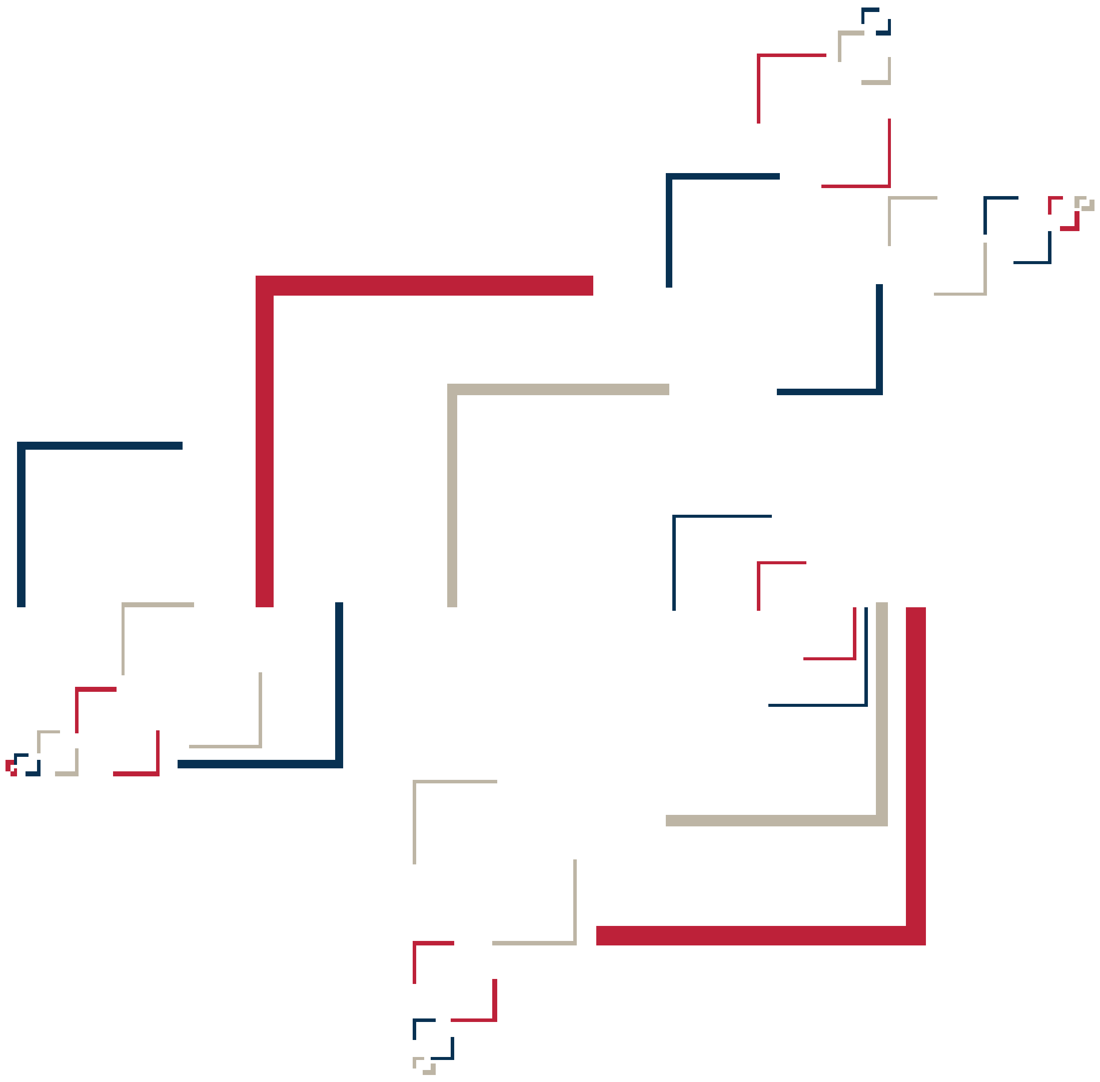


Micro Focus®

Modernization Workbench™

Transforming Applications

MICRO FOCUS®

Contents

Preface

The Modernization Workbench is a suite of PC-based software products for analyzing, re-architecting, and transforming legacy applications. The products are deployed in an integrated environment with access to a common repository of program objects. Language-specific parsers generate repository models that serve as the basis for a rich set of diagrams, reports, and other documentation.

The Modernization Workbench suite consists of customizable modules that together address the needs of organizations at every stage of legacy application evolution: maintenance/enhancement, renovation, and modernization.

Audience

This guide assumes that you are a corporate Information Technology (IT) professional with a working knowledge of the legacy platforms you are using the product to analyze. If you are transforming a legacy application, you should also have a working knowledge of the target platform.

Organization

This guide contains the following chapters:

- Chapter 1, "Overview of Transformation," contains common description of Modernization Workbench transformation tools and phases.
- Chapter 2, "Data Transition," describes Database Schema tool and Data Transition phase, when you examine how the data is stored in your legacy system and then generate object-oriented access methods for the new database schema.
- Chapter 3, "Interface Transition," describes User Interface tool and Interface Transition phase, when you analyze the interactions between the legacy screens on a project basis and generate the target code for the interface to create a new graphical user interface (GUI) based on the window flow.
- Appendix A, "Data Access Object Support (CRUD Runtime Facilities)," describes Data Access Object support for Java.
- The Glossary defines the names, acronyms, and special terminology used in this guide.

Conventions

This guide uses the following typographic conventions:

- **Bold type**: indicates a specific area within the graphical user interface, such as a button on a screen, a window name, or a command or function.
- *Italic type*: indicates a new term. Also indicates a document title. Occasionally, italic type is used for emphasis.
- `Monospace type`: indicates computer programming code.
- **`Bold monospace type`**: indicates input you type on the computer keyboard.
- **1A**/**1B**, **2A**/**2B**: in task descriptions, indicates mutually exclusive steps; perform step A or step B, but not both.

Related Manuals

This document is part of a complete set of Modernization Workbench manuals. Together they provide all the information you need to get the most out of the system.

- *Getting Started* introduces the Modernization Workbench. This guide provides an overview of the workbench tools, discusses basic concepts, and describes how to use common product features.
- *Preparing Projects* describes how to set up Modernization Workbench projects. This guide describes how to load applications in the repository and how to use reports and other tools to ensure that the entire application is available for analysis.
- *Analyzing Projects* describes how to analyze applications at the project level. This guide describes how to create diagrams of applications and how to perform impact analysis across applications. It also describes how to estimate project complexity and effort.
- *Analyzing Programs* describes how to analyze applications at the program level. This guide describes how to use HyperView tools to view programs interactively and perform program analysis in stages. It also describes how to analyze procedure and data flows, search the repository, and extract business rules with HyperView.
- *Managing Application Portfolios* describes how to build enterprise dashboards that track survey-based metrics for applications in your portfolio. It also describes how to use Enterprise View Express to browse Web-generated views of application repositories.
- *Creating Components* describes how to extract program components from a legacy application.
- *Error Messages* lists the error messages issued by Modernization Workbench, with a brief explanation of each and instructions on how to proceed.

Online Help

In addition to the manuals provided with the system, you can learn about the product using the integrated online help. All GUI-based tools include a standard Windows **Help** menu.

You can display:

- The entire help system, with table of contents, index, and search tool, by selecting **Help:Help Topics**.
- Help about a particular Modernization Workbench window by clicking the window and pressing the **F1** key.

Many Modernization Workbench tools have *guides* that you can use to get started quickly in the tool. The guides are help-like systems with hyperlinks that you can use to access functions otherwise available only in menus and other program controls.

To open the guide for a tool, choose **Guide** from the **View** menu. Use the table of contents in the **Page** drop-down to navigate quickly to a topic.

Overview of Transformation

1

The Modernization Workbench provides a powerful solution to help with legacy application transformation to modern platforms and computing paradigms. The Modernization Workbench application is an integrated environment that lets you centrally locate and analyze the massive amounts of legacy components. Once you decide which portions of the applications you want to preserve and migrate, Modernization Workbench can transform the selected code into components targeted to the new environment.

Modernization Workbench uses a *repository,* which is a database of objects. You store the various components of your legacy application, and the new code and models that you create from them, in this repository. The repository is created when you are setting up your project. For detailed information of this phase, refer to *Preparing Projects* book in the workbench documentation set.

By using analysis tools of Modernization Workbench, parsers analyze the legacy components to help you understand the relationships and dependencies among system components. Graphical tools of Modernization Workbench help you to visualize the existing system by giving you a better understanding of how the system flows. A legacy system can be transformed without performing any analysis, however, the benefits of modernizing such a system would be much less. For detailed information about legacy applications analysis with Modernization Workbench, refer to *Analyzing Projects* and *Analyzing Programs* books in the workbench documentation set.

Data Transition

During Data Transition phase (Chapter 2, “Data Transition”), you examine how the data is stored in your legacy system. Modernization Workbench creates a database schema that describes the design of the database within the application.

After Modernization Workbench creates a model of the database, you can make changes to that database — you can change the tables, columns, and keys and the relationships among them, all on a project basis.

Finally, you can generate object-oriented access methods for the new database schema using Data Active Objects (DAO) or Java Database Connectivity (JDBC) and export them out of Modernization Workbench.

Database Overview

A database is a structured set of persistent data. A simple database might be a single file containing many records, each of which contains the same set of fields where each field is a certain fixed width.

Hierarchical, network, and relational databases are the three most common methods of organizing data:

- ***Hierarchical databases*** link records together similar to an organization chart. A record type can be owned by only one owner. For example, in a hierarchical database for processing customer orders, the *order* record type is owned only by *customer*. Hierarchical structures were widely used in the first mainframe database management systems. However, due to their restrictions, they often cannot be used to relate structures that exist in the real world.

- ***Network database structures*** contain a record type that can have multiple owners. In the order processing example, the *orde*r record type is owned by both *customer* and *product* because that's the way they relate in the business.
- ***Relational databases*** do not link records together physically. The design of the records must provide a common field, such as account number, to allow for matching. Quite often, the fields used for matching are indexed to speed up the process. In the order-processing example, customers, orders, and products are linked by comparing data fields and/or indexes when information from more than one record type is needed.

