description |
|--------|---|
| object | An <code>AppConn</code> object (for example, <code>AppConnModel</code> , <code>AppConnTerm</code> , and <code>AppConnTable</code>) |

Remarks

Use the `GetMinorVersion` method to get the minor version number of the `AppConn` connector.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, tmpString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If
tmpString = Verastream_Session.GetMinorVersion
MsgBox ("Minor version value = " & tmpString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[GetMajorVersion](#)

GETMODELENTITIES METHOD

Retrieves the names of the entities for the model.

Syntax

```
Set ModelEntities = object.GetModelEntities
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------------|---|
| ModelEntities | An AppConnStringList with the names of the entities |

Remarks

Use the GetModelEntities method to get the a list of the entities defined in the model.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim EntityList As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set EntityList = Verastream_Session.GetModelEntities
For I = 1 To EntityList.Count
    MsgBox ("Model entities = " & EntityList(I))
Next

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

LASTERRORMESSAGELIST PROPERTY

Returns the error message strings associated with the most recent failed API call.

Syntax

```
object.LastErrorMessageList
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the LastErrorMessageList method to get a list of the error message strings associated with the most recent failed API call.

MODELNAME PROPERTY

Returns the Verastream model name for the current session.

Syntax

```
object.ModelName
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the ModelName property to get the name of the Host Integrator model for the current session. The model name is available even though a model name was not used to connect to the session (e.g. ConnectToSession)

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, tmpString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If
tmpString = Verastream_Session.ModelName
MsgBox ("The Verastream model name is = " & tmpString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[GetSessionID](#)

[ServerName](#)

GETMODELVARIABLENAMES METHOD

Method used to return the names of all non-hidden model variables in the Host Integrator model.

Syntax

```
Set MVNames = object.GetModelVariableNames
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,AppConnTable, and AppConnSessionEx) |

| Part | Description |
|---------|--|
| MVNames | An AppConnStringList with the variable names |

Remarks

Use the GetModelVariableNames method to get a list of all non-hidden model variables that are defined in the model.

GETMODELVARIABLES METHOD

Retrieves the model variable names and values.

Syntax

```
Set Variables = object.GetModelVariables
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-----------|---|
| Variables | An AppConnStringMap with name/valuepairs of variables |

Remarks

Use the GetModelVariables method to get the an AppConnStringMap of the names and values of the variables.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set ModelVars = Verastream_Session.GetModelVariables
For I = 1 To ModelVars.Count
    MsgBox ("Model variable " & ModelVars.Keys(I) & " " & ModelVars.Item(I))
Next
ModelVars.Clear

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETOPERATIONMETADATA METHOD

Retrieves the metadata for the given operation on the given entity.

Syntax

```
Set OperationMetaData = object.GetOperationMetaData(EntityName, OperationName)
```

| Part | Description |
|---------------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |
| OperationName | The name of the operation |

| Part | Description |
|-------------------|--------------------------------|
| OperationMetaData | The metadata for the operation |

Remarks

Use the GetOperationMetaData method to get the metadata for the given operation of the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strOperationName As String
Dim objMetaData As OperationMetaData
Dim Destinations, AttributesUsed, VariablesUsed As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
strOperationName = "ToNameSearch"

Set objMetaData = Verastream_Session.GetOperationMetaData(strEntityName, _
    strOperationName)

MsgBox ("Operation metadata name is " & objMetaData.Name)
MsgBox ("Operation metadata Description is " & objMetaData.Description)
MsgBox ("Operation metadata Destination is " & objMetaData.Destination)
MsgBox ("Operation metadata IsDefault is " & objMetaData.IsDefault)
MsgBox ("Operation metadata MetaDataType is " & objMetaData.MetaDataType)
MsgBox ("Operation metadata Timeout is " & objMetaData.Timeout)

Set Destinations = objMetaData.AltDestinations
For I = 1 To Destinations.Count
    MsgBox ("Alternate destination: " & Destinations(I))
Next
Set Destinations = Nothing

Set AttributesUsed = objMetaData.AttributesUsed
For I = 1 To AttributesUsed.Count
    MsgBox ("Attributes used: " & AttributesUsed(I))
Next
Set AttributesUsed = Nothing

Set VariablesUsed = objMetaData.VariablesUsed
For I = 1 To VariablesUsed.Count
    MsgBox ("Variables used: " & VariablesUsed(I))
Next
Set VariablesUsed = Nothing
```

GETPATTERNLOCATIONS METHOD

Used to get the locations for the given patterns.

Syntax

```
Locations = object.GetPatternLocations(PatternNames)
```

| Part | Description |
|--------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |
| PatternNames | The names of the patterns |

| Part | Description |
|-------------|-------------------------------|
| Locations | The locations of the patterns |

Remarks

Use the GetPatternLocations method to get locations of the given patterns.

GETPROCEDUREMETADATA METHOD

Used to retrieve the metadata from the named table for the named procedure.

Syntax

```
Set ProcedureMetaData = object.GetProcedureMetaData(TableName, ProcedureName)
```

| Part | Description |
|---------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| TableName | The name of the table defined in Verastream |
| ProcedureName | The name of the procedure defined in the table |
| | |

| Part | Description |
|-------------------|-----------------------------|
| ProcedureMetaData | The metadata for the column |

Remarks

Use the `GetProcedureMetaData` method to get the metadata for the given procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim ProcMetaData As ProcedureMetaData
Dim ProcStringList As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ProcMetaData = Verastream_Session.GetProcedureMetaData("Transactions", _
    "GetTransactions")

MsgBox ("Procedure name is " & ProcMetaData.Name)
MsgBox ("Procedure description is " & ProcMetaData.Description)
MsgBox ("Procedure metadata type is " & ProcMetaData.MetadataType)
MsgBox ("Procedure type is " & ProcMetaData.ProcedureType)
MsgBox ("Is the procedure used for SQL? " & ProcMetaData.UsedForSQL)

Set ProcStringList = ProcMetaData.FilterColumns
For I = 1 To ProcStringList.Count
    MsgBox ("Filter column " & I & " is " & ProcStringList(I))
    MsgBox ("Is " & ProcStringList(I) & " required?" & _
        ProcMetaData.IsRequiredFilter(ProcStringList(I)))`
Next
ProcStringList.Clear

Set ProcStringList = ProcMetaData.InputColumns
For I = 1 To ProcStringList.Count
    MsgBox ("Input column " & I & " is " & ProcStringList(I))
    MsgBox ("Is " & ProcStringList(I) & " required?" & _
        ProcMetaData.IsRequiredInput(ProcStringList(I)))
Next
ProcStringList.Clear

Set ProcStringList = ProcMetaData.OutputColumns
For I = 1 To ProcStringList.Count
    MsgBox ("Output column " & I & " is " & ProcStringList(I))
Next
ProcStringList.Clear

Verastream_Session.Disconnect
Set ProcStringList = Nothing
Set ProcMetaData = Nothing
Set Verastream_Session = Nothing
```

GETRECORDSETMETADATA METHOD

Method used to retrieve the metadata for the named Recordset on the named entity.

Syntax

```
Set RecordSetMetaData = object.GetRecordSetMetaData (EntityName, RecordSetName)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |

| Part | Description |
|-------------------|-------------------------------|
| RecordSetName | The name of the recordset |
| RecordSetMetaData | The metadata for therecordset |

Remarks

Use the GetRecordSetMetaData method to get the metadata for the given recordset of the given entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName, strRecordsetName, strScrollOper As String
Dim objMetaData As RecordSetMetaData
Dim appStringList As appConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "AcctTransactions"
strRecordsetName = "AcctTransData"

Set objMetaData = Verastream_Session.GetRecordSetMetaData(strEntityName, strRecordsetName)

MsgBox ("Recordset name is " & objMetaData.Name)
MsgBox ("Recordset description is " & objMetaData.Description)
MsgBox ("Recordset type is " & objMetaData.MetaDataType)
MsgBox ("Recordset SupportsDirectInserts is " & objMetaData.SupportsDirectInserts)
MsgBox ("Recordset SupportsSelect is " & objMetaData.SupportsSelect)

Set appStringList = objMetaData.FieldNames
For I = 1 To appStringList.Count
    MsgBox ("Field name: " & appStringList(I))
Next
appStringList.Clear

Set appStringList = Nothing

Set objMetaData = Nothing
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETRECORDSETLOCATIONS METHOD

Used to get the locations for the given record sets.

Syntax

```
Locations = object.GetRecordSetLocations(RecordSetNames)
```

| Part | Description |
|----------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |
| RecordSetNames | The names of the record sets |

| Part | Description |
|-------------|----------------------------------|
| Locations | The locations of the record sets |

Remarks

Use the GetRecordSetLocations method to get locations of the given record sets.

GETSESSIONID METHOD

Retrieves the identification number of the current session.

Syntax

```
SessionID = object.GetSessionID
```

| Part | Description |
|-------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |

| Part | Description |
|-----------|---|
| SessionID | The unique identification number of the session |

Remarks

Use the GetSessionID method to get the current session id, which can be used in referencing logging information.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, tmpString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If
tmpString = Verastream_Session.GetSessionID
MsgBox ("Session ID value = " &tmpString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[ModelName](#)

[ServerName](#)

GETSTRINGATOFFSET METHOD

Used to get a string from the current entity on the Verastream server starting at an offset of the given length.

Syntax

```
String = object.GetStringAtOffset(Offset, Length)
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| Offset | The offset of the string on the entity |
| Length | The length of the string on the entity |

| Part | Description |
|--------|--|
| String | The text string from the entity at the given offset for the given length |

Remarks

Use the GetStringAtOffset method to get strings where attributes or recordset fields have not been defined.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.InsertStringAtOffset "testname", 1487

MsgBox ("The string at row 19, column 30 is " & _
    Verastream_Session.GetStringAtOffset(1487, 8))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETSTRINGATROWCOLUMN METHOD

Used to get a string within a rectangular region from the current entity on the Verastream Server.

Syntax

```
String = object.GetStringAtRowColumn(TopRow, LeftColumn, NumRows, NumColumns)
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| TopRow | The top row of the entity (for example, screen) |
| LeftColumn | The left column of the entity (for example, screen) |
| NumRows | The number of rows of the entity (for example, screen) |
| NumColumns | The number of columns of the entity (for example, screen) |

| Part | Description |
|--------|--|
| String | The text string from the entity at the given offset for the given length |

Remarks

Use the GetStringAtRowColumn method to get strings where attributes or recordset fields have not been defined.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Verastream_Session.InsertStringAtRowColumn "test name", 19, 48
MsgBox ("The string at row 19 column 48 is " & _
    Verastream_Session.GetStringAtRowColumn(19, 48, 1, 12))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

GETTABLECOLUMNS METHOD

Used to retrieve the column names for a given table.

Syntax

```
Set TableColumns = object.GetTableColumns(TableName)
```

| Part | Description |
|-----------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| TableName | The name of the table defined in Verastream |

| Part | Description |
|--------------|--|
| TableColumns | An AppConnStringList with the names of the columns |

Remarks

Use the GetTableColumns method to get a list of the columns for the given procedure.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String
Dim TableCol As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set TableCol = Verastream_Session.GetTableColumns("Transactions")
For I = 1 To TableCol.Count
    MsgBox ("Transactions table column " & I & " is " & TableCol(I))
Next
TableCol.Clear

Verastream_Session.Disconnect
Set TableCol = Nothing
Set Verastream_Session = Nothing
```

GETTABLEDESCRIPTION METHOD

Used to retrieve the description for the named table.

Syntax

```
Description = object.GetTableDescription(TableName)
```

| Part | Description |
|-----------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| TableName | The name of the table defined in Verastream |

| Part | Description |
|-------------|-------------------------------|
| Description | The description for the table |

Remarks

Use the `GetTableDescription` method to get the description for the given table.

Example

```
Dim Verastream_Session As AppConnTable
Dim TableNames As AppConnStringList
Dim strModelName, strServerName, strTableDescr As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set TableNames = Verastream_Session.GetTableNames

For i = 1 To TableNames.Count
    strTableDescr = Verastream_Session.GetTableDescription(TableNames(i))
    MsgBox ("The description for the table named " & TableNames(i) _
        & " is " & strTableDescr)
Next

Verastream_Session.Disconnect
Set TableNames = Nothing
Set Verastream_Session = Nothing
```

GETTABLENAMES METHOD

Method used to retrieve the table names.

Syntax

```
Set TableNames = object.GetTableNames
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|------------|---|
| TableNames | An AppConnStringList with the table names |

Remarks

Use the GetTableNames method to get a list of the names of the tables that are defined in the model.

Example

```
Dim Verastream_Session As AppConnModel
Dim TableNames As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set TableNames = Verastream_Session.GetTableNames

For i = 1 To TableNames.Count
    MsgBox ("Table " & i & " is " & TableNames(i))
Next

Verastream_Session.Disconnect
Set TableNames = Nothing
Set Verastream_Session = Nothing
```

GETTABLEPROCEDURES METHOD

Used to retrieve the tables defined for the named table.

Syntax

```
Set TableProcedures = object.GetTableProcedures
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-----------------|---|
| TableProcedures | An AppConnStringList of the procedure names |

Remarks

Use the GetTableNames method to get a list of the names of the tables that are defined in the model.

Example

```
Dim Verastream_Session As AppConnTable
Dim ProcedureNames, TableNames As AppConnStringList
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set TableNames = Verastream_Session.GetTableNames

For i = 1 To TableNames.Count
    Set ProcedureNames = Verastream_Session.GetTableProcedures(TableNames(i))
    For j = 1 To ProcedureNames.Count
        MsgBox ("The table " & TableNames(i) & " has a procedure named " _
            & ProcedureNames(j))
    Next
Next

Verastream_Session.Disconnect
Set ProcedureNames = Nothing
Set TableNames = Nothing
Set Verastream_Session = Nothing
```

GETTERMINALFIELDATCURSOR METHOD

Used to get the field at the cursor defined in the Verastream model for the host application.

Syntax

```
TerminalField = object.GetTerminalFieldAtCursor
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|---------------|------------------|
| TerminalField | A terminal field |

Remarks

Use the GetTerminalFieldAtCursor method to get the field at the cursor defined in the Verastream model for the host application.

GETVARIABLEMETADATA METHOD

Retrieves the metadata for the named variable.

Syntax

```
Set VariableMetaData = object.GetVariableMetaData (VariableName)
```

| Part | Description |
|--------------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |
| VariableName | The name of the variable |

| Part | Description |
|------------------|-------------------------------|
| VariableMetaData | The metadata for the variable |

Remarks

Use the `GetVariableMetaData` method to get the metadata for the given variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)

MsgBox ("Variable metadata name is " & objVariableMetaData.Name)
MsgBox ("Variable metadata default value is " & objVariableMetaData.DefaultValue)
MsgBox ("Variable metadata description is " & objVariableMetaData.Description)
MsgBox ("Variable metadata initialization is " & objVariableMetaData.Initialization)
MsgBox ("Variable metadata IsEncrypted is " & objVariableMetaData.IsEncrypted)
MsgBox ("Variable metadata IsHidden is " & objVariableMetaData.IsHidden)
MsgBox ("Variable metadata IsReadable is " & objVariableMetaData.IsReadable)
MsgBox ("Variable metadata IsWriteable is " & objVariableMetaData.IsWriteable)
MsgBox ("Variable metadata meta data type is " & objVariableMetaData.MetaDataType)
MsgBox ("Variable metadata variable type is " & objVariableMetaData.VariableType)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

GETVERSIONSTRING METHOD

Gets the version string for the AppConn object.

Syntax

```
object.GetVersionString
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the `GetVersionString` method to get the version information about the AppConn connector.

Example


```

Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, tmpString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If
tmpString = Verastream_Session.GetVersionString
MsgBox ("Version string value = " & tmpString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

INSERTRECORD METHOD

Inserts a record into a Verastream recordset.

Syntax

```
object.InsertRecord Record
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|--|
| Record | An AppConnStringMap with name/value pairs for the fields of a record |

Remarks

Use the InsertRecord method to insert a record into a Verastream recordset.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String
Dim recordMap As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) = 0 Then
    Set recordMap = New AppConnStringMap
    recordMap.Clear
    recordMap.Add "Date", "04-18-03"
    recordMap.Add "Code", "B6"
    recordMap.Add "Amount", "100.00"
    Verastream_Session.InsertRecord recordMap
Else
    MsgBox ("Set Current Entity Error")
End If

Verastream_Session.Disconnect
Set recordMap = Nothing
Set Verastream_Session = Nothing

```

INSERTRECORDS METHOD

Used to insert a set of records into a record set defined in the Verastream model.

Syntax

```
object.InsertRecords Records
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|-------------|--------------------------------------|
| Records | A set of records (AppConnStringMaps) |

Remarks

Use the InsertRecords method to insert a set of records into a record set defined in the Verastream model.

INSERTSTRINGATCURSOR METHOD

Used to insert a string to a host application at the cursor.

Syntax

```
object.InsertStringAtCursor String
```

| Part | Description |
|-------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|--------|-------------|
| String | A string |

Remarks

Use the `InsertStringAtCursor` method to insert a string to a host application at the cursor.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String
Dim Verastream_StringMap As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

' This is adding the string into the "UserID" attribute location
Verastream_Session.InsertStringAtCursor ("bjones")

Set Verastream_StringMap = New AppConnStringMap
Verastream_StringMap.Add "password", "bjones"
Verastream_Session.SetAttributes Verastream_StringMap
Verastream_StringMap.Clear
Verastream_Session.SetCurrentEntity ("MainMenu")

MsgBox ("The host is on entity " & Verastream_Session.GetCurrentEntity)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Verastream_StringMap = Nothing
```

INSERTSTRINGATOFFSET METHOD

Used to insert a string into the current entity on the Verastream server starting at an offset.

Syntax

```
object.InsertStringAtOffset String, Offset
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| String | The text string to put on the entity at the given offset |

| Part | Description |
|--------|--|
| Offset | The row and column coordinates on a terminal screen. |

Remarks

Use the `InsertStringAtOffset` method to put strings where attributes or recordset fields have not been defined.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.InsertStringAtOffset "testname", 1487

MsgBox ("The string at row 19, column 30 is " & _
    Verastream_Session.GetStringAtOffset(1487, 8))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

INSERTSTRINGATROWCOLUMN METHOD

Used to insert a string into the current entity on the Verastream server starting at the given row and column.

Syntax

```
object.InsertStringAtRowColumn String, Row, Column
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| String | The text string to insert on the entity at the given offset. |
| Row | The row number on the entity. |
| | |

| Part | Description |
|--------|----------------------------------|
| Column | The column number on the entity. |

Remarks

Use the `InsertStringAtRowColumn` method to put strings where attributes or recordset fields have not been defined.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Verastream_Session.InsertStringAtRowColumn "test name", 19, 48
MsgBox ("The string at row 19 column 48 is " & _
    Verastream_Session.GetStringAtRowColumn(19, 48, 1, 12))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing`
```

ISCONNECTED PROPERTY

Returns whether or not the object is connected to a Verastream session.

Syntax

```
object.IsConnected
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the `IsConnected` property to determine if the object is connected to a Verastream session.

Example

```

Dim Verastream_Session As AppConnModel
Dim strServerName, strModelName, strUserID, strPassword As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"

Set ModelVars = New AppConnStringMap

strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

See also

- [ConnectToModel](#)
- [ConnectToModelViaDomain](#)
- [ConnectToSession](#)
- [ConnectToSessionViaDomain](#)
- [SuspendConnection](#)
- [ResumeConnection](#)
- [Disconnect](#)

ISSECURECONNECTION PROPERTY

Returns whether or not the connection to a Verastream session is encrypted.

Syntax

```
object.IsSecureConnection
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the IsSecureConnection property to determine if the connection to a Verastream session is encrypted.

MODELVERSIONSTRING PROPERTY

Gets the version string for the current model.

Syntax

```
object.ModelVersionString
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

The string returned uniquely identifies the version of the model deployed to the Host Integrator Server.

MOVECURRENTRECORDINDEX METHOD

Moves the record index for a Verastream recordset.

Syntax

```
object.MoveCurrentrecordIndex Movement
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|----------|---|
| Movement | An enumeration value defined below indicating the scroll movement |

Remarks

Use the MoveCurrentRecordIndex method to move the index in a Verastream recordset. The enumeration values for AppConnScrollMovement are defined in the Type Library.

Scroll Movement values - ScrollHome, ScrollEnd, ScrollLineUp, ScrollLineDown, ScrollPageUp and ScrollPageDown

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strAttrName, strAttrValue As String
Dim modRecord As AppConnModelRecord
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Set Attributes = Nothing

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Verastream_Session.MoveCurrentRecordIndex ScrollPageDown
Set modRecord = Verastream_Session.GetCurrentRecord

For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next
Set modRecord = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
Set modRecord = Nothing
```

NEXTCORD METHOD

Moves the record index for a Verastream recordset to the next record that matches the filter expression and retrieves that record.

Syntax

```
Set Record = object.NextRecord([FilterExpression])
```

| Part | Description |
|------------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| FilterExpression | [optional] An expression used to filter which record to get |

| Part | Description |
|--------|---|
| Record | A record with the fields names and values |

Remarks

Use the NextRecord method to set the record index in a Verastream recordset to the next record that matches the filter expression (if no filter expression is given, it moves to the next record) and retrieve that record. If no record is found a record with index of -1 is returned.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName, strAttrName, _
    strAttrValue, strFilter As String
Dim modRecord As AppConnModelRecord
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Set Attributes = Nothing

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

' this gets the first record in the record set since there is no filter criteria
Set modRecord = Verastream_Session.NextRecord
For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next

' this gets the next record based on the filter expression
strFilter = "AcctTransactions.Code = E6"
Set modRecord = Verastream_Session.NextRecord(strFilter)
For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next
Set modRecord = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

PERFORMAIDKEY METHOD

Used to enter the aid key (for example, a Program Function key) into the current Verastream server session.

Syntax

```
object.PerformAidKey AidKey
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|---|
| AidKey | The AppConnAidKeyenumerated value, like IBM_Enter for the Enter key, that represents an aidkey that will be executed on the host system. The enumeration values forAppConnAidKey enumeration are defined in the Type Library. |

Remarks

Use the PerformAidKey method to have the host system process the selected aid key.

Example

```

On Error GoTo error_Handler

Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "SuperPool"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToSession strServerName, strModelName
Verastream_Session.PerformEntityOperation "SetCursor"
Verastream_Session.InsertStringAtCursor "tst"
Verastream_Session.PerformAidKey IBM_Tab_Key
Verastream_Session.InsertStringAtCursor "testpass"
Verastream_Session.PerformAidKey IBM_Enter_Key

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Exit Sub

error_Handler:
MsgBox "Verastream error", vbExclamation, "Verastream Error"

```

PERFORMENTITYOPERATION METHOD

Used to perform an entity operation.

Syntax

```
object.PerformEntityOperation OperationName
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------------|--|
| OperationName | The name of the operation to performed on the current entity |

Remarks

Use the InsertRecord method to perform an operation defined for an entity.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strOperationName, strCurrentEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strOperationName = "ToMainMenu"
Verastream_Session.PerformEntityOperation strOperationName
strCurrentEntity = Verastream_Session.GetCurrentEntity
MsgBox ("The current entity is " & strCurrentEntity)

Verastream_Session.Disconnect
```

PERFORMTABLEPROCEDURE METHOD

Used to perform a procedure on the Verastream server.

Syntax

```
Set RecordSet = object.PerformTableProcedure(TableName, ProcedureName,
[DataInputValues], [FilterValues], [FilterIsCaseSensitive], [OutputColumnNames],
[MaxRows])
```

| Part | Description |
|-----------------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| TableName | The name of the table defined in Verastream |
| ProcedureName | The name of the procedure for the table |
| DataInputValues | [optional] An AppConnStringMap with name/value pairs for the procedure inputs |
| FilterValues | [optional] An AppConnStringMap with name/value pairs for the procedure filters |
| FilterIsCaseSensitive | [optional] An indicator whether or not the filter is case sensitive |
| OutputColumnNames | [optional] An AppConnStringList with the names of the columns to retrieve |
| MaxRows | [optional] The maximum number of rows that will be fetched |

| Part | Description |
|-----------|---|
| RecordSet | The set of records returned for the procedure |

Remarks

Use the PerformTableProcedure method to execute a procedure.

Example

```
Dim Verastream_Session As AppConnTable
Dim strModelName, strServerName, strTableName, strProcedureName As String
Dim strMap As AppConnStringMap
Dim ReturnRS As AppConnRecordSet

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnTable
Verastream_Session.ConnectToModel strServerName, strModelName

Set strMap = New AppConnStringMap
Set ReturnRS = New AppConnRecordSet

strTableName = "Transactions"
strProcedureName = "GetTransactions"
strMap.Add "AcctNumber", 167439459

Set ReturnRS = Verastream_Session.PerformTableProcedure(strTableName, strProcedureName, Null, strMap)

Dim VHIRecord As AppConnRecord
For i = 1 To ReturnRS.Count
    Set VHIRecord = ReturnRS.Item(i)
    MsgBox (VHIRecord!Date)
Next
```

PROCESSTRING METHOD

Calls the ProcessString event handler on the server.

Syntax

```
StringA = object.ProcessString(StringB)
```

| Part | Description |
|---------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| StringB | The inputstring to be processed by the ProcessString event handler. |

| Part | Description |
|---------|--|
| StringA | The results returned by the ProcessString event handler. |

REQUIRESECURECONNECTION METHOD

This method is kept for backwards compatibility. Connections to a Host Integrator Server are always encrypted.

Syntax

```
object.RequireSecureConnection require
```

| Part | Description |
|---------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| require | True or False |

RESUMECONNECTION METHOD

Resumes the connection to a Verastream session that has been suspended earlier.

Syntax

```
object.ResumeConnection(ConnectionToken)
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-----------------|--|
| ConnectionToken | The token that was returned from theSuspendConnection method |

Remarks

Use the ResumeConnection method to resume a Verastream session where state has been maintained.

Example

```

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.RequireSecureConnection True

Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

' The session token would be written out to a session variable, a cookie or maybe a DB
strTimeout = "1"
strConnectionToken = Verastream_Session.SuspendConnection(strTimeout)
MsgBox ("The Verastream session is suspended and the token value is " _
    & strConnectionToken)

' The session token would be fetched from where it was stored
Verastream_Session.ResumeConnection (strConnectionToken)
MsgBox ("The Verastream connection has been resumed")

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

See also

- [ConnectToModel](#)
- [ConnectToModelViaDomain](#)
- [ConnectToSession](#)
- [ConnectToSessionViaDomain](#)
- [SuspendConnection](#)
- [Disconnect](#)

SELECTCURRENTRECORD METHOD

Used to perform the select operation defined in Verastream model for the current record in the current recordset.

Syntax

```
object.SelectCurrentRecord
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the SelectCurrentRecord method to perform the select operation defined in a Verastream record set where the host application has select functionality defined for that recordset. To determine if the record set is selectable, check the Operations definitions for a record set defined within the modeling tool.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strIndex As String
Dim Attributes As AppConnStringMap
Dim modRecord As AppConnModelRecord

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Verastream_Session.SetCurrentRecordIndex (1)
Verastream_Session.SelectCurrentRecord
For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next
Set modRecord = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SELECTRECORDBYFILTER METHOD

Used to perform the select operation on the first record in the current recordset which matches the filter expression.

Syntax

```
object.SelectRecordByFilter FilterExpression
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|------------------|--|
| FilterExpression | An expression used to filter which record will be selected |

Remarks

Use the `SelectRecordByFilter` method to perform the select operation defined in Verastream on the first record that matches the filter expression where the host application has select functionality defined for that recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, _
    strAttrName, strAttrValue, strFilterExpression As String
Dim Attributes As AppConnStringMap
Dim modRecord As AppConnModelRecord

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

strFilterExpression = "AcctTransData.Code=""B6"""
MsgBox (strFilterExpression)
Verastream_Session.SelectRecordByFilter strFilterExpression
Set modRecord = Verastream_Session.GetCurrentRecord

For I = 1 To modRecord.Count
    MsgBox (modRecord.ElementNames(I) & " has a value of " & modRecord.Item(I))
Next
Set modRecord = Nothing

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SELECTRECORDBYINDEX METHOD

Used to perform the select operation on the first record in the current recordset for the given index.

Syntax

```
object.SelectCurrentRecord Index
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-------|-------------------------|
| Index | The index of the record |

Remarks

Use the `SelectRecordByIndex` method to perform the select operation on the record for the given index defined in a Verastream record set where the host application has select functionality defined for that recordset. To determine if the record set is selectable, check the Operations definitions for a record set defined within the modeling tool.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, _
    strAttrName, strAttrValue, strIndex As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
strIndex = "1"
Verastream_Session.SelectRecordByIndex (strIndex)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SERVERNAME PROPERTY

Returns the Verastream server name for the current session.

Syntax

```
object.ServerName
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

Remarks

Use the `ServerName` property to get the name of Verastream server for the current session. The server name is available even though a server name was not used to connect to the session (for example, `ConnectToModelViaDomain`).

Example

```

Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, strTempString As String
Dim ModelVars As AppConnStringMap
strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

strTempString = Verastream_Session.ServerName
MsgBox ("The Verastream server is named " & strTempString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

See also

[GetSessionID](#)

[ModelName](#)

SETLOCALE METHOD

Sets the Locale for the AppConn object.

Syntax

```
object.SetLocale Locale
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the SetLocale method to set the locale of the AppConn object, which may be retrieved with the GetLocale method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, strLocale, strTempString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

strLocale = "en_us"
Verastream_Session.SetLocale (strLocale)

strTempString = Verastream_Session.GetLocale
MsgBox ("The Verastream server locale is " & strTempString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[GetLocale](#)

SETLOGGINGLEVEL METHOD

Used to set the logging level for the session.

Syntax

```
object.SetLoggingLevel LoggingLevel
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |

| Part | Description |
|--------------|---|
| LoggingLevel | A logging level (i.e. Errors,ErrorsAndWarnings, or All) |

Remarks

Use the SetLoggingLevel method to set the logging level for a session.

Example

```

On Error GoTo error_Handler

Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Verastream_Session.SetLoggingLevel All
MsgBox ("The logging level for the Verastream Server is =" _
& Verastream_Session.GetLoggingLevel)

Verastream_Session.SetLoggingLevel Errors
MsgBox ("The logging level for the Verastream Server is =" _
& Verastream_Session.GetLoggingLevel)

Verastream_Session.SetLoggingLevel ErrorsAndWarnings
MsgBox ("The logging level for the Verastream Server is =" _
& Verastream_Session.GetLoggingLevel)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Exit Sub

error_Handler:
MsgBox "Verastream error", vbExclamation, "Verastream Error"

```

See also

[SetLoggingLevel](#)

SESSIONTYPE PROPERTY

Gets the terminal emulation type of the host session.

Syntax

```
object.SessionType
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |

Remarks

Use the SessionType property to get the terminal emulation type of the host session. The enumeration values for AppConnSessionType are defined in the Type Library.

| |
|--------------------|
| SessionType |
| SessionType3270 |
| SessionType5250 |
| SessionTypeHP |
| SessionTypeVT |

SETATTRIBUTES METHOD

Used to set attributes on the current entity.