  This method is the most flexible for ad hoc inquiries but may be too slow for heavy transaction processing environments.

Modernization Workbench currently supports:

- DB2
- SQLServer
- Oracle
- Access

Interface Transition

In the Interface Transition phase (Chapter 3, "Interface Transition"), you analyze the interactions between the legacy screens on a project basis. You can use the User Interface tool to understand how the screen objects and the program logic interact.

During Interface Transition Modernization Workbench creates a window flow, which describes the interface of the application. Once Modernization Workbench has created the window flow, you can modify it.

Finally, you generate the target code for the interface to create a new graphical user interface (GUI) based on the window flow.

Data Transition

2

The Modernization Workbench enables you to transform structural information about your data into a database schema on a project basis by using the Database Schema tool. You can then make changes to the database schema. This phase of transformation is described in "Editing a Database Schema" on page 2-8.

Once you have finished the database schema, you can generate (if the Transformation Assistant component of Modernization Workbench is installed) new DDL statements and object-oriented access methods for the new database schema using Java Database Connectivity (JDBC), entity Enterprise Java Beans (EJB), or XML document and export them out of Modernization Workbench.

Starting Database Schema Tool

Start the Database Schema tool by selecting **Transform:Data Transition** or by clicking the **Database Schema** button on the Modernization Workbench main toolbar. The Database Schema tool main window is displayed (Figure 2-1).

Figure 2-1 *Database Schema Tool Main Window*

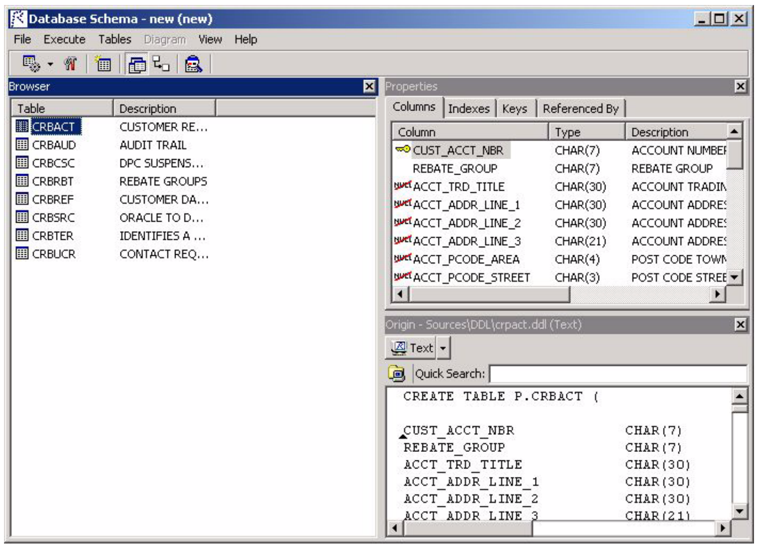


Table	Description
CRBACT	CUSTOMER RE...
CRBAUD	AUDIT TRAIL
CRBCSC	DPC SUSPENS...
CRBRBT	REBATE GROUPS
CRBREF	CUSTOMER DA...
CRBSRC	ORACLE TO D...
CRBTER	IDENTIFIES A ...
CRBUCR	CONTACT REQ...

Column	Type	Description
CUST_ACCT_NBR	CHAR(7)	ACCOUNT NUMBEI
REBATE_GROUP	CHAR(7)	REBATE GROUP
ACCT_TRD_TITLE	CHAR(30)	ACCOUNT TRADIN
ACCT_ADDR_LINE_1	CHAR(30)	ACCOUNT ADDRE:
ACCT_ADDR_LINE_2	CHAR(30)	ACCOUNT ADDRE:
ACCT_ADDR_LINE_3	CHAR(21)	ACCOUNT ADDRE:
ACCT_PCODE_AREA	CHAR(4)	POST CODE TOWN
ACCT_PCODE_STREET	CHAR(3)	POST CODE STREE

```
CREATE TABLE P.CRBACT (

CUST_ACCT_NBR                    CHAR(7)
REBATE_GROUP                     CHAR(7)
ACCT_TRD_TITLE                   CHAR(30)
ACCT_ADDR_LINE_1                 CHAR(30)
ACCT_ADDR_LINE_2                 CHAR(30)
ACCT_ADDR_LINE_3                 CHAR(21)
```

Navigating the Main Window

The Database Schema tool window can include the following panes:

- **Browser** pane — contains the hierarchy of the current project's database schema. The hierarchy tree automatically updates as objects are added, modified, or deleted from the project.
- **Diagram** pane — contains the graphical representation of the current project's database schema.

Note: To switch between **Browser** and **Diagram** panes, select the corresponding item in the **View** menu or click the respective button on the Database Schema tool window's toolbar.

- **Properties** pane — contains the subparts of whatever object is selected in the Browser or Diagram pane. For example, if **Table** is selected, it displays a list of tables.

- **Origin** pane — displays the source code and is enabled for Interactive Analysis. For information about using HyperView, see the *Analyzing Projects* book of the workbench documentation set.
- **Errors** pane — contains an errors list generated by the last verification procedure. You can navigate to the error's location by double-clicking on the error message.

Note: Furthermore, the Database Schema tool can display the **Activity Log** window, which displays the information about the results of Modernization Workbench operations.

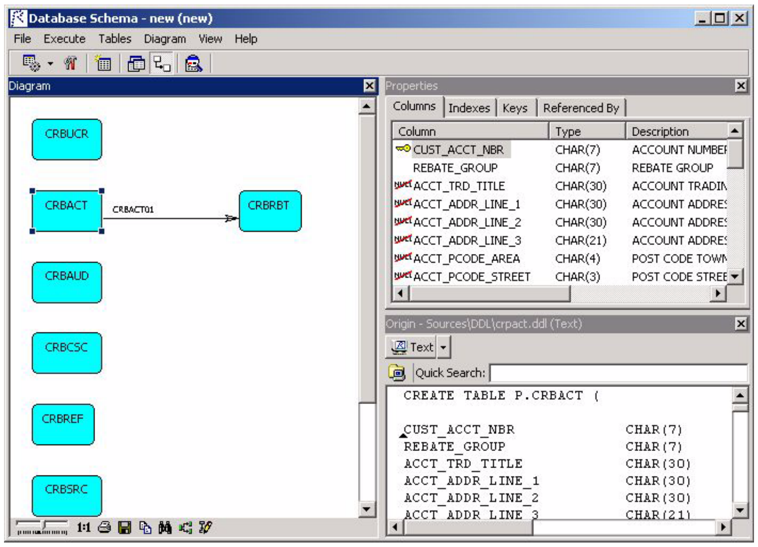
Arranging the Panes

The Database Schema tool window is separated into panes by splitters. All splitters are supplied with a pop-up menu. To open it, move the cursor onto a splitter (as you would for dragging it, which is also possible) and click the right mouse button. For additional information about using the splitters, refer to *Getting Started* book of the workbench documentation set.

Selecting a View

After a schema is created, you can view data by either a list view (Figure 2-1 on page 2-2) (default) or as a diagram (Figure 2-2 on page 2-4). To choose, select **View:Browser** or **View:Diagram** from the menu or click the corresponding button on the toolbar. Of course, you can open both panes.

Figure 2-2 *Diagram View*


Database Schema - new (new)
File Execute Tables Diagram View Help
Diagram
CRBUCR
CRBACT
CRBACT01
CRBRBT
CRBAUD
CRBCSC
CRBREF
CRBSRC
1:1
Properties
Columns Indexes Keys Referenced By
Origin - Sources\DDL\crpact.ddl (Text)
Text
Quick Search:

Column	Type	Description
CUST_ACCT_NBR	CHAR(7)	ACCOUNT NUMBEI
REBATE_GROUP	CHAR(7)	REBATE GROUP
ACCT_TRD_TITLE	CHAR(30)	ACCOUNT TRADIN
ACCT_ADDR_LINE_1	CHAR(30)	ACCOUNT ADDRES
ACCT_ADDR_LINE_2	CHAR(30)	ACCOUNT ADDRES
ACCT_ADDR_LINE_3	CHAR(21)	ACCOUNT ADDRES
ACCT_PCODE_AREA	CHAR(4)	POST CODE TOWN
ACCT_PCODE_STREET	CHAR(3)	POST CODE STREE

```
CREATE TABLE P.CRBACT (

CUST_ACCT_NBR                     CHAR(7)
REBATE_GROUP                      CHAR(7)
ACCT_TRD_TITLE                    CHAR(30)
ACCT_ADDR_LINE_1                  CHAR(30)
ACCT_ADDR_LINE_2                  CHAR(30)
ACCT_ADDR_LINE_3                  CHAR(21)
```

Locating Errors

When you Verify your table edits, if any errors are present, an errors list is generated. To display the list, select **View:Errors** from the menu. You can navigate to the error's location by double-clicking on the error message.

Setting Database Schema Options

Database Schema options specify import and generation operations for the Database Schema tool.

To set Database Schema options:

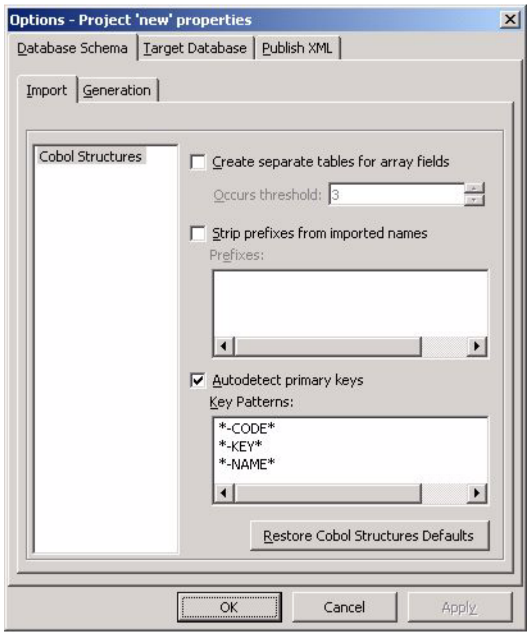
1 In the **View** menu, choose **Options**. The Options window is displayed.

2 Click the **Import** tab, then click Cobol Structures (Figure 2-3 on page 2-6)

> ***Note***
>
> This tab is COBOL-specific, so you should use only the next tab to set PL/I options.

- **Create separate tables for array fields** — indicate whether to create additional tables for structure or field (these tables contain references to the main table) by checking this selection (**on**). If unchecked (off), then during data structures import, each array field defined in a COBOL OCCURS phrase is repeated *n* times, for example, mycol1, mycol2,... mycol*n*)
    - **Occurs threshold** — if **Create separate tables for array fields** is checked (**on**), then indicate the boundary value of OCCURS *n*, such that when n is less or equal to the threshold, each column is repeated and when *n* is greater than the threshold, a new table is created
- **Strip prefixes from imported names** — indicate prefixes to be stripped from the fields' names. With the help of this option, you can more easily tell one data structure from another by their names, which could originally start with a common prefix. The prefix list can be modified by right-clicking in the **Prefixes** box. From the displayed pop-up menu, you can **Add** a new pattern, **Delete** a selected pattern, of **Edit** a selected pattern's text.
- **Autodetect primary keys** — indicates that specified pattern references to VSAM data items that are referenced in copybooks should be selected as primary keys. The Key Patterns list can be modified by right-clicking in the **Key Patterns** box. From the displayed pop-up menu, you can **Add** a new pattern, **Delete** a selected pattern, of **Edit** a selected pattern's text.
- **Restore Cobol Structures Defaults** — click button to restore Modernization Workbench defaults

Figure 2-3 *Project Options — Database Schema:Import:Cobol Structures*

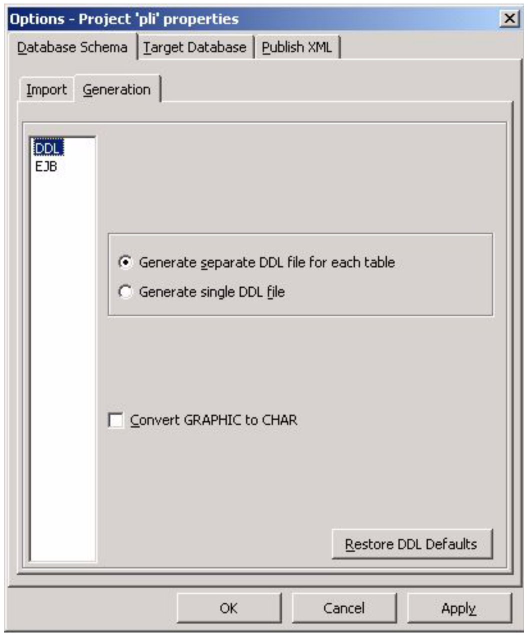


☐ Create separate tables for array fields
Occurs threshold: 3
☐ Strip prefixes from imported names
Prefixes:
☑ Autodetect primary keys
Key Patterns:
-CODE
-KEY
-NAME

3 Click the **Generation** tab (Figure 2-4 on page 2-7). From the list in the left pane, select the type of object to be generated. The options are different for each object-type:

- **DDL** — select the object-type by clicking the corresponding radio button (Figure 2-4 on page 2-7). When executing DDLs at a target database environment, the order of execution has to be known, for example, a table has to be created before references to it (foreign keys) are created. For a large system, creating one table could become too complicated to find the correct order for executing the DDLs. However, by creating a single DDL file, you can ensure that “main” tables are created before a table that references them. Also, for target environments where execution of multiple SQL statements in one query is not possible, such as MS Access, you can, with one file, inherit a means of executing separate DDLs in correct order.

Figure 2-4 *Project Options — Database Schema:Generation:DDL*



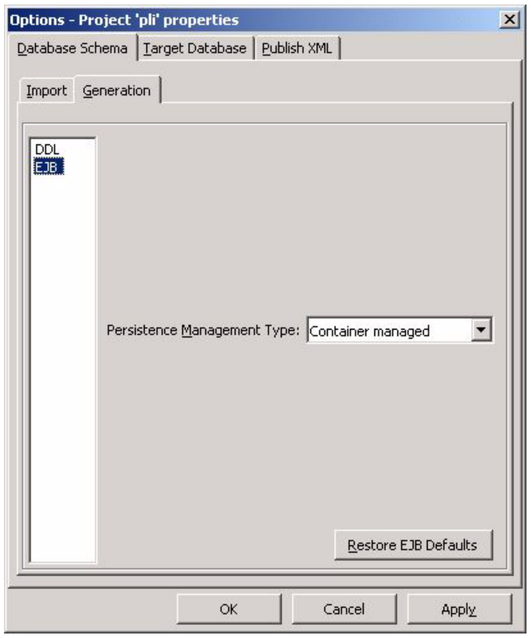
(●) Generate separate DDL file for each table
( ) Generate single DDL file

[ ] Convert GRAPHIC to CHAR

- **Generate separate DDL file for each table** — select the radio button to generate multiple DDL for each table
- **Generate single DDL file** — select the radio button to generate only one DDL file
- **Convert GRAPHIC to CHAR** — when selected, the GRAPHIC data structures are converted to CHARs (one byte), otherwise to NCHARs (two bytes).
- **Restore DDL Defaults** — click button to restore Modernization Workbench defaults

- **EJB** (Figure 2-5 on page 2-8)
  - **Persistence Management Type** — Container managed (default) or Bean managed. To change the type, select the pull-down menu and click on your choice.
  - **Restore EJB Defaults** — click button to restore Modernization Workbench defaults

Figure 2-5 *Project Options — Database Schema:Generation:EJB*



Persistence Management Type: Container managed

Editing a Database Schema

After the Database Schema tool creates a database schema or executes the conversion to the relational database, you can make changes to it.

Note: If you make the changes in the structure of the relational database, you should perform in future all actions concerned with maintenance of this new software.

You can, for example, create a table definition from a COBOL copybook. COBOL copybooks represent a nonrelational data structure that is used to read and write to a file.

By creating a schema from one or more copybooks or data definition modules, you can create a relational model from a flat or hierarchical model. To refine the model, you can identify additional structures, such as foreign keys, and new tables representing many-to-many relationships. It is possible to import data structures from COBOL programs, copybooks, data definition modules, to any of the existing tables or to a new one, if you like. Also, you can create a data map to prescribe subsequent physical transfer of certain data fields to table columns.

If the database schema is already relational, you might want to modify the schema based on the transformation target. For example, if you have a table in the schema that will not be transformed, you could delete this table and references to it (foreign keys) from the schema.

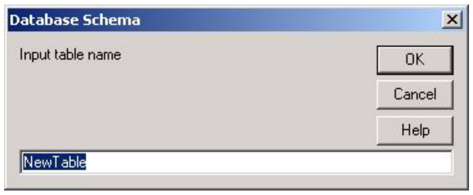
Creating a Table

To create a new table in your project:

1. Select **Tables:New...** from the main menu or right-click in the Browser pane and select **New...** from a pop-up menu, or click the **Create New Table** button on the toolbar.

   The **New Table** dialog box (Figure 2-6) is displayed.

Figure 2-6 *Table Name Dialog Box*

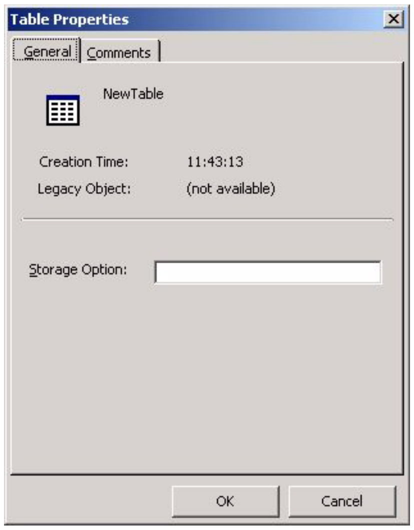


2. Type a name for the table in the table name field.
3. Click **OK**.

   The new table is created and is displayed in the left pane tree and in the Table tab.
4. Select the table name in the Browser pane and select **Tables:Properties** from the menu or right-click the table name and select **Properties** from the pop-up menu. The **Table Properties** dialog box (Figure 2-7 on page 2-10) is displayed.

   You can enter Storage options and comments.

Figure 2-7 *Table Properties Window*



NewTable

Creation Time: 11:43:13
Legacy Object: (not available)

Storage Option:

Editing a Table

After you create a table, you can edit its items by selecting them in the corresponding tab in Properties pane. You can select:

- **Columns** — create, edit, and drop columns
- **Indexes** — create, edit, and drop indexes
- **Keys** — create, edit, and drop keys
- **Referenced By** — create, edit and drop reference keys

To create any of the items, select the appropriate tab, right-click in the tab pane, and select **New** from the pop-up menu. The new item is added to the pane's list (Figure 2-8 on page 2-11).

Figure 2-8 *New Column*

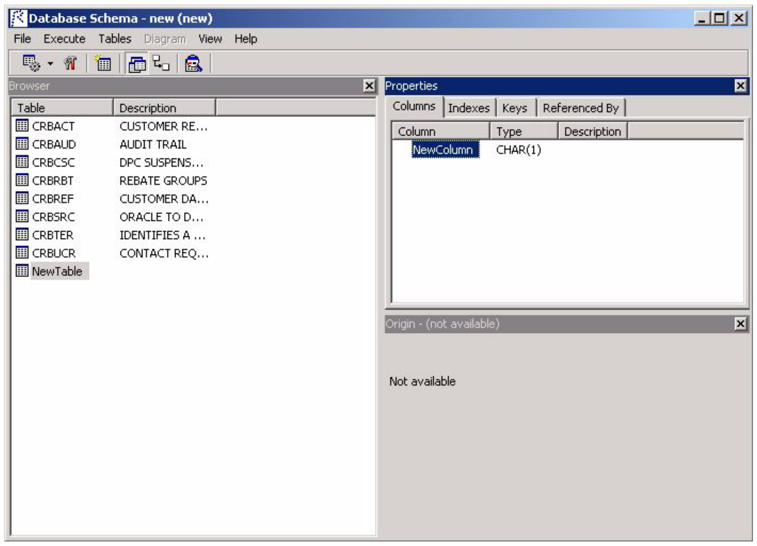

Database Schema - new (new)
File Execute Tables Diagram View Help
Browser
Properties
Columns Indexes Keys Referenced By
Origin - (not available)

Table	Description
CRBACT	CUSTOMER RE...
CRBAUD	AUDIT TRAIL
CRBCSC	DPC SUSPENS...
CRBRBT	REBATE GROUPS
CRBREF	CUSTOMER DA...
CRBSRC	ORACLE TO D...
CRBTER	IDENTIFIES A ...
CRBUCR	CONTACT REQ...
NewTable	

Column	Type	Description
NewColumn	CHAR(1)	

Not available

Working with Columns

Creating a column

To create a column:

1 Click the **Columns** tab.

2 Right-click and select **New** from the pop-up menu.

After you create a column, the new column is displayed in the column list (Figure 2-8). You can modify the column’s name, its type, scale, and precision and add comments.