Syntax

```
object.SetAttributes Attributes
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|------------|---|
| Attributes | An AppConnStringMap with name/valuepairs for the attributes |

Remarks

Use the SetAttributes method to set the attributes of the host application.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim strEntityName As String
Dim strAttrName, strAttrValue As String

strModelName = "CCSDemo"
strServerName = "localhost"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"

Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SETATTRIBUTESDELAYED METHOD

Used to set attributes for the named entity when that entity is reached.

Syntax

```
object.SetAttributesDelayed Attributes, EntityName
```

| Part | Description |
|------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| Attributes | An AppConnStringMap with name/valuepairs for the attributes |

| Part | Description |
|------------|------------------------|
| EntityName | The name of the entity |

Remarks

Use the `SetAttributesDelayed` method to set the attributes of the host application.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, _
    strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
strEntityName = "CustInquiryPanel"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue

Verastream_Session.SetAttributesDelayed Attributes, strEntityName
Verastream_Session.SetCurrentEntity strEntityName

strEntityName = "AcctProfile"
Verastream_Session.SetCurrentEntity strEntityName

strEntityName = Verastream_Session.GetCurrentEntity
MsgBox ("The current entity in the host session is " & strEntityName)
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SETCURRENTENTITY

Used to traverse to an entity (for example, screen) of the host application.

Syntax

```
object.SetCurrentEntity EntityName
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|------------|------------------------|
| EntityName | The name of the entity |

Remarks

Use the SetCurrentEntity method to traverse to the named entity. Reasons for failure include:

- Server session has not been established
- Invalid entity name
- No traversal path to entity

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntityName As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntityName = "CustInquiryPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Verastream_Session.SetCurrentEntity strEntityName
If StrComp(Verastream_Session.GetCurrentEntity, strEntityName) <> 0 Then
    MsgBox ("Set Current Entity Error")
End If

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SETCURRENTRECORDINDEX METHOD

Used to set the index of the current recordset.

Syntax

```
object.SetCurrentRecordIndex Index
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|-------|-------------------------|
| Index | The index of the record |

Remarks

Use the `SetCurrentRecordIndex` method to set the record index for the current recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strIndex As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName
Verastream_Session.SetCurrentRecordIndex (1)
strIndex = Verastream_Session.GetCurrentRecordIndex
MsgBox ("Current record index = " & strIndex)
Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SETCURRENTRECORDSETBYNAME METHOD

Used to set the current recordset by name.

Syntax

```
object.SetCurrentRecordSetByName RecordSetName
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------------|---------------------------|
| RecordSetName | The name of the recordset |

Remarks

Use the `SetCurrentRecordSetByName` method to set the current recordset.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity As String
Dim strAttrName, strAttrValue, strRecordset, strRecordsetOut As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes

strEntityName = "AcctTransactions"
Verastream_Session.SetCurrentEntity strEntityName

strRecordset = "AcctTransData"
Verastream_Session.SetCurrentRecordSetByName strRecordset

strRecordsetOut = Verastream_Session.GetCurrentRecordSetName
MsgBox ("Current record set is " & strRecordsetOut)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
```

SETMETHODTIMEOUT METHOD

Sets the timeout value set on the AppConn connection.

Syntax

```
object.SetMethodTimeout Timeout
```

| Part | Description |
|--------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|---------|--|
| Timeout | The amount of time (milliseconds) methods will process before timing out |

Remarks

Use the `SetMethodTimeout` method to set the timeout for methods of the `AppConn` object, which may be retrieved with the `GetMethodTimeout` method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strServerName, strDomainName, strModelName, strUserID, _
    strPassword, strMethodTimeout, strTempString As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
strDomainName = "localhost"

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

strMethodTimeout = "20000"
Verastream_Session.SetMethodTimeout (strMethodTimeout)
strTempString = Verastream_Session.GetMethodTimeout
MsgBox ("The Verastream server method timeout is " & strTempString)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing
```

See also

[GetMethodTimeout](#)

SETMODEL VARIABLES METHOD

Used to set model variables.

Syntax

```
object.SetModelVariables Variables
```

| Part | Description |
|--------|---|
| object | An <code>AppConn</code> object (for example, <code>AppConnModel</code> , <code>AppConnTerm</code> , and <code>AppConnTable</code>) |

| Part | Description |
|-----------|--|
| Variables | An AppConnStringMap with name/valuepairs for the variables |

Remarks

Use the SetModelVariables method to set model variables.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName As String
Dim ModelVars As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set ModelVars = New AppConnStringMap
ModelVars.Clear
ModelVars.Add "userID", "test1"
ModelVars.Add "password", "test2"
Verastream_Session.SetModelVariables ModelVars
ModelVars.Clear

Set ModelVars = Verastream_Session.GetModelVariables

For I = 1 To ModelVars.Count
    MsgBox ("Model variable " & ModelVars.Keys(I) & " " & ModelVars.Item(I))
Next

Verastream_Session.Disconnect
Set Verastream_Session = Nothing`
```

SUSPENDCONNECTION METHOD

Suspends the connection to a Verastream session so that the connection to that session can be resumed later.

Syntax

```
ConnectionToken = object.SuspendConnection(Timeout)
```

| Part | Description |
|---------|---|
| object | An AppConn object (for example, AppConnModel, AppConnTerm,and AppConnTable) |
| Timeout | The amount of time (minutes) the suspended connection willbe held suspended |

| Part | Description |
|-----------------|--|
| ConnectionToken | The token used to resume the connection with ResumeConnection method |

Remarks

Use the SuspendConnection method to maintain the state of a Verastream session while an AppConn connector is persisted with the intent that the connection to that session will be resumed.

Example

```

Set ModelVars = New AppConnStringMap
strUserID = ""
strPassword = ""
ModelVars.Add "userID", "bjones"
ModelVars.Add "password", "bjones"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName, _
    strUserID, strPassword, ModelVars
If Not Verastream_Session.IsConnected Then
    MsgBox ("Verastream Not Connected")
Else
    MsgBox ("Verastream Connected")
End If

' The session token would be written out to a session variable, a cookie or maybe a DB
strTimeout = "1"
strConnectionToken = Verastream_Session.SuspendConnection(strTimeout)
MsgBox ("The Verastream session is suspended and the token value is " _
    & strConnectionToken)

' The session token would be fetched from where it was stored
Verastream_Session.ResumeConnection (strConnectionToken)
MsgBox ("The Verastream connection has been resumed")

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set ModelVars = Nothing

```

See also

- [ConnectToModel](#)
- [ConnectToModelViaDomain](#)
- [ConnectToSession](#)
- [ConnectToSessionViaDomain](#)
- [ResumeConnection](#)
- [Disconnect](#)

UPDATECURRENTRECORD METHOD

Used to update the current record in a Verastream recordset.

Syntax

```
object.UpdateCurrentRecord Record
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| Record | An AppConnStringMap with name/valuepairs for the record fields |

Remarks

Use the UpdateCurrentRecord method to update the current record in a Verastream recordset.

UPDATERECORDBYFILTER METHOD

Used to update a record in a Verastream recordset that matches a filter condition.

Syntax

```
Updated = object.UpdateRecordByFilter(Record, FilterExpression)
```

| Part | Description |
|------------------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| Record | An AppConnStringMap with name/valuepairs for the record fields |
| FilterExpression | An expression used to filter which record will be selected |
| Update | Indicator whether or not a record is updated |

Remarks

Use the UpdateRecordByFilter method to update the first record in a Verastream recordset that matches a filter condition.

UPDATERECORDBYINDEX METHOD

Used to update a record in a Verastream recordset for the given index.

Syntax

```
Updated = object.UpdateRecordByIndex(Record, Index)
```

| Part | Description |
|--------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |

| Part | Description |
|--------|--|
| Record | An AppConnStringMap with name/valuepairs for the record. |
| Index | The index of a record |
| Update | Indicator whether or not a record is updated |

Remarks

Use the UpdateRecordByIndex method to update a record in a Verastream recordset for a given index.

UPDATERECORDS METHOD

Used to update records in a Verastream recordset match a filter condition.

Syntax

```
NumRecordsUpdated = object.UpdateRecords (Record, FilterExpression)
```

| Part | Description |
|------------------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| Record | An AppConnStringMap with name/valuepairs for the record fields |
| FilterExpression | An expression used to filter which record will be selected |

| Part | Description |
|-------------------|---|
| NumRecordsUpdated | Then number of records that wereupdated |

Remarks

Use the UpdateRecords method to update records in a Verastream recordset that match the filter expression.

WAITFORCONDITION METHOD

Used to wait for the given conditions to be met on the given entity.

Syntax

```
object.WaitForCondition Timeout, [Expression], [EntityName]
```

| Part | Description |
|------------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, AppConnTable, and AppConnSessionEx) |
| Timeout | The amount of time to wait for theconditions to be meet |
| Expression | [optional] A conditionalexpression |
| EntityName | [optional] The name of theentity |

Remarks

Use the WaitForCondition method to wait for conditions to be met on the given entity. If the entity name is not supplied the current entity will be assumed. If the timeout expires before the conditions are met an error will be raised.

WAITFORENTITYCHANGE METHOD

Used to wait for the host application to reach an entity.

Syntax

```
object.WaitForEntityChange EntityName, Timeout
```

| Part | Description |
|------------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| EntityName | The name of the entity |

| Part | Description |
|---------|---|
| Timeout | The amount of time in seconds to wait before timing out |

Remarks

Use the `WaitForEntityChange` method to synchronize session actions by waiting for the host to reach an entity.

WAITFORCURSOR METHOD

Used to synchronize a host application by waiting for the terminal cursor at the given row and column.

Syntax

```
object.WaitForCursor RowNum, ColumnNum, Timeout
```

| Part | Description |
|-----------|--|
| object | An AppConn object (for example, AppConnModel, AppConnTerm, and AppConnTable) |
| RowNum | The row on the entity (for example, screen) |
| ColumnNum | The column on the entity (for example, screen) |

| Part | Description |
|---------|---|
| Timeout | The amount of time in seconds to waitfor the cursor to reach the given position |

Remarks

Use the WaitForCursor method to synchronize processing with a host session.

Example

```

On Error GoTo error_Handler

Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName As String
Dim Verastream_StringMap As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Verastream_Session.WaitForCursor 19, 45, 1 ' will error out
Verastream_Session.WaitForCursor 19, 48, 1 ' will succeed

MsgBox ("Cursor it at the correct location.")

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Exit Sub

error_Handler:
MsgBox "The cursor is not in the correct location after 1 second.", _
vbExclamation, "Verastream Error"

```

WAITFORSTRING METHOD

Used to synchronize a host application by waiting for a string at the given row and column.

Syntax

```
object.WaitForString String, RowNum, ColumnNum, Timeout
```

| Part | Description |
|-----------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| String | The text string to wait toappear |
| RowNum | The row on the entity (for example,screen) |
| ColumnNum | The column on the entity (forexample, screen) |

| Part | Description |
|---------|---|
| Timeout | The amount of time in seconds to waitfor the cursor to reach the given position |

Remarks

Use the WaitForString method to synchronize processing with a host session.

Example

```
Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName, strField As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strField = "Consolidated" ' this will work
strField = "Consolidated2" ' this will throw an error
Verastream_Session.WaitForString strField, 17, 25, 1
MsgBox ("WaitForString completed successfully!")

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Exit Sub

error_Handler:
MsgBox "String " & strField & " Not Found", vbExclamation, "Verastream Error"
```

WAITFORSTRINGRELCURSOR METHOD

Used to synchronize a host application by waiting for a string at an offset relative to the cursor position.

Syntax

```
object.WaitForStringRelCursor String, RowOffset, ColumnOffset, Timeout
```

| Part | Description |
|--------------|---|
| object | An AppConn object (for example,AppConnModel, AppConnTerm, and AppConnTable) |
| String | The text string to wait toappear |
| RowOffset | The offset from the cursor for the row on the entity |
| ColumnOffset | The offset from the cursor for the column on the entity |

| Part | Description |
|---------|---|
| Timeout | The amount of time in seconds to waitfor the cursor to reach the given position |

Remarks

Use the WaitForStringRelCursor method to synchronize processing with a host session.

Example

```

On Error GoTo error_Handler`

Dim Verastream_Session As AppConnSessionEx
Dim strModelName, strServerName, strField As String

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strField = "Userid" ' this will work
strField = "Userid2" ' this will throw an error

' On the same row as the cursor and 18 columns to the left
Verastream_Session.WaitForStringRelCursor strField, 0, -18, 1
MsgBox ("WaitForStringRelCursor completed successfully!")

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Exit Sub

error_Handler:
MsgBox "String " & strField & " Not Found", vbExclamation, "Verastream Error"

```

STRINGLIST OBJECT

AppConnStringList Object

Use the AppConnStringList object to manage lists of strings. This object is similar to the AppConnStringMap except the AppConnString map deals with name/value pairs instead of single strings.

Click a method to see more information on its use, syntax, and parameters:

[Add](#)

[Clear](#)

[Clone](#)

[Count](#)

[GetLocale](#)

[Item](#)

[Remove](#)

[SetLocale](#)

ADD METHOD

Used to add an element to an AppConnStringList.

Syntax

```
object.Add String
```

| Part | Description |
|-------------|----------------------|
| object | An AppConnStringList |
| | |

| Part | Description |
|--------|---------------|
| String | A text string |

Remarks

Use the Add method to add an element to an AppConnStringList.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
MsgBox ("There are " & AttributeList.Count & " attributes on the entity.")

AttributeList.Add ("NewAttributeName")
MsgBox ("There are now " & AttributeList.Count & " attributes on the entity.")

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

CLEAR METHOD

Used to remove all elements from an AppConnStringList.

Syntax

```
object.Clear
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |

Remarks

Use the Clear method to remove all of the elements from an AppConnStringList.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)

MsgBox ("There are " & AttributeList.Count & " attributes on the entity.")
AttributeList.Clear
MsgBox ("There are now " & AttributeList.Count & " attributes in the list.")

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

CLONE METHOD

Used to clone an AppConnStringList.

Syntax

```
Set Copy = object.Clone
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |
| | |

| Part | Description |
|------|---------------------------------|
| Copy | A copy of the AppConnStringList |

Remarks

Use the Clone method to create a clone of an AppConnStringList.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList, AttributeListClone As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
MsgBox ("There are " & AttributeList.Count & " attributes in AttributeList.")

Set AttributeListClone = AttributeList.Clone
MsgBox ("There are now " & AttributeListClone.Count & _
    " attributes in AttributeListClone.")

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set AttributeListClone = Nothing
Set Verastream_Session = Nothing
```

COUNT PROPERTY

Used to get the number of elements in the AppConnStringList.

Syntax

```
object.Count
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |

Remarks

Use the Count property to get the number of elements in an AppConnStringList.

Example

```

Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName
Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)

MsgBox ("There are " & AttributeList.Count & " attributes on the entity.")

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing

```

GETLOCALE METHOD

Retrieves the Locale for the AppConnStringList.

Syntax

```
Locale = object.GetLocale
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |
| | |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the GetLocale method to get the locale of the AppConnStringList, which may be set with the SetLocale method.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
AttributeList.SetLocale ("en_us")
MsgBox ("The locale is " & AttributeList.GetLocale)

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

ITEM PROPERTY

Used to get and set an element in an AppConnStringList.

Syntax

```
object.Item(Index) or object.Item("Name")
```

```
object.(Index) or object.("Name")
```

```
object!Name
```

| Part | Description |
|--------|---|
| object | An AppConnStringList |
| Index | The name or index(starting at 1) of an element in the AppConnStringList |

| Part | Description |
|------|---|
| Name | The name of an element in the AppConnStringList |

Remarks

Use the Item property to get or set an element in an AppConnStringList.

Example

```

Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
MsgBox ("The first attribute is " & AttributeList.Item(1))
MsgBox ("The first attribute is " & AttributeList.Item("UserID"))
MsgBox ("The first attribute is " & AttributeList(1))
MsgBox ("The first attribute is " & AttributeList("UserID"))
MsgBox ("The first attribute is " & AttributeList!UserID)

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing

```

REMOVE METHOD

Used to remove an element from an AppConnStringList.

Syntax

```
object.Remove Index
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |

| Part | Description |
|-------|---|
| Index | The name or index(starting at 1) of an element in the AppConnStringList |

Remarks

Use the Remove method to add an element to an AppConnStringList.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
MsgBox ("There are " & AttributeList.Count & " attributes in AttributeList.")
AttributeList.Remove (1) ' Removes the first attribute in the list
MsgBox ("There are " & AttributeList.Count & " attributes in AttributeList.")

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

SETLOCALE METHOD

Sets the Locale for the AppConnStringList.

Syntax

```
object.SetLocale Locale
```

| Part | Description |
|--------|----------------------|
| object | An AppConnStringList |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the SetLocale method to set the locale of the AppConnStringList, which may be retrieved with the GetLocale method.

Example

```
Dim Verastream_Session As AppConnModel
Dim AttributeList As AppConnStringList
Dim strModelName, strServerName, strEntity As String
Dim I As Integer

strModelName = "CCSDemo"
strServerName = "localhost"
strEntity = "SignonPanel"

Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

Set AttributeList = Verastream_Session.GetEntityAttributes(strEntity)
AttributeList.SetLocale ("en_us")
MsgBox ("The locale is " & AttributeList.GetLocale)

Verastream_Session.Disconnect
Set AttributeList = Nothing
Set Verastream_Session = Nothing
```

STRINGMAP OBJECT

AppConnStringMap Object

Use the AppConnStringMap object to manage maps of strings. This object is similar to the AppConnStringList except the AppConnString map deals with name/value pairs. This object is mainly used with relation to setting or getting the Attributes of a model.

Click a method to see more information on its use, syntax, and parameters:

- [Add](#)
- [Clear](#)
- [Clone](#)
- [Count](#)
- [GetLocale](#)
- [Item](#)
- [Keys](#)
- [Remove](#)
- [SetLocale](#)

ADD METHOD

Used to add an element to an AppConnStringMap.

Syntax

```
object.Add Name, Value
```

| Part | Description |
|--------|----------------------------|
| object | An AppConnStringMap object |
| Name | The name for an element |
| | |

| Part | Description |
|-------|--------------------------|
| Value | The value for an element |

Remarks

Use the Add method to add an element to an AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
MsgBox (AttributesClone.Item("AcctNumber"))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

CLEAR METHOD

Used to remove all elements from an AppConnStringMap.

Syntax

```
object.Clear
```

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |

Remarks

Use the Clear method to remove all of the elements from an AppConnStringMap.

Example


```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
MsgBox (Attributes.Item("AcctNumber"))

Attributes.Clear

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing

```

CLONE METHOD

Used to clone an AppConnStringMap.

Syntax

```
Set Copy = object.Clone
```

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |
| | |

| Part | Description |
|------|--------------------------------|
| Copy | A copy of the AppConnStringMap |

Remarks

Use the Clone method to create a clone of an AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes, AttributesClone As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Verastream_Session.SetAttributes Attributes
Set AttributesClone = Attributes.Clone
MsgBox (AttributesClone.Item("AcctNumber"))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
Set AttributesClone = Nothing
Count Property
Used to get the number of elements in the AppConnStringMap.
```

Syntax

`object.Count`

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |

Remarks

Use the Count property to get the number of elements in an AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes, AttributesClone As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
MsgBox ("count of records = " & Attributes.Count)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

GETLOCALE METHOD

Retrieves the Locale for the AppConnStringMap.

Syntax

```
Locale = object.GetLocale
```

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |
| | |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the GetLocale method to get the locale of the AppConnStringMap, which may be set with the SetLocale method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.SetLocale ("en_us")
MsgBox ("Locale is = " & Attributes.GetLocale)
Attributes.Add strAttrName, strAttrValue

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

ITEM PROPERTY

Used to get an element in an AppConnStringMap.

Syntax

```
object.Item(Index)
```

```
object.Item("Name")
```

```
object(Index)
```

```
object("Name")
```

```
object!Name
```

| Part | Description |
|--------|--|
| object | An AppConnStringMap |
| Index | The name or index(starting at 1) of an element in the AppConnStringMap |

| Part | Description |
|------|--|
| Name | The name of an element in the AppConnStringMap |

Remarks

Use the Item property to get or set an element in a string AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
MsgBox ("Account Number = " & Attributes.Item(1))
MsgBox ("Account Number = " & Attributes.Item("AcctNumber"))
MsgBox ("Account Number = " & Attributes(1))
MsgBox ("Account Number = " & Attributes("AcctNumber"))
MsgBox ("Account Number = " & Attributes!AcctNumber)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

KEYS PROPERTY

Method used to get a list of the keys of the elements in the string AppConnStringMap.

Syntax

```
Set Keys = object.Keys
```

| Part | Description |
|------|---|
| Keys | An AppConnStringList of the keys for the AppConnStringMap |

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |

Remarks

Use the Keys property to get a list of the keys in a string AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap
Dim Keys As AppConnStringList

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.Add strAttrName, strAttrValue
Set KeyList = Attributes.Keys
MsgBox ("key list is " & KeyList(1))

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
Set Keys = Nothing
```

REMOVE METHOD

Used to remove an element from a string AppConnStringMap.

Syntax

```
object.Remove Index
```

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |

| Part | Description |
|-------|---|
| Index | The name or index (starting at 1) of an element in the list |

Remarks

Use the Remove method to add an element to a string AppConnStringMap.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

Set Attributes = New AppConnStringMap
strAttrName = "AcctNumber"
strAttrValue = "167439459"
Attributes.Add strAttrName, strAttrValue
strAttrName = "AcctNumber2"
strAttrValue = "167439459"
Attributes.Add strAttrName, strAttrValue
MsgBox ("Attribute count = " & Attributes.Count)
Attributes.Remove (2)
MsgBox ("Attribute count = " & Attributes.Count)

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

SETLOCALE METHOD

Sets the Locale for the AppConnStringMap.

Syntax

```
object.SetLocale Locale
```

| Part | Description |
|--------|---------------------|
| object | An AppConnStringMap |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the `SetLocale` method to set the locale of the `AppConnStringMap`, which may be retrieved with the `GetLocale` method.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strEntity, strAttrName, strAttrValue As String
Dim Attributes As AppConnStringMap

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strEntityName = "CustInquiryPanel"
Verastream_Session.SetCurrentEntity strEntityName

strAttrName = "AcctNumber"
strAttrValue = "167439459"
Set Attributes = New AppConnStringMap
Attributes.SetLocale ("en_us")
MsgBox ("Locale is = " & Attributes.GetLocale)
Attributes.Add strAttrName, strAttrValue

Verastream_Session.Disconnect
Set Verastream_Session = Nothing
Set Attributes = Nothing
```

STRINGMAPSET OBJECT

AppConnStringMapSet Object

Click a method to see more information on its use, syntax, and parameters:

- [Add](#)
- [Clear](#)
- [Clone](#)
- [Count](#)
- [GetLocale](#)
- [Item](#)
- [Remove](#)
- [SetLocale](#)

ADD METHOD

Used to add an element to an `AppConnStringMapSet`.

Syntax

```
object.Add StringMap
```


| Part | Description |
|------------------|-------------------------|
| object | A string map set object |
| AppConnStringMap | A string map |

Remarks

Use the Add method to add an element to an AppConnStringMapSet.

CLEAR METHOD

Used to remove all elements from an AppConnStringMapSet.

Syntax

```
object.Clear
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |

Remarks

Use the Clear method to remove all of the elements from an AppConnStringMapSet.

CLONE METHOD

Used to clone an AppConnStringMapSet.

Syntax

```
Set Copy = object.Clone
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |

| Part | Description |
|------|-----------------------------------|
| Copy | A copy of the AppConnStringMapSet |

Remarks

Use the Clone method to create a clone of an AppConnStringMapSet.

COUNT PROPERTY

Used to get the number of elements in the AppConnStringMapSet.

Syntax

```
object.Count
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |

Remarks

Use the Count property to get the number of elements in an AppConnStringMapSet.

GETLOCALE METHOD

Retrieves the Locale for the AppConnStringMapSet object.

Syntax

```
Locale = object.GetLocale
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |
| | |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the GetLocale method to get the locale of the AppConnStringMapSet object, which may be set with the SetLocale method.

ITEM PROPERTY

Used to get and set an element in an AppConnStringMapSet.

Syntax

```
object.Item(Index) [= AppConnStringMap]
```

```
object.(Index) [= AppConnStringMap]
```

| Part | Description |
|------------------|--|
| object | A string map set object |
| Index | The name or index(starting at 1) of an element in the list |
| AppConnStringMap | The AppConnStringMap used to set an element in the list |

Remarks

Use the Item property to get or set an element in an AppConnStringMapSet.

REMOVE METHOD

Used to remove an element from an AppConnStringMapSet.

Syntax

```
object.Remove Index
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |

| Part | Description |
|-------|--|
| Index | The name or index(starting at 1) of an element in the list |

Remarks

Use the Remove method to add an element to an AppConnStringMapSet.

SETLOCALE METHOD

Sets the Locale for the AppConnStringMapSet object.

Syntax

```
object.GetLocale Locale
```

| Part | Description |
|--------|-------------------------------|
| object | An AppConnStringMapSet object |

| Part | Description |
|--------|------------------------|
| Locale | The locale designation |

Remarks

Use the SetLocale method to set the locale of the AppConnStringMapSet object, which may be retrieved with the GetLocale method.

TERMINALATTRIBUTES OBJECT

AppConnTerminalAttributes Object

Use the AppConnTerminalAttributes object to get information on terminal attributes.

Click a method to see more information on its use, syntax, and parameters:

[Color](#)

[IsBlinking](#)

[IsColumnSeparated](#)

[IsHalfBrite](#)

[IsIntense](#)

[IsNondisplay](#)

[IsNumeric](#)

[IsPenDetect](#)

[IsProtected](#)

[IsReverse](#)

[IsUnderscore](#)

COLOR PROPERTY

Used to get the color value terminal attribute.

Syntax

```
object.Color
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the Color property to get the color terminal attribute.

ISBLINKING PROPERTY

Used to get the blinking terminal attribute.

Syntax

```
object.IsBlinking
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsBlinking property to get the blinking terminal attribute.

ISCOLUMNSEPARATED PROPERTY

Used to get the column separated terminal attribute.

Syntax

```
object.IsColumnSeparated
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsColumnSeparated property to get the comma separated terminal attribute.

ISHALFBRITE PROPERTY

Used to get the half brite terminal attribute.

Syntax

```
object.IsHalfBrite
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsHalfBrite property to get the half brite terminal attribute.

ISINTENSE PROPERTY

Used to get the intense terminal attribute.

**Syntax8*

```
object.IsIntense
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsIntense property to get the intense terminal attribute.

ISNONDISPLAY PROPERTY

Used to get the nondisplay terminal attribute.

Syntax

```
object.IsNondisplay
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsNondisplay property to get the nondisplay terminal attribute.

ISNUMERIC PROPERTY

Used to get the numeric terminal attribute.

Syntax

```
object.IsNumeric
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsNumeric property to get the numeric terminal attribute.

ISPENDETECT PROPERTY

Used to get the pen detect terminal attribute.

Syntax

```
object.IsPenDetect
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsPenDetect property to get the pen detect terminal attribute.

ISPROTECTED PROPERTY

Used to get the protected terminal attribute.

Syntax

```
object.IsProtected
```

[Part |Description| object |A terminal attributes object

Remarks

Use the IsProtected property to get the protected terminal attribute.

ISREVERSE PROPERTY

Used to get the reverse terminal attribute.

Syntax

```
object.IsReverse
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsReverse property to get the reverse terminal attribute.

ISUNDERScore PROPERTY

Used to get the underscore terminal attribute.

Syntax

```
object.IsUnderscore
```

| Part | Description |
|--------|------------------------------|
| object | A terminal attributes object |

Remarks

Use the IsUnderscore property to get the underscore terminal attribute.

TERMINAL FIELD OBJECT

TERMINALFIELD OBJECT

Use the TerminalField object to obtain information about the field.

Click a method to see more information on its use, syntax, and parameters:

[TopRow](#)

[LeftColumn](#)

[Offset](#)

[TerminalAttributes](#)

TOPROW PROPERTY

Used to get the location of the top row flag of the field.

Syntax

```
object.TopRow
```

| Part | Description |
|--------|-------------------------|
| object | A terminal field object |

Remarks

Use the TopRow property to get the location of the top row of a field.

LEFTCOLUMN PROPERTY

Used to get the location of the left column of the field.

Syntax

```
object.LeftColumn
```

| Part | Description |
|--------|-------------------------|
| object | A terminal field object |

Remarks

Use the LeftColumn property to get the location of the left column of a field.

OFFSET PROPERTY

Used to get the offset of the field.

Syntax

```
object.Offset
```

| Part | Description |
|--------|-------------------------|
| object | A terminal field object |

Remarks

Use the Offset property to get the offset for a field. Length Property Used to get the length of the field.

Syntax

```
object.Length
```

| Part | Description |
|--------|-------------------------|
| object | A terminal field object |

Remarks

Use the Length property to get the length for a field.

TERMINALATTRIBUTES PROPERTY

Used to get the terminal attributes for the field.

Syntax

```
object.TerminalAttributes
```

| Part | Description |
|--------|-------------------------|
| object | A terminal field object |

VARIABLEMETADATA OBJECT

VariableMetaData Object

Use the VariableMetaData object to manage Verastream model variable metadata.

Click a method to see more information on its use, syntax, and parameters:

[DefaultValue](#)

[Description](#)

[Initialization](#)

[IsEncrypted](#)

[IsHidden](#)

[IsReadable](#)

[IsWriteable](#)

[MetaDataType](#)

[Name](#)

[VariableType](#)

DEFAULTVALUE PROPERTY

Used to get the default value of a Verastream model variable.

Syntax

```
object.DefaultValue
```

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the DefaultValue property to get the default value of a Verastream model variable.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData =
Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata description is " & objVariableMetaData.Description)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing

```

DESCRIPTION PROPERTY

Used to get the description of a Verastream model variable.

Syntax

```
object.Description
```

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the Description property to get the description of a Verastream model Verastream model variable.

Example

```

Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata description value is " & objVariableMetaData.Description)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing

```

INITIALIZATION PROPERTY

Used to get the type initialization of the Verastream model variable.

Syntax

```
object.Initialization
```

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the Initialization property to get the initialization type of a Verastream model variable. The enumeration values for AppConnInitialization are defined in the Type Library.

| |
|-----------------------|
| Initialization |
| AtConnection |
| ByDefault |
| |

Initialization

WhenEncountered

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata initialization is " & objVariableMetaData.Initialization)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

ISENCRYPTED PROPERTY

Used to get the encrypted flag of the Verastream model variable.