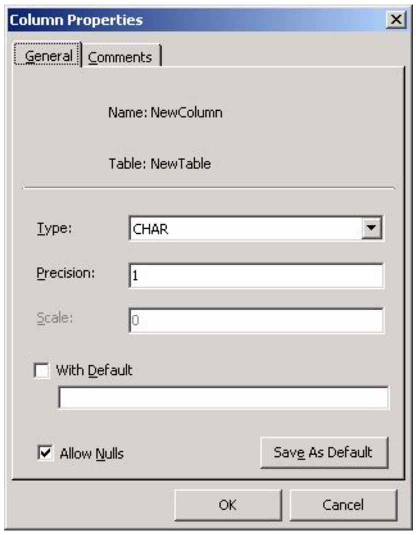
Changing a Column Name

To change a column name, right-click on the column name you want to change and select **Rename** from the pop-up menu. Type the new name in the column’s text box.

Editing a Column

To edit a column's attributes, right-click on the column you want to change, select **Properties** from the pop-up menu. The **Column Properties** dialog (Figure 2-9) is displayed. Type or select the new value.

Figure 2-9 *Column Properties dialog*



Name: NewColumn
Table: NewTable
Type: CHAR
Precision: 1
Scale: 0
☐ With Default
☑ Allow Nulls

Dropping a Column

To drop a column, select a column, and right-click, and select **Delete** from the pop-up menu.

Changing Column Order

If you have several columns, you can change their order in the table by selecting a column, holding the mouse key down, and dragging the column to change its position.

Working with Indexes

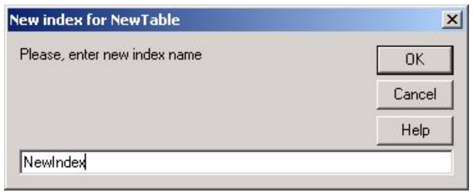
Creating an Index

To create an index:

1 Click the **Indexes** tab.

2 Right-click and select **New** from the pop-up menu. The **New index for...** dialog box (Figure 2-10) is displayed.

Figure 2-10 *Create New Index Dialog*



Please, enter new index name

NewIndex

3 Type the name for the index in the dialog box and click **OK**.

4 The new index is added to the index list (Figure 2-11).

Figure 2-11 *Indexes Tab*

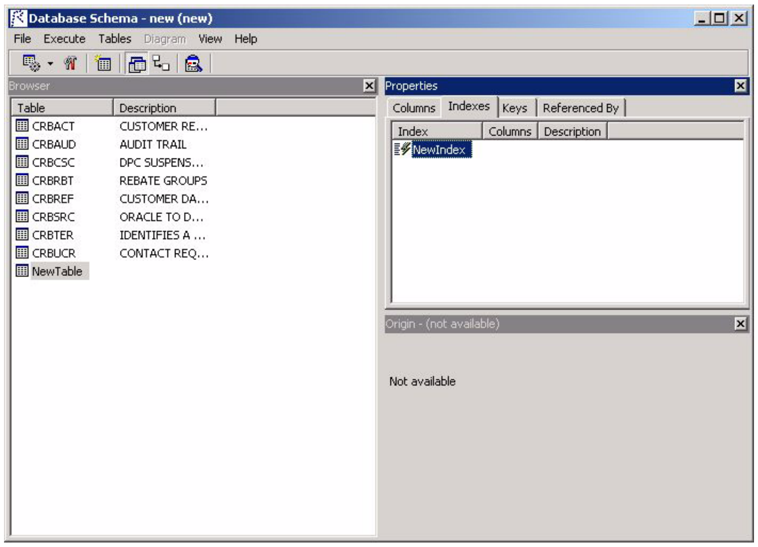

Database Schema - new (new)
File Execute Tables Diagram View Help
Browser
Properties
Columns Indexes Keys Referenced By
Origin - (not available)

Table	Description
CRBACT	CUSTOMER RE...
CRBAUD	AUDIT TRAIL
CRBCSC	DPC SUSPENS...
CRBRBT	REBATE GROUPS
CRBREF	CUSTOMER DA...
CRBSRC	ORACLE TO D...
CRBTER	IDENTIFIES A ...
CRBUCR	CONTACT REQ...
NewTable	

Index	Columns	Description
NewIndex		

Not available

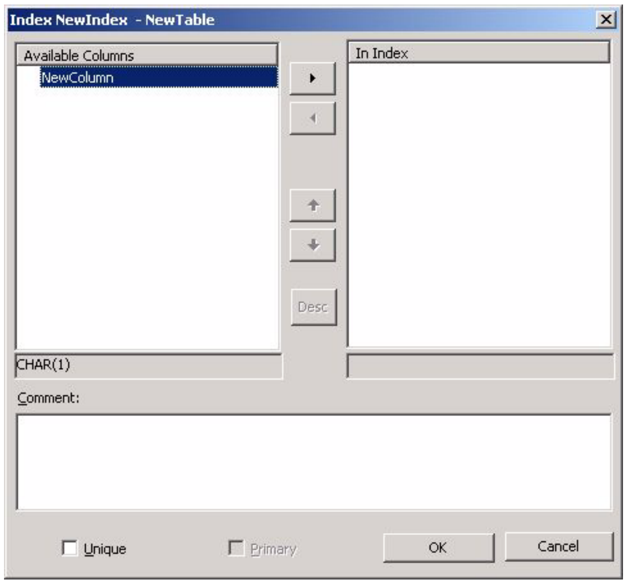
You can edit the index by selecting it from the index list, double-clicking or right-clicking, and selecting **Properties** from the pop-up menu. The **Index** dialog (Figure 2-12 on page 2-14) is displayed.

Adding a Column to an Index

To add a column to an index:

1. Select the column you want to add from the **Available Columns** list.
2. Repeat for each column.
3. Click the **Add** button.
4. Adjust the order of the index by selecting it in the **In Index list and using Up or Down arrows.**
5. Select whether the index is unique by checking the **Unique** check-box. If it is checked, the **Primary** checkbox is also available.
6. Click **OK**.

Figure 2-12 *Edit Index Dialog*



Removing a Column from an Index

To remove a column from an index:

1. Select the column you want to remove from the **In Index** list.

2 Repeat for each column.

3 Click the **Remove** button.

4 Click **OK**.

Working with Keys

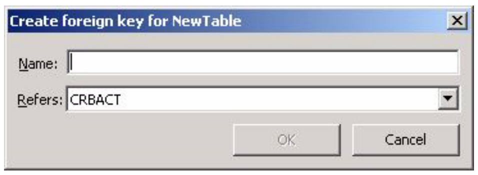
Creating a Key

To create a key:

1 Click the **Keys** tab.

2 Right-click and select **New** from the pop-up menu. The **Create foreign key for...** dialog box (Figure 2-13) is displayed.

Figure 2-13 *Create Foreign Key Dialog*



3 Type the name for the key in the dialog box and click **OK**.

4 The new key is added to the key list (Figure 2-14).

Figure 2-14 *New Foreign Key*

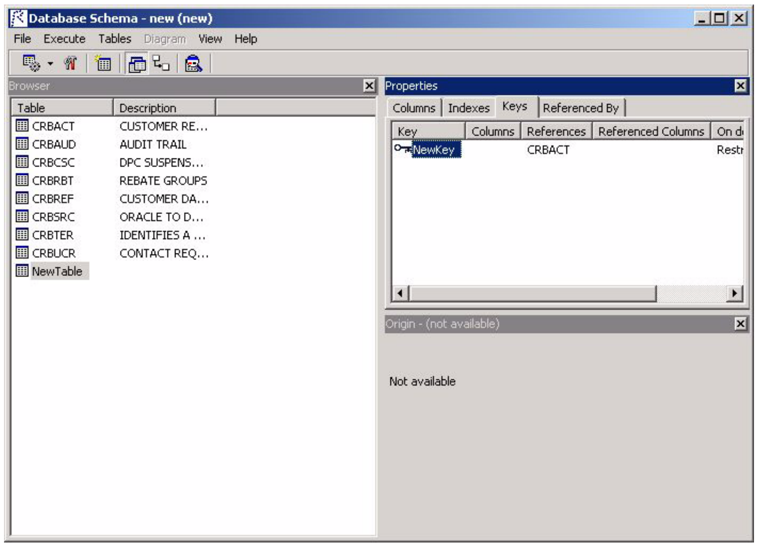

Database Schema - new (new)
File Execute Tables Diagram View Help
Browser
Properties
Columns Indexes Keys Referenced By
Origin - (not available)
Not available

Table	Description
CRBACT	CUSTOMER RE...
CRBAUD	AUDIT TRAIL
CRBCSC	DPC SUSPENS...
CRBRBT	REBATE GROUPS
CRBREF	CUSTOMER DA...
CRBSRC	ORACLE TO D...
CRBTER	IDENTIFIES A ...
CRBUCR	CONTACT REQ...
NewTable	

Key	Columns	References	Referenced Columns	On d
NewKey		CRBACT		Restr

To edit the foreign key, select it from the key list, double-click or right-click, and select the **Properties** item from the pop-up menu. The **Foreign Key** dialog box (Figure 2-15) is displayed.

Adding a Column to a Key

To add a column to a key:

1 Select the column in the list of columns on the left of the dialog box.

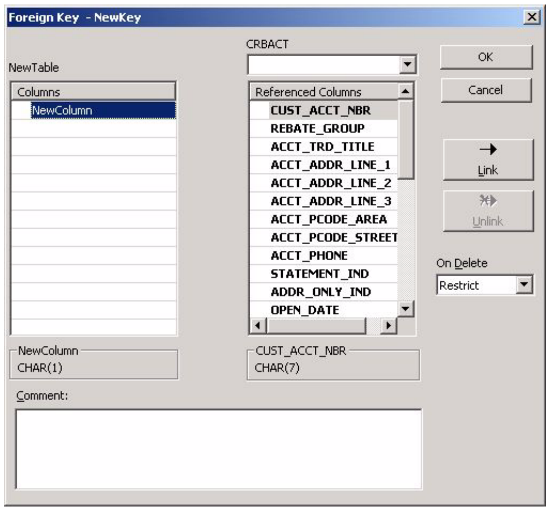
2 Select the referenced column on the right of the dialog box.

3 Select the action to perform from the **On Delete** drop-down menu if the key is deleted.

4 Click **Link**. An arrow is drawn to illustrate the link.

5 Click **OK**.

Figure 2-15 *Foreign Key Dialog*



Dropping a Column from a Key

To drop a column from a key:

1. Select the column in the list of columns on the left of the dialog box.
2. Click **Unlink**.

The column is dropped from the key.

Dropping a Key

To drop a key, select the key in the **Key** tab on the Database Schema tool window, right-click, and select **Delete** from the pop-up menu.

Editing Referenced Keys

To edit referenced keys:

1. Click the **Referenced By** tab.
2. Select a key from the key list.

3. Double-click or right-click and select **Properties** from the pop-up menu. The **Foreign key for...** dialog box (Figure 2-15 on page 2-17) is displayed.
4. Follow directions from "Creating a Key" on page 2-15.

Dropping a Table

To drop a table:

1. Select the table in the tree in the Browser pane.
2. **Right-click and click Delete from the pop-up menu.**

The table is dropped from the project.

Importing Structures

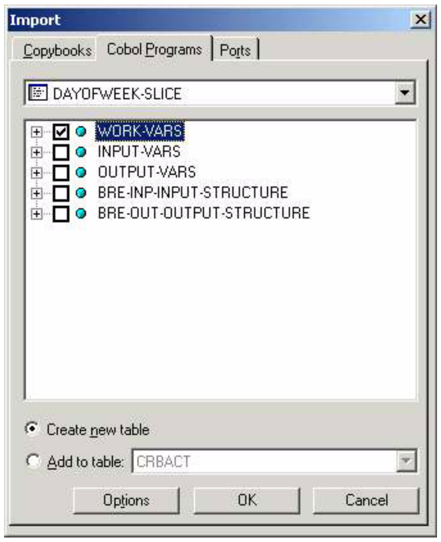
The Database Schema tool allows you to import tables from different files included in your project. It can import Cobol structures and DMS schemas as described below.

Importing Cobol Structures

To import COBOL structure from Cobol programs, copybooks or ports in COBOL project, you should do the following steps:

1. Select **Execute:Import Cobol Structures** from the main Database Schema tool menu or click the **Execute** button and select the same item from the drop down list. The **Import** dialog will appear:

Figure 2-16 *Import Cobol Structures Dialog*



2 Choose a tab with necessary Cobol structure.

3 Select the name of desired Cobol structure in the drop-down list.

4 Check necessary tree branch (you can expand and collapse the tree by clicking a plus sign to the left of desired branch). Only elements that have children should be checked.

5 Choose whether creating new table or adding to existent.

6 Press **OK**.

All chosen branches will be imported as tables. If you chose **Create new table** then a new table with the same name will be created for every checked branch. You can also set Import options for COBOL structures by clicking **Options** in the Import dialog. You can find detailed description in “Setting Database Schema Options” on page 2-4.

Importing DMS Schemas

> ***Note***
>
> This action is available only for COBOL projects.

If you add DMS DDL files to your project and verified them then corresponding DMS schemas will be created and included in your project automatically. After verification you can import DMS schemas and convert them to tables, for example, for further using during Cobol-SQL generation from Unisys-CDML.

To import DMS schemas, you should select **Execute:Import DMS Schemas** from the main Database Schema tool menu or click the **Execute** button and select the same item from the drop down list. All DMS schemas contained in your project will be imported (it means that you cannot choose a specific schema to be imported).

Generating DDL Statements

To generate DDL statements for your database schema:

1. Select a database schema.
2. Select **Execute:Generate DDL for <target database type>** from the Database Schema tool main window's menu or click the **Execute** button and select the same name item from the drop down list.

   You can select the following target database types:

   - **Microsoft SQL Server**
   - **Microsoft Access**
   - **Oracle**
   - **DB2**

The generated models appear in the **Target DDL Models** folder.

Generating Java Access Methods

To generate Java access methods for your database schema:

1 Select a database schema.

2 Select **Execute:Generate Access for Java/JDBC** from the Database Schema tool main window’s menu or click the **Execute** button and select the same name item from the drop down list.

The new methods appear in the **Target Access Models** folder.

Access Methods for Java (JDBC) Usage

To create the access methods for Java (JDBC):

1 Apply **Generate Access for Java/JDBC** method to the selected database schema as it is described in “Generating Java Access Methods”.

2 Export the generated files (*<Table>Service.java* for each table in schema) to new directory.

3 Create your application using the generated classes in the same directory and import the com.relativity.rescuent.dao run-time support classes in your sources (import *com.relativity.rescuent.dao.**).

4 To compile the created application, you should specify the path to Java run-time support classes: *…\RunTime\Java\classes.zip* (for example, using the -CLASSPATH option for the Sun Microsystems compiler).

5 Run the program <java compiler> [-options]* <Test>.java.

6 You also need to create DSN for your database by means of the ODBC32 Manager in your Windows' Control Panel.

7 Pass DSN as the only parameter for table constructor, if connection to your DSN without User/Password is enabled; otherwise specify DSN, User, Password as the parameters of the table constructor.

8 The only way to transfer data from/to the user is by using row class instance, which has all necessary methods.

9 Run the program *<java interpreter> [-options]* <Test>*, where *<Test>* is your application file name.

The options depend on Java VM being used:

- adding to classpath *DIR=.\RunTime\Java\classes.zip;*.
- for Sun javac: *java -classpath%classpath%;DIR, for jre -cp DIR*;
- for MS jvc: *jview -cp DIR*

Implementing DAO Classes

A Data Access Object is generated as a class, one per relational table. Class. Methods are:

- **Insert** — method to insert new row in the table

  Input: Single row

  Output: SQL code
- **Singleton Select** — method to search any table row by primary key

  Input: Primary Key

  Output: Single Row + SQL code
- **Cursor Select** — iteration method to sequentially fetch all rows in the table

  Input: Cursor

  Output: Single Row + Cursor + SQL code
- **Update** — method to update table row

  Input: Primary Key + Row with values for update

  Output: SQL code
- **Delete** — method to delete any row by primary key

  Input: Primary key

  Output: SQL code

Note: **Create table** and **Drop table** actions are not supported.

Note: Referential Integrity Constraints are enforced by the DBMS system.

Note: The current implementation of cursor allows browsing the whole table only, without any selection criteria. For example, if you specify that you want to browse a table for values between 100 and 200, the entire table is returned.

Generating EJB Access Methods

To generate EJB (Enterprise Java Bean) access methods for your database schema:

1. Select a database schema.
2. Select **Execute:Generate Entity EJBs** from the Database Schema tool main window's menu or click the **Execute** button and select the same name item from the drop down list.

The new methods appear in the **Target Access Models** folder.

Generating Entity Java Beans

Entity Beans generation highlights

1. For each table from a DB Schema, Modernization Workbench generates an entity Java bean (EJB) that corresponds to a row of this table. Each bean resides in its own subdirectory of the target generation directory.
2. Table fields become the entity bean's private members.
3. The remote interface should contain methods set/get for all the table fields, and this is the only business logic of the generated entity bean.
4. If a table from DB Schema has no primary key, then no entity bean is generated for it.
5. The home interface should contain the following methods:
   - create(...)

with all table fields' values as the parameters; this is a method to create a new entity instance (and therefore to insert a new row in the table)

- remove(<*primary-key*>)

  to remove an entity bean instance (and therefore to delete the corresponding row from table)

- findByPrimaryKey(<*primary-key*>)

  to get an entity bean instance by primary key (i.e., to search a table row by primary key)

- findAll()

  to get collection with all entity bean instances from a persistent storage (i.e., to open cursor with all table records)

6 Modernization Workbench provides two models of entity bean persistence by a user option:

- bean-managed
- managed by container

7 Transaction management attribute for the generated entity bean by default is TX_REQUIRED, i.e. the generated bean will be executed inside the client transaction context (if it exists), otherwise it starts its own transaction. Access DBMS does not support transactions, so for this DBMS a default transaction attribute is TX_NOT_SUPPORTED.

8 The generation uses templates located in *<Modernization Workbench>\Templates\Gen\EJB\*.**, so the user can slightly modify the text to be generated; however, in that case he is responsible for possible errors.

9 Modernization Workbench generates the entity bean source code and default deployment descriptor, as well as a *Build.cmd* file to compile and create a jar from. Also, for each DB Schema a master *Build.cmd* file is generated, which allows to compile all beans generated for the DB Schema (it subsequently calls Build.cmd of each generated bean). These *.cmd* files use paths to Java compiler and

server platform, which paths are taken from the *<Modernization Workbench>\Data\Environment.ini* file. Generated jar file placed in *<WebLogic>\config\<Domain>\applications\* directory (for WebLogic) or in *%WAS_HOME%\classes\* (for WebSphere) and is automatically deployed by server platform. As additional for every bean Modernization Workbench generates a small test which performs simple operation using the generated bean. This test is placed in the *TEST* subdirectory.