Syntax

object.IsEncrypted

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the IsEncrypted property to get the encrypted flag of a Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata IsEncrypted is " & objVariableMetaData.IsEncrypted)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

ISHIDDEN PROPERTY

Used to get the hidden flag of the Verastream model variable.

Syntax

`object.IsHidden`

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the IsHidden property to get the hidden flag of a Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata IsHidden is " & objVariableMetaData.IsHidden)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

ISREADABLE PROPERTY

Used to get the readable flag of the Verastream model variable.

Syntax

`object.IsReadable`

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the IsReadable property to get the readable flag of a Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata IsReadable is " & objVariableMetaData.IsReadable)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

ISWRITEABLE PROPERTY

Return a flag indicating whether the Verastream model variable is writeable.

Syntax

```
object.IsWriteable
```

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the IsWriteable property to get the writeable flag of a Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata IsWriteable is " & objVariableMetaData.IsWriteable)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

METADATATYPE PROPERTY

Used to get the metadata type of the Verastream model variable.

Syntax

```
object.MetadataType
```

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the MetadataType property to get the type of metadata within a Verastream table. The enumeration values for AppConnMetadataType are defined in the Type Library.

| Metadata Types |
|----------------|
| AttributeMeta |

| Metadata Types |
|-----------------------|
| OperationMeta |
| RecordSetMeta |
| FieldMeta |
| VariableMeta |
| TableMeta |
| ColumnMeta |
| |

Metadata Types

ProcedureMeta

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata meta data type is " & objVariableMetaData.MetaDataType)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

NAME PROPERTY

Used to get the name of a Verastream model variable.

Syntax

object.Name

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the DefaultValue property to get the name of a Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata name is " & objVariableMetaData.Name)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

VARIABLETYPE PROPERTY

Used to get the type of the Verastream model variable.

Syntax

object.VariableType

| Part | Description |
|--------|----------------------------|
| object | A variable metadata object |

Remarks

Use the VariableType property to get the type of the Verastream model variable.

Example

```
Dim Verastream_Session As AppConnModel
Dim strModelName, strServerName, strVariableName As String
Dim objVariableMetaData As VariableMetaData

strModelName = "CCSDemo"
strServerName = "localhost"
Set Verastream_Session = New AppConnModel
Verastream_Session.ConnectToModel strServerName, strModelName

strVariableName = "cursorPosition"
Set objVariableMetaData = Verastream_Session.GetVariableMetaData(strVariableName)
MsgBox ("Variable metadata variable type is " & objVariableMetaData.VariableType)

Verastream_Session.Disconnect
Set objVariableMetaData = Nothing
Set Verastream_Session = Nothing
```

4.16.7 C Connector

Use the Host Integrator C connector to create C or non-COM C++ applications for Windows- or Linux-based development platforms that integrate host data into web applications or `client/server` applications.

The [interfaces](#) supported in the Host Integrator COM connector are not available in the AppConn C connector. Instead, the connector implements a session interface, which includes all of the methods included in the `IAppConnTable`, `IAppConnModel`, `IAppConnTerm`, and `IAppConnChannel` interfaces. Use the [CreateSession](#) method in the AppConn C connector to create a session object.

Intended Audience

This manual is written for developers who are familiar with developing C or C++ applications on Linux or Windows-based development platforms.

Getting Started

The C API requires inclusion of `appconnapi.h*` and `appconndef.h` in the source file. The file `appconnapi.h` contains the AppConn API function definitions, and `*appconndef.h` contains AppConn enum and constant definitions.

[Accessing host data](#)

[Building a C application](#)

[Error handling](#)

Linking C API Programs With the AppConn Shared Library

In order to link a C API program with an AppConn shared library, you must declare the library in the PATH variable. Declaring the PATH variable is handled differently on each platform.

ON WINDOWS

Programs using the AppConn C API must link to:

- `appconn.dll` which implements the AppConn C API functions. `appconn.dll` is installed in your `\Winnt\system32` directory
- `appconn.lib` which contains the linkage definitions necessary to use the AppConn C API functions in `appconn.dll`, is installed in your `<install directory>\lib` directory

ON LINUX

Programs on Linux that use the AppConn C API must link with the `libappconn.so` located where the AppConn C API functions are implemented.

When using the AppConn C API, the directory where the libappconn is located must be declared in the library variable. For example:

```
LD_LIBRARY_PATH=`{usr/local<install directory>/lib} (Linux)`  
export LD_LIBRARY_PATH (Linux)
```

The library file associated with the AppConn C API is installed in your <install directory>/lib directory.

Creating C API Objects

The C API uses C mechanisms at the interface level. C API objects are created with CreateXXX functions, which return a handle to the object. The handle is used as the first parameter to functions that perform methods on the object. The objects must be destroyed with ReleaseXXX functions. For example, use the following code to create a session object to interact with the Host Integrator Server:

```
SessionHandle hSess;  
hSess = CreateSession();  
ReleaseSession(hSess);  
...  
  
The C API functions use the following four basic types:  
  
- **int**- is used for boolean values.  
- **long**- is used for numeric values.  
- **CharType**- is used for string values and is defined as char *, but may also be defined as wchar_t in the future.  
- **Handles**- are defined for each of the C API objects. The handles are all defined as void *, so a handle can be assigned to any other handle type; however, the handle will be checked for the proper type whenever it is used in a functional call.  
  
!!! note  
CharType * and handle parameters that are not required may be 0 or null.  
  
```javascript  
int nRet;
nRet = ConnectToModel(hSess, "localhost", "CCSDemo",
"bjones", "bjones", 0);
if (nRet)
Disconnect(hSess, 0);
```

Strings are returned from the C API by passing a string buffer and the buffer's length into a function. The actual length of the AppConn string is returned. The C API function will only copy into the buffer up to the length that was given. If the buffer is 0 or null then nothing will be copied. If the size of the buffer that is being returned is known, then the C API function can be called with a static or preallocated buffer. If the size of the buffer is not known, the C API function can be called with a null buffer to get the length of the string. Then, a string buffer can be allocated of that length, and the C API function can be called with that allocated buffer. For example:

```
long nLen;
char szCurrentEntity[80];
char *pszCurrentEntity;
nLen = GetCurrentEntity(hSess, szCurrentEntity,
sizeof(szCurrentEntity));
nLen = GetCurrentEntity(hSess, 0, 0);
pszCurrentEntity = new char[nLen + 1];
GetCurrentEntity(hSess, pszCurrentEntity, nLen);
Delete [] pszCurrentEntity;
```

Handles to objects that are returned from C API functions must be released just like objects that are created explicitly. For example:

```
StringMapHandle hVariables;
hVariables = GetModelVariables(hSess);
ReleaseStringMap(hVariables);
```

## Error Handling with the C Connector

Errors in function calls are indicated by the return value. If the function returns an int, 0 indicates an error. If the function returns a long, -1 indicates an error. If the function returns a handle, 0 indicates an error. When an error is detected, use `GetNumMessages`, `GetErrorMessage`, and `GetLocalizedMessage` to access the error messages. `GetNumMessages` returns the number of the error message. `GetErrorMessage` and `GetLocalizedMessage` can be used to retrieve each error message. `GetLocalizedMessage` will return a localized message if localization is enabled, otherwise it will return the default error message.

```
long nMessageLen;
char *strMessage;
for (int i = 0; i < GetNumMessages(hSess); i++)
{
 nMessageLen = GetErrorMessage(hSess, i, 0, 0);
 strMessage = malloc(nMessageLen + 1);
 GetErrorMessage(hSess, i, strMessage, nMessageLen);
 printf("Apptrieve error: %s\n", strMessage);
 free(strMessage);
}
```

## FORK CONSIDERATIONS ON LINUX

There are restrictions to the use of the `fork()` system call when using the AppConn C for Linux connector. The `fork()` system call cannot be used while any AppConn session handles are held by the user's application. The system call may be used prior to creating a session with `CreateSession()` or after all session handles have been released with `ReleaseSession()`.

## Example

A C sample application, based on the Consolidated Credit Services (CCSDemo) mainframe application, is included in the AppConn C API installation; however, all the principles and methodology described in the sample application are equally applicable to any other host application.

The sample application, called CCSTutor, performs the following operations:

- Allows the user to connect to the server.
- Allows the user to login to the host.
- Provides the user with the ability to look up customers by their account number or name to review their account information.

For Windows, the complete source code of CCSTutor can be found in the `<VHI install folder>\examples\C-Sample` folder. For Linux, the complete source code of CCSTutor can be found in the `/usr/local/vhi/examples/C-Sample` directory. Use the command line to open the project called CCSTutor.cpp.

To access data using CCSTutor, use the following search criteria:

Userid: bjones

Password: bjones

Last name: smith

First initial: c

State: ri

Account number: 167439459

 **Note**

The data is case sensitive.

### **AppConn C Method Reference**

Click a link below for any AppConn C method.

AddStringMapToSet  
AddStringToList  
AddStringToMap  
ClearStringList  
ClearStringMap  
ClearStringMapSet  
ConnectToModel  
ConnectToModelViaDomain  
ConnectToSession  
ConnectToSessionViaDomain  
CreateRecord  
CreateRecordSet  
CreateSession  
CreateStringList  
CreateStringMap  
CreateStringMapSet  
Disconnect  
EnableTerminalAttributes  
ExecuteSQLStatement  
ExecuteSQLStatementWithMaxrows  
FetchRecords  
FieldAttributesEnabled  
GetAttributeAtCursor  
GetAttributeLocations  
GetAttributeLength  
GetAttributeMetaData  
GetAttributeReadVariable  
GetAttributes  
GetAttributesUsed  
GetAttributeWriteVariable  
GetColor  
GetColumnMax  
GetColumnMetaData



GetColumnMin  
GetColumnType  
GetConnectionTimeout  
GetCurrentEntity  
GetCurrentRecord  
GetCurrentRecordIndex  
GetCurrentRecordSetName  
GetDefaultValue  
GetDestination  
GetElementByIndex  
GetElementByName  
GetElementMetaDataByIndex  
GetElementMetaDataByName  
GetElementName  
GetElements  
GetElemLocAt  
GetElemLocElementName  
GetElemLocElementType  
GetElemLocLeftColumn  
GetElemLocLength  
GetElemLocNumColumns  
GetElemLocNumRows  
GetElemLocOffset  
GetElemLocRegionType  
GetElemLocTopRow  
GetEntityAttributes  
GetEntityDescription  
GetEntityOperations  
GetEntityRecordSets  
GetErrorCode  
GetErrorMessage  
GetErrorMessageInfo  
GetErrorMessageCode

GetErrorMessageParameterCount  
GetErrorMessageParameter  
GetErrorScreen  
GetErrorScreenColumns  
GetErrorScreenCursorColumn  
GetErrorScreenCursorPosition  
GetErrorScreenCursorRow  
GetErrorScreenEntityName  
GetErrorScreenRows  
GetErrorScreenFullText  
GetErrorScreenText  
GetErrorScreenTextRow  
GetFieldLength  
GetFieldLocations  
GetFieldMetaData  
GetFieldNames  
GetFilterColumns  
GetHomeEntityName  
GetInitialization  
GetInputColumns  
GetKey  
GetLastRequestID  
GetLocale  
GetLocalizedMessage  
GetLoggingLevel  
GetMetaDataDescription  
GetMetaDataOnlyFlag  
GetMajorVersion  
GetMetaDataName  
GetMetaDataType  
GetMethodTimeout  
GetMinorVersion  
GetModelEntities

GetModelName  
GetModelVariableNames  
GetModelVariables  
GetModelVersionString  
GetNumElements  
GetNumElemLocs  
GetNumMessages  
GetNumRecordElements  
GetNumRecords  
GetNumStringMaps  
GetNumStringPairs  
GetNumStrings  
ReplaceStringAt  
GetOperationMetaData  
GetOperationTimeout  
GetOutputColumns  
GetPatternLocations  
GetProcedureMetaData  
GetProcedureType  
GetRecord  
GetRecordIndex  
GetRecordSetLocations  
GetRecordSetMetaData  
GetScrollMovements  
GetScrollOperation  
GetServerName  
GetSessionID  
GetSessionType  
GetStringAt  
GetStringAtOffset  
GetStringAtRowColumn  
GetStringByIndex  
GetStringByKey

GetStringMapAt  
GetTableColumns  
GetTableDescription  
GetTableNames  
GetTableProcedures  
GetTermAttrsByIndex  
GetTermAttrsByName  
GetTermFieldLeftColumn  
GetTermFieldLength  
GetTermFieldOffset  
GetTermFieldTermAttr  
GetTermFieldTopRow  
GetTerminalFieldAtCursor  
GetVariableMetaData  
GetVariablesUsed  
GetVariableType  
GetVersionString  
InsertRecord  
InsertRecords  
InsertStringAtCursor  
InsertStringAtOffset  
InsertStringAtRowColumn  
IsAttributeReadable  
IsAttributeWriteable  
IsBlinking  
IsColumnKey  
IsColumnSeparated  
IsConnected  
IsDefaultOperation  
IsEncrypted  
IsFieldKey  
IsFieldReadable  
IsHidden

IsIntense  
IsNondisplay  
IsNumeric  
IsPenDetect  
IsProtected  
IsReadable  
IsRequiredFilter  
IsRequiredInput  
IsReverse  
IsSecureConnection  
IsUnderscore  
IsWriteable  
NextRecord  
MoveCurrentRecordIndex  
PerformAidKey  
PerformEntityOperation  
PerformTableProcedure  
ProcessString  
RecordFromXML  
RecordSetFromXML  
RecordSetToXML  
RecordToXML  
ReleaseElemLoc  
ReleaseElemLocList  
ReleaseErrorMessageInfo  
ReleaseErrorScreen  
ReleaseMetaData  
ReleaseRecord  
ReleaseRecordSet  
ReleaseSession  
ReleaseStringList  
ReleaseStringMap  
ReleaseStringMapSet

ReleaseTermField  
ReleaseTerminalAttributes  
RemoveStringAt  
RemoveStringByIndex  
RemoveStringByKey  
RemoveStringMapAt  
ReplaceStringAt  
ReplaceStringByIndex  
ReplaceStringByKey  
ReplaceStringMapAt  
RequireSecureConnection  
ResumeConnection  
SelectCurrentRecord  
SelectRecordByIndex  
SetAttributes  
SetAttributesDelayed  
SetConnectionTimeout  
SetCurrentEntity  
SetCurrentRecordIndex  
SetCurrentRecordSetByName  
SetLocale  
SetLoggingLevel  
SetMetaDataOnlyFlag  
SetMethodTimeout  
SetModelVariables  
SupportsDirectInserts  
SupportsSelect  
SuspendConnection  
TerminalAttributesEnabled  
UpdateCurrentRecord  
UpdateRecordByFilter  
UpdateRecordByIndex  
UpdateRecords

UsedForSQL

WaitForCondition

WaitForCursor

WaitForEntityChange

WaitForString

WaitForStringRelCursor

```

```

## CreateSession

Create an AppConn session.

```
SessionHandle CreateSession();
```

Return Value

A handle to a Host Integrator session.

---

```

```

## ConnectToModel

Method used to establish a connection to a Host Integrator Server and create or allocate a host session with the specified model. The model must be specified.

```
int ConnectToModel (SessionHandle hSession, const CharType *strServer, const
CharType *strModelName, const CharType *strUserID, const CharType
*strPassword, StringMapHandle hModelVariables);
```

### PARAMETERS

**hSession**

[in] The Host Integrator session.

**strServer**

[in] The name of the Host Integrator Server to connect to.

**strModelName**

[in] The name of the Host Integrator model to use.

**strUserID**

[in] The user ID that Host Integrator uses to authenticate the user.

#### **strPassword**

[in] The password that Host Integrator uses to authenticate the user.

#### **hModelVariables**

[in] An optional list of variable names paired with values (handle to a StringMap Object).

#### **Return Value**

A 1 is returned if the function succeeds, and a 0 is returned if the function does not succeed.

#### **Remarks**

Reasons for failure include:

- Server is not running.
- Invalid server address.
- Invalid model name.
- Invalid user ID or password.

See the [Error Handling](#) topic for details.

The parameters user ID and password are used by Host Integrator Server if the security option is ON.

---

```

```

## **ConnectToModelViaDomain**

Method used to establish a connection to a Host Integrator Server and create or allocate a host session with the specified model in the specified Host Integrator Domain via a Management server.

The model must be specified.

```
int ConnectToModelViaDomain (SessionHandle hSession, const CharType
*strDirectoryServer, const CharType *strDomainName, const CharType
*strModelName, const CharType *strUserID, const CharType *strPassword,
StringMapHandle hModelVariables);
```

### **PARAMETERS**

#### **hSession**

[in] The Host Integrator session.

#### **strDirectoryServer**



[in] The name of the Directory Server.

#### **strDomainName**

[in] The name of the Host Integrator Domain to use.

#### **strModelName**

[in] The name of the Host Integrator model to use.

#### **strUserID**

[in] The user ID that Host Integrator uses to authenticate the user.

#### **strPassword**

[in] The password that Host Integrator uses to authenticate the user.

#### **hModelVariables**

[in] An optional list of variable names paired with values (handle to a StringMap Object).

#### **Return Value**

A 1 is returned if the function succeeded, and 0 if the function did not succeed.

#### **Remarks**

Reasons for failure include:

- Server is not running.

- Invalid server address.

- Invalid model name.

- Invalid user ID or password.

See the [Error Handling](#) topic for details.

The parameters user ID and password are used by Host Integrator Server if the security option is ON.

## **ConnectToSession**

Method used to establish a connection to a Host Integrator Server and create or allocate a host session with the specified session. The session must be specified.

```
int ConnectToSession (SessionHandle hSession, const CharType *strServer, const
CharType *strSession, const CharType *strUserID, const CharType
*strPassword, StringMapHandle hModelVariables);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

### **strServer**

[in] The name of the Host Integrator Server to connect to.

### **strSession**

[in] The name of the Host Integrator session to use.

### **strUserID**

[in] The user ID that Host Integrator uses to authenticate the user.

### **strPassword**

[in] The password that Host Integrator uses to authenticate the user.

### **hModelVariables**

[in] An optional list of variable names paired with values (handle to a StringMap Object).

## Return Value

A 1 is returned if the function succeeds, and 0 is returned if the function does not succeed.

## Remarks

Reasons for failure include:

- Server is not running.

- Invalid server address.

- Invalid model name.

- Invalid user ID or password.

See the [Error Handling](#) topic for details.

The parameters user ID and password are used by the Host Integrator Server if the security option is ON.

## ConnectToSessionViaDomain

Method used to establish a connection to a Host Integrator Server and create or allocate a host session with the specified session in the specified Host Integrator Domain via a Management server. The session must be specified.

```
int ConnectToSessionViaDomain (SessionHandle hSession, const CharType
*strDirectoryServer, const CharType *strDomainName, const CharType
*strSession, const CharType *strUserID, const CharType *strPassword,
StringMapHandle hModelVariables);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

### **strDirectoryServer**

[in] The name of the directory server.

### **strDomainName**

[in] The name of the Host Integrator Domain to use.

### **strSession**

[in] The name of the Host Integrator session to use.

### **strUserID**

[in] The user ID that Host Integrator uses to authenticate the user.

### **strPassword**

[in] The password that Host Integrator uses to authenticate the user.

### **hModelVariables**

[in] An optional list of variable names paired with values (handle to a StringMap Object).

## Return Value

A flag indicating whether the function succeeded or not.

## Remarks

Reasons for failure include:

- Server is not running.

- Invalid directory server.

- Invalid domain.

- Invalid session name.

- Invalid user ID or password.

See the [Error Handling](#) topic for details.

The parameters user ID and password are used by Host Integrator Server if the security option is ON.

## Disconnect

Method used to disconnect from a Host Integrator session.

```
int Disconnect (SessionHandle hSession, int nReserved);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nReserved

[in] Reserved parameter: use 0.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### Remarks

Reasons for failure include:

- Server session has not been established.

See the [Error Handling](#) topic for details.

## EnableTerminalAttributes

Method used to enable terminal attributes to be returned from the server with model recordsets.

```
int EnableTerminalAttributes (SessionHandle hSession, int bEnable);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### bEnable

[in] A flag.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## Remarks

Reasons for failure include:

Server session has not been established.

See the [Error Handling](#) topic for details.

## ExecuteSQLStatement

Method used to execute an SQL statement on a table defined in the model.

```
RecordSetHandle ExecuteSQLStatement(SessionHandle hSession, const CharType
*strSQLStatement);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strSQLStatement

[in] The SQL statement to execute.

### Return Value

A recordset handle which contains the results of the executed SQL statement. If the statement fails the return value is null. The handle must be destroyed with the ReleaseRecordSet method after it is no longer in use.

## Remarks

Reasons for failure include:

Server session has not been established.

Unable to fetch rows.

See the [Error Handling](#) topic for details.

## ExecuteSQLStatementWithMaxrows

Method used to execute an SQL statement on a table defined in the model and specifying maximum number of rows.

```
RecordSetHandle ExecuteSQLStatementWithMaxrows(SessionHandle hSession, const
CharType *strSQLStatement, long maxRows);
```

### Parameters

**hSession**

[in] The Host Integrator session.

**strSQLStatement**

[in] The SQL statement to execute.

**maxRows**

[in] Maximum number of rows to be fetched, or zero if all rows are to be returned.

**Return Value**

A recordset handle which contains the results of the executed SQL statement. If the statement fails, the return value is -1.

## FetchRecords

Method used to fetch up to max rows of data from the Host Integrator Server for the current recordSet of the current entity. If the number of rows to fetch is specified as 0 then the number of rows returned is not limited. If this is the first fetch following an explicit position of the current record, fetchRecords includes the current record in the returned set of records. Otherwise, the set of records returned begins with the first record following the current record. After a fetchRecords, the current record is the last record included in returned set of records. If fieldNames is null, fetchRecords returns all fields for each record.

```
RecordSetHandle FetchRecords(SessionHandle hSession, long nMaxRows,
StringListHandle hFieldNames, const CharType *strFilterExpression);
```

**Parameters****hSession**

[in] The Host Integrator session.

**nMaxRecords**

[in] The maximum number of rows to return with the fetch.

**hFieldNames**

[in] List of the field names to fetch from the entity

**strFilterExpression**

[in] An expression that qualifies the records to fetch. See [Using Filter Expressions](#).

**Return Value**

A recordset handle which contains the results of the fetch. If the statement fails the return value is null. The handle must be destroyed with the `ReleaseRecordSet` method after it is no longer in use.

## Remarks

Reasons for failure include:

- Server session has not been established.

- Unable to fetch rows.

See the [Error Handling](#) topic for details.

## GetAttributeAtCursor

Method used to get the name of the attribute at the current cursor position. This method returns the length of the attribute name string that was copied into the `strAttributeName` parameter. *If the cursor is not at a defined attribute, the return value is 0 and the `strAttributeName` parameter is set to "".*

```
long GetAttributeAtCursor(SessionHandle hSession, CharType
*strAttributeName, long nBufferLen);
```

*\*Parameters*

**hSession**

[in] The Host Integrator session.

**strAttributeName**

[out] Buffer to retrieve the attribute name.

**nBufferLen** [in] The length of the buffer.

*\*Return Value*

The length of the attribute name retrieved.

## GetAttributeMetaData

Method used to get a metadata object for an attribute for an entity in the Host Integrator model.

```
AttributeMetaDataHandle GetAttributeMetaData(SessionHandle hSession, const
CharType *strEntityName, const CharType *strAttributeName);
```

**Parameters**

**hSession**

[in] The Host Integrator session.

#### **strEntityName**

[in] Name of the entity in the Host Integrator model.

#### **strAttributeName**

[in] Name of the attribute of the entity in the Host Integrator model.

#### **Return Value**

A attribute metadata handle which contains the metadata for the given entity and attribute. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

### **GetAttributes**

Method used to get the attributes of the current entity. If attributeNames is null, getAttributes returns all attributes of the entity.

```
RecordHandle GetAttributes(SessionHandle hSession, StringListHandle
hAttributeNames);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

##### **hAttributeNames**

[in] List of attribute names to get from the entity.

#### **Return Value**

A record handle which contains the attributes on the current entity. If the attributes names parameter is non-null, then only named attributes are retrieved. If the statement fails the return value is null. The handle must be destroyed with the ReleaseRecord method after it is no longer in use.

### **GetColumnMetaData**

Method used to get a metadata object for a column of a table in the Host Integrator model. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

```
ColumnMetaDataHandle GetColumnMetaData(SessionHandle hSession, const CharType
*strTableName, const CharType *strColumnName);
```



## Parameters

### hSession

[in] The Host Integrator session.

### strTableName

[in] Name of the table in the Host Integrator model.

### strColumnName

[in] Name of the column of the table in the Host Integrator model.

## Return Value

A column metadata handle which contains the metadata for the given table and column. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## GetConnectionTimeout

Gets how long Host Integrator continues attempting to establish a connection if for any reason the connection cannot be established on the first try (in seconds).

The default value is 30 seconds.

```
long GetConnectionTimeout(SessionHandle hSession);
```

## Parameters

### hSession

[in] The Host Integrator session.

## Return Value

The timeout value for connections.

## GetCurrentEntity

Method used to get the current entity (i.e. screen) of the legacy application.

```
long GetCurrentEntity(SessionHandle hSession, CharType *strEntityName, long nBufferLen);
```

## Parameters

### hSession

[in] The Host Integrator session.

### **strEntityName**

[out] Buffer to retrieve the entity name.

### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the entity name retrieved. If there is an error -1 is returned.

## **GetCurrentRecord**

Method used to get the current record in the current recordSet.

```
RecordHandle GetCurrentRecord(SessionHandle hSession);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

### **Return Value**

A record handle which contains the contents of the current record. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseRecord method after it is no longer in use.

## **GetCurrentRecordIndex**

Method used to get the index number of the current record in the current recordSet.

```
long GetCurrentRecordIndex(SessionHandle hSession);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

### **Return Value**

The index of the current record.

## **GetCurrentRecordSetName**

Method used to get the name of the current recordSet.

```
long GetCurrentRecordSetName(SessionHandle hSession, CharType
*strRecordSetName, long nBufferLen);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **strRecordSetName**

[out] Buffer to retrieve the recordset name.

#### **nBufferLen**

[in] The length of the buffer.

### Return Value

The length of the recordset name retrieved. If there is an error -1 is returned.

## GetEntityAttributes

Method used to get the attribute names of an entity in the Host Integrator model.

```
StringListHandle GetEntityAttributes(SessionHandle hSession, const CharType
*strEntityName);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **strEntityName**

[in] Name of the entity in the Host Integrator model.

### Return Value

A string list handle which contains the names of the entity attributes. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetEntityDescription

Method used to get the description for an entity in the Host Integrator model.

```
long GetEntityDescription(SessionHandle hSession, const CharType
*strEntityName, CharType*strDescription, long nBufferLen);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

### **strEntityName**

[in] The display offset.

### **strDescription**

[out] Buffer to retrieve the entity description.

### **nBufferLen**

[in] The length of the buffer.

## Return Value

The length of the entity description retrieved. If there is an error -1 is returned.

## GetEntityOperations

Method used to get the operation names of an entity in the Host Integrator model.

```
StringListHandle GetEntityOperations(SessionHandle hSession, const CharType
*strEntityName);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

### **strEntityName**

[in] Name of the entity in the Host Integrator model.

## Return Value

A string list handle which contains the names of the entity operations. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetEntityRecordSets

Method used to get the recordset names of an entity in the Host Integrator model.

```
StringListHandle GetEntityRecordSets(SessionHandle hSession, const CharType
*strEntityName);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

### **strEntityName**

[in] Name of the entity in the Host Integrator model.

## Return Value

A string list handle which contains the names of the entity recordsets. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseStringList` method after it is no longer in use.

## GetErrorScreen

Returns a handle to access a terminal screen dump made when an error was reported. If a non-zero handle is returned by this function, it must be released after use.

```
ErrorScreenHandle GetErrorScreen(SessionHandle hSession);
```

## Parameters

### **hSession**

[in] The Host Integrator session.

## Return Value

ErrorScreenHandle. This is a non-zero value if there is an error screen available.

## GetErrorScreenColumns

Returns the number of columns in an error screen.

```
int GetErrorScreenColumns(ErrorScreenHandle hErrorScreen);
```

## Parameters

### **ErrorScreenHandle**

Handle for the error screen.

## Return Value

Number of columns, or zero if there is no error screen

## GetErrorScreenCursorColumn

Returns the zero-based column position of the cursor of an error screen.

```
int GetErrorScreenCursorColumn(ErrorScreenHandle hErrorScreen);
```

#### Parameters

##### ErrorScreenHandle

Handle for the error screen.

#### Return Value

Zero-based column position, or -1 if there is no error screen.

### GetErrorScreenCursorPosition

Returns the zero-based cursor offset an error screen.

```
int GetErrorScreenCursorPosition(ErrorScreenHandle hErrorScreen);
```

#### Parameters

##### ErrorScreenHandle

Handle for the error screen.

#### Return Value

Zero-based cursor offset, or -1 if there is no error screen.

### GetErrorScreenCursorRow

Returns the zero-based row position an error screen.

```
int GetErrorScreenCursorRow(ErrorScreenHandle hErrorScreen);
```

#### Parameters

##### ErrorScreenHandle

Handle for the error screen.

#### Return Value

Zero-based row position, or -1 if there is no error screen.

### GetErrorScreenEntityName

Returns the entity name, if there is one, for an error screen.

```
int GetErrorScreenEntityName(ErrorScreenHandle hErrorScreen, CharType
*strEntityName, long nBufferLen);
```

### Parameters

#### ErrorScreenHandle

Handle for the error screen.

#### StrEntityName

Buffer to receive the entity name.

#### nBufferLen

Maximum number of characters the buffer can accept, including the terminating NUL character.

### Return Value

Number of characters in the entity name, zero if there is no entity, or -1 if there is no error screen.

## GetErrorScreenRows

Returns the number of rows in an error screen.

```
int GetErrorScreenRows(ErrorScreenHandle hErrorScreen);
```

### Parameters

#### ErrorScreenHandle

Handle for the error screen.

### Return Value

Number of rows in an error screen, or zero if there is no error screen.

## GetErrorScreenFullText

Returns all of the screen text from an error screen.

```
int GetErrorScreenFullText(ErrorScreenHandle hErrorScreen, CharType *strText, long
nBufferLen);
```

### Parameters

#### ErrorScreenHandle

Handle for the error screen.

#### strText

Buffer to receive the text.

#### **nBufferLen**

Maximum number of characters the buffer can accept, including the terminating NUL character. A good number would be rows - columns + 1.

#### **Return Value**

Number of characters returned, or zero if there is no error screen.

### **GetErrorScreenText**

Returns a selected portion of the screen text from an error screen.

```
int GetErrorScreenText(ErrorScreenHandle hErrorScreen, int offset, int length, CharType *strText, long nBufferLen);
```

#### **Parameters**

##### **ErrorScreenHandle**

Handle for the error screen.

##### **offset**

Zero-based offset within the error screen for the selected text.

##### **length**

Number of characters of text to be obtained.

##### **strText**

Buffer to receive the text.