Step-by-step Entity Beans Generation

Assume existence of a DB Schema to generate EJBs from. EJBs will be generated only for tables with primary keys.

The order of execution is as follows:

1. Apply the **Generate Access for Java/ADBC** or **Generate Entity EJBs** action to the chosen DB Schema.
2. This method has the option **Persistence Management Type** with two values:
   - **Bean managed** — to generate Entity Bean with bean-managed persistence
   - **Container managed** — to generate Entity Bean with container-managed persistence
3. When the sources of EJBs are generated, the stage of compilation and deployment ensues. Modernization Workbench provides batch files to compile and deploy EJBs using WebLogic classes, but the deployment descriptor and file with WebLogic properties should be modified manually.

Customization of Deployment Descriptor

The deployment descriptor contains some parameters common for both persistence management types. Among such parameters is transaction attribute that specifies how the container should manage the transaction boundaries when delegating method invocation to a business method of an enterprise bean.

Modernization Workbench uses the value required (TX_REQUIRED or Required in EJB v1.1 specification) for the default one, but some databases (e.g., Microsoft Access) do not support this feature, so in that case, the value not supported is used (TX_NOT_SUPPORTED or NotSupported in EJB v1.1 specification).

Having information about the database type, it is possible to generate the transaction attribute more accurately.

Bean-Managed Persistence

Because an EJB with bean-managed persistence has to "know" the data source name and account to connect with, Modernization Workbench generates environment values with this information. If the information is absent at the moment, it should be specified before deployment.

database	database type (Access | SQLServer | Oracle | DB2)	SQLServer
driverClass	JDBC driver class (e.g. sun.jdbc.odbc.JdbcOdbcDriver)	sun.jdbc.odbc. JdbcOdbcDriver
subProtocol	subprotocol for JDBC driver (e.g. odbc)	odbc
dataSource-Name	DSN registered in ODBC administrator	DSN
user	user name to use for connection	""
password	user password to use for connection	""

Because the EJB generation does not have these options available for the user now, the info should be specified manually before deploying. Nevertheless, the default values have an effect. They are shown in the right column.

Container-Managed Persistence

An EJB with container-managed persistence does not contain code for interaction with any data source. Thus, the values described in the previous section are unnecessary for these beans. The container, however, should know the parameters of the bean's storage. So, this is how the concept of a connection pool appears. A server platform connection pool provides a pool of connections that are created when the server starts up. Then, when an authorized user needs a connection to a database, he requests and uses an already existing connection from the pool, rather than create a direct, specific connection to the database. When persistence is container-managed, such a user is the container itself.

The deployment descriptor should contain the name of connection pool, and this pool must be described in server platform properties. Modernization Workbench generates the default name “Pool” for the connection pool.

Customization of WebLogic Properties

Weblogic 6.0 performs automatic deploying of EJBs if they are placed in directory *<WebLogic>\config\<Domain>\applications\*.

For more details, refer to *WebLogic User’s Guide*.

Bean-Managed Persistence

There are no additional options for EJBs with bean-managed persistence.

Container-Managed Persistence

You must describe the connection pool.

For example, do the following:

1 In administrative console choose **Domain:Services:JDBC:Connection Pool**, then choose **Create a new Connection Pool**.

2 Set pool name — the same as used in EJBs.

3 Specify the values for **url** and **driver** fields (for example jdbc:odbc:DSN and sun.jdbc.odbc.JdbcOdbcDriver).

4. In **properties** field specify user name and password.
5. Confirm creating by pressing **Apply**.
6. In **TARGETS**, choose target server from list.

For more details, refer to *WebLogic User's Guide*.

After customizing the WebLogic properties, the server can be started and any client application will be able to use the deployed EJBs.

Customization of WebSphere Properties

To work with EJBs you need installed IBM WebSphere 4.0 server, Java compiler and configured Database DNS's.

For Entity bean designed with container managed persistence you must add a connection pool in a server. There are two examples of creating a pool. The first one is for Oracle database and uses Oracle driver for a connection pool. The second one uses Modernization Workbench driver located in the *<Modernization Workbench>\RunTime\Java\rwsql.jar*. This driver uses JdbcOdbc bridge to connect Data Source Name (DSN) specified in the system.

Example 1

1. Start server.
2. Start console.
3. Switch to **Resources:JDBC Drivers**.
4. Choose **Create a New**.
   - Set some **Name** of resource.
   - Specify **Implementation Classname** — the Java class name with ConnectionPoolDataSource interface — Oracle.jdbc.pool.OracleConnectionPoolDataSource.
   - Specify **Server Class Path** — the path to JAR files containing driver classes (for example, *d:\Oracle\jdbc\lib\classes12.zip*).
   - Press OK.

5 Switch to **Resources:JDBC Drivers:<Your new driver name>:Data Sources**.

6 Choose **Create a New**.

- Set some **Name** of this resource.
- Set the same JNDI name as a Pool name in generated EJBs.
- Set minimum pool size.
- Set maximum pool size.
- Set timeouts.
- Set default user ID and password.
- Press **OK**.

7 Go to the Property Set of this new Data source.

8 Add new Property:

- Name = 'URL'
- Type = 'java.lang.string'
- Value = connection url (for example, jdbc:oracle:thin:user/password@host:1521:orcl)
- Press OK.

Example 2

1 Start server.

2 Start console.

3 Switch to **Resources:JDBC Drivers**.

4 Choose **Create a New**.

- Set some **Name** of resource.
- Specify **Implementation Classname** — javax.rw.RWDataSource.
- Specify **Server Class Path** — the path to *rwsql.jar* file.
- Press **OK**.

5. Switch to **Resources:JDBC Drivers:<Your new driver name>:Data Sources**.
6. Choose **Create a New**.
    - Set some **Name** of this resource.
    - Set the same JNDI name as Pool name in generated EJBs.
    - Set minimum pool size.
    - Set maximum pool size.
    - Set timeouts.
    - Press **OK**.
7. Go to the Property Set of this new Data source.
8. Add new Property:
    - Name = url
    - Type = String
    - Value = connection url (that is ODBC Data Source name, for example, DSND for DB2)
9. Add new Property:
    - Name = user
    - Type = String
    - Value = user name
10. Add new Property:
    - Name = password
    - Type = String
    - Value = user password
11. Add new Property:
    - Name = driverclass
    - Type = String
    - Value = driver class - sun.jdbc.odbc.JdbcOdbcDriver
12. Add new Property

- Name = subprotocol
- Type = String
- Value = subprotocol - odbc

An Example of Client Program

```
import javax.naming.*;
import javax.ejb.*;
import java.rmi.RemoteException;
import java.util.*;

public class Client
{
  static String url       = "t3://localhost:7001";
  static String user      = null;
  static String password  = null;

  public static void println(String S)
  {
    System.out.println(S);
  }

 public static void main(String[] args)
  {
    println("Beginning...");

    try
    {
      Context ctx = getInitialContext();
      BooksHome home = (BooksHome) ctx.lookup("BooksTable");
      Books books;

      if (args != null)
      {
```

```
    switch(args.length)
    {
      case 0:
        {
          println("Fetching all instances...");
          Enumeration e = home.findAll();
          println("Fetching ended...");

          if (e != null)
          {
            while (e.hasMoreElements())
            {
              println("Getting element from enumeration...");
              books = (Books) e.nextElement();
              println("Output element contents...");
          println("" + books.getId() + ", " + books.getAuthor()
                      + ", " + books.getTitle());
              books.remove();
            }
          }


        }
        break;
      case 1:
      case 2:
        break;
      case 3:
        println("Creating new instance...");
       books = home.create(Integer.parseInt(args[0]), args[1],
                             args[2]);
        break;
    }
  }
}
```

```
    catch (Exception e)
    {
     println(":::::::::::: Unexpected Error :::::::::::::::");
      e.printStackTrace();
    }
    finally { println("End..."); }
  }

 static public Context getInitialContext() throws Exception
  {
    Hashtable h = new Hashtable();
    h.put(Context.INITIAL_CONTEXT_FACTORY,
          "weblogic.jndi.WLInitialContextFactory");
    h.put(Context.PROVIDER_URL, url);
    if (user != null)
    {
      System.out.println ("user: " + user);
      h.put(Context.SECURITY_PRINCIPAL, user);
      if (password == null) password = "";
      h.put(Context.SECURITY_CREDENTIALS, password);
    }
    return new InitialContext(h);
  }
}
```

Generating XML

To generate XML from objects of the "DB Schema" type, select **Execute:Generate XML** from the Database Schema tool menu, or click the **Execute** button and choose **Generate XML** from a drop-down menu.

XML file and corresponding schema file will be generated for every table from DB Schema. The names of files are formed using the next rule:

<*file-name*> = xml + <*table-name*> + <*extension*>. For example for table “A” there will be generated two files: *xmlA.xml* and *xmlA.xsd*.

XML Generation Results

During this process Modernization Workbench generates:

In XML document:

1 A general heading with the XML version and encoding indicated:

```
<?xml version="1.0" encoding="UTF-8"?> - for English
version
```

```
<?xml version="1.0" encoding="Shift_JIS"?> - for Japanese
version
```

2 A root element named after this table and consisting of sequence of elements each of them corresponds to table column. The root element has only the one attribute named Comments which contains comments specified in DBSchema.

Example:

```
<Table1 Comments="">
... //columns go here
</Table1>
```

3 Sub elements for each column of this table named after this column.

Each element has the Value attribute. If the field has some default value, then this value will be generated, otherwise it will be made a null string.