##### **nBufferLen**

Maximum number of characters the buffer can accept, including the terminating NUL character. Ideally, this would be length + 1.

#### **Return Value**

Number of characters returned, or zero if there is no error screen.

### **GetErrorScreenTextRow**

Returns a single row of text from an error screen.

```
int GetErrorScreenTextRow(ErrorScreenHandle hErrorScreen, int row, CharType *strText, long nBufferLen);
```



## Parameters

### ErrorScreenHandle

Handle for the error screen.

### row

Zero-based row index for which the text is obtained.

### strText

Buffer to receive the text.

### nBufferLen

Maximum number of characters the buffer can accept, including the terminating NUL character. Ideally, this would be number of columns + 1.

## Return Value

Number of characters returned, or zero if there is no error screen.

## GetFieldMetaData

Method used to get a metadata object for a field for an entity recordset in the Host Integrator model.

```
FieldMetaDataHandle GetFieldMetaData(SessionHandle hSession, const CharType
*strEntityName, const CharType*strRecordSetName, const CharType
*strFieldName);
```

## Parameters

### hSession

[in] The Host Integrator session.

### strEntityName

[in] Name of the entity in the Host Integrator model.

### strRecordSetName

[in] Name of the recordset.

### strFieldName

[in] Name of the field.

## Return Value

A field metadata handle which contains the metadata for the given field on the given recordset for the given entity. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## GetLastRequestID

Method used to get an integer identifier for the most recent transaction performed against the current Host Integrator Server Session.

```
long GetLastRequestID(SessionHandle hSession);
```

### Parameters

#### hSession

[in] The Host Integrator session.

### Return Value

A unique id for the last request performed against the server. If a request has not yet been performed in the current session, this method returns 0. If not connected, this method returns -1.

## GetMethodTimeout

Method used to get the method timeout for an Host Integrator Server Session.

```
long GetMethodTimeout(SessionHandle hSession);
```

### Parameters

#### hSession

[in] The Host Integrator session.

### Return Value

The timeout value for methods.

## GetModelEntities

Method used to get the entity names of the legacy application.

```
StringListHandle GetModelEntities(SessionHandle hSession);
```

### Parameters

#### hSession

[in] The Host Integrator session.

## Return Value

A string list handle which contains the names of the entities for the Host Integrator model. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseStringList` method after it is no longer in use.

## GetModelName

Method used to get the name of the Host Integrator model for the current session.

```
long GetModelName(SessionHandle hSession, CharType *strModelName, long nBufferLen);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strModelName

[out] Buffer to retrieve the model name.

#### nBufferLen

[in] The string length.

### Return Value

The length of the model name retrieved. If there is an error -1 is returned.

## GetModelVariables

Method used to get the names and values of a variables in the Host Integrator model.

```
StringMapHandle GetModelVariables(SessionHandle hSession);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nOffset

### Return Value

A string map handle which contains the names and values of the variables for the Host Integrator model. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseStringMap` method after it is no longer in use.

## GetOperationMetaData

Method used to get an metadata object for an operation of an entity.

```
OperationMetaDataHandle GetOperationMetaData(SessionHandle hSession, const
CharType *strEntityName, const CharType*strOperationName);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strEntityName

[in] Name of an entity in the Host Integrator model.

#### strOperationName

[in] Name of the operation.

### Return Value

A operation metadata handle which contains the metadata for the given operation for the given entity. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## GetProcedureMetaData

Method used to get a metadata object for a procedure of a table.

```
ProcedureMetaDataHandle GetProcedureMetaData(SessionHandle hSession, const
CharType *strTableName, const CharType*strProcedureName);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strTableName

[in] Name of the table in the Host Integrator model.

#### strProcedureName

[in] Name of the procedure.

### Return Value

A procedure metadata handle which contains the metadata for the given procedure for the given table. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## GetRecordSetMetaData

Method used to get an metadata object for a recordset of an entity.

```
RecordSetMetaDataHandle GetRecordSetMetaData(SessionHandle hSession, const
CharType *strEntityName, const CharType*strRecordSetName);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strEntityName

[in] Name of the entity in the Host Integrator model.

#### strRecordSetName

[in] Name of the recordset.

### Return Value

A recordset metadata handle which contains the metadata for the given record set for the given entity. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## GetServerName

Method used to get the name of the Host Integrator Server for the current session.

```
long GetServerName(SessionHandle hSession, CharType *strServerName, long
nBufferLen);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strServerName

[out] Buffer to retrieve the server name.

#### nBufferLen

[in] The length of the buffer.

#### **Return Value**

The length of the server name retrieved. If there is an error -1 is returned.

### **GetSessionID**

Method used to get an integer identifier for the current Host Integrator Server Session.

```
long GetSessionID(SessionHandle hSession);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

#### **Return Value**

A unique id for the session established with the server. If a session has not yet been established, this method returns 0.

### **GetSessionType**

Gets the terminal emulation type of the host session.

The returned value can be:

SessionType3270 (1)

SessionType5250 (2)

SessionTypeVT (3)

SessionTypeHP (4)

```
long GetSessionType(SessionHandle hSessionType);
```

#### **Parameters**

##### **hSessionType**

[in] The Host Integrator session.

#### **Return Value**

The session type. If there is an error, -1 is returned.

### **GetStringAtOffset**

Get the string at the given offset of the given length.

```
long GetStringAtOffset(SessionHandle hSession, long nOffset, long
nLength, CharType *strString, long nBufferLen);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **nOffset**

[in] The offset on the entity (i.e. terminal screen)..

#### **nLength**

[in] The string length.

#### **strString**

[out] Buffer to retrieve the string.

#### **nBufferLen**

[in] The length of the buffer.

### Return Value

The length of the string retrieved. If there is an error -1 is returned.

## GetStringAtRowColumn

Method used to get a string of the given length from the current entity on the host session starting at a given row and column.

```
long GetStringAtRowColumn(SessionHandle hSession, long nTopRow, long
nLeftColumn, long nNumRows, long nNumColumns, CharType *strString,
long nBufferLen);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **nTopRow**

[in] The starting row on the entity (i.e. terminal screen).

#### **nLeftColumn**

[in] The starting column on the entity (i.e. terminal screen).

**nNumRows**

[in] The number of rows to get relative to the starting row.

**nNumColumns**

[in] The number of columns to get relative to the starting column.

**strString**

[out] Buffer to retrieve the string.

**nBufferLen**

[in] The length of the buffer.

**Return Value**

The length of the string retrieved. If there is an error -1 is returned.

## GetTableColumns

Method used to get the column names of a table.

```
StringListHandle GetTableColumns(SessionHandle hSession, const CharType
*strTableName);
```

**Parameters****hSession**

[in] The Host Integrator session.

**strTableName**

[in] Name of the table in the Host Integrator model.

**Return Value**

A string list handle which contains the names of the columns for the given table. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetTableDescription

Method used to get the description for a table in the Host Integrator model.

```
long GetTableDescription(SessionHandle hSession, const CharType *strTableName,
CharType*strDescription, long nBufferLen);
```

**Parameters**



**hSession**

[in] The Host Integrator session.

**strTableName**

[in] Name of the table in the Host Integrator model.

**strDescription**

[out] Buffer to retrieve the description.

**nBufferLen**

[in] The length of the buffer.

**Return Value**

The length of the description retrieved. If there is an error -1 is returned.

## GetTableNames

Method used to get the table names in the Host Integrator model.

```
StringListHandle GetTableNames(SessionHandle hSession);
```

**Parameters****hSession**

[in] The Host Integrator session.

**Return Value**

A string list handle which contains the names of the tables in the Host Integrator model. If the retrieval fails the return value is an empty string. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetTableProcedures

Method used to get the procedures names for a table.

```
StringListHandle GetTableProcedures(SessionHandle hSession, const CharType
*strTableName);
```

**Parameters****hSession**

[in] The Host Integrator session.

## **strTableName**

[in] Name of the table in the Host Integrator model.

### **Return Value**

A string list handle which contains the names of the procedure for the given table. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## **GetVariableMetaData**

Method used to get a metadata object for a model variable in the Host Integrator model.

```
VariableMetaDataHandle GetVariableMetaData(SessionHandle hSession, const
CharType *strVariableName);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

#### **strVariableName**

[in] Name of the variable in the Host Integrator model.

### **Return Value**

A variable metadata handle which contains the metadata for the given variable in the Host Integrator model. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## **InsertRecord**

Method used to perform the insert operation for the current recordset.

```
int InsertRecord(SessionHandle hSession, StringMapHandle hRecord);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

#### **hRecord**

[in] A map containing name-value pairs for the record elements to insert.

### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## InsertStringAtOffset

Method used to insert a string into the current entity on a host session starting at an offset.

```
int InsertStringAtOffset(SessionHandle hSession, const CharType *strString, long nOffset);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **strString**

[in] The string.

#### **nOffset**

[in] The offset on the entity (i.e. terminal screen).

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## InsertStringAtRowColumn

Method used to insert a string into the current entity on the host session starting at the given row and column.

```
int InsertStringAtRowColumn(SessionHandle hSession, const CharType *strString, long nRowNum, long nColumnNum);
```

### Parameters

#### **hSession**

[in] The Host Integrator session.

#### **strString**

[in] The string.

#### **nRowNum**

[in] The row number of the string on the entity (i.e. terminal screen).

#### **nColumnNum**

[in] The column number of the string on the entity (i.e. terminal screen).

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **IsConnected**

Method used to determine if the Host Integrator Server Session is connected.

```
int IsConnected(SessionHandle hSession);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

#### **Return Value**

An integer value indicating whether or not the session is connected (true = !0, false = 0).

### **IsSecureConnection**

Method used to determine if the Host Integrator Server Session is using an encrypted connection.

```
int IsSecureConnection(SessionHandle hSession);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

#### **Return Value**

An integer value indicating whether or not the connection is encrypted (true = !0, false = 0).

### **MoveCurrentRecordIndex**

Method used to move the current record index to a new position in the current recordSet. The movementCode can be one of the following enumerated values defined in appconndef.h:

ScrollHome  
ScrollEnd  
ScrollLineUp  
ScrollLineDown  
ScrollPageUp  
ScrollPageDown

```
int MoveCurrentRecordIndex(SessionHandle hSession, unsigned long nMovement);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nMovement

[in] The enumerated value for the type of movement to perform.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## NextRecord

Method used to get the next record in the current recordSet

```
RecordHandle NextRecord(SessionHandle hSession, const CharType
*strFilterExpression);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strFilterExpression

[in] An expression that qualifies the conditions for the next record. See Using Filter Expressions.

### Return Value

A record handle which contains the fields of the next record. If the statement fails the return value is null. The handle must be destroyed with the ReleaseRecord method after it is no longer in use.

## PerformAidKey

Method used to enter an aid key (for example, a Program Function key) into the current host session. The key is one of the terminal key enumerated values defined in appconndef.h.

```
int PerformAidKey(SessionHandle hSession, unsigned long nKey);
```

#### Parameters

##### hSession

[in] The Host Integrator session.

##### nKey

[in] The enumerated value for the perform. aid key.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### PerformEntityOperation

Method used to perform an operation on the current entity.

```
int PerformEntityOperation(SessionHandle hSession, const CharType
*strOperationName);
```

#### Parameters

##### hSession

[in] The Host Integrator session.

##### strOperationName

[in] The operation name.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### PerformTableProcedure

Method used to perform a procedure on a table defined in the model. hDataInputValues are required only for update and insert procedures, while hFilterValues are required for all procedures except update. If hOutputColumnNames is non-null, then the names in this supplied list determine which table columns are returned in the resulting AppConnRecordSet. If nMaxRows is zero then the number of rows returned is unlimited.

```
RecordSetHandle PerformTableProcedure(SessionHandle hSession, const CharType
*strTableName, const CharType*strProcedureName, StringMapHandle
hDataInputValues, StringMapHandle hFilterValues, int
bFilterIsCaseSensitive, StringListHandle hOutputColumnNames, long
nMaxRows);
```

## Parameters

### hSession

[in] The Host Integrator session.

### strTableName

[in] Name of the table in the Host Integrator model.

### strProcedureName

[in] Name of the table procedure to perform.

### hDataInputValues

[in] Set of data input column name-value pairs to use.

### hFilterValues

[in] Set of filter column name-value pairs to use.

### bFilterIsCaseSensitive

[in] Integer value used to determine if comparison is case sensitive(true = !0, false = 0)..

### hOutputColumnNames

[in] List of the column names of the entity to return as output.

### nMaxRows

[in] The maximum number of rows to be returned.

## Return Value

A recordset handle which contains the results of the procedure. If the statement fails the return value is null. The handle must be destroyed with the ReleaseRecordSet method after it is no longer in use.

## RequireSecureConnection

This method is kept for backwards compatibility. Connections to a Host Integrator Server are always encrypted.

```
int RequireSecureConnection(SessionHandle hSession, int bRequire);
```

#### Parameters

##### **hSession**

[in] The Host Integrator session.

##### **bRequire**

[in] Integer value used require secure connection(true = !0, false = 0).

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### ResumeConnection

Method used to resume the execution of a currently suspended Host Integrator server session. If the requested session is already running, this method does nothing.

```
int ResumeConnection(SessionHandle hSession, const CharType *strConnectionToken);
```

#### Parameters

##### **hSession**

[in] The Host Integrator session.

##### **strConnectionToken**

[in] The token received from the SuspendConnection method.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### SelectCurrentRecord

Method used to perform the selection operation for the current recordset.

```
int SelectCurrentRecord(SessionHandle hSession);
```

#### Parameters

##### **hSession**

[in] The Host Integrator session.

#### Return Value



An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SelectRecordByFilter

Method used to find a record in the current recordset and perform its selection operation.

```
int SelectRecordByFilter(SessionHandle hSession, const CharType
*strFilterExpression);
```

Parameters hSession [in] The Host Integrator session. strFilterExpression [in] An expression that qualifies the record to select. See Using Filter Expressions. Return Value An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SelectRecordByIndex

Method used to find a record in the current recordset and perform its selection operation.

```
int SelectRecordByIndex(SessionHandle hSession, long nIndex);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nIndex

[in] The index of the record to select.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetAttributes

Method used to set the attributes in the current entity. The map contains attributes ( name/value pairs ) that are to be inserted in the entity. The map does not need to contain all of the attributes for the entity, but should contain all of the required attributes for the entity.

```
int SetAttributes(SessionHandle hSession, StringMapHandle hAttributes);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### hAttributes

[in] A map containing attributes ( name/value pairs ).

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetAttributesDelayed

Method used to set the attributes of an entity. The map contains attributes ( name/value pairs ) that are to be set in the entity. The map does not need to contain all of the attributes for the entity, but should contain all of the required attributes for the entity. If the entity is not specified then the current entity is used.

```
int SetAttributesDelayed(SessionHandle hSession, StringMapHandle
hAttributes, const CharType *strEntityName);
```

Parameters hSession [in] The Host Integrator session. hAttributes [in] A map containing attributes ( name/value pairs ). strEntityName [in] Name of the entity in the Host Integrator model. Return Value An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetConnectionTimeout

Method used to specify how long Host Integrator continues attempting to establish a connection if for any reason the connection cannot be established on the first try (in seconds).

This method is useful, for example, if the server is temporarily unable to allow any more sessions, or if the domain load has been reached. Connection attempt information, include the number of connection attempts and the time of those attempts, is written to the log.

The default value is 30 seconds.

```
int SetConnectionTimeout(SessionHandle hSession, long nTimeout);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nTimeout

[in] The timeout period in seconds.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetCurrentEntity

Method used to set the current entity. The legacy application traverses to the screen that corresponds to that entity.

```
int SetCurrentEntity(SessionHandle hSession,
 const CharType *strEntityName);
```

#### Parameters

**hSession**

[in] The Host Integrator session.

### SetCurrentRecordIndex

Method used to change the index number of the current record in the current recordset.

```
int SetCurrentRecordIndex(SessionHandle hSession, long nIndex);
```

#### Parameters

**hSession**

[in] The Host Integrator session.

**nIndex**

[in] The index of the record to make current.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### SetCurrentRecordSetByName

Method used to set the current recordset by name. This is only needed if there is more than one recordSet for the current entity.

```
int SetCurrentRecordSetByName(SessionHandle hSession, const CharType
 *strRecordSetName);
```

#### Parameters

**hSession**

[in] The Host Integrator session.

**strRecordSetName**

[in] The name of the recordset to use for recordset methods.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetMetaDataOnlyFlag

Defines the current session as a "metadata-only" session. A metadata-only session does not require a host connection and allows only methods and properties that interact with metadata. You must set SetMetaDataOnlyFlag to true before you use a Connect method.

Once a connect method has been called, and before the Disconnect method has been called, any attempt to change the value of the SetMetaDataOnlyFlag property generates an error with this text: "The MetaDataOnly property cannot be changed while a connection is active."

When a client connects with SetMetaDataOnlyFlag set to true, the server does not report the connection as a session, and does not allocate a new session or a session from a pool.

Use the [GetMetaDataOnlyFlag] (need link) property to determine the current value of the metadata-only flag.

### Syntax

```
int SetMetaDataOnlyFlag(SessionHandle hSession, int nMetaDataOnly);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nMetaDataOnly

[in] An integer value specifying whether or not to set the metadata-only flag (yes = !0, no = 0).

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### Example

```
void myTestRoutine(void){ SessionHandle hSession; int mdoflag; hSession = CreateSession(); SetMetaDataOnlyFlag(hSession, 1); // Make this a metadata-only connection mdoflag = GetMetaDataOnlyFlag(hSession); printf("Metadata-only flag = %d\n", mdoflag); ConnectToModel(hSession, srvr, model, userid, password, 0); ... Disconnect(hSession, 0); ReleaseSession(hSession);}
```

## SetMethodTimeout

Method used to set the method timeout for a Host Integrator Server Session.

```
int SetMethodTimeout(SessionHandle hSession, long nTimeout);
```

## Parameters

### hSession

[in] The Host Integrator session.

### nTimeout

[in] The timeout period in milliseconds.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SetModelVariables

Method used to set the values of variables in the Host Integrator model.

```
int SetModelVariables(SessionHandle hSession, StringMapHandle hVariables);
```

## Parameters

### hSession

[in] The Host Integrator session.

### hVariables

[in] A collection of name/value pairs for the variables.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## SuspendConnection

Method used to suspend the execution of a currently running Host Integrator Server Persistent Session. On successful suspension, the method returns a token to be used when resuming the session. If the current session is already suspended, this method does nothing. The suspended session automatically terminates if it is not resumed before the specified timeout period elapses.

```
long SuspendConnection(SessionHandle hSession, long nTimeout, CharType
*strConnectionToken, long nBufferLen);
```

## Parameters

### hSession

[in] The Host Integrator session.

### nTimeout

[in] The timeout value to use for the suspended session in minutes.

#### **strConnectionToken**

[out] Buffer to retrieve the description.

#### **nBufferLen**

[in] The length of the buffer.

#### **Return Value**

The length of the connection token retrieved. If there is an error -1 is returned.

### **UpdateCurrentRecord**

Method used to change the values of the current record in the current record set.

```
int UpdateCurrentRecord(SessionHandle hSession, StringMapHandle hRecord);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

##### **hRecord**

[in] A map containing name-value pairs for the record elements to insert.

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **UpdateRecordByFilter**

Method used to change the values of a record in the current recordset. The record to change is the first one after the current record that satisfies the filter expression.

```
int UpdateRecordByFilter(SessionHandle hSession, StringMapHandle hRecord, const CharType *strFilterExpression);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

##### **hRecord**

[in] A map containing name-value pairs for the record elements to insert.

## UpdateRecordByIndex

Method used to change the values of a record in the current recordSet. The record to change is selected by the provided index number.

```
int UpdateRecordByIndex(SessionHandle hSession, StringMapHandle hRecord, long nIndex);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### hRecord

[in] A map containing name-value pairs for the record elements to insert.

#### nIndex

[in] The index of the record to update.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## UpdateRecords

Method used to change the values of a record in the current recordSet. The record to change is the first one after the current record that satisfies the filter expression.

```
long UpdateRecords(SessionHandle hSession, StringMapHandle hRecord, const CharType *strFilterExpression);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### hRecord

[in] A map containing name-value pairs for the record elements to insert.

#### strFilterExpression

[in] An expression that qualifies the record to update. See Using Filter Expressions.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## WaitForCursor

Method used to wait for the presence of the terminal cursor at a particular terminal screen offset. Exception occurs if timeout period elapses before the cursor is at the offset.

```
int WaitForCursor(SessionHandle hSession, long nRowNum, long nColumnNum, long nTimeout);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### nRowNum

[in] The row on the entity (i.e. terminal screen) for the cursor.

#### nColumnNum

[in] The column on the entity (i.e. terminal screen) for the cursor.

#### nTimeout

[in] The timeout period in milliseconds.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## WaitForEntityChange

Method used to wait for the recognition of a new entity that is not 'strEntityName'. If entityName is null or empty, the method returns upon recognition of any new entity. Exception occurs if timeout period elapses before the Verastream Server recognizes a new entity.

```
int WaitForEntityChange(SessionHandle hSession, const CharType *strEntityName, long nTimeout);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strEntityName

[in] The name of the entity that cannot satisfy the entity change condition, typically the entity that was current when the method is called.

#### nTimeout



[in] The timeout period in milliseconds. Return Value An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## WaitForString

Method used to wait for the presence of a string starting at a particular terminal screen row and column. Exception occurs if timeout period elapses before the string appears at the row and column.

```
int WaitForString(SessionHandle hSession, const CharType *strString, long nRowNum, long nColumnNum, long nTimeout);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strString

[in] The string.

#### \*nRowNum8

[in] The row on the entity (i.e. terminal screen) of the beginning of the string.

#### nColumnNum

[in] The column on the entity (i.e. terminal screen) of the beginning of the string.

#### nTimeout

[in] The timeout period in milliseconds.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## WaitForStringRelCursor

ethod used to wait for the presence of a string relative to the current terminal screen cursor. Exception occurs if timeout period elapses before the string appears at the offset.

```
int WaitForStringRelCursor(SessionHandle hSession, const CharType *strString, long nRowOffset, long nColumnOffset, long nTimeout);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### **strString**

[in] The string.

#### **nRowOffset**

[in] The row offset from the cursor (i.e. terminal screen) of the beginning of the string.

#### **nColumnNum**

[in] The column offset from the cursor (i.e. terminal screen) of the beginning of the string.

#### **nTimeout**

[in] The timeout period in milliseconds.

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **ReleaseSession**

Method used to release an AppConn session.

```
int ReleaseSession(SessionHandle hSession);
```

#### **Parameters**

##### **hSession**

[in] The Host Integrator session.

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **CreateRecordSet**

Create an AppConn recordset.

```
RecordSetHandle CreateRecordSet(void);
```

#### **Return Value**

A handle to a recordset.

### **CreateRecord**

Create an AppConn record.

RecordHandle CreateRecord(void);

#### **Return Value**

A handle to a record.

#### **GetRecordIndex**

Method used to get Host Integrator recordset index of the record.

```
long GetRecordIndex(RecordHandle hRecord);
```

#### **Parameters**

##### **hRecord**

[in] AppConn record.

#### **Return Value**

The Host Integrator recordset index. If record did not come from a recordset the index is -1.

Indexing of records is based on model-runtime indexing, which starts with 1. (Contrast with GetRecord, which uses 0-based indexing). So the index of the first record in a recordset is 1.

### **CreateRecordSet**

Create an AppConn recordset.

RecordSetHandle CreateRecordSet(void);

#### **Return Value**

A handle to a recordset.

### **GetNumRecords**

Method used to get the number of records in the recordset.

```
long GetNumRecords(RecordSetHandle hRecordSet);
```

#### **Parameters**

##### **hRecordSet**

[in] An AppConn recordset.

#### **Return Value**

The number of records in the recordset.

## GetNumRecordElements

Method used to get the number of fields in the records of the recordset.

```
long GetNumRecordElements(RecordSetHandle hRecordSet);
```

### Parameters

#### hRecordSet

[in] An AppConn recordset.

### Return Value

The number of fields in the records of the recordset

## GetRecord

Method used to get a record of the recordset.

```
RecordHandle GetRecord(RecordSetHandle hRecordSet, long nRecordNo);
```

### Parameters

#### hRecordSet

[in] An AppConn recordset.

#### nRecordNo

[in] Index of the record to get.

### Return Value

A record handle which contains the fields of the retrieved record. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseRecord method after it is no longer in use.

Indexing of records through the Verastream API is 0-based, so to get the first record in your recordset you do a GetRecord(myRecordSet, 0). (Contrast with [GetRecordIndex] (need link), which uses 1-based indexing.)

## GetElementMetaDataByName

Method used to get the metadata for an element in the records of the recordset.

```
ElementMetaDataHandle GetElementMetaDataByName(RecordSetHandle hRecordSet,
const CharType *strElementName);
```

### Parameters

## **hRecordSet**

[in] An AppConn recordset.

## **strElementName**

[in] Name of the element to get the metadata.

## **Return Value**

A metadata handle (column or field depending on the recordset) which contains the metadata for the given element. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## **GetElementMetaDataByIndex**

Method used to get the metadata for an element in the records of the record set.

```
ElementMetaDataHandle GetElementMetaDataByIndex(RecordSetHandle hRecordSet, long nElementNo);
```

## **Parameters**

### **hRecordSet**

[in] An AppConn recordset.

### **nElementNo**

[in] Index of the element to get the metadata.

## **Return Value**

A metadata handle (column or field depending on the recordset) which contains the metadata for the given element. If the statement fails the return value is null. The handle must be destroyed with the ReleaseMetaData method after it is no longer in use.

## **RecordSetToXML**

Method used to convert from an AppConn recordset to an XML String.

```
long RecordSetToXML(RecordSetHandle hRecordSet, const CharType *strDTDURL, CharType *strXMLString, long nBufferLen);
```

## **Parameters**

### **hRecordSet**

[in] An AppConn recordset.

### **strDTDURL**

[in] The URL of the DTD to be used for the XML result.

### **strXMLString**

[out] Buffer to retrieve the XML string. If strXMLString is passed in as NULL, RecordSetToXML returns the length of buffer necessary to accommodate the XML string.

### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the XML string retrieved. If there is an error -1 is returned.

## **RecordSetFromXML**

Method used to convert from an XML String to an AppConn recordset.

```
int RecordSetFromXML(RecordSetHandle hRecordSet, const CharType *strXMLString);
```

### **Parameters**

#### **hRecordSet**

[in] An AppConn recordset.

#### **strXMLString**

[in] Buffer to retrieve the XML string.

### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## **ReleaseRecordSet**

MMethod used to release an AppConn recordset. All elements of the recordset are released.

```
int ReleaseRecordSet(RecordSetHandle hRecordSet);
```

### **Parameters**

#### **hRecordSet**

[in] AppConn recordset.

### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **CreateRecord**

Create an AppConn record.

```
RecordHandle CreateRecord(void);
```

### **Return Value**

A handle to a record.

## **GetRecordIndex**

Method used to get Host Integrator recordset index of the record.

```
long GetRecordIndex(RecordHandle hRecord);
```

### **Parameters**

#### **hRecord**

[in] AppConn record.

### **Return Value**

The Host Integrator recordset index. If record did not come from a recordset the index is -1.

Indexing of records is based on model-runtime indexing, which starts with 1. (Contrast with GetRecord, which uses 0-based indexing). So the index of the first record in a recordset is 1.

## **GetModelVersionString**

Method used to get version information for the current model. The string returned uniquely identifies the version of the model deployed to the Host Integrator Server.

```
GetModelVersionString(SessionHandle hSession, CharType * strModelVersionString, long nBufferLen);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

#### **strModelVersionString**

[in] Buffer to retrieve the model version string

#### **nBufferLen**

[in] The length of the buffer.

#### **Return Value**

The length of the version string retrieved.

### **GetNumElements**

Method used to get the number of elements in the record.

```
long GetNumElements(RecordHandle hRecord);
```

#### **Parameters**

##### **hRecord**

[in] AppConn record.

#### **Return Value**

The number of elements in the AppConn record.

### **GetElementName**

Method used to get the name of an element in the record.

```
long GetElementName(RecordHandle hRecord, long nElementNo, CharType
*strElementName, long nBufferLen);
```

#### **Parameters**

##### **hRecord**

[in] AppConn record.

##### **nElementNo**

[in] Index of the element to get.

##### **strElementName**

[out] Buffer to retrieve the element name.

##### **nBufferLen**

[in] The length of the buffer.

#### **Return Value**

The length of the element name retrieved. If there is an error -1 is returned.



## GetElementByIndex

Method used to get an element in the record.

```
long GetElementByIndex(RecordHandle hRecord, long nElementNo, CharType *strElementValue, long nBufferLen);
```

### Parameters

#### hRecord

[in] AppConn record.

#### nElementNo

[in] Index of the element to get.

#### strElementName

[out] Buffer to retrieve the element name.

#### nBufferLen

[in] The length of the buffer.

### Return Value

The length of the element retrieved. If there is an error -1 is returned.

## GetElementByName

Method used to get an element in the record.

```
long GetElementByName(RecordHandle hRecord, const CharType *strElementName, CharType *strElementValue, long nBufferLen);
```

### Parameters

#### hRecord

[in] AppConn record.

#### strElementName

[in] Name of the element to get.

#### strElementValue

[out] Buffer to retrieve the element value.

#### nBufferLen

[in] The length of the buffer.

#### **Return Value**

The length of the element retrieved. If there is an error -1 is returned.

### **GetElements**

Method used to get the elements (key/value pairs) in the record.

```
StringMapHandle GetElements(RecordHandle hRecord);
```

#### **Parameters**

##### **hRecord**

[in] AppConn record.

#### **Return Value**

A string map handle which contains the names and values for the record. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringMap method after it is no longer in use.

### **GetTermAttrsByIndex**

Method used to get the terminal attributes of an element in the record.

```
TerminalAttributesHandle GetTermAttrsByIndex(RecordHandle hRecord, long
nElementNo);
```

#### **Parameters**

##### **hRecord**

[in] AppConn record.

##### **nElementNo**

[in] Index of the element to get.

#### **Return Value**

A terminal attributes handle for the element. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseTerminalAttributes method after it is no longer in use.

### **GetTermAttrsByName**

Method used to get the terminal attributes of an element in the record.

```
TerminalAttributesHandle GetTermAttrsByName(RecordHandle hRecord, const
CharType *strElementName);
```

### Parameters

#### **hRecord**

[in] AppConn record.

#### **strElementName**

[in] Name of the element to get.

### Return Value

A terminal attributes handle for the element. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseTerminalAttributes method after it is no longer in use.

### ProcessString

Calls the ProcessString event handler on the server.

```
long ProcessString(SessionHandle hSession, const CharType *strIn, const
CharType *strOut, long nBufferLen);
```

#### **hSession**

[in] The Host Integrator session.

#### **strIn**

[in] The string to be passed to the ProcessString event handler.

#### **strOut**

[in] Buffer to retrieve the results returned by the ProcessString event handler.

#### **nBufferLen**

[in] The length of the buffer.

### Return Value

The length of the results returned from the ProcessString event handler.

## RecordToXML

Method used to convert from an AppConn record to an XML String.

```
long RecordToXML(RecordHandle hRecord, const CharType *strDTDURL, CharType
*strXMLString, long nBufferLen);
```

## Parameters

### **hRecord**

[in] AppConn record.

### **strDTDURL**

[in] The URL of the DTD to be used for the XML result.

### **\*strXMLString8**

[out] Buffer to retrieve the XML string.

### **nBufferLen**

[in] The length of the buffer.

## Return Value

The length of the XML string retrieved. If there is an error -1 is returned.

## RecordFromXML

Method used to convert from an XML String to an AppConn record.

```
int RecordFromXML(RecordHandle hRecord, const CharType *strXMLString);
```

## Parameters

### **hRecord**

[in] AppConn record.

### **strXMLString**

[in] Buffer to retrieve the XML string.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## ReleaseRecord

Method used to release an AppConn record. All elements of \* the record are released.

```
int ReleaseRecordSet(RecordSetHandle hRecordSet);
```

## Parameters

### **hRecord**

[in] AppConn record.

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

#### **CreateStringList**

Create an AppConn string list.

```
StringListHandle CreateStringList(void);
```

#### **Return Value**

A handle to a string list.

### **CreateStringList**

Create an AppConn string list.

```
StringListHandle CreateStringList(void);
```

#### **Return Value**

A handle to a string list.

### **AddStringToList**

Method used to add a string to the string list.

```
int AddStringToList(StringListHandle hStringList, const CharType *strString);
```

#### **Parameters**

##### **hStringList**

[in] AppConn string list.

##### **strString**

[in] String to add to the list.

##### *\*Return Value*

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0)

### **ClearStringList**

Method used to remove all the strings from the string list.

```
int ClearStringList(StringListHandle hStringList);
```

## Parameters

### hStringList

[in] AppConn string list.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetNumStrings

Method used to get the number of strings in the string list.

```
long GetNumStrings(StringListHandle hStringList);
```

## Parameters

### hStringList

[in] AppConn string list.

## Return Value

The number of elements in the AppConn string list.

## GetStringAt

Method used to get a string in the string list.

```
long GetStringAt(StringListHandle hStringList, long nIndex, CharType *strString, long nBufferLen);
```

## Parameters

### hStringList

[in] AppConn string list.

### nIndex

[in] Index of the string to get.

### strString

[out] Buffer to retrieve the string.

### nBufferLen

[in] The length of the buffer.

## Return Value

The length of the element retrieved. If there is an error -1 is returned.

## ReplaceStringAt

Method used to replace a string in the string list.

```
int ReplaceStringAt(StringMapHandle hStringMap, long nIndex, const CharType *strString);
```

### Parameters

#### hStringList

[in] AppConn string list.

#### nIndex

[in] Index of the string to get.

#### strString

[in] String to replace the existing entry.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## RemoveStringAt

Method used to remove a string from the string list.

```
int RemoveStringAt(StringListHandle hStringList, long nIndex);
```

### Parameters

#### hStringList

[in] AppConn string list.

#### nIndex

[in] Index of the string to remove.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## ReleaseErrorScreen

Release an error screen and free the resources associated with it.

```
int ReleaseErrorScreen(ErrorScreenHandle hErrorScreen);
```

#### Parameters

##### ErrorScreenHandle

Handle of error screen to be released.

#### Return Value

1 with success of error screen release, 0 on failure.

### ReleaseStringList

Method used to release an AppConn string list. All elements of the string list are released.

```
int ReleaseStringList(StringListHandle hStringList);
```

#### Parameters

##### hStringList

[in] AppConn string list.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### CreateStringMap

Create an AppConn string map.

```
StringMapHandle CreateStringMap(void);
```

#### Return Value

A handle to a string map.

### AddStringToMap

Method used to add a key/value pair to the string map.

```
int AddStringToMap(StringMapHandle hStringMap, const CharType *strKey, const CharType *strString);
```

#### Parameters

##### hStringMap



[in] AppConn string map.

### **strstrKeyString**

[in] Key to add to the map.

### **strString**

[in] String to add to the map.

### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## **ClearStringMap**

Method used to remove all the strings from the string map.

```
int ClearStringMap(StringMapHandle hStringMap);
```

### **Parameters**

#### **hStringMap**

[in] AppConn string map.

### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## **GetNumStringPairs**

Method used to get the number of string/value pairs in the string map.

```
long GetNumStringPairs(StringMapHandle hStringMap);
```

### **Parameters**

#### **hStringMap**

[in] AppConn string map.

### **Return Value**

The number of elements in the AppConn string map.

## **GetKey**

Method used to get a key in the string map.

```
long GetKey(StringMapHandle hStringMap, long nIndex, CharType *strKey, long nBufferLen);
```

### Parameters

#### **hStringMap**

[in] AppConn string map.

#### **nIndex**

[in] Index of the key to get.

#### **strKey**

[out] Buffer to retrieve the key.

#### **nBufferLen**

[in] The length of the buffer.

### Return Value

The length of the key retrieved. If there is an error -1 is returned.

## GetStringByIndex

Method used to get a string in the string map.

```
long GetStringByIndex(StringMapHandle hStringMap, long nIndex, CharType *strString, long nBufferLen);
```

Parameters hStringMap [in] AppConn string map. nIndex [in] Index of the key to get. strString [out] Buffer to retrieve the string. nBufferLen [in] The length of the buffer. Return Value The length of the string retrieved. If there is an error -1 is returned.

## GetStringByKey

Method used to get a string in the string map.

```
long GetStringByIndex(StringMapHandle hStringMap, long nIndex, CharType *strString, long nBufferLen);
```

### Parameters

#### **hStringMap**

[in] AppConn string map.

#### **nIndex**

[in] Index of the key to get.

#### **strString**

[out] Buffer to retrieve the string.

#### **nBufferLen**

[in] The length of the buffer.

#### **Return Value**

The length of the string retrieved. If there is an error -1 is returned.

### **ReplaceStringByIndex**

Method used to replace a string in the string map.

```
int ReplaceStringByIndex(StringMapHandle hStringMap, long nIndex, const CharType *strString);
```

#### **Parameters**

##### **hStringMap**

[in] AppConn string map.

##### **nIndex**

[in] Index of the string to get.

##### **strString**

[in] String to replace the existing entry.

#### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### **ReplaceStringByKey**

Method used to replace a string in the string map.

```
int ReplaceStringByKey(StringMapHandle hStringMap, const CharType *strKey, const CharType *strString);
```

#### **Parameters**

##### **hStringMap**

[in] AppConn string map.

**strKey**

[in] Key of the string to get.

**strString**

[in] String to replace the existing entry.

**Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## RemoveStringByIndex

Method used to remove a string from the string map.

```
int RemoveStringByIndex(StringMapHandle hStringMap, long nIndex);
```

**Parameters****hStringMap**

[in] AppConn string map.

**nIndex**

[in] Index of the string to remove.

**Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## RemoveStringByKey

Method used to remove a string from the string map.

```
int RemoveStringByKey(StringMapHandle hStringMap, const CharType *strKey);
```

**Parameters****hStringMap**

[in] AppConn string map.

**strKey**

[in] Key of the string to remove.

**Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## ReleaseStringMap

Method used to release an AppConn string map. All elements of the string map are released.

```
int ReleaseStringMap(StringMapHandle hStringMap);
```

### Parameters

#### hStringMap

[in] AppConn string map.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetMetaDataName

Method used to get the metadata name.

```
long GetMetaDataName(MetaDataHandle hMetaData, CharType *strName, long
nBufferLen);
```

### Parameters

#### hMetaData

[in] AppConn metadata.

#### strName

[out] Buffer to retrieve the name.

#### nBufferLen

[in] The length of the buffer.

### Return Value

The length of the name retrieved. If there is an error -1 is returned.

## GetMetaDataDescription

Method used to get the metadata description.

```
long GetMetaDataDescription(MetaDataHandle hMetaData, CharType *strDescription,
long nBufferLen);
```

### Parameters

#### hMetaData

[in] AppConn metadata.

### **strDescription**

[out] Buffer to retrieve the description.

### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the description retrieved. If there is an error -1 is returned.

## **GetMetaDataOnlyFlag**

Determines whether the current session is "metadata only." A metadata only session does not require a host connection and allows only methods and properties that interact with metadata.

Use [SetMetaDataOnlyFlag] (need link) to set the metadata-only flag.

```
int GetMetaDataOnlyFlag(SessionHandle hSession);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

### **Return Value**

An integer value indicating whether or not the metadata-only flag is set (true = !0, false = 0).

### **Example**

```
void myTestRoutine(void){ SessionHandle hSession; int mdoflag; hSession = CreateSession(); SetMetaDataOnlyFlag(hSession, 1); // Make this a metadata-only connection mdoflag = GetMetaDataOnlyFlag(hSession); printf("Metadata-only flag = %d\n", mdoflag); ConnectToModel(hSession, srvr, model, userid, password, 0); ... Disconnect(hSession, 0); ReleaseSession(hSession);}
```

## **GetMetaDataType**

Method used to get the metadata type.

The returned value is one of:

AttributeMeta  
OperationMeta  
RecordSetMeta  
FieldMeta  
VariableMeta  
TableMeta  
ColumnMeta  
ProcedureMeta

```
long GetMetaDataType(MetaDataHandle hMetaData);
```

#### Parameters

##### hMetaData

[in] AppConn metadata.

#### Return Value

The type of the metadata. If there is an error -1 is returned.

### ReleaseMetaData

Method used to release an AppConn metadata.

```
int ReleaseMetaData(MetaDataHandle hMetaData);
```

#### Parameters

##### hStringMap

[in] AppConn string map.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### IsAttributeReadable

Method used to determine if the attribute is readable.

```
int IsAttributeReadable(AttributeMetaDataHandle hAttributeMetaData);
```

#### Parameters

##### hAttributeMetaData

[in] Attribute metadata.

#### Return Value

An integer value indicating whether or not the attribute is readable(true = !0, false = 0).

### IsAttributeWriteable

Method used to determine if the attribute is writeable.

```
int IsAttributeWriteable(AttributeMetaDataHandle hAttributeMetaData);
```

#### Parameters

##### hAttributeMetaData

[in] Attribute metadata.

#### Return Value

An integer value indicating whether or not the attribute is readable(true = !0, false = 0).

### GetAttributeLength

Method used to get the length of the attribute.

```
long GetAttributeLength(AttributeMetaDataHandle hAttributeMetaData);
```

#### Parameters

##### hAttributeMetaData

[in] Attribute metadata.

#### Return Value

The length of the attribute. If there is an error -1 is returned.

### GetAttributeReadVariable

Method used to get the read variable of the attribute.

```
long GetAttributeReadVariable(AttributeMetaDataHandle hAttributeMetaData,
CharType *strReadVariable, long nBufferLen);
```

#### Parameters

##### hAttributeMetaData

[in] Attribute metadata.



### **strReadVariable**

[out] Buffer to retrieve the read variable.

### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the read variable retrieved. If there is an error -1 is returned.

## **GetAttributeWriteVariable**

Method used to get the write variable of the attribute.

```
long GetAttributeWriteVariable(AttributeMetaDataHandle hAttributeMetaData,
CharType *strWriteVariable, long nBufferLen);
```

### **Parameters**

#### **hAttributeMetaData**

[in] Attribute metadata.

#### **strWriteVariable**

[out] Buffer to retrieve the write variable.

#### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the write variable retrieved. If there is an error -1 is returned.

## **TerminalAttributesEnabled**

Method used to determine if terminal attribute are enabled on the attribute.

```
int TerminalAttributesEnabled(AttributeMetaDataHandle hAttributeMetaData);
```

### **Parameters**

#### **hAttributeMetaData**

[in] Attribute metadata.

### **Return Value**

An integer value indicating whether or not terminal attributes are enabled(true = !0, false = 0).

## IsFieldKey

Method used to determine if the field is a key. A key value is unique for the associated field in the recordset--no other records will have the same value for that field.

```
int IsFieldKey(FieldMetaDataHandle hFieldMetaData);
```

### Parameters

#### hFieldMetaData

[in] Field metadata.

### Return Value

An integer value indicating whether or not the field is a key(true = !0, false = 0).

## IsFieldReadable

Method used to determine if the field is readable--that is, whether the field can be read by a client.

```
int IsFieldReadable(FieldMetaDataHandle hFieldMetaData);
```

### Parameters

#### hFieldMetaData

[in] Field metadata.

### Return Value

An integer value indicating whether or not the field is readable(true = !0, false = 0).

## IsFieldWritable

Method used to determine if the field is writable--that is, whether Host Integrator allows this field to have data written to it.

```
int IsFieldWritable(FieldMetaDataHandle hFieldMetaData);
```

### Parameters

#### hFieldMetaData

[in] Field metadata.

### Return Value

An integer value indicating whether or not the field is writeable(true = !0, false = 0).

## GetFieldLength

Method used to get the length of the field.

```
long GetFieldLength(FieldMetaDataHandle hFieldMetaData);
```

### Parameters

#### hFieldMetaData

[in] Field metadata.

### Return Value

The length of the attribute. A return value of -1 indicates that the field length is variable.

## FieldAttributesEnabled

Method used to determine if terminal attribute are enabled for the field. The terminal attributes enabled flag indicates whether Host Integrator can return the terminal attributes of the field. Terminal attributes specify various properties the text can have, including color, whether the text is blinking, and whether the text is displayed in reverse video.

```
int FieldAttributesEnabled(FieldMetaDataHandle hFieldMetaData);
```

### Parameters

#### hFieldMetaData

[in] Field metadata.

### Return Value

An integer value indicating whether or not the field attributes are enabled(true = !0, false = 0).

## GetColumnType

Method used to get the type of the column.

The returned value can be:

IntegerColumn

TextColumn

FloatColumn

```
long GetColumnType(ColumnMetaDataHandle hColumnMetaData);
```

## Parameters

### hColumnMetaData

[in] Column metadata.

## Return Value

The type of the column. If there is an error -1 is returned.

## IsColumnKey

Method used to determine if the column is a key.

```
int IsColumnKey(ColumnMetaDataHandle hColumnMetaData);
```

## Parameters

### hColumnMetaData

[in] Column metadata.

## Return Value

An integer value indicating whether or not the column is a key(true = !0, false = 0).

## GetColumnMin

Method used to get the min for the column. For integer columns the min is the minimum value that can be entered for that column. For text columns the min is the minimum length of a string that can be entered in that column.

```
long GetColumnMin(ColumnMetaDataHandle hColumnMetaData);
```

## Parameters

### hColumnMetaData

[in] Column metadata.

## Return Value

The min of the column. If there is an error -1 is returned.

## GetColumnMax

Method used to get the max for the column. For integer columns the max is the maximum value that can be entered for that column. For text columns the max is the maximum length of a string that can be entered in that column.

```
long GetColumnMax(ColumnMetaDataHandle hColumnMetaData);
```

#### Parameters

##### **hColumnMetaData**

[in] Column metadata.

#### Return Value

The max of the column. If there is an error -1 is returned.

### GetAltDestinations

Method used to get a list of alternative destinations of the operation.

```
StringListHandle GetAltDestinations(OperationMetaDataHandle hOperationMetaData);
```

#### Parameters

##### **hOperationMetaData**

[in] Operation metadata.

#### Return Value

A string list handle which contains the names of the alternative destinations. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

### GetAttributesUsed

Method used to get a list of the attributes used in the operation.

```
StringListHandle GetAttributesUsed(OperationMetaDataHandle hOperationMetaData);
```

#### Parameters

##### **hOperationMetaData**

[in] Operation metadata.

#### Return Value

A string list handle which contains the names of the attributes used in the operation. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

### GetDestination

Method used to get the destination of the operation.

```
long GetDestination(OperationMetaDataHandle hOperationMetaData, CharType
*strDestination, long nBufferLen);
```

#### Parameters

##### **hOperationMetaData**

[in] Operation metadata.

##### **strDestination**

[out] Buffer to retrieve the destination.

##### **nBufferLen**

[in] The length of the buffer.

#### Return Value

The length of the destination retrieved. If there is an error -1 is returned.

### IsDefaultOperation

Method used to determine if the operation is the default.

```
int IsDefaultOperation(OperationMetaDataHandle hOperationMetaData);
```

#### Parameters

##### **hOperationMetaData**

[in] Operation metadata.

#### Return Value

An integer value indicating whether or not the operation is the default(true = !0, false = 0).

### GetVariablesUsed

Method used to get a list of the variables used in the operation.

```
StringListHandle GetVariablesUsed(OperationMetaDataHandle hOperationMetaData);
```

#### Parameters

##### **hOperationMetaData**

[in] Operation metadata.

### Return Value

A string list handle which contains the names of the variables used by the operation. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetFilterColumns

Method used to get a list of the filter columns for the procedure.

```
StringListHandle GetFilterColumns(ProcedureMetaDataHandle hProcedureMetaData);
```

### Parameters

#### hProcedureMetaData

[in] Procedure metadata.

### Return Value

A string list handle which contains the names of the filter columns used by the procedure. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetInputColumns

Method used to get a list of the input columns for the procedure.

```
StringListHandle GetInputColumns(ProcedureMetaDataHandle hProcedureMetaData);
```

### Parameters

#### hProcedureMetaData

[in] Procedure metadata.

### Return Value

A string list handle which contains the names of the input columns used by the procedure. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetOutputColumns

Method used to get a list of the output columns for the procedure.

```
StringListHandle GetOutputColumns(ProcedureMetaDataHandle hProcedureMetaData);
```

### Parameters

## **hProcedureMetaData**

[in] Procedure metadata.

### **Return Value**

A string list handle which contains the names of the output columns of the procedure. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseStringList` method after it is no longer in use.

## **GetProcedureType**

Method used to get the type of the procedure.

The returned value is one of:

- DeleteProcedure
- UpdateProcedure
- SelectProcedure
- InsertProcedure

```
long GetProcedureType(ProcedureMetaDataHandle hProcedureMetaData);
```

### **Parameters**

#### **hProcedureMetaData**

[in] Procedure metadata.

### **Return Value**

The type of the procedure. If there is an error -1 is returned.

## **UsedForSQL**

Method used to determine if the procedure is a used for SQL.

```
int UsedForSQL(ProcedureMetaDataHandle hProcedureMetaData);
```

### **Parameters**

#### **hProcedureMetaData**

[in] Procedure metadata.

### **Return Value**

An integer value indicating whether or not the procedure is used for SQL(true = !0, false = 0).



## IsRequiredFilter

Method used to determine if the given column is a required filter.

```
int IsRequiredFilter(ProcedureMetaDataHandle hProcedureMetaData, const CharType *strColumnName);
```

### Parameters

#### hProcedureMetaData

[in] Procedure metadata.

#### strColumnName

[in] column name.

### Return Value

An integer value indicating whether or not the column is a required filter(true = !0, false = 0).

## IsRequiredInput

ethod used to determine if the given column is a required input.

```
int IsRequiredInput(ProcedureMetaDataHandle hProcedureMetaData, const CharType *strColumnName);
```

### Parameters

#### hProcedureMetaData

[in] Procedure metadata.

#### strColumnName

[in] column name.

### Return Value

An integer value indicating whether or not the column is a required input(true = !0, false = 0).

## SupportsDirectInserts

Method used to determine if the recordset supports direct inserts.

```
int SupportsDirectInserts(RecordSetMetaDataHandle hRecordSetMetaData);
```

### Parameters

#### hRecordSetMetaData

[in] Recordset metadata.

### Return Value

An integer value indicating whether or not the recordset supports direct inserts(true = !0, false = 0).

## SupportsSelect

Method used to determine if the recordset supports select.

```
int SupportsSelect(RecordSetMetaDataHandle hRecordSetMetaData);
```

Parameters  
hRecordSetMetaData [in] Recordset metadata. Return Value An integer value indicating whether or not the recordset supports select(true = !0, false = 0).

## GetFieldNames

Method used to get a list of the field names for the recordset.

```
StringListHandle GetFieldNames(RecordSetMetaDataHandle hRecordSetMetaData);
```

### Parameters

#### hRecordSetMetaData

[in] Recordset metadata.

### Return Value

A string list handle which contains the names of the fields in the recordset. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseStringList method after it is no longer in use.

## GetScrollMovements

Method used to get a list of the scroll movements implemented for the record set.

```
long GetScrollMovements(RecordSetMetaDataHandle hRecordSetMetaData, long *pnScrollMovements, long nBufferLen);
```

### Parameters

#### hRecordSetMetaData

[in] Recordset metadata.

#### pnScrollMovements

[out] Buffer to retrieve the scroll movements.

#### nBufferLen

[in] The length of the buffer.

#### Return Value

The length of the scroll movements retrieved. If there is an error -1 is returned.

### GetScrollOperation

Method used to get the operation for the given scroll movement.

Scroll Movement values:

ScrollHome

ScrollEnd

ScrollLineUp

ScrollLineDown

ScrollPageUp

ScrollPageDown

```
long GetScrollOperation(RecordSetMetaDataHandle hRecordSetMetaData, long
nScrollMovement, CharType *strScrollOperation, long nBufferLen);
```

#### Parameters

##### hRecordSetMetaData

[in] Recordset metadata.

##### nScrollMovements

[in] Buffer to retrieve the destination.

##### strScrollOperation

[out] Buffer to retrieve the scroll operation.

##### nBufferLen

[in] The length of the buffer.

#### Return Value

The length of the scroll operation retrieved. If there is an error -1 is returned.

### GetDefaultValue

Method used to get the default value of the variable.

```
long GetDefaultValue(VariableMetaDataHandle hVariableMetaData, CharType
*strDefaultValue, long nBufferLen);
```

### Parameters

#### **hVariableMetaData**

[in] Variable metadata.

#### **strDefaultValue**

[out] Buffer to retrieve the scroll value.

**nBufferLen** [in] The length of the buffer.

### Return Value

The length of the default value retrieved. If there is an error -1 is returned.

## GetInitialization

Method used to get the method of initialization for the variable.

The returned value is one of:

AtConnection

ByDefault

Uninitialized

```
long GetInitialization(VariableMetaDataHandle hVariableMetaData);
```

### Parameters

#### **hVariableMetaData**

[in] Variable metadata.

### Return Value

The type of initialization for the variable. If there is an error -1 is returned.

## IsEncrypted

Method used to determine if the variable is encrypted.

```
int IsEncrypted(VariableMetaDataHandle hVariableMetaData);
```

### Parameters

#### **hVariableMetaData**

[in] Variable metadata.

#### **Return Value**

An integer value indicating whether or not the variable is encrypted(true = !0, false = 0).

### **IsHidden**

Method used to determine if the variable is hidden.

```
int IsHidden(VariableMetaDataHandle hVariableMetaData);
```

#### **Parameters**

##### **hVariableMetaData**

[in] Variable metadata.

#### **Return Value**

An integer value indicating whether or not the variable is hidden(true = !0, false = 0).

### **IsReadable**

Method used to determine if the variable is readable.

```
int IsReadable(VariableMetaDataHandle hVariableMetaData);
```

#### **Parameters**

##### **hVariableMetaData**

[in] Variable metadata.

#### **Return Value**

An integer value indicating whether or not the variable is readable(true = !0, false = 0).

### **IsWritable**

Method used to determine if the variable is writable.

```
int IsWritable(VariableMetaDataHandle hVariableMetaData);
```

#### **Parameters**

##### **hVariableMetaData**

[in] Variable metadata.

#### **Return Value**

An integer value indicating whether or not the variable is writeable(true = !0, false = 0).

## GetVariableType

Method used to get the type of the variable.

The returned value is one of:

SettingVariable

UserVariable

```
long GetVariableType(VariableMetaDataHandle hVariableMetaData);
```

### Parameters

#### hVariableMetaData

[in] Variable metadata.

### Return Value

The type of the variable. If there is an error -1 is returned.

## IsBlinking

Method to determine if the terminal attribute is blinking.

```
int IsBlinking(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### hTerminalAttributes

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is blinking(true = !0, false = 0).

## IsReverse

Method to determine if the terminal attribute is reverse.

```
int IsReverse(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### hTerminalAttributes

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is reverse(true = !0, false = 0).

## IsUnderscore

Method to determine if the terminal attribute is underscore.

```
int IsUnderscore(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### hTerminalAttributes

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is underscore(true = !0, false = 0).

## IsHalfBrite

Method to determine if the terminal attribute is half brite.

```
int IsHalfBrite(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### hTerminalAttributes

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is half brite(true = !0, false = 0).

## IsProtected

Method to determine if the terminal attribute is protected.

```
int IsProtected(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### hTerminalAttributes

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is protected(true = !0, false = 0).

## IsNumeric

Method to determine if the terminal attribute is numeric.

```
int IsNumeric(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### **hTerminalAttributes**

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is numeric(true = !0, false = 0).

## IsNondisplay

Method to determine if the terminal attribute is nondisplay.

```
int IsNondisplay(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### **hTerminalAttributes**

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is nondisplay(true = !0, false = 0).

## IsColumnSeparated

Method to determine if the terminal attribute is column separated.

```
int IsColumnSeparated(TerminalAttributesHandle hTerminalAttributes);
```

### Parameters

#### **hTerminalAttributes**

[in] Terminal attribute.

### Return Value

An integer value indicating whether or not the terminal attribute is column separated(true = !0, false = 0).

## IsPenDetect



Method to determine if the terminal attribute is pen detect.

```
int IsPenDetect(TerminalAttributesHandle hTerminalAttributes);
```

#### Parameters

##### **hTerminalAttributes**

[in] Terminal attribute.

#### Return Value

An integer value indicating whether or not the terminal attribute is pen detect(true = !0, false = 0).

### IsIntense

Method to determine if the terminal attribute is intense.

```
int IsIntense(TerminalAttributesHandle hTerminalAttributes);
```

#### Parameters

##### **hTerminalAttributes**

[in] Terminal attribute.

#### Return Value

An integer value indicating whether or not the terminal attribute is intense(true = !0, false = 0).

### GetColor

Method used to get the terminal attribute's color value.

```
long GetColor(TerminalAttributesHandle hTerminalAttributes);
```

#### Parameters

##### **hTerminalAttributes**

[in] Terminal attribute.

#### Return Value

The type of terminal attributes color value. If there is an error -1 is returned.

### ReleaseTerminalAttributes

Method used to release a terminal attribute.

```
int ReleaseTerminalAttributes(TerminalAttributesHandle hTerminalAttributes);
```

## Parameters

### hTerminalAttributes

[in] Terminal attribute.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetAttributeLocations

Method used to the locations for a given set of attributes.

```
ElementLocationListHandle GetAttributeLocations(SessionHandle hSession,
StringListHandle hAttributeNames);
```

## Parameters

### hSession

[in] The Host Integrator session.

### hAttributeNames

[in] Names of the attributes to get locations.

## Return Value

A element location list handle which contains the element locations for the given attributes. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseElemLocList method after it is no longer in use.

## GetPatternLocations

Method used to the locations for a given set of patterns.

```
ElementLocationListHandle GetPatternLocations(SessionHandle hSession,
StringListHandle hPatternNames);
```

## Parameters

### hSession

[in] The Host Integrator session.

### hPatternNames

[in] Names of the patterns to get locations.

## Return Value

A element location list handle which contains the element locations for the given attributes. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseElemLocList method after it is no longer in use.

## GetFieldLocations

Method used to the locations for a given set of fields.

```
ElementLocationListHandle GetFieldLocations(SessionHandle hSession,
StringListHandle hFieldNames);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### hPatternNames

[in] Names of the fields to get locations.

### Return Value

A element location list handle which contains the element locations for the given attributes. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseElemLocList method after it is no longer in use.

### Remarks

Recordset fields are always linear regions.

This method is guaranteed to work correctly only when a record in the recordset is selected. If no record is selected:

- For recordsets containing fixed records, the field locations returned are those for the first record in the recordset, based upon the information stored in the model.
- For recordsets containing variable-length records, the information returned is not meaningful.

## GetRecordSetLocations

Method used to the locations for a given set of recordsets.

```
ElementLocationListHandle GetRecordSetLocations(SessionHandle hSession,
StringListHandle hRecordSetNames);
```

### Parameters

### **hSession**

[in] The Host Integrator session.

### **hPatternNames**

[in] Names of the recordsets to get locations.

### **Return Value**

A element location list handle which contains the element locations for the given attributes. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseElemLocList method after it is no longer in use.

## **GetHomeEntityName**

Get the name of the home entity.

```
long GetHomeEntityName(SessionHandle hSession, CharType *strEntityName, long nBufferLen);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

#### **strEntityName**

[out] Buffer to retrieve the entity name.

#### **nBufferLen**

[in] The length of the buffer.

### **Return Value**

The length of the string retrieved. If there is an error -1 is returned.

## **GetTerminalFieldAtCursor**

Method used to the terminal field at the cursor.

```
TerminalFieldHandle GetTerminalFieldAtCursor(SessionHandle hSession);
```

### **Parameters**

#### **hSession**

[in] The Host Integrator session.

## Return Value

A terminal field handle. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseTermField` method after it is no longer in use.

## InsertRecords

Method used to perform the insert operation for the current recordset.

```
int InsertRecords(SessionHandle hSession, StringMapSetHandle hRecords);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### hRecords

[in] A list of maps containing name-value pairs for the records to insert.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## InsertStringAtCursor

Method used to insert a string at the cursor on the current entity on a host session.

```
int InsertStringAtCursor(SessionHandle hSession, const CharType *strString);
```

### Parameters

#### hSession

[in] The Host Integrator session.

#### strString

[in] The string.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetLoggingLevel

Method used to get the logging level for an Host Integrator Server Session.

The returned value is one of:

Errors  
ErrorsAndWarnings  
All

```
long GetLoggingLevel(SessionHandle hSession);
```

#### Parameters

##### **hSession**

[in] The Host Integrator session.

##### **Return Value**

The logging level value.

### SetLoggingLevel

Method used to set the logging level for a Host Integrator Server Session.

The value to set is one of:

Errors  
ErrorsAndWarnings  
All

```
int SetLoggingLevel(SessionHandle hSession, long nLoggingLevel);
```

#### Parameters

**hSession** [in] The Host Integrator session.

##### **nLoggingLevel**

[in] The logging level.

##### **Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### WaitForCondition

Method used to wait on the given conditions. Exception occurs if timeout period elapses before the conditions are met.

```
int WaitForCondition(SessionHandle hSession, long nTimeout, const CharType
*strExpression, const CharType *strEntityName);
```

#### Parameters

**hSession**

[in] The Host Integrator session.

**nTimeout**

[in] The timeout period in milliseconds.

**strExpression**

[in] The condition expression.

**strEntityName**

[in] The entity name.

**Return Value**

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetElemLocElementName

Method used to get the name of the element location.

```
long GetElemLocElementName(ElementLocationHandle hElementLocation, CharType
*strElementName, long nBufferLen);
```

**Parameters****hElementLocation**

[in] Element location.