Example:

```
<Table1 Comments="User comments">
  <A1 Value=""/>
  <A2 Value=""/>
</Table1>
```

In XML Schema document:

1 A general heading with the XML version and encoding indicated:

```
<?xml version="1.0" encoding="UTF-8"?> - for English
version
```

```
<?xml version="1.0" encoding="Shift_JIS"?> - for Japanese
version
```

2 schema header containing ‘xs’ namespace definition:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

3 simpleType element with the name SQLType defining SQL types. This type defines enumeration with the next values: Character, Date, Time, Float, Real, Double, Integer, LongVarCharacter, Numeric, SmallInt, TinyInt, Timestamp, VarCharacter, Graphic, VarGraphic and LongGraphic.

```
<xs:simpleType name="SQLType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Character"/>
    <xs:enumeration value="Date"/>
    <xs:enumeration value="Time"/>
    <xs:enumeration value="Float"/>
    <xs:enumeration value="Real"/>
    <xs:enumeration value="Double"/>
    <xs:enumeration value="Integer"/>
    <xs:enumeration value="LongVarCharacter"/>
    <xs:enumeration value="Numeric"/>
    <xs:enumeration value="SmallInt"/>
    <xs:enumeration value="TinyInt"/>
    <xs:enumeration value="Timestamp"/>
    <xs:enumeration value="VarCharacter"/>
    <xs:enumeration value="Graphic"/>
    <xs:enumeration value="VarGraphic"/>
    <xs:enumeration value="LongGraphic"/>
   </xs:restriction>
</xs:simpleType>
```

4 Element description for every table column.

```
<xs:element name="A1">
```

Element has the name according to column name and the following attributes:

- Name — character name of the column. This attribute has a string type.
- Type — type of the column, that can have one of SQL types: Character, Float, Real etc. This attribute has a SQLType type.
- Length — length of the field. This attribute has a string type.
- Prec — precision. This attribute has a string type.
- Scale — scale. This attribute has a string type.
- Comments — comments entered by the user in the DB schema editor. This attribute has a string type.
- Value — if the field has some default value, then this value will be generated, otherwise it will be made a null string. This attribute has a string type.

Note: Attributes except Comments and Value are fixed.

Note: If two columns in the DB schema have identical names, then one of those names will be changed (by adding some number to its end), because XML does not allow non-unique names of elements.

Example:

```
 <xs:element name="A1">

    <xs:complexType>

      <xs:attribute name="Name" type="xs:string"
fixed="A1"/>

      <xs:attribute name="Type" type="SQLType"
fixed="SmallInt"/>

      <xs:attribute name="Length" type="xs:string"
fixed="0"/>

      <xs:attribute name="Prec" type="xs:string"
fixed="0"/>
```

```
      <xs:attribute name="Scale" type="xs:string"
fixed="0"/>

      <xs:attribute name="Comments" type="xs:string"
default=""/>

      <xs:attribute name="Value" type="xs:string"
default=""/>

    </xs:complexType>

 </xs:element>
```

5 Root element description. This element has the same name as a table.

Description of this element consists of sequence of references to subelements. Each element represents table column.

```
 <xs:element name="Table1">

    <xs:complexType>

      <xs:sequence minOccurs="0" maxOccurs="unbounded">

        <xs:element ref="A1"/>

        <xs:element ref="A2" minOccurs="0"/>

      </xs:sequence>

      <xs:attribute name="Comments" type="xs:string"
default=""/>

    </xs:complexType>

 </xs:element>
```

The sequence may not occur. This is identified by minOccurs and maxOccurs attributes.

Elements corresponded to columns that are not a part of a primary key or unique index may not occur. This is identified by minOccurs attribute in element reference.

Also the root element has attribute named Comments and a string type. This attribute has default value specified in DBSchema.

Exporting Source Files

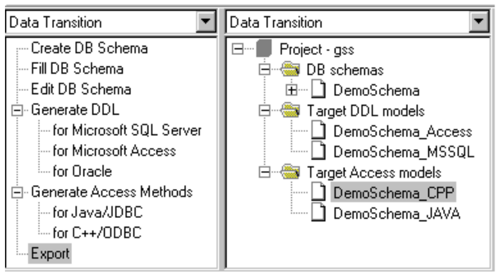
When you have generated the new access methods or DDL statements, you can export them out of the repository. This makes them available to your development environment.

To export source files:

1. Select the access method that you want to export (Figure 2-17 on page 2-38).
2. Right-click to display the pop-up menu, and select **Export**

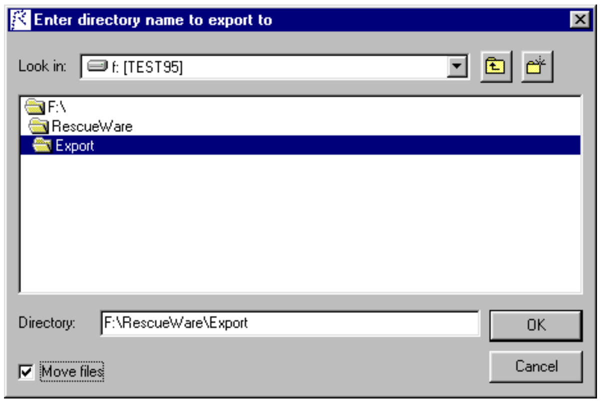
Note: Click **View** to view the generated object in text form.

Figure 2-17 *Select the item to export*



3. Select a directory and click **OK.** To move the files instead of copying them, check the **Move Files** check box (Figure 2-18).

Figure 2-18 *Select a directory for exported files*



Directory: F:\RescueWare\Export

[✓] Move files

Interface Transition

3

The Interface Transition phase lets you analyze the interaction among programs, maps, and screens (and COBOL copybooks in COBOL projects), and also find all events that may occur on a mainframe 3270 panel. The outputs from the Interface Transition phase are interfaces that you can implement in Java or HTML.

The Interface Transition phase enables you to edit the window flow to transfer your legacy application's 3270 interface to a new type of interface. While editing, you can change which events cause windows to be displayed, change the appearance of the interface, and emulate it. Once you are satisfied, you generate the new interfaces. Finally, you export the model of your interface to an external directory.

A window flow is edited with the help of User Interface tool. When you start this tool, it first scans the maps and screens, programs, COBOL copybooks (for COBOL projects) to find the events inside the legacy code that define the interactions between screen objects and the program logic. This action adds your legacy application interface data to the window flow.

Then, based on the discovered events, User Interface creates emulated window views for the programs. Using this tool, you can change the window flow and add buttons, menu options, and different text.

Events can be added to the new interface in the following ways:

- **Programmatically**

  Modernization Workbench determines the mapping between the field in the screen and the field in the program, then finds all the occurrences of that field in the program. This occurs during the Event Mining.

- **Manually associating fields and events**

  If the field in the legacy program that maps directly to the field in the screen is not the one that gets tested because it is assigned to another field, you can add an event-sensitive data item to the field in the screen that is tested.

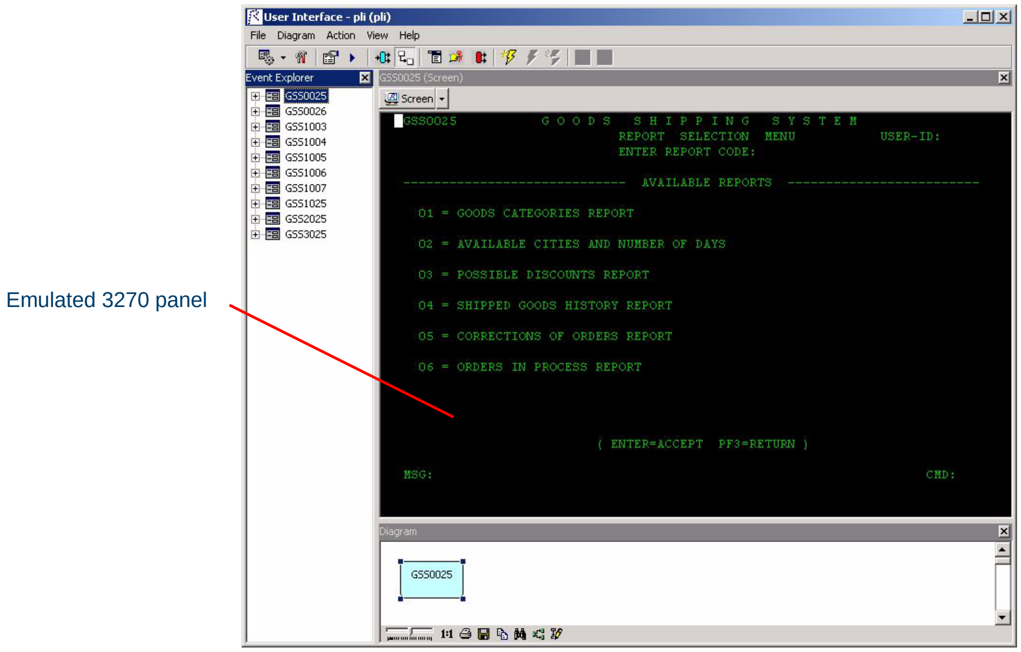
- **Manually adding new events**

  If you want to add new events that did not appear in the legacy system, you can go directly to the screen and add events. These events have no associated fields in the legacy system but will appear in the new system.

Starting User Interface Tool

To start the User Interface tool, click **Transform:User Interface** or click the **User Interface** button on the Modernization Workbench main toolbar. The User Interface tool main window (Figure 3-1 on page 3-3) is displayed.

Figure 3-1 *User Interface Tool Main Window*



```
GSS0025          G O O D S   S H I P P I N G   S Y S T E M
                           REPORT  SELECTION  MENU           USER-ID:
                           ENTER REPORT CODE:

----------------------------  AVAILABLE REPORTS  -------------------------

  01 = GOODS CATEGORIES REPORT

  02 = AVAILABLE CITIES AND NUMBER OF DAYS

  03 = POSSIBLE DISCOUNTS REPORT

  04 = SHIPPED GOODS HISTORY REPORT

  05 = CORRECTIONS OF ORDERS REPORT

  06 = ORDERS IN PROCESS REPORT



                        ( ENTER=ACCEPT  PF3=RETURN )

MSG:                                                             CMD:
```

At this point, you may name your screens following the recommendations of the "Naming the Screens" section below.

Setting User Interface Options

User Interface options specify import and generation operations for the User Interface Transition tool.