**strElementName**

[out] Buffer to name.

**nBufferLen**

[in] The length of the buffer.

**Return Value**

The length of the default value retrieved. If there is an error -1 is returned.

## GetElemLocElementType

Method used to get the type of the element.

The returned value is one of:

Attribute  
Field  
Pattern  
RecordSet

```
long GetElemLocElementType(ElementLocationHandle hElementLocation);
```

#### Parameters

##### **hElementLocation**

[in] Element location.

#### Return Value

The type of the element location. If there is an error -1 is returned.

### GetElemLocTopRow

Method used to get the top row of the location.

```
long GetElemLocTopRow(ElementLocationHandle hElementLocation);
```

#### Parameters

##### **hElementLocation**

[in] Element location.

#### Return Value

The top row of the element location. If the region is linear-1 is returned.

### GetElemLocLeftColumn

Method used to get the left column of the location.

```
long GetElemLocLeftColumn(ElementLocationHandle hElementLocation);
```

#### Parameters

##### **hElementLocation**

[in] Element location.

#### Return Value

The left column of the element location. If the region is linear-1 is returned.



## GetElemLocNumRows

Method used to get the number of rows of the location.

```
long GetElemLocNumRows(ElementLocationHandle hElementLocation);
```

### Parameters

#### hElementLocation

[in] Element location.

### Return Value

The number of rows of the element location. If the region is linear-1 is returned.

## GetElemLocNumColumns

Method used to get the number of columns of the location.

```
long GetElemLocNumColumns(ElementLocationHandle hElementLocation);
```

### Parameters

#### hElementLocation

[in] Element location.

### Return Value

The number of columns of the element location. If the region is linear-1 is returned.

## GetElemLocLength

Method used to get the offset of the location.

```
long GetElemLocOffset(ElementLocationHandle hElementLocation);
```

### Parameters

#### hElementLocation

[in] Element location.

### Return Value

The offset of the element location. If the region is rectangular -1 is returned.

## GetElemLocRegionType

Method used to get the type of the region.

The returned value is one of:

Linear

Rectangular

```
long GetElemLocRegionType(ElementLocationHandle hElementLocation);
```

#### Parameters

##### **hElementLocation**

[in] Element location.

#### Return Value

The type of the region. If there is an error -1 is returned.

### ReleaseErrorMessageInfo

Release an ErrorInfoHandle and free resources associated with it.

```
int ReleaseErrorMessageInfo(ErrorInfoHandle hError);
```

#### Parameters

##### **hErrorInfo**

ErrorInfoHandle obtained by calling GetErrorMessageInfo()

#### Return Value

int - (succeeded = !0, failed = 0)

### ReleaseElemLoc

Method used to release an Element Location.

```
int ReleaseElemLoc(ElementLocationHandle hElementLocation);
```

#### Parameters

##### **hElementLocation**

[in] Element location.

#### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetNumElemLocs

Method used to get the number of element locations in the list.

```
long GetNumElemLocs(ElementLocationListHandle hElementLocationList);
```

### Parameters

#### **hElementLocationList**

[in] An element location list handle.

### Return Value

The number of element locations in the list.

## GetElemLocAt

Method used to get an element location in the list by index.

```
ElementLocationHandle GetElemLocAt(ElementLocationListHandle
hElementLocationList, long nIndex);
```

### Parameters

#### **hElementLocationList**

[in] An element location list handle.

#### **nIndex**

[in] Index for an element location.

### Return Value

A element location handle. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseElemLoc method after it is no longer in use.

## ReleaseElemLocList

Method used to release an Element Location List.

```
int ReleaseElemLocList(ElementLocationListHandle hElementLocationList);
```

### Parameters

#### **hElementLocationList**

[in] Element location list.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

### GetTermFieldTopRow

Method used to get the top row of the terminal field.

```
long GetTermFieldTopRow(TerminalFieldHandle hTerminalField);
```

#### Parameters

##### hTerminalField

[in] Terminal field.

#### Return Value

The top row of the terminal field. If there is an error -1 is returned.

### GetTermFieldLeftColumn

Method used to get the left column of the terminal field.

```
long GetTermFieldLeftColumn(TerminalFieldHandle hTerminalField);
```

#### Parameters

##### hTerminalField

[in] Terminal field.

#### Return Value

The left column of the terminal field. If there is an error -1 is returned.

### GetTermFieldOffset

Method used to get the offset of the terminal field.

```
long GetTermFieldOffset(TerminalFieldHandle hTerminalField);
```

#### Parameters

##### hTerminalField

[in] Terminal field.

#### Return Value

The offset of the terminal field. If there is an error -1 is returned.

## GetTerminalFieldTermAttr

Method used to the terminal attributes of the terminal field.

```
TerminalAttributesHandle GetTerminalFieldTermAttr(TerminalFieldHandle hTerminalField);
```

### Parameters

#### hTerminalField

[in] Terminal field.

### Return Value

A terminal attributes handle. If the retrieval fails the return value is null. The handle must be destroyed with the ReleaseTerminalAttributes method after it is no longer in use.

## ReleaseTermField

Method used to release a Terminal Field

```
int ReleaseTermField(TerminalFieldHandle hTerminalField);
```

### Parameters

#### hTerminalField

[in] Terminal field.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## CreateStringMapSet

Create an AppConn string map list.

```
StringMapSetHandle CreateStringMapSet(void);
```

### Return Value

A handle to a string map list.

## AddStringMapToSet

Method used to add a string map to the string map list.

```
int AddStringMapToSet(StringMapSetHandle hStringMapSet, const StringMapHandle
hStringMap);
```

## Parameters

### **hStringMapSet**

[in] AppConn string map list.

### **hStringMap**

[in] String map to add to the list.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## ClearStringMapSet

Method used to remove all the strings maps from the string map list.

```
int ClearStringMapSet(StringMapSetHandle hStringMapSet);
```

## Parameters

### **hStringMapSet**

[in] AppConn string map list.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetNumStringMaps

Method used to get the number of string maps in the string map list.

```
long GetNumStringMaps(StringMapSetHandle hStringMapSet);
```

## \*\*Parameters

### **hStringMapSet**

[in] AppConn string map list.

## Return Value

The number of elements in the AppConn string map list.

## GetStringMapAt

Method used to get a string in the string list.

```
StringMapHandle GetStringMapAt(StringMapSetHandle hStringMapSet, long nIndex);
```

## Parameters

### **hStringMapSet**

[in] AppConn string map list.

### **nIndex**

[in] Index of the string to get.

## Return Value

A string map handle. If the retrieval fails the return value is null. The handle must be destroyed with the `ReleaseStringMap` method after it is no longer in use.

## ReplaceStringMapAt

Method used to replace a string map in the string map list.

```
int ReplaceStringMapAt(StringMapSetHandle hStringMapSet, long nIndex, const StringMapHandle hStringMap);
```

## Parameters

### **hStringMapSet**

[in] AppConn string map list.

### **nIndex**

[in] Index of the string to get.

### **hStringMap**

[in] String Map to replace the existing entry.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## RemoveStringMapAt

Method used to remove a string map from the string map list.

```
int RemoveStringMapAt(StringMapSetHandle hStringMapSet, long nIndex);
```

## Parameters

### **hStringMapSet**

[in] AppConn string map list.

## nIndex

[in] Index of the string map to remove.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## ReleaseStringMapSet

Method used to release an AppConn string map list. All elements of the string map list are released.

```
int ReleaseStringMapSet(StringMapSetHandle hStringMapSet);
```

## Parameters

### hStringMapSet

[in] AppConn string map list.

## Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetErrorCode

Method used to get the error code for an AppConn object.

```
long GetErrorCode(SessionHandle hSession);
```

## Parameters

### hSession

[in] AppConn object

## Return Value

The error code for the last reported error.

## GetErrorMessage

Method used to get the error message for an AppConn object.

```
long GetErrorMessage(SessionHandle hSession, long nIndex, CharType *strString,
long nBufferLen);
```

## Parameters



**hSession**

[in] AppConn object

**nIndex**

[in] The index of the message to select.

**strString**

[out] Buffer to retrieve the string.

**nBufferLen**

[in] The length of the buffer.

**Return Value**

The length of the message retrieved.

**GetErrorMessageInfo**

Obtain an Error Info handle for a given error message in the last set of reported errors.

```
ErrorInfoHandle GetErrorMessageInfo(SessionHandle hSession, long index);
```

**Parameters****hSession**

Session handle.

**Index**

0-based index of the error for which information is to be obtained.

**Return Value**

Handle for accessing error information, or NULL if the session handle is invalid, there are no errors, or the index value is out of range.

**GetErrorMessageCode**

Return the error code of a given error message.

```
long GetErrorMessageCode(ErrorInfoHandle hErrorInfo);
```

**Parameters****hError**

ErrorInfoHandle obtained by calling GetErrorMessageInfo()

### Return Value

Error code, 0 if there is no error, or -1 if hError is invalid.

## GetErrorMessageCode

Return the error code of a given error message.

```
long GetErrorMessageCode(ErrorInfoHandle hErrorInfo);
```

### Parameters

#### hError

ErrorInfoHandle obtained by calling GetErrorMessageInfo()

### Return Value

Error code, 0 if there is no error, or -1 if hError is invalid.

## GetErrorMessageParameterCount

Return the number of parameter strings associated with an error message.

```
long GetErrorMessageParameterCount(ErrorInfoHandle hErrorInfo);
```

### Parameters

#### hErrorInfo

ErrorInfoHandle obtained by calling GetErrorMessageInfo()

### Return Value

Number of error parameters, of -1 if hErrorInfo is invalid

## GetErrorMessageParameter

Return a parameter string for an error message, given an index.

```
long GetErrorMessageParameter(ErrorInfoHandle hErrorInfo, long index, CharType
*strString, long nBufferLen);
```

### Parameters

#### hErrorInfo

ErrorInfoHandle obtained by calling GetErrorMessageInfo()

#### index

0-based index of the parameter to be obtained.

### **strString**

Buffer to receive the parameter string.

### **nBufferLen**

Maximum number of characters the buffer can accept.

### **Return Value**

number of characters returned, or -1 if argument(s) are invalid.

## **ReleaseErrorMessageInfo**

Release an ErrorInfoHandle and free resources associated with it.

```
int ReleaseErrorMessageInfo(ErrorInfoHandle hError);
```

### **Parameters**

#### **hErrorInfo**

ErrorInfoHandle obtained by calling GetLastErrorInfo()

### **Return Value**

```
int - (succeeded = !0, failed = 0)
```

## **GetLocale**

Method used to get the locale for an AppConn session. By default, this method returns -1 and internally posts an "Unknown Error" message. If the user has set a locale string by a prior call to SetLocale(), that string is returned by GetLocale().

```
long GetLocale(SessionHandle hSession, CharType *strLocale, long nBufferLen);
```

### **Parameters**

#### **hSession**

[in] AppConn object

#### **strLocale**

[out] buffer to retrieve the locale

#### **nBufferLen**

[in] The length of the buffer.

### Return Value

The length of the locale retrieved.

## SetLocale

Method used to set the locale for an AppConn object.

```
int SetLocale(SessionHandle hSession, const CharType *strLocale);
```

### Parameters

#### hSession

[in] AppConn object.

#### strLocale

[in] the locale.

### Return Value

An integer value indicating whether or not the call succeeded (succeeded = !0, failed = 0).

## GetLocalizedMessage

Method used to get the localized error message for an AppConn object.

```
long GetLocalizedMessage(Handle hError, long nIndex, CharType *strString, long
nBufferLen);
```

### Parameters

#### hError Handle

[in] AppConn object.

#### long nIndex

The zero-based index into the message list

#### strString

[out] buffer to retrieve the message.

#### nBufferLen

[in] The length of the buffer.

### Return Value

The length of the message retrieved.

### **GetMajorVersion**

Method used to get version information for the AppConn connector.

```
long GetMajorVersion(void);
```

#### **Return Value**

The major version number.

### **GetMinorVersion**

Method used to get version information for the AppConn connector.

```
long GetMinorVersion(void);
```

#### **Return Value**

The minor version number.

### **GetModelVariableNames**

Method used to get the names of all variables in the Host Integrator model.

```
StringListHandle GetModelVariableNames(SessionHandle hSession);
```

#### **Parameters**

**hSession** SessionHandle

[in] AppConn object.

**strStringMapHandle**

[out] buffer to retrieve the model variables.

#### **Return Value**

The variable names.

#### **Remarks**

One potential reason for failure is that the server session has not been established.

### **GetNumMessages**

Method used to get the number of error messages for an AppConn object.

```
long GetNumMessages(SessionHandle hSession);
```

#### Parameters

##### hSession

[in] AppConn object.

#### Return Value

The number of error messages.

### GetOperationTimeout

Method used to get the timeout (in seconds) of the operation.

```
long GetOperationTimeout(OperationMetaDataHandle hOperationMetaData);
```

#### Parameters

hOperationMetaData OperationMetaDataHandle

[in] AppConn object.

#### Return Value

The timeout.

### GetTermFieldLength

Method used to get the length of a field.

```
long GetTermFieldLength(TerminalFieldHandle hTerminalField);
```

#### Parameters

hTerminalField TerminalFieldHandle

[in] AppConn Terminal Field

#### Return Value

The length of the field.

### GetVersionString

Method used to get version information for the AppConn connector.

```
long GetVersionString(CharType *strVersionString, long nBufferLen);
```

#### strVersionString

[in] Buffer to retrieve the version string

**nBufferLen**

[in] The length of the buffer.

**Return Value**

The length of the version string retrieved.

## **EXAMPLE**

**Building a C Application on Any Development Platform**

## 4.16.8 Event Handler API

---

Event handling is a feature in Verastream Host Integrator that extends the capabilities of models by allowing you to define specific events that suspend the interpretation of a model and turn control over to user-supplied procedural code.

The [Event Handler Guidelines](#) topic provides basic information on designing and implementing event handlers.

### Java Event Handlers API

The Java Event Handler API (Javadoc) provides detailed information on the Java classes and interfaces that you can use in your event handler code.

The public API for event handling is largely contained in the `com.wrq.vhi.script.api` package. Most of the classes and interfaces found in this package fall into one of the following categories.

### .NET Event Handlers API

The .NET Event Handler API (available in a Windows help file) provides detailed information on the .NET classes and methods you can use in your event handler code.

### Event Handler Classes

There are eight event handler classes, one for each type of object in Host Integrator that can have an event handler attached:

LifeCycleEventHandler

ModelEventHandler

EntityEventHandler

AttributeEventHandler

OperationEventHandler

RecordSetEventHandler

FieldEventHandler

ProcedureEventHandler

Any class that extends one of these base classes is an event handler and is, by definition, attached to an appropriate object (entity, attribute, etc) in the model. You typically create such a class using the Design Tool, and then modify it using the editor of your choice. Each event handler class defines a set of methods, with each method corresponding to an event in Host Integrator. To implement an event, a derived class simply needs to override one of these methods. For performance reasons, however, it is important to only override those methods that do something beyond the default action.

### Event Interfaces



Each method on an event handler class takes a single event interface as an argument. This event interface provides the contextual information for the event and is different for each method. Therefore there is one event interface per event in Host Integrator.

In addition, there is an inheritance hierarchy of intermediate event interfaces that mirrors the object hierarchy in models. Either directly or indirectly, all event interfaces extend the base Event interface. There are intermediate interfaces that are shared by many events such as the EntityEvent interface or the AttributeEvent interface. In fact, the AttributeEvent interface extends the EntityEvent interface because all attributes are defined within the context of an entity.

### Callback Interfaces

A few of the interfaces in the `com.wrq.vhi.script.api` package are for making callbacks to Host Integrator, in order to obtain contextual information, modify the model state, or modify the terminal. When available, these interfaces are obtained via an event's event interface. The callback interfaces are:

`ModelContext` - Use the methods in this interface to obtain information about the model.

`ScriptHostSession` - Use this interface to get information about the host session and to manipulate the terminal at various levels (direct, model, table).

`ClientSession` - Use this interface to obtain information about the currently connected client.

`RemoteHostSession (Java Only)` - Use this interface to establish a session with a Host Integrator model other than the one containing the event handler.

`Logger` - Use this interface to make entries in the Host Integrator log file.

## 4.16.9 Managing Model Variable Lists

---

The `com.wrq.vhi.sconfig` package contains methods for managing model variable lists on Host Integrator Servers. This makes it possible for you to update username/password lists for hosts that enforce regular expiration of usernames and passwords, hosts that only allow a credentialed user one concurrent host session, and so forth. Use the classes in the `com.wrq.vhi.sconfig` package to manage the model variable lists on any Host Integrator Server in your domain.

### Code Samples

[Show model variable lists](#)

Reverse model variable list value strings

## **SHOW MODEL VARIABLE LISTS**

```

/* Example Program :
NOTE: As of the 7.0 release AADS has been replaced with the Management Server. Although references to
AADS still remain in the APIs the code samples will work with the Management Server.

Display model variable lists and their contents.

Compile: javac -classpath sconfig.jar;bc-fips-1.0.2.3.jar;bctls-fips-1.0.14.1.jar ShowMVLs.java

Run: java -classpath .;sconfig.jar;wcp.jar;bc-fips-1.0.2.3.jar;bctls-fips-1.0.14.1.jar ShowMVLs <ManagementServerName> <ServerName>

Before running this program, edit the username and password variables to provide credentials for a user that is
part of the Administrator profile.
*/

import java.io.IOException;
import java.util.*;
import com.wrq.vhi.sconfig.*;
import java.security.Security;

import org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider;
import org.bouncycastle.jsse.provider.BouncyCastleJsseProvider;
import org.bouncycastle.crypto.CryptoServicesRegistrar;

class ShowMVLs
{
 public static final String DIVIDER = "=====";

 public static final String username = "";
 public static final String password = "";

 /* Display the contents of a single model variable list. */
 static void showMVL(IMvlBrowser mvlb, String mvlname)
 {
 VhiMvl mvl = mvlb.getMvl(mvlname);

 /* For each MVL entry */
 for(int i=0; i<mvlb.size(); i++) {
 Map mvle = mvlb.get(i);
 /* For each variable/value pair in the MVL entry */
 for(Iterator it=mvle.entrySet().iterator(); it.hasNext();) {
 Map.Entry item = (Map.Entry)it.next();
 String vname = (String)item.getKey();
 String vval = (String)item.getValue();
 System.out.println(" Entry " + (i+1) + " : Variable : " + vname + " Value : " + vval);
 }
 System.out.println("");
 }
 }

 /* Display all of the MVLs defined on a Host Integrator Server. */
 static void displayMVLs(String mgmtServerName, String serverName)
 throws SCEException, IOException, ClassNotFoundException
 {
 IAADSConnection mgmtServerConn =
 ServerConfig.newAAADSConnection(mgmtServerName);
 IServerAdminSession sas =
 ServerConfig.newServerAdminSession(serverName, mgmtServerConn, username, password);

 /* Register the Bouncy Castle providers */
 registerProviders();

 sas.open(); /* Connect to the Host Integrator Server */

 System.out.println(DIVIDER);
 System.out.println("Management Server: " + mgmtServerName);
 System.out.println("Server: " + serverName);

 IMvlBrowser mvlb = ServerConfig.newMvlBrowser(sas);

 /* Read the MVL configuration from the Host Integrator Server. */
 mvlb.readConfiguration();

 /* Get a list of the names of MVLs defined on this server. */
 List mvlNames = mvlb.getMvlNames();
 System.out.println("Number of MVLs : " + mvlNames.size());

 /* Display the variables/values for each MVL */
 for(Iterator it1=mvlNames.iterator(); it1.hasNext();) {
 String mvlName = (String)it1.next();
 System.out.println(DIVIDER);
 System.out.println("MVL Name: " + mvlName);
 showMVL(mvlb, mvlName);
 }

 sas.close(); /* Disconnect from the Host Integrator Server. */
 }

 static void registerProviders()
 {
 // initialize a few properties used by BCFIPS and BCJSSE
 Security.setProperty("ssl.KeyManagerFactory.algorithm", "PKIX");
 System.setProperty("org.bouncycastle.ec.disable_mqv", "true");
 System.setProperty("org.bouncycastle.jsse.ec.disableChar2", "true");

 // the next line shows how to place BCFIPS in approved-only mode.
 // CryptoServicesRegistrar.setApprovedOnlyMode(true);
 Security.insertProviderAt(new BouncyCastleFipsProvider(), 1);
 Security.insertProviderAt(new BouncyCastleJsseProvider(), 2);
 }
}

```

```
}

static public void main(String [] args)
{
 if(args.length != 2) {
 System.out.println("Usage: ShowMVLs <ManagementServerName> <Servername>");
 return;
 }
 try {
 displayMVLs(args[0], args[1]);
 }
 catch(Exception e) {
 System.err.println("Exception caught: " + e);
 e.printStackTrace();
 }
}
}
```

## REVERSE MODEL VARIABLE LIST VALUE STRINGS

```

/* Example Program :
NOTE: As of the 7.0 release AADS has been replaced with the Management Server. Although references to
AADS still remain in the APIs the code samples will work with the Management Server.

Reverse the model variable list value strings, of each MVL on a Host Integrator Server.

Compile: javac -classpath sconfig.jar;bc-fips-1.0.2.3.jar;bctls-fips-1.0.14.1.jar ReverseMVLs.java

Run: java -classpath .;sconfig.jar;wcp.jar;bc-fips-1.0.2.3.jar;bctls-fips-1.0.14.1.jar ReverseMVLs <ManagementServerName> <ServerName>

Use with care! This program changes the Host Integrator Server's configuration.

Before running this program, edit the username and password variables to provide credentials for a user that is
part of the Administrator profile.
*/
import com.wrq.vhi.sconfig.*;

import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Security;

import org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider;
import org.bouncycastle.jsse.provider.BouncyCastleJsseProvider;
import org.bouncycastle.crypto.CryptoServicesRegistrar;

class ReverseMVLs
{
 public static final String DIVIDER = "=====";
 public static final String username = "";
 public static final String password = "";

 /* Display the contents of a single model variable list. */
 static void reverseMvlValues(IMvlBrowser mvlb, String mvlName)
 throws SCEException
 {
 VhiMvl mvl = mvlb.getMvl(mvlName);
 for(int i=0; i<mvl.size(); i++) {
 Map mvle = mvl.get(i);

 for(Iterator it=mvle.entrySet().iterator(); it.hasNext();) {
 Map.Entry item = (Map.Entry)it.next();
 String vname = (String)item.getKey();
 String vval = (String)item.getValue();

 /* Reverse the characters in the value string. */
 String newval = new StringBuffer(vval).reverse().toString();

 /* Change the value of an MVL entry. */
 mvle.put(vname, newval);
 System.out.println(" Entry " + (i+1) + " : Variable : " + vname + " Old Value : " + vval + " New Value : " + newval);
 }
 System.out.println("");

 mvl.set(i, mvle); /* Update the MVL entry. */
 }

 mvlb.setMvl(mvl); /* Update the MVL with our changes. */
 }

 /* Display all of the MVLs defined on a Host Integrator Server. */
 static void reverseMVLs(String mgmtServerName, String serverName)
 throws SCEException, IOException, ClassNotFoundException
 {
 IAADSConnection mgmtServerConn =
 ServerConfig.newAADSConnection(mgmtServerName);
 IServerAdminSession sas =
 ServerConfig.newServerAdminSession(serverName, mgmtServerConn, username, password);

 /* Register the Bouncy Castle providers */
 registerProviders();

 sas.open(); /* Connect to the Host Integrator Server. */

 System.out.println("Management Server: " + mgmtServerName);
 System.out.println("Server: " + serverName);

 IMvlBrowser mvlb = ServerConfig.newMvlBrowser(sas);

 mvlb.readConfiguration(); /* Read the MVLs from the Host Integrator Server. */

 List mvlNames = mvlb.getMvlNames();

 for(Iterator it1=mvlNames.iterator(); it1.hasNext();) {
 String mvlName = (String)it1.next();
 System.out.println(DIVIDER);
 System.out.println("MVL Name: " + mvlName);
 reverseMvlValues(mvlb, mvlName);
 }

 mvlb.updateConfiguration(); /* Write the updated MVLs to the Host Integrator Server. */

 sas.close(); /* Disconnect from the Host Integrator Server. */
 }

 static void registerProviders()
 {

```

```

// initialize a few properties used by BCFIPS and BCJSSE
Security.setProperty("ssl.KeyManagerFactory.algorithm", "PKIX");
System.setProperty("org.bouncycastle.ec.disable_mqv", "true");
System.setProperty("org.bouncycastle.jsse.ec.disableChar2", "true");

// the next line shows how to place BCFIPS in approved-only mode.
// CryptoServicesRegistrar.setApprovedOnlyMode(true);
Security.insertProviderAt(new BouncyCastleFipsProvider(), 1);
Security.insertProviderAt(new BouncyCastleJsseProvider(), 2);
}

static public void main(String [] args)
{
 if(args.length != 2) {
 System.out.println("Usage: ReverseMVLs <ManagementServerName><Servername>");
 return;
 }

 try {
 reverseMVLs(args[0], args[1]);
 }
 catch(Exception e) {
 System.err.println("Exception caught: " + e);
 e.printStackTrace();
 }
}
}
}

```

## 4.16.10 Using HLLAPI Adapters to Migrate Legacy Programs

---

By migrating traditional terminal emulation programs to Host Integrator's powerful integration platform, you make them available to Web applications, mobile apps, and SOA Web services. Centralized server based solutions mean that you no longer have to install terminal emulation programs on individual desktops.

The two Host Integrator HLLAPI (High Level Language Application Program Interface) Adapters provide a simple migration path:

Host Integrator contains a .NET class library, EXTRA! COM, that is also accessible through a COM interface and provides you with the ability to migrate programs from EXTRA! Extreme 9.3. The EXTRA! COM library currently supports only 3270 terminal type sessions.

Enterprise Access Objects provide a migration path from the e-Vantage SDK, using Visual Basic, to Host Integrator.

### Migrating EXTRA! Extreme Applications

By migrating EXTRA! terminal emulation applications to Host Integrator's powerful integration platform, you make them available to Web applications, mobile apps, and SOA Web services. Host Integrator contains a .NET class library, EXTRA! COM, that is also accessible through a COM interface and provides you with the ability to do just that.

The EXTRA! COM library currently supports only 3270 terminal type sessions.

#### How does it work?

The EXTRA! COM library is a .NET class library that contains methods and properties that closely resemble the EXTRA COM API. Your VB.NET or C# programs use the EXTRA API to make .NET to COM calls to invoke methods provided by EXTRA. Since the EXTRA! COM library does not cache, both EXTRA! COM and VHI connector calls work together seamlessly and provide a smooth transition from EXTRA to VHI.

You can migrate VB.NET, C#, or Microsoft Office Visual Basic for Applications (VBA) programs.

### HOW TO MIGRATE EXTRA! PROGRAMS

The classes, methods, and properties that comprise the EXTRA! COM HLLAPI Adapter are documented in a MSDN-style compiled help system that is included with Verastream Host Integrator.

#### Migrating VB.NET or C# Programs

1. In your .NET EXTRA program, remove the reference to the EXTRA! library, and add a reference to both the EXTRA! COM library and the .NET connector:

```
%VHI_ROOT%\lib\dotnet\Attachmate.Verastream.HostIntegrator.E2V.dll
%VHI_ROOT%\lib\dotnet\WRQ.Verastream.HostIntegrator.dll
```

- 1.
2. In your code, add using directives for these classes. The type "Screen" exists in both connectors. To remedy this confusion, use an using alias directive:

```
using Attachmate.Verastream.HostIntegrator.E2V;
using Screen = Attachmate.Verastream.HostIntegrator.E2V.Screen;
using WRQ.Verastream.HostIntegrator;
```

- 2.

In the VHI Design Tool, create a model that connects to the same host and port as the EXTRA! program. Leave the model empty (no entities) to get started.

Deploy the model.

Change your program to call Sessions.open with the model name.

Compile and debug your code.

### Migrating Microsoft Office Visual Basic for Applications (VBA) Programs

The COM interface to the EXTRA! COM library is not based on the Verastream Host Integrator COM connector. It is a .NET class library. While it is possible to use both the EXTRA! COM interface and the Host Integrator COM connector in your VBA program, these two connectors are unrelated; you cannot share sessions between the two. Microsoft Office VBA interfaces use COM; if you are not using Microsoft Office VBA, we recommend the use of .NET because it offers more flexibility.

In your Microsoft Office VBA program, remove the reference to the EXTRA! library, and add a reference to the EXTRA! COM library:

```
%VHI_ROOT%\lib\dotnet\Attachmate.Verastream.HostIntegrator.E2V.tlb
```

2. In the VHI Design Tool, create a model that connects to the same host and port as the EXTRA! program. Leave the model empty (no entities) to get started.
3. Deploy the model.
4. Change your program to call Sessions.open with the model name.
5. Compile and debug your code.

## TROUBLESHOOTING YOUR MIGRATION

Depending on the situation, there may be additional changes that you will have to make to the EXTRA! client. The following points may be helpful in successfully migrating your EXTRA! programs to Verastream Host Integrator.



- Sessions: The EXTRA! method `Sessions.Open()` takes a string which is a file name, typically a .EDP file. VHI cannot read the EDP file. To navigate in VHI, you create and deploy a model. The EXTRA! COM library offers two `Open()` methods, one that takes a string (either "model" or "model@sesssrvr"), and another that takes a `HostIntegratorRejuvenationSession`. When you use the `HostIntegratorRejuvenationSession` method you have access to all VHI features: model variables, security configuration, session pools and domains. You must add a reference to the VHI connector.
- Timing: Every call is a round trip to the server, that includes simple calls such as getting the current cursor position. You may notice that the change in timing causes functional problems or causes the application to run slower than expected. Errors: There may be differences in error conditions between EXTRA! and VHI. For example, if you place the cursor in a protected area of the screen and write some text, EXTRA! may flag this error when you write the text, but VHI may flag the error when you move the cursor.
- Exceptions: VHI uses different exceptions than EXTRA! and your code may not catch them all. Uncaught exceptions can cause a program to terminate.
- Scaling: After migration, the EXTRA! client still creates the same number of sessions as it did prior to migration. To get the same scalability in Host Integrator, you may need to create session pools.

## Migrating e-Vantage SDK Programs

The e-Vantage SDK HLLAPI Adapter is a .NET class library that contains methods and properties that closely resemble the e-Vantage SDK API. Your VB.NET or C# programs use the e-Vantage SDK API to make .NET to COM calls to invoke methods provided by e-Vantage SDK. Since the e-Vantage SDK library does not cache, both e-Vantage SDK and VHI connector calls work together seamlessly and provide a smooth transition from e-Vantage SDK programs to VHI.

You can migrate VB.NET, C#, or Microsoft Office Visual Basic for Applications (VBA) programs.

The e-Vantage SDK library currently supports only 3270 terminal type sessions.

## MIGRATING VB.NET OR C# PROGRAMS

1. In your .NET e-Vantage SDK program, remove the reference to the e-Vantage SDK library, and add a reference to both the EA02V library and the .NET connector:

1.

```
%VHI_ROOT%\lib\dotnet\Attachmate.Verastream.HostIntegrator.EA02V.dll
%VHI_ROOT%\lib\dotnet\WRQ.Verastream.HostIntegrator.dll
```

2. In your code, add using directives for these classes.

1.

1.

```
using Attachmate.Verastream.HostIntegrator.EA02V;
using WRQ.Verastream.HostIntegrator;
```

3. You do not need to deploy a model as the e-Vantage SDK adapter uses the standard model Terminal3720, which is already deployed. You can use a different model if necessary. To use a different model, In the VHI Design Tool, create a model that connects to the same host and port as the e-Vantage SDK program. Leave the model empty (no entities) to get started. Set the

property `AtmConTN3270.Name` to a string in the form `modelName@sesssrvhostname`. 4. Compile and debug your code.

1.

## MIGRATING MICROSOFT OFFICE VISUAL BASIC FOR APPLICATIONS (VBA) PROGRAMS

The COM interface to the EA02V library is not based on the Verastream Host Integrator COM connector. It is a .NET class library. While it is possible to use both the EA02V COM interface and the Host Integrator COM connector in your VBA program, these two connectors are unrelated; you cannot share sessions between the two. Microsoft Office VBA interfaces use COM; if you are not using Microsoft Office VBA, we recommend the use of .NET because it offers more flexibility.

1. In your Microsoft Office VBA program, remove the reference to the e-Vantage SDK library, and add a reference to the EA02V COM library:

1. `%VHI_ROOT%\lib\dotnet\Attachmate.Verastream.HostIntegrator.EA02V.tlb`

2. You do not need to deploy a model as the e-Vantage SDK adapter uses the standard model `Terminal3720`, which is already deployed. You can use a different model if necessary.

2. To use a different model:

In the VHI Design Tool, create a model that connects to the same host and port as the e-Vantage SDK program. Leave the model empty (no entities) to get started. Set the property `AtmConTN3270.Name` to a string in the form `modelName@sesssrvhostname`

Clear the properties `AtmConTN3270.RemoteHostAddress` and `AtmConTN3270.DestinationPort` in your program. If you do not clear these properties, the values will be used to set model variables "hostName" and "hostPort", respectively.

3. All e-Vantage SDK programs start with two assignments: assigning both the screen and connection objects to the session. Change your VBA code to use "Set" statements.

Old:

```
Sess.Screen = Scr
Sess.Connection = Conn
```

New:

```
Set Sess.Screen = Scr
Set Sess.Connection = Conn
Compile and debug your code.
```

4. Compile and debug your code.

## TROUBLESHOOTING YOUR E-VANTAGE SDK MIGRATION

Depending on the situation, there may be additional changes that you will have to make to the e-Vantage SDK! client. The following points may be helpful in successfully migrating your e-Vantage SDK programs to Verastream Host Integrator.

The e-Vantage SDK COM library currently supports only 3270 terminal type sessions.

**Sessions:** The e-Vantage SDK method `AtmSession.Connect()` uses the host name and port you specify. In VHI, these values are passed to the model `Terminal3270`. You also have the option to create and deploy a model. The EAO2V library offers two options:

You can use the overloaded `AtmSession.Connect()` that takes an argument of type `HostIntegratorRejuvenationSession`. When you use the `HostIntegratorRejuvenationSession` method you have access to all VHI features: model variables, security configuration, session pools and domains.

Or you can specify the model name in the `AtmConTN3270.Name` property using the form `model@sesssrvr`. If you use this method, you may experience connection errors. The error will be visible in the Model Debug Messages file as `Error [VHI 2922] Model variable hostName does not exist in model`. In this case, either clear the values of properties `AtmConTN3270.RemoteHostAddress` and `AtmConTN3270.DestinationPort` in your program, or add two model variables named `hostName` and `hostPort` to your model.

**Timing:** Every call is a round trip to the server, including trivial calls such as getting the current cursor position. You may notice that the change in timing causes functional problems or causes the application to run slower than expected.

**Errors:** There may be differences in error conditions between e-Vantage SDK and VHI. For example, if you place the cursor in a protected area of the screen and write some text, e-Vantage SDK may flag this error when you write the text, but VHI may flag the error when you move the cursor.

**Exceptions:** VHI uses different exceptions than e-Vantage SDK and your code may not catch them all. Uncaught exceptions can cause a program to terminate. **Scaling:** After migration, the e-Vantage SDK client still creates the same number of sessions as it did prior to migration. To get better scalability in Host Integrator, you may need to create session pools.

**Set statement required:** If you are using VBA (COM) and see this error, `Object variable or With block variable not set`, you must change your code to use Set statements. This is described in [How to Migrate e-Vantage SDK Programs to VHI](#).

# 5. Web Builder

---

## 5.1 Using Web Builder

---

## 5.1.1 Web Builder

---

Web Builder is a component of the [Host Integrator Development Kit](#) that enables you to generate and deploy web applications and component interfaces from a Host Integrator model. Web Builder uses the model to complete the encapsulation process that enables you to write new applications using legacy functionality.

Web Builder provides a variety of abstraction levels that enable you to deliver host functionality in the appropriate format for your business needs.

### Overview

Web Builder is installed with the Host Integrator Development Kit.

To open Web Builder:

- From the Design Tool: Click File > Web Builder or click the Generate New Web Project button on the standard toolbar.
- From the Start menu: Click Start > Programs > Micro Focus Verastream > Host Integrator > Web Builder.

### Generating Web Builder Projects

Web Builder creates Web applications and objects (Java beans and a .NET class library) that provide Web and application developers programmatic access to host data and functionality. Programmers use these generated Web applications to test models of host applications, to prototype a Web application that accesses host data, and to generate sample code for incorporating host data into their Web or client/server applications.

### Basic Steps to Build and Deploy a Verastream Web Builder Project

To create Web applications and component interfaces:

Use the Host Integrator Design Tool to create a model.

In the Design Tool, deploy the model to Web Builder.

Set the Web Builder and project properties you want to use to generate a project.

Use Web Builder to generate a project from the model.

Implement the project into a Web or client/server application.

### Learn about Web Builder

- [Configuring Project Properties](#)
- [Building Web Builder projects](#)
- [Customizing HTML 5 Web applications](#)
- [Working with Web projects](#)



## 5.1.2 Generating Web Builder Projects

---

Web Builder creates Web applications and objects (Java beans and a .NET class library) that provide Web and application developers programmatic access to host data and functionality. Programmers use these generated Web applications to test models of host applications, to prototype a Web application that accesses host data, and to generate sample code for incorporating host data into their Web or client/server applications.

[Steps to build a Web project](#)

[Rebuilding a project](#)

[Creating new project types](#)

### Using the Host Integrator Model

Using Web Builder, you can work with models that are deployed on any Host Integrator server or domain that is accessible from the network, on a local Host Integrator server, or with an .xml model file that has been exported from the Design Tool

### Project Types

Using a Host Integrator model, you can build either a ready-to-use Web application project, or use objects that provide developers with the building blocks necessary to create their own custom applications.

There are two current project types; applications and objects. The different project types provide:

An HTML 5 Web application that is optimized, not only for the desktop, but for mobile devices. You can access 3270 or 5250 terminal sessions from the Windows Start menu; you do not have to open the Verastream Design Tool. Simply click 3270 (5250) Terminal Session. The HTML 5 Web application provides increased terminal functionality including Hotspots, improved Aid key usage, and increased screen display space.

A choice of Java bean or a .NET class library to provide programmatic access to information residing on the host.

Host functionality exposed using tables and procedures that can span multiple host screens and applications.

The level of abstraction necessary to create new applications that are independent of the existing host application work flow.

A third project option is legacy Java and .NET Web applications.

### WEB APPLICATION PROJECTS

Web Builder projects provide a wide range of customization for the presentation and flow of the Web application you create. Web applications are complete and ready-to-use or you can modify them using Web/HTML development and design tools. Web application projects can:

- Expose both procedures and screen functionality in one Web-based presentation

- Gather procedure inputs via input forms and display outputs as tables

- Rejuvenate both modeled and unmodeled screens

  - Modeled screens can be displayed as HTML-based forms or as a simulated host screen

  - Unmodeled screens are displayed as simulated host screens

- Generate a project capable of automatically rejuvenating entire host applications by only specifying connection information (no entities) in the model

When you build a Web application project you can choose to include all screens, as well as all procedures, to provide navigation through the host application.

## OBJECTS

Web Builder wraps the model functionality into a reusable service that application developers can access through the object. These objects include:

- .NET (C#) Class Library Routines that can be used by .NET programmers to access host data through the Host Integrator model.

- Java beans Components that can be incorporated into Java-based applications.

## USER-DEFINED PROJECT TYPES

Web Builder uses project types to define and generate project files. By creating your own user-defined project types, you can build consistent-looking projects from the same customized project type.

On the Options menu, click User-Defined Project Types to view any user-defined project types. To create a new project type, click New on the User-Defined Project Types dialog box.

## Steps to build a Web project

To generate a project:

1. In Web Builder, click Project > New (or click New) to open the New Project dialog box.
2. Click the tab (Application, Object, or Legacy) corresponding to the type of project you want to generate.
3. Select the Host Integrator model or session name you want to use to generate the project. Models or session names are available as options after you have deployed them in the Verastream Host Integrator Design Tool.
4. A project name is automatically generated. You can change the name of the project. The name you enter here is used to generate folder and file names. Web Builder may alter this name to accommodate your operating system. For example, an underscore is used if spaces are not allowed, and numbers are appended to projects with the same name.

Set properties and build the project:

To use default settings, click Build on the New Project dialog box to generate the project.

To set specific properties for this project, click Properties and change the settings in the Project Properties dialog box. Click Build to generate the project.

As Web Builder generates the project, the Build Project window displays information about the success or failure of the project generation process.

## Rebuilding a project

The build process generates the project files. If you change the project settings and properties, the project is automatically rebuilt to incorporate those changes. Anytime you rebuild a project, the project files are overwritten from the starting point of the rebuild process.

During the testing of a project, you may rebuild a project many times. After you put the project into production, rebuilding a project may overwrite files you have modified. If you have modified the generated code, select to rebuild the project from the compiling project step. This compiles and deploys the project without overwriting your changes. However, if the model changed, you need to completely rebuild the project to pick up the changes in the model.

To rebuild a project:

In Web Builder, select the project to rebuild and on the Project menu, click Rebuild to open the Rebuild Project dialog box.

Select one of the processing steps as the starting point for the build. The processing steps vary depending on the type of project you are rebuilding.

Click Build.

During the rebuild process, Web Builder detects any modified files and creates a backup folder in the project folder. Depending on the type and scope of your modifications, you can select to:

- Keep your changes

Select this option if you have made complex modifications; use the file in the backups folder to manually update your modified file with any features that may have been overwritten.

- Discard your changes

Select this option to use the original file generated by Web Builder; use the file saved in the backups folder to copy your modifications into the original file.

- Apply to all modified files

To avoid multiple prompts for keeping or discarding changes as the build proceeds, select this option.

## TO REBUILD A PROJECT FROM THE COMMAND LINE

You can rebuild a project without opening Web Builder by using the command line option. Before proceeding, verify that JAVA\_HOME is set to a Java 7 or higher JDK. You can point JAVA\_HOME to the JDK that is installed with Host Integrator which is located here `<install-location>\java\jdk1.8.x.`

Open a command line prompt in the project directory of the Web application you want to rebuild. For example:

```
C:\Users\<user>\Documents\Micro Focus\Verastream\HostIntegrator\projects\<project>
```

where `<user>` is the name of the person who generated the application and `<project>` is the name of the project you want to rebuild.

At the command prompt, type:

```
<install-location>\HostIntegrator\ant\bin\ant.bat -lib
"<install-location>\HostIntegrator\lib\webbuilder\coreplugins;<install-location>\HostIntegrator\
lib\webbuilder" -Dparent.dir=. -Dautorun=false
```

where `<install-location>` is the directory where the product is installed.

## Creating new project types

Instead of using the project types available from Web Builder, you can create your own project type that will be available to use with subsequent new projects. When you create a new project type, you can customize the way Web Builder generates a project.

1. Open Web Builder, click Options, and then User-defined Project Types.
2. On the User-defined Project Types dialog box, click New.
3. From the Copy this project type list, select the project type that will be the template for your new project type.
4. Enter the name and description for the project type.
5. Click Build.

To modify the project type:

After creating a new project type, you can modify the files that are used to build the project. The resources folder contains files that you can modify for each project.

1. In the User-Defined Project Types dialog box, select the project type you want to modify, click Explore. The <Documents>\Micro Focus\Verastream\HostIntegrator\userpluginprojects folder is now accessible.
2. Make changes to the files in the subfolders in the Resources folder:
  - UI – To change the icons
  - tasks – To change xsl
  - project – To recreate build.xml, an ANT build file
3. In the User-Defined Project Types dialog box, select your new project type and click Rebuild to incorporate the changes into the new project type.

### **Keyboard mapping for Web applications**

You can map your keyboard so specific workstation keyboard keys will function as terminal keys for a host application. Keyboard mapping is available for Java or .NET Web applications.

You can implement keyboard mapping after a project is built, or incorporate it into a User-Defined Project Type, which can be used to build subsequent Web applications using the same keyboard map

## 5.1.3 Generating a Web Application

---

Web Builder uses model information to generate Web applications. You determine the look and feel of the Web application by the model and Web Builder settings you choose. HTML 5 Web applications are platform and technology independent.

The complexity and functionality of the Web application depends on the host application model that you create with the Design Tool. Using a model that contains only connection information, has no defined entities, and uses the default settings, Web Builder generates a Web application matching the terminal screen layout. However, if you use a model with defined entities, attributes, operations, recordsets, and procedures, you have more control over the Web application that is generated.

### How does it work?

A Web page is generated for each included entity and procedure in the model. The pages are named `entity_<ENTITYNAME>` and `procedure_<PROCEDURENAME>`. Each procedure is exposed through a link in the generated Web application. When you click the link the application either displays a form to input the procedure parameters and an Execute button, or if there are no more parameters, a link that automatically executes the procedure.

When you include procedures in a model, Web Builder can generate a Web application that exposes the host application functionality, rather than just the host screens. Procedures encapsulate host data and logic so that it can be accessed as database tables. A set of files is generated that provides a Web-based front-end to the procedures in the table of your host application model. Programmers can then retrieve information from the host application using a familiar database interface.

### Running a Web Application

To run a selected Web application:

Click Run on the bottom of the Web Builder dialog box

-or-

Open a Web browser and enter the URL for the Web application. The URL takes this format:

`http://<serverName>/<ProjectName>`.

### Hosting an HTML 5 Web Application on a Non-Windows Verastream Installation

Verastream Web applications are generated on Windows platforms, however you can copy and run them from any supported Verastream installation platform. Web Builder automatically deploys Web applications to your local Windows Host Integrator Web server, which makes it easy to test projects under development.

## Note

During deployment the Microsoft User Account Control (UAC) utility will prompt you to continue deployment using elevated privileges. Click Continue to dismiss the dialog box and continue deploying the Web application.

## ABOUT THE WAR FILE

The Web application , in the form of a WAR file, is copied into the Web server folder:

```
<VHI install folder>\servletengine\webapps
```

A Web archive (.war) file is a convenient way to deploy the generated Web application to any JSP 1.2 compliant servlet engine or application server. A .war file is generated for Web applications in the <Documents>\Micro Focus\Verastream\HostIntegrator\projects\<ProjectName>\webapp folder.

## MOVING THE VERASTREAM HTML 5 WEB APPLICATION TO ANOTHER SERVER

To move a Web application to either a Windows or non-Windows test or production environment:

1. Confirm that the deployed application has access to the Host Integrator model:

If you created the application using a model in your production environment, the Web application's access information is correct.

- If you created the application using a model outside your production environment or from an exported XML file, make the following changes:

If you are connecting with the model via the session server: modify the name of the session server and, if necessary, the model name.

If you are connecting with the model via a domain: modify the name of the management server and, if necessary, the domain name.

3. To make these changes open the Web Builder Project Properties dialog box and modify the connection information.

2. To deploy the Web application to the VHI Web Server on a different machine (either Linux or Windows), manually copy the .WAR file to <VHI install folder>/servletengine/webapps . To deploy the Web application to the VHI Web Server on Windows platforms, manually copy the .WAR file to <VHI install folder>\servletengine\webapps .

## Note

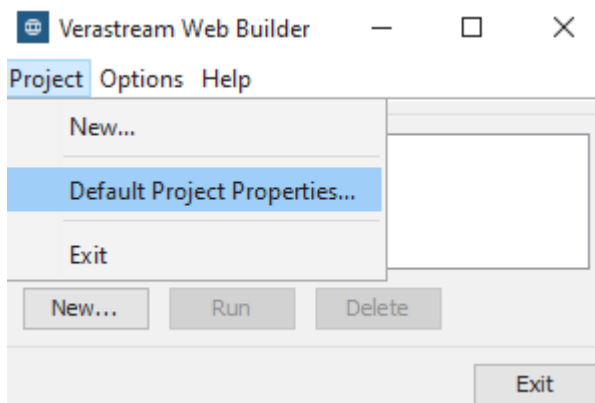
If the web application has already been deployed, you can make these changes manually by editing the <VHI install folder>\servletengine\webapps\<ProjectName>\WEB-INF\classes\<ProjectName>\<ProjectName>Session.properties file.

## 5.1.4 Configuring Project Properties

You can customize how Web Builder generates a project by setting project properties. Set default properties for each type of project, or set properties for specific, individual projects.

### Default project properties

Web Builder generates a project based on the default settings for that project type. You can modify the default project properties using the Default Project Properties dialog box. Any new project you generate uses the new settings.



### Project Properties for Specific Projects

To configure properties for a specific project prior to building that project, click Properties on the New Project dialog box and enter information into the Project Properties dialog box.

To configure properties for a specific project after the project has been created, right-click the project and select Properties, or click Properties on the Web Builder menu. The project will automatically rebuild to incorporate the property changes.

### Common Property Settings

These options affect how Web Builder generates all projects:

Property	Description
Launch browser when build succeeds	Select this option to automatically open projects in a browser after the build successfully generates them.
Logging	Select Use verbose logging to expand the information in the log file. This option provides information that can be helpful in debugging problems with the build process.



Property	Description
Store projects in this directory	You can specify the directory in which Web Builder finds and builds projects. The default is <code>&lt;Documents&gt;\Micro Focus\Verastream\HostIntegrator\projects</code> . Click Browse to specify another directory. The directory you specify applies to all projects you open and save in the Web Builder. Any changes to the default folder will not affect the location of previous Web Builder projects.

Property	Description
Defaults button	To return the common settings to default values, click the Defaults button at the bottom of the Web Builder Settings dialog box.

## Microsoft Tools Settings

Set the following options for Microsoft tools and utilities:

1. From the Options menu, click Web Builder Settings. Select Microsoft Tools in the right pane. The fields are initially populated with the default Microsoft install locations.

2. To adjust the settings, modify the following:

.NET Framework directory

1. Enter the location of the .NET Framework directory (C:\Windows\Microsoft.NET\Framework64\v4.0.30319 by default), or click Browse to find the directory. Web Builder uses this information to compile and open .NET projects in Visual Studio .NET.

1. Web Builder supports new .NET projects. Previously created .NET 2.0 projects can still be modified. Use this setting to change to the appropriate location for your project if it is not already selected as the default.

1. The first time it runs, Web Builder searches for the latest version of .NET Framework. If you install a new or more recent Framework, you may have to change this setting.

Root directory for IIS

1. Enter the location of the Internet Information Services (IIS) root directory, or click Browse to find the directory. Web Builder creates project files that need to be copied into the IIS folder structure. The default is C:\inetpub.

3. You can click Default at anytime to return to the original Microsoft default install paths.

4. Click OK.

For more details on configuring Web Builder project settings, see:

[HTML 5 Web application model connection properties](#)

[Using Objects](#)

## 5.1.5 About HTML 5 Applications

---

Host Integrator HTML 5 Web applications are platform and technology independent. They are easily generated and deployed using Web Builder. The complexity and functionality of the Web application depends on the host application model that you create with the Design Tool.

Verastream Web applications are generated on Windows platforms, however you can copy and run them from any supported Verastream installation platform. Web Builder automatically deploys Web applications to your local Windows Host Integrator Web server, which makes it easy to test projects under development. See [Hosting an HTML 5 Web Application on a Non-Windows Verastream Installation](#) for information on deploying to non-Windows platforms.

### Web application features

The web application you generate and deploy has a number of features which aid in its use.

- **Cursor-based navigation**

The Verastream HTML5 Web applications support cursor-based navigation on 3270 and 5250 terminal screens. You can set the cursor position using arrow keys, mouse clicks, or by tapping the screen.

- **Hotspot and Light Pen Support**

Verastream Host Integrator HTML 5 Web applications support two types of hotspots; aid key support, enabled by default, and light pen support. To use hotspots, select **Enable Aid Keys** on the Screen properties panel after you set the Display type to **Terminal**.

- **Zero footprint terminal sessions for 3270 and 5250 hosts**

You can install VHI and have immediate access to your host. From the Start menu, choose either 3270 Terminal Session or 5250 Terminal Session, depending on your host type. You can also enter a known address such as `http://server:port/HostApp` into your browser. The terminal session Web application displays an host connection dialog where you can enter the appropriate connection information, complete the host connection, and interact with your green screen host application. If you pre-configure these options in Web Builder, in the terminal model property pages, then the host connection page is no longer required and won't display. This feature gives you access to your host applications without having to open the Design Tool or Web Builder and without having to create and deploy a model.

- **Keyboard mapping**

You can map your keyboard so specific workstation keyboard keys will function as terminal keys for a host application. Keyboard mapping is available for Web applications.

### More information

[HTML 5 Web application properties](#)

## 5.1.6 HTML 5 Web Application Properties

---

Use these properties to configure how you want the generated Web application to handle model connections, procedures, as well as screen and project properties. You can also customize HTML 5 web applications using project properties.

You can set these properties for HTML 5 web applications:

[Procedure properties](#)

[Model connection properties](#)

[Screen properties](#)

[HTML 5 Project Properties](#)

[Customizing HTML 5 web applications](#)

### Procedure properties

You can create a custom output format for procedures.

By default all procedures are included in the Web application. You can exclude all procedures from the Web application, by clearing the Procedures checkbox. Selecting a table automatically selects all the procedures associated with that table. However, you must select procedures individually if you want to change their "Display" settings.

#### Default procedure settings

Display

- Form - Select this option to have the procedure output display as an HTML form. Specify the display format for the output of the procedure. Choose either to:

Display outputs as table. A single table is generated containing columns for each of the procedure output parameters and rows for the values.

Display output as list. A separate table containing the procedure output parameters is generated for each value returned.

- Screen - This option displays the screen where the the procedure ended as opposed to the procedure output. This is useful for procedures that are solely created to navigate to a location in the host application. The display for the specified screen is configured on the Screens tab.
- Custom (legacy Web applications only) - Using this option you can take complete control of how the procedure output is handled in the Web application.

Specify the URL of the page you want displayed when this entity is encountered. A relative URL is assumed to be relative to the root of the Web application and the page must be included with the Web application. You can do this manually or by using the Custom Content options.

If you want to return to a single format after having made changes to an individual procedure, click Reset all.

### Model Features

These options are meant for testing purposes only and should not be enabled in production environments.

- Execute SQL statement

Select this option to have the Web application provide a user option to perform a query by typing in an SQL statement. To use this feature, the procedure must be configured as available for SQL queries.

- ProcessString event handler

Select this option to for users to provide a string for a Process String event handler within the Web application. This option requires that a ProcessString event handler is included in the model.

## Model connection properties

When you select Model Connection Properties in the Project Properties dialog box, you can configure the settings below. These properties will vary depending on the project type you have selected.

### Connect method

Select the appropriate connect method for your project:

Connect to model via server

Connect to model via domain

Connect to session pool via server

Connect to session pool via domain

### **Host Integrator model or session pool name**

Select the model or session pool that is the basis of this Web Builder project.

Depending on the Connect method you select, these options are available:

- Server - When connecting through the server, enter the name of the Host Integrator Server where the model is installed.
- Management Server - When connecting via domain, enter the name of the server that is handling domain requests.
- Domain - When connecting via domain, enter the name of the domain.
- Obtain model data from Design Tool exported file - When creating a new project, you can use this option to select an exported XML model file. Enter the name of the file Web Builder is to use to generate the project, or click Browse to locate the XML model file.
- Host settings - Select the host name and port number this model is using.

## **Screen properties**

Use the Screens tab to configure how you want to handle unrecognized screens and to specify how to display recognized screens.

### **Unrecognized screens**

If a model does not have defined entities for host application screens, specify how Host Integrator should handle these unrecognized screens.

### **Recognized screens (entities)**

You can set up the way your web applications handle recognized screens (entities) on a global basis, or you can select individual entities and specify a different display. You can use Reset All to return all recognized entities to the default values.


- Form

Web Builder generates a web application that handles the data fields differently than it handles the screen display. This is the default. The dynamic data on an entity (the defined attributes, recordsets, or operations) comes from the host application, and is updated as needed.

The appearance of the user interface remains static and is generated into a web application that can be modified by a web designer

### Common Terminal settings for both unrecognized and recognized screens (entities)

- Enable Aid Keys

Select this option to provide aid key functionality in your Web application. You can invoke Aid keys using either the F keys on your physical keyboard or by using the Aid key palette in your Web application. To display the Aid key palette, click  on the toolbar. Any host screen that is configured to display as a terminal has aid keys enabled by default. If you disable this option Enable Hotspots and Enable Light Pen Hotspots will also be disabled.

- Enable Hotspots

Hotspots are buttons that appear over common host commands in terminal sessions. Any host screen that is configured to display as a terminal has hotspots enabled by default. Selecting Enable Aid Keys is a prerequisite for using hotspots. See About Hotspots(/wb-hotspots/) for more information on this option.

- Enable Light Pen Hotspots

You can enable light pen support for all screens or individual screens configured to display as a terminal. By default, light pen support is not enabled. If you have HTML 5 web applications that were generated using past versions of Host Integrator, you must rebuild your project to enable light pen support. Selecting Enable Aid Keys is a prerequisite for using light pen hotspots. See About Hotspots for more information on this option.

### Additional settings

- Hide - Select this option to hide screens. Type in the text that is displayed with this terminal screen.
- Error - Rather than display a screen, display an error message. You can configure the following:

Message to display - Type in the text that is displayed with this terminal screen.

Disconnect from the host - Select this option if you want the web application to disconnect from the host when the unrecognized screen is encountered.

## HTML 5 project properties

You can generate an HTML 5 Web application, using project properties to determine what type of information to display. You can set default properties for each type of project, or set properties for specific individual projects.

<b>Property</b>	<b>Description</b>
Include source code in archive (.jar) file	Web Builder includes the source files with the compiled classes in the .jar file. By default, the source files are not included.
Generate javadoc documentation	Web Builder generates JavaDoc documentation of the methods in your generated code. By default, these files are not included. The JavaDoc documentation is generated using the descriptions from the Procedure Editor; therefore, make sure the procedure descriptions are as complete as possible. See the Tables Overview for more information.
Java package name	Create unique package names. This option is only available for individual projects.
Method timeout	How long to wait for a method to complete before timing out. The default is 30 seconds
Locale	The locale (or language) that the generated Java Beans use to communicate with the Host Integrator Server. Change this setting only if you provide locale-specific resources or files for the Host Integrator Server. The default uses the current operating system locale.
Require username and password	Specifies whether generated Web applications use Host Integrator security to connect to the host. If you select this option, users are prompted for a valid user ID and password when a project attempts to connect to the host. Selecting this option does not override the security requirements for the application. An application that requires no security credentials will allow a user to continue without providing password information. Conversely, an application requiring security credentials will include a password prompt even if this option is not selected. Use this option, for example, when you generate a Web application using a server that does not have security enabled, but will be exporting that generated application to a system that does have security enabled.



<b>Property</b>	<b>Description</b>
Require encrypted connection to the VHI server	This option uses a secure connection between the Web server and the Verastream Host Integrator session server. This option forces a secure connection even if the server setting is not set to require a secure channel. It is selected by default. This setting does not affect the client (browser) connection to the Web server, which is handled using the correct HTTPS port. Host Integrator session server HTTPS/HTTP connection settings for Web services are configured in the Administrative Console. See the online help topic, Setting Web Services Session Server Properties.

## Customizing HTML 5 web applications

Use the Customization tab on the Project Properties dialog box to import custom content for your generated Web application. You can specify a directory location containing customized content which you can use for layouts and other display elements. For example, you can import custom pages, images, and additional support libraries.

### Custom Content

Use this feature to import custom content and resources into your Web application. For example, you can import custom pages, images, and additional support libraries.

### How does importing custom content work?

When a directory is specified, the contents of the specified directory is overlaid on top of the generated Web application before it is deployed. Files in the directory itself end up in the base of the Web application directory. Since all subdirectories are copied as well, a directory structure matching that of the Web application can be created, and you can import files into any area of the generated Web application. Existing files are overwritten with files from the custom content directory. The Web application automatically provides the same parameters to the custom page as it does to the generated page.

You can modify the generated Web application and utilize elements provided in the custom content to expand the Web application's functionality. When you rebuild the project, modifications are preserved and the custom content is included in the generated application.

### To import custom content

Create a directory in which to place your custom content. This is the source from which you will copy your custom content.

In Web Builder, click Options >Customization >Custom Content. Specify the directory that you just created.

Place a file, a collection of files, or an entire directory structure inside the source directory you created in step 1.

Rebuild your project.

When the project rebuilds, the contents of the source directory is added to and deployed with your Web application. Each time you rebuild, the overlay directory is inserted into the built Web application.

## 5.1.7 Using Legacy Web Applications

Web Builder has two legacy Web applications that are available on the Legacy tab of the New Project dialog box. They are:

- .NET 2.0 Web Application

- Java Web Application

Each of these options has it's own set of properties that you can configure.

[.NET](#)

[Java](#)

### Using HTML 5 Web Applications

In the 7.5 release of Verastream Host Integrator HTML 5 Web applications are available. These are platform and technology independent. To generate HTML 5 Web applications, click the Application tab, under Project Types, on the the New Project dialog box.

[.NET](#)

When you are working with a .NET Web application project type, the Project Properties contains all the settings and configurations you need to access the Host Integrator model.

Property	Description
Method timeout	Specifies how long to wait for a method to complete before timing out. The default is 30 seconds.

<b>Property</b>	<b>Description</b>
Locale	The locale (or language) that the generated files use to communicate with the Host Integrator Server. Change this setting only if you provide locale-specific resources or files for the Host Integrator Server. The default uses the current operating system locale.
Require username and password*	Specifies whether generated Web applications use Host Integrator security to connect to the host. If you select this option, users are prompted for a valid user ID and password when a project attempts to connect to the host.

Property	Description
Require encrypted connection to VHI server	This option uses a secure connection between the Web server and the Verastream Host Integrator server. This option forces a secure connection even if the server setting is not set to require a secure channel. It is selected by default. This setting does not affect the client (browser) connection to the Web server, which is handled using the correct HTTPS port. Host Integrator session server HTTPS/HTTP connection settings for Web services are configured in the Administrative Console. See the online help topic, Setting Web Services Session Server Properties.

### Note

Selecting this option does not override the security requirements for the application. An application that requires no security credentials will allow a user to continue without providing password information. Conversely, an application requiring security credentials will include a password prompt even if this option is not selected. Use this option, for example, when you generate a web application using a server that does not have security enabled, but will be exporting that generated application to a system that does have security enabled.

## ADDITIONAL REQUIREMENTS

To generate .NET Web applications on computers running Microsoft IIS 7.0, select the IIS 6 Management Compatibility option when installing IIS. For Vista, install IIS 7. under Programs and Features. For Windows Server 2008, use Manage Server to Add a Role to install IIS 7.

To use .NET Web applications in Web Builder, APS .NET must be installed. In older versions of Windows, APS .NET is installed by default when you install Internet Information Services (IIS). This is not the case when running Windows 2008 and Windows 7. To use .NET Web applications on these platforms:

From the Control Panel, open Programs | Turn Windows Features On or Off.

Expand Internet Information Services, then World Wide Web Services, and finally Application Development Features.

Enable ASP.NET, and click OK. This also enables some dependant features. The features are installed.

When the installation is complete, re-deploy your .NET Web service running Web Builder as an administrator.

## Java

When you are working with a Java Web application project type, the Project Properties contains all the settings and configurations you need to access the Host Integrator model.

After you select the Java Web Application project type from the New Project dialog, click Properties to access the Project Properties dialog box.

<b>Property</b>	<b>Description</b>
Include source code in archive (.jar) file	Web Builder includes the source files with the compiled classes in the .jar file. By default, the source files are not included.
Generate javadoc documentation	Web Builder generates JavaDoc documentation of the methods in your generated code. By default, these files are not included. The JavaDoc documentation is generated using the descriptions from the Procedure Editor; therefore, make sure the procedure descriptions are as complete as possible. See the Tables Overview for more information.
Java package name	Create unique package names. This option is only available for individual projects.
Method timeout	How long to wait for a method to complete before timing out. The default is 30 seconds
Locale	The locale (or language) that the generated Java Beans use to communicate with the Host Integrator Server. Change this setting only if you provide locale-specific resources or files for the Host Integrator Server. The default uses the current operating system locale.
Require username and password	Specifies whether generated Web applications use Host Integrator security to connect to the host. If you select this option, users are prompted for a valid user ID and password when a project attempts to connect to the host. Selecting this option does not override the security requirements for the application. An application that requires no security credentials will allow a user to continue without providing password information. Conversely, an application requiring security credentials will include a password prompt even if this option is not selected. Use this option, for example, when you generate a Web application using a server that does not have security enabled, but will be exporting that generated application to a system that does have security enabled.

<b>Property</b>	<b>Description</b>
Require encrypted connection to the VHI server	This option uses a secure connection between the Web server and the Verastream Host Integrator session server. This option forces a secure connection even if the server setting is not set to require a secure channel. It is selected by default. This setting does not affect the client (browser) connection to the Web server, which is handled using the correct HTTPS port. Host Integrator session server HTTPS/HTTP connection settings for Web services are configured in the Administrative Console. See the online help topic, <a href="#">Setting Web Services Session Server Properties</a> .

### **ADDITIONAL CONFIGURATION**

The legacy Java and .NET screen, procedure properties and model connection properties are documented in the [HTML 5 Web Application Properties](#) topic.

## 5.1.8 Working with Web Projects

---

Web Builder projects are meant to be incorporated into Web or client/server applications. Web Builder provides standardized interfaces of incorporating host data into your applications. The general process is to:

Use Web Builder to create the project, making modifications and customizations with Web Builder.

Deliver the project to developers who will enhance the Web application or use the Java or .NET objects to access host data.

See [Setting HTML 5 Properties](#) for information on deploying these applications.

### Deploying a Java legacy Web application

Because Web Builder automatically deploys Web applications to your local Host Integrator Web Server, you can test projects that are under development on your local machine, rather than deploying them to production servers. Use the Project menu to deploy and undeploy a project on your local server.

The Web application, in the form of a WAR file, is copied into the Web server folder:

```
<VHI install folder>\servletengine\webapps.
```

To deploy the Web application to the VHI Web Server on a different machine (either Linux or Windows), manually copy the .WAR file to `<VHI install folder>/servletengine/webapps`. To deploy the Web application to the VHI Web Server on Windows platforms, manually copy the .WAR file to `<VHI install folder>\servletengine\webapps`.

During the deployment process the Microsoft User Account Control (UAC) utility prompts you to continue deployment using elevated privileges. Click Continue to dismiss the dialog box and continue deploying the Java Web application.

#### About the WAR File

A Web archive (.war) file is a convenient way to deploy the generated Web application to any JSP 1.2 compliant servlet engine or application server. A .war file is generated for Web applications in the `<documents>\Micro Focus\Verastream\HostIntegrator\projects\<ProjectName>\webapp` folder.

#### Moving the Java Web Application Files to Another Server

**To move a Web application to another test or production environment:**

Confirm that the deployed application has access to the Host Integrator model:

If you created the application using a model in your production environment, the Web application's access information is correct.

- If you created the application using a model outside your production environment or from an exported XML file, you must correct the appropriate Management, server, and domain statements. You can do this by either:
- Opening the Web Builder Project Properties dialog box and modifying the connection information. -or-

Editing the bean.properties file in the `<VHI install folder>\projects\<ProjectName>\javabeans\src\<ProjectName>` folder, and rebuilding the project.

Copy the `<ProjectName>.war` file, following the steps specific to the servlet runner.

## Deploying a .NET legacy Web application

Because Web Builder automatically deploys .NET Web applications to your local IIS Server, you can test projects that are under development on your local machine, rather than deploying them to production servers.

During the deployment process the Microsoft User Account Control (UAC) utility prompts you to continue deployment using elevated privileges. Click Continue to dismiss the dialog box and continue deploying the .NET Web application.

A `<ProjectName>.sln` file is generated in the `<documents>\Micro Focus\Verastream\HostIntegrator\projects\<ProjectName>\solution` directory. You can use this file to edit the .NET Web application. Other generated files are copied to `Inetpub\wwwroot\<ProjectName>`.

### To move the .NET Web Application Files to another server

Confirm that the deployed application has access to the Host Integrator model:



- If you created the application using a model in your production environment, the Web application's access information is correct.
- If you created the application using a model outside your production environment or from an exported XML file, you must correct the appropriate Management server, server, and domain statements. You can do this by either:
  - Opening the Web Builder Project Properties dialog box and modifying the connection information. -or-
  - Editing the `<ProjectName>` Session object, and rebuilding the project.
- Verify that the .NET connector (AppConn connector interfaces) is installed on the production server.
- Copy the `<ProjectName>` folder from `Inetpub\wwwroot` to the same location on the production Web server.
- Configure it as an application using Internet Services Manager.

## Using Objects

Providing an object to your host applications enables rapid reuse of the business logic that exists on these hosts. The .NET class library, or Java Beans interface becomes a reusable object for creating portals, Web applications, and other business solutions.

Programmers are able to use their language and development tool of choice because you can use a single Host Integrator model to generate these component interfaces:

### .NET Class Library Setup

The Verastream .NET Class Library is a library of classes and interfaces that comply with the Microsoft .NET Framework SDK. You can add this library as a reference in Visual Studio .NET to access host data and functionality.

Prior to using the Verastream .NET Class Library for a production application:

1. If necessary, rebuild the .NET Class Library project to access the model on the production server:

In Web Builder, open the Project Properties dialog box for the project.

In the Project Properties dialog box, enter the correct server name.

Click Build to regenerate the .NET Class Library using the production server name.

2. Add the project as a reference in Visual Studio .NET. The `<ProjectName>.dll` is located in the `<documents>\Micro Focus\Verastream\HostIntegrator\projects\<ProjectName>\classlibrary\bin` folder.

### .NET Class Library Code Example

This example illustrates how to use a Web Builder generated .NET Class Library. The example uses a project named CICSAccts\_Class\_Library, which generated a .NET Class Library for the CICSAccts model. The GetAccount procedure takes an account number as an input parameter and returns account information.

```
using System.Data;
using CICSAccts_Class_Library;
// Create an object to represent the session
CICSAccts_Class_LibrarySession session = new CICSAccts_Class_LibrarySession();
// Create an object to hold the filter values
GetAccountFilters filters = new GetAccountFilters();
// Initialize the filters object with the values you want to pass to the host filters.AcctNum = 31415;
// Call the procedure on the session, passing in the filters
DataSet ds = session.GetAccount(filters, null, 0);
// Access the data set and process through them
for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
{
 Console.WriteLine(ds.Tables[0].Rows[i]["AcctNum"]);
 Console.WriteLine(ds.Tables[0].Rows[i]["LastName"]);
 //...
}
```

## Java Beans Setup

Java Bean components, or Beans, are reusable software components that can be manipulated visually in a Java application development tool. Beans can be combined to create traditional applications and incorporated into Web applications as Java servlets.

When you generate, the following files are copied to the `<documents>\Micro Focus\Verastream\HostIntegrator\projects\<ProjectName>\javabeans\src\<ProjectName>` directory, for each model and its associated procedures:

```
<ProjectName>Session.java
<ProcedureName>Filters.java
<ProcedureName>Inputs.java
<ProcedureName>Record.java
<ProcedureName>RecordSet.java
```

The corresponding class files are located in the bld directory.

Prior to using Verastream Java Beans for a production application:

1. If necessary, rebuild the Java Beans project to access the model on the production server:

In Web Builder, open the Project Properties dialog box for the Java Beans project.

In the Project Properties dialog box, enter the correct server name.

Click Build to regenerate the Java Beans using the production server name.

2. Copy the `<documents>\Micro`

`Focus\Verastream\HostIntegrator\projects\<projectname>\javabeans` folder to the web server.

3. Change the server's CLASSPATH to include these lines:

```
<VHI_install_folder>\lib\java\wcp.jar
<VHI_install_folder>\lib\java\vhprop.jar
<VHI_install_folder>\lib\java\aptrieve.jar
`<documents>\Micro Focus\Verastream\HostIntegrator;
\projects\<projectname>\javabeans\<projectname>.jar
```

### Java Beans Code Example

- 3.

This example illustrates how to use Web Builder generated Java Beans. The example uses a project named CICSAccts\_beans, which generated Java Beans for the CICSAccts model. The GetAccount procedure takes an account number as an input parameter and returns account information.

```
// Create an object to represent the session
CICSAccts_beans session = new CICSAccts_beansSession();

// Create a bean to hold the filter values
GetAccountFilters filters = new GetAccountFilters();

// Initialize the filters bean with the values you want to pass to the host
filters.setAcctNumber(new Integer (20000));

// Call the procedure on the session, passing in the filters
GetAccountRecordSet recordSet = session.getAccount(filters);

// Access the array of records and process through them

GetAccountRecord[] records = recordSet.getGetAccountRecords();
for (int i = 0; i < records.length; i++)
{
System.out.println(records[i].getAcctNum());
System.out.println(records[i].getLastName());
//...
}
```

## 5.1.9 Using Objects

You can determine how Web Builder generates a project by setting project properties. You can set default properties for every type of project, or set properties for specific, individual projects.

### Default Project Properties

Web Builder generates a project based on the default settings for that project type.

To modify default project properties:

From the Project menu, click Default Project Properties.

In the left pane, select the project type you want to configure.

Set the default project properties in the right pane. These options vary depending on the project type you select. Click Help on each property page for information specific to that property.

Each new project you generate will use the new settings.

### Project Properties for Specific Projects

You can configure properties for a specific project before you build it.

From the Project menu, click New.

In the left pane of the New Project dialog box, select the project type you want to configure

Click Properties in the right pane. Set the project properties. These options vary depending on the project type you select. Click Help on each property page for information specific to that property.

After you have created the project, to configure the project's specific settings:

Right-click the project name.

Select Properties, or alternatively, select Properties from the Web Builder Project menu.

The project automatically rebuilds to incorporate the property changes.

### .NET Class Library Properties

Property	Description
Method timeout	Specifies how long to wait for a method to complete before timing out. The default is 30 seconds.
Locale	The locale (or language) that the generated files use to communicate with the Host Integrator server. Change this setting only if you provide locale-specific resources or files for the Host Integrator server. The default uses the current operating system locale.

Property	Description
Require encrypted connection to the VHI server	Select this option to use a secure connection between the Verastream Host Integrator connector and the Verastream Host Integrator session server. This option forces a secure connection even if the server setting is not set to require a secure channel. This setting does not affect the client (browser) connection to the Web server, which is handled using the correct HTTPS port. Host Integrator session server HTTPS/HTTP connection settings for Web services are configured in the Administrative Console.
Procedures automatically connect to Session Server	Automatically connects and disconnects from the Host Integrator session server when a model procedure is accessed. If there is a lot of connection overhead, or you are making several procedure calls one after the other, you may want to clear this check box and handle the connection in your code.

## Java Beans Properties

Property	Description
Include source code in archive (.jar) file	Web Builder includes the source files with the compiled classes in the .jar file. By default, the source files are not included.
Generate javadoc documentation	Web Builder generates Javadoc documentation of the methods in your generated code. By default, these files are not included. The Javadoc documentation is generated using the descriptions from the Procedure Editor; therefore, make sure the procedure descriptions are as complete as possible. See the Tables Overview for more information.
Java package name	Create unique package names. This option is only available for individual projects.
Method timeout	How long to wait for a method to complete before timing out. The default is 30 seconds
Locale	The locale (or language) that the generated Java Beans use to communicate with the Host Integrator Server. Change this setting only if you provide locale-specific resources or files for the Host Integrator Server. The default uses the current operating system locale.

<b>Property</b>	<b>Description</b>
Procedures automatically connect to Session Server	Automatically connects and disconnects from the Host Integrator Session Server when a model procedure is accessed. If there is a lot of connection overhead, or you are making several procedure calls one after the other, you may want to clear this check box and handle the connection in your code.
Require encrypted connection to the VHI server	This option uses a secure connection between the Web server and the Verastream Host Integrator session server. This option forces a secure connection even if the server setting is not set to require a secure channel. It is selected by default. This setting does not affect the client (browser) connection to the Web server, which is handled using the correct HTTPS port. Host Integrator session server HTTPS/HTTP connection settings for Web services are configured in the Administrative Console.

Property	Description
Execute post-build step	Select this option if you want Web Builder to execute an additional step after the building process is complete. Identify which command or batch file you want to run; it is invoked with two arguments, the project name and the path to the newly-built .jar file. See the example below.

Execute post-build step example:

The post-build script, which you would point to in Web Builder, copies the jar file to a local Maven environment:

```
SET projectuiname=%1
SET jarpath=%2
mvn install:install-file -Dfile=%jarpath% -DgroupId=org.example -DartifactId=%projectuiname% -Dversion=master-SNAPSHOT -Dpackaging=jar
```

## 5.1.10 Troubleshooting Web Builder

---

This section provides tips for troubleshooting problems you may encounter with Web Builder.

- [Host cannot connect when you attempt to run a browser-based Web application](#)
- [Partial screens are received from the host](#)
- [Changing the default Host Integrator Web Server port number](#)
- [Data missing in the Web Builder log file](#)
- [Interpreting HTML Markup Characters in Web applications](#)
- [Rebuilding Web Application Fails When Renaming or Deleting Procedures](#)

### Host cannot connect when you attempt to run a browser-based Web application

Host Integrator Servers in the Development Kit can run only five concurrent host sessions. If you don't use the Disconnect button to close a Web application connected to a host session and just close the browser, the session will persist its connection to the host from the Host Integrator Server and you can run out of available sessions. To close a session that is persisting its connection to the host, follow these steps:

1. From the Administrative Console, go to the Session Server Explorer and in the list of session servers, select the session server that the Web application used for its host session.
2. From the toolbar, click Remove or Shut Down the Session Server.
3. Repeat step 2 for each session you want to close.

### Partial screens are received from the host

If the Web application displays incomplete screens, use these settings that define how long Web Builder waits before displaying a new host screen:

- **Timeout**—If the host is slow to respond, increase the timeout setting.
- **Duration**—If the host sends the screen in chunks and pauses between each transmission, increase the duration setting.

### Changing the default Host Integrator Web Server port number

By default, the Web Builder assumes that the Host Integrator Web Server is listening on port 8081. This port is known to be used by Network Associates' McAfee software (ePolicy Orchestrator "Agent Wake-Up Call" service). You must change either VHI or the McAfee product's configuration to avoid a port conflict.



To change this value, follow these steps:

1. Open the common.properties file in the %LOCALAPPDATA%\Attachmate\Verastream\HostIntegrator\webbuilder\common folder, and change this line to match the port number used by your Host Integrator Web Server:
  1. `servletcontainerport=8081`.
2. Open the server configuration file, `container.properties`, found in the <VHI install folder>\servletengine\conf folder, and change these lines to the port you want to use:
  2. `servletengine.port=8081`
  2. `servletengine.ssl.port=8443`
3. Restart the VHI Web Server and Web Builder.

## Data missing in the Web Builder log file

A log file is generated whenever Web Builder generates a project. When building Java projects, the Java compiler sends information to the log file on the first build, but not on any subsequent builds. Therefore, to generate output from the Java compiler, close and restart Web Builder, then rebuild the Java project.

## Interpreting HTML Markup Characters in Web applications

Displaying data containing HTML markup characters may have varying results depending on the type of Web application you build and the browser used to display the application. Generally, Web applications convert HTML markup characters correctly, but terminal screen layout Web applications display the characters directly to the screen.

Here are two examples where HTML markup character issues may occur when building terminal screen layout Web applications:

- Event handlers that embed HTML in returned data. For example, a script displays all invalid data as red text by using `<font color=red>`.
- Host data containing characters that could be interpreted as an HTML markup character. For example, the host data "commissions `<40" could be truncated to "commissions" because the browser tries to render `<` as an HTML character.

## Rebuilding Web Application Fails When Renaming or Deleting Procedures

When you rebuild a .NET or Java Web application, you may encounter an error if you have renamed or deleted procedures in the model. To correct this:

1. Delete the previously generated source files that reference the old procedure name.

In Java Web applications the source files are located in `<VHI install dir>\projects\<<projectname>\webapp`.

In .NET Web applications the source files are located in `<Inetpub>\<project name>`.

2. Rebuild the Web application.

## 5.1.11 Modeling Tips for Building Web Applications

---

When building a Host Integrator model, use these tips for enhancing your Web Builder projects.

### Button and Link Names

The names you assign while you are creating a model determine what appears in the resulting Web application:

Operation names are displayed on buttons.

Procedure names appear as navigation links.

Descriptions appear as tool tips.

Underscores and mid-word capitalizations are replaced by spaces. For example, *your\_procedure* is replaced with *your procedure*, and *MyProcedure* is replaced with *My Procedure*.

### Attributes

In the Web application, attributes are labeled using names defined in the model. To create a more detailed description of an attribute, open the Advanced Attribute Properties dialog box from the Design Tool's Attribute tab. Supply a description to be used as a tool tip for the attribute.

Read-only attributes appear in the Web application as text that cannot be edited; writable attributes appear as HTML input fields that permit the user to type in a value. Writable attributes that need encryption, such as passwords, use an HTML password input field to ensure that the user-entered value is not echoed to the screen.

### DISPLAYING OPERATIONS

In a Web application, operations are displayed as buttons. The operation names defined in the model appear as text on the buttons. Descriptions appear as tool tips.

When a user selects a recordset scrolling operation, the recordset will scroll through its records appropriately. When the end user selects an operation not associated with a recordset, the Web application sends the value of any attribute that the end user has changed to the Host Integrator Server. The Web application communicates to the Host Integrator Server to execute the entity operation corresponding to the selected button.

### Displaying Host Data in Recordsets

Recordsets are labeled using the names you define in the model. Recordsets appear as read-only text in the Web application. You can update recordsets by creating a procedure to handle the update processing.

If scrolling operations have been defined for a recordset, the Web application displays buttons that correspond to these operations. Users can navigate through the recordset display using these buttons.

If recordset selection is defined in the model, the generated Web application supports selecting a specific record to display record details.

## **Event Handlers and Terminal Screen Layout**

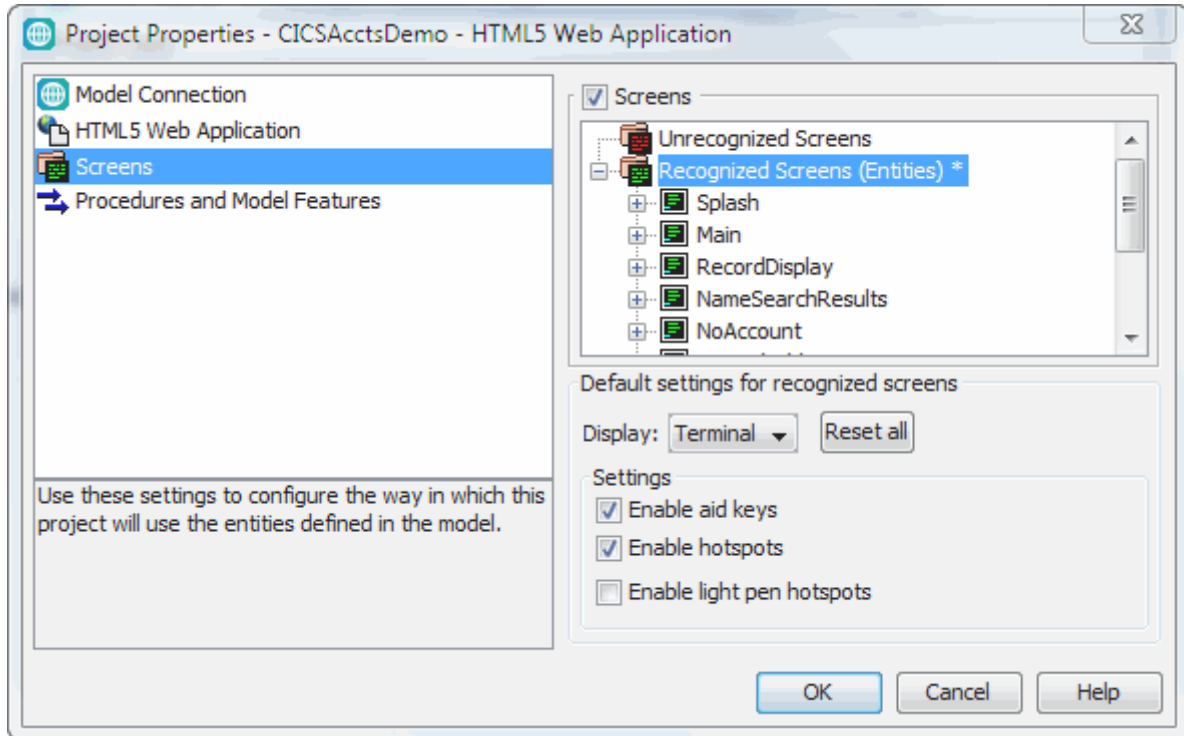
When Web Builder generates a Web application from a model containing event handlers, there are several limitations, depending on the type of Web application you are building. Web applications using a terminal screen layout have these limitations:

- Attribute reads are truncated to fit the attribute size that is defined in the model.

- Attribute input fields only allow input of characters up to the attribute size that is defined in the model.

## 5.1.12 About Hotspots

Verastream Host Integrator HTML 5 Web applications support two types of hotspots; aid key support, enabled by default, and light pen support. To use hotspots, select Enable Aid Keys on the Screen properties panel after you set the Display type to Terminal.



### Hotspots for Aid Keys

Hotspots are buttons that appear over common host commands in terminal sessions. By using hotspots, you can control your terminal session with the mouse instead of the keyboard. Clicking a hotspot transmits a terminal key or command to the host. By default, hotspots are configured for the most common 3270 or 5250 commands. You can customize the aid key hotspots to provide more functionality.

When you are building an HTML 5 Web application project, any host screen that is configured to display as a terminal has hotspots enabled by default. Selecting Enable Aid Keys is a prerequisite for using hotspots.

#### Default hotspots for 3270 and 5250 sessions

Host type	Hotspot	Does this...
3270	PF1...PF24	Transmits a PF1...PF24 to the host
3270	1=...24=	Transmits a PF1...PF24 to the host
3270	clear	Transmits a Clear key to the host

<b>Host type</b>	<b>Hotspot</b>	<b>Does this...</b>
3270/5250	F1....F24	Transmits a PF1...PF24 to the host
3270	PA1,PA2,or PA3	Transmits a PA1, PA2, or PA3 to the host
3270/5250	enter	Transmits an Enter key to the host

Host type	Hotspot	Does this...
5250	more...	Transmits a Roll Up key to the host (scrolls down one page)

## Light Pen Hotspots

To use light pen hotspots, set Display type to Terminal, and then select Enable aid keys. Light pen support is disabled by default.

If you have HTML5 web applications that were generated in versions of the product prior to the initial release of light pen support, you must rebuild the application to enable light pen support.

### Interacting with light pen fields

You interact with HTML 5 web application light pen fields using a mouse or by touch on a mobile device.

Light pen fields are determined by the host application. There are two types of light pen fields; selection light pen fields (which are visible as checkboxes) and immediate action light pen fields that, when activated, cause some action to occur.

When you send a screen containing light pen fields to the host by either invoking an aid key or by invoking an immediate action light pen field, the following occurs:

- When the modified state of a selection-type light pen field differs from the state designated by the host application, the cursor select aid action is invoked at the position of the first character in the light pen field.

- All modified input fields are sent to the host.

- When aid keys are invoked they are sent to the host.

All light pen fields have tooltips associated with them.

## Configuring Hotspots

You can configure specific hotspots for each HTML 5 Web application that you generate. Each Web application has a hotspot.configurations file, which you can find in the

`Verastream\HostIntegrator\servletengine\webapps\<ProjectName>\WEB-INF\classes` directory. This file contains information explaining the file format. For detailed information on how to create your own hotspots, see [Knowledge Base 7021326](#).

## 5.1.13 Keyboard Maps

You can map your keyboard so specific workstation keyboard keys will function as terminal keys for a host application. Keyboard mapping is available for Web applications.

You can implement keyboard mapping after a project is built, or incorporate it into a User-Defined Project Type, which you can use to build subsequent Web applications using the same keyboard map.

### Keyboard mapping requirements

Each time you build a Web application using these settings a default keyboard mapping is generated for 3270 and 5250 keyboards.

Keyboard mapping requires the:

- Host Integrator model to be connected to either a 3270 or 5250 terminal
- Client has JavaScript enabled
- Terminal option selected for the screen display
- Enable option selected for the Aid key settings

### Default settings

Keyboard key	3270 Terminal Key	5250 Terminal Key
F1-F12	PF1-PF12	PF1-PF12
Shift + F1-F12	PF13-PF24	PF13-PF24
Scroll lock	Clear	Clear
Escape	Reset	Reset
Ctrl-Enter	Enter	Enter
Page Up	PA1	Page Up
Page Down	PA2	Page Down
KP Minus (5250)	n/a	Field Minus



Keyboard key	3270 Terminal Key	5250 Terminal Key
KP Plus (5250)	n/a	Field Plus

When visible, the Aid keys have automatically generated tooltips that show the keyboard mapping.

## Modify the Keyboard Mapping

Use the following information to edit the keyboard mapping for 3270 or 5250 host applications:

### AidKey Class Constructor

`aidKey`— The `AidKey3270/5250.Codes` enumeration to map to the terminal key

`key`— The local keyboard `Key.Codes` enumeration to map to the terminal key

`keyModifier`— The local keyboard `Key.Modifiers` enumerations are required when mapping the local key. The single modifier `Key.Modifiers.SHIFT` requires only the Shift key to be pressed. Multiple `Key.Modifiers.SHIFT|Key.Modifiers.CTRL` requires the Shift and Control keys to be pressed.

### Using keyboard.js

The `keyboard.js` file defines the local keyboard code enumeration, `Key.Codes`, and the modifier key, `Key.Modifiers`:

```
Key.Modifiers = {
 NONE: 0,
 ALT: 1,
 CTRL: 2,
 SHIFT: 4,
 META: 8 //Windows or Apple Command keys
};

Key.Codes = {
 ESCAPE: 27, //ESC
 F1: 112,
 F2: 113,
 F3: 114,
 F4: 115,
 F5: 116,
 F6: 117,
};
```

### Using `aidkeys_3270.js` and `aidkeys_5250.js`

Both the `aidkeys_3270.js` and `aidkeys_5250.js` define the `aidKeysxxxx`. Codes as well as the mapping between the Aid keys and the local keyboard.

```
AidKey3270.Codes = {
 ATTN: 257,
 CLEAR: 277,
 SYSRQ: 504,
 RESET: 393,
};
```

### To edit your keyboard mapping configuration

Open `aidkeys_3270.js` or `aidkeys_5250.js` (depending on your terminal type), in a text or JavaScript editor.

- To modify a key

Find the entry in the Keyboard3270 or Keyboard5250 constructor. `new`

```
AidKey(AidKey3270.Codes.PF1, Key.Codes.F1, Key.Modifiers.NONE) Modify the key or
keyModifier parameter as needed. new AidKey(AidKey3270.Codes.PF1, Key.Codes.F1,
Key.Modifiers.ALT)
```

- To add a key

At the end of the Keyboard3270 or Keyboard5250 constructor, add a new entry.

```
new AidKey (AidKey3270.Codes.PA2, Key.Codes.PAGE_DOWN, Key.Modifiers.NONE),
new AidKey (AidKey3270.Codes.SYSRQ, Key.Codes.PGUP, KeyModifiers.SHIFT)
));
```

- To remove a key

Remove the line that contains the Aid key you want to remove. For example, change this:

```
new AidKey (AidKey3270.Codes.PF23, Key.Codes.F11, Key.Modifiers.SHIFT),
new AidKey (AidKey3270.Codes.PF24, Key.Codes.F12, Key.Modifiers.SHIFT),
new AidKey (AidKey3270.Codes.PA1, Key.Codes.PAGE_UP, KeyModifiers.NONE)
```

To this:

```
new AidKey(AidKey3270.Codes.PF23, Key.Codes.F11, Key.Modifiers.SHIFT),
new AidKey (AidKey3270.Codes.PA1, Key.Codes.PAGE_UP, KeyModifiers.NONE)
```

## - Troubleshoot a problem

Edit the browser.js file by setting the logging settings, located near the beginning of the file, to True.

```
var LOGGER = true;
```

```
var ERROR = true;`
var WARN = true;
var INFO = true;
var DEBUG = true;
var TRACE = true;
```

Set the logging setting back to False when you are finished testing.

## 6. Legal Notice

---

Copyright 2023 Open Text.

The only warranties for products and services of Open Text and its affiliates and licensors (“Open Text”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Open Text shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.