To set User Interface options:

1 In the **View** menu, choose **Options**. The Options window is displayed.

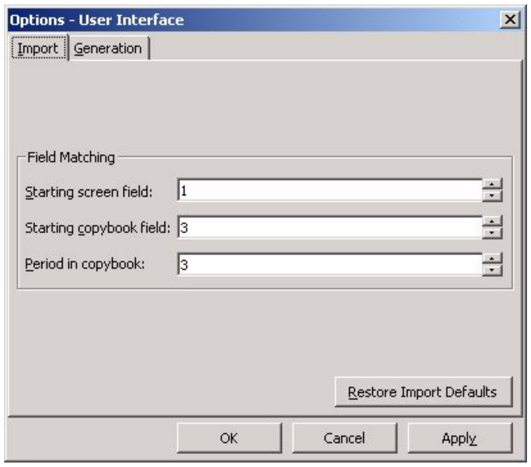
2 Click the **Import** tab (Figure 3-2 on page 3-4).

Note

This tab is COBOL-specific and so you should use only the next tabs to set PL/I options.

- **Field Matching** panel
  - **Starting screen field** — indicate the first field you want to map.
  - **Starting copybook field** — indicate first field in the COBOL copybook that BMS maps match.
  - **Period in copybook** — the number of fields in a COBOL copybook group. Each item in a BMS map matches a group of fields in the COBOL copybook.
- **Restore Import Defaults** — click button to restore Modernization Workbench defaults.

Figure 3-2 *Project Options — User Interface:Import*



Starting screen field: 1
Starting copybook field: 3
Period in copybook: 3

3 Click the **Generation** tab (Figure 3-3 on page 3-5).

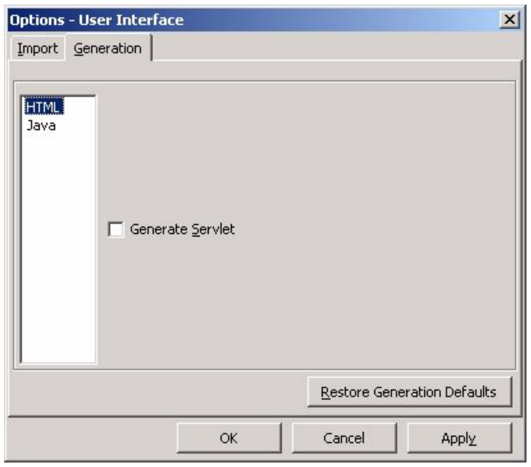
Select the type of generation that you want to set options for from the available types in the left-side panel.

- **HTML** — Generate Servlet (Figure 3-3 on page 3-5)

  By default it is set to **off** (unchecked). If you want to generate standard servlets providing user's access via Internet to the transformed interface (and written in Java), click the option's checkbox to display a checkmark (**on**). For more details on generating

Java servlets refer to “Generating Java Servlets for HTML” on page 3-36.

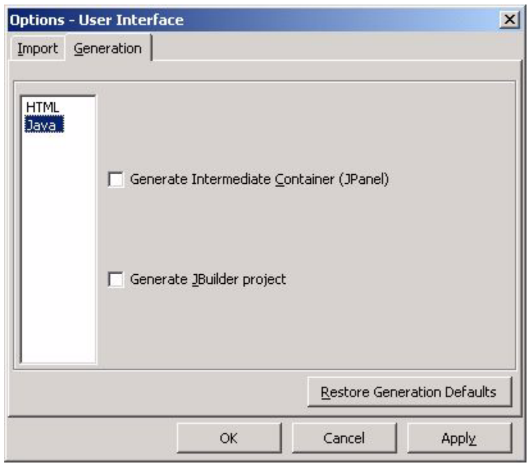
Figure 3-3 *Project Options — User Interface:Generation:HTML*



☐ Generate Servlet

- **Java** (Figure 3-4 on page 3-6)
  - **Generate Intermediate Container (JPanel)** — By default it is set to **off** (unchecked). If you want to generate JPanels, click the option’s checkbox to display a checkmark (**on**).
  - **Generate JBuilder project** — By default it is set to **off** (unchecked). If you want to generate a JBuilder project, click the option’s checkbox to display a checkmark (**on**).

Figure 3-4 *Project Options — User Interface:Generation:Java*



Options - User Interface

Import | Generation

HTML
Java

☐ Generate Intermediate Container (JPanel)

☐ Generate JBuilder project

Restore Generation Defaults

OK | Cancel | Apply

- **Restore Generation Defaults** — click button to restore Modernization Workbench defaults

If you do not want to change any of the settings, click **Cancel** to discard any changes and return to the main Modernization Workbench window. Otherwise, change the settings as necessary, click **Apply** to save settings and keep the window open or click **OK** to save the changes and return to the main Modernization Workbench window.

Navigating the panes of the User Interface Tool Window

The User Interface tool window can be divided into three different panes. Moreover, the Activity Log can be displayed in this window. You can reduce the number of panes in the window by hiding them, as well as restore any pane by displaying it again. You can also rearrange the panes within the window as you want. Figure 3-1 shows the window with all the panes and Activity Log. The panes are:

- **Event Explorer** pane

  In Figure 3-1 on page 3-3, this is the left pane. You can display/hide it by selecting **View:Event Explorer** from the menu.

  The Event Explorer pane is the leading pane of the User Interface tool window. However, the Diagram Pane, described below, can also be a leading window. The display in the other panes depends on what you select in the hierarchy (event tree) drawn by the Explorer. **The hierarchy levels for COBOL projects in the** Event Explorer pane **are:**

1. BMS maps (these correspond to 3270 panels), MFS maps and AS 400 screens
2. COBOL programs (if available; otherwise, just the “Fields” group label)
3. Screen fields
4. COBOL fields (data items; these correspond to fields on the screen)
5. Events

Note: The last two levels are displayed only if the corresponding COBOL program is available.

Note: MFS and DPS maps can be displayed in the same way as BMS maps or AS 400 screens, including the fields highlighting/selection, but no other actions described below are currently supported for MFS or DPS maps.

Note: You can right-click on any of the items in the hierarchy to display context-sensitive pop-up menus, which are always copied by the main **Action** menu.

- **Source** pane

  In Figure 3-1 on page 3-3, this is the right upper pane. You can display/hide it by selecting **View:Source** from the menu.

  The display in the Source pane depends on what you select in the Event Explorer pane:

  - A topmost-level selection (map) shows you the emulated screen, which is a visual display of the map, with the selector positioned in the first line (as in Figure 3-5 on page 3-11)
  - A second-level selection (source COBOL program, if available), shows this program’s code in HyperView, with the selector indi-

cating the statement that relates to this map, as in Figure 3-5 below (selecting the “Fields” group label on the second level does not change the display at all)

- A third-level selection (field) shows the emulated screen for the map, with the selector positioned on this field
- A fourth-level selection (COBOL variable) shows the source program code, with the selector positioned on the declaration of this variable
- A bottom-level selection (event) shows the source program code, with the selector positioned on the code line corresponding to this event

- **Diagram** pane

  In Figure 3-1 on page 3-3, this is the right lower pane. You can display/hide it by selecting **View:Diagram** from the menu.

  Two different views are provided for the Diagram pane:

  - Current

    This view is set by checking **Current window,** unchecking **Whole diagram** in the **Diagram** menu, or by clicking the **Current Window Links** button on the toolbar (each of these immediately switches the other two).

    In this view, the pane contains the object for a map as it appears in the window flow diagram.

    No modes or tools specific for the Diagram pane are provided in the Current view.

  - Diagram

    This view is set by checking **Whole diagram**, unchecking **Current window** in the **Diagram** menu, or by clicking the **Window Flow Diagram** button on the toolbar (each of these immediately switches the other two).

    In this view, the whole window flow diagram is displayed (if any part of the diagram is not seen immediately, you can always scroll the pane to it).

The window flow diagram consists of map rectangles and directed event lines that connect some of them (characterization of the corresponding event is provided near each line). The current (selected map's) rectangle casts a shadow and, thus, is always distinguishable in the diagram.

Note: Event lines in a window flow diagram are drawn only for those events which you have created yourself, as described below. The events originally existing in your window flow are not reflected in the diagram.

When you click any rectangle it becomes the current one. It immediately appears selected as the topmost item in the Event Explorer pane. The display in the Source pane also switches to the corresponding screen. Thus, the Diagram pane in the Diagram view is the second leading pane of the User Interface window. For the purpose of controlling your navigation within the window flow, sometimes it may be convenient to alternate between the Diagram and Event Explorer panes.

For rearranging the diagram, you can manually select the rectangles in the diagram and drag them to other locations in the pane. To do so, click on a rectangle, then hold down the mouse button (you can use either mouse button) while you drag the rectangle to its new location. Modernization Workbench redraws the event lines automatically. You can also drag a line or its segments in a similar way: where ever you place a line, it will always connect the same two rectangles. In addition, you can resize a rectangle as you like by dragging any of its sides in the same manner.

Note: For editing a diagram, you can click the **Edit diagram** button on the Diagram pane's toolbar. As a result, the Diagram Editor window will be displayed.

If you move the items and then decide that you want Modernization Workbench to realign the diagram, select **Diagram:Auto place** in the menu.

You can also zoom your diagram by selecting a desired scale with the help of the zoom slider on the Diagram pane's toolbar or by clicking the **Normal Zoom** 1:1 button.

Working with User Interface Tool

> ***Note***
>
> These actions are available only for COBOL projects.

To proceed with editing your window flow:

1 Click on the plus sign (+) next to a BMS map (for example, GSS1003 — see Figure 3-5) in the hierarchy to see the COBOL programs that receive the map.

For this application, there is a COBOL program named GSS. Note the name "via GSS1003I". That means this COBOL program receives this map in an area that has 01-level in the COBOL working storage called GSS1003I.

2 Click a COBOL program to display its source code. The selected COBOL program in Figure 3-5 on page 3-11 is "GSS via GSS1003I," which means the COBOL program GSS receives this map in an area that has 01-level in the COBOL working storage called GSS1003I. The cursor is positioned at the RECEIVE statement in the code, which shows you where the COBOL program received the BMS map in the code.

3 *(does not apply to AS400)* Right-click on the COBOL program to display the pop-up menu and select **View INTO Record.** This repositions the cursor in the source code to the area where the COBOL program receives data (Figure 3-6 on page 3-12).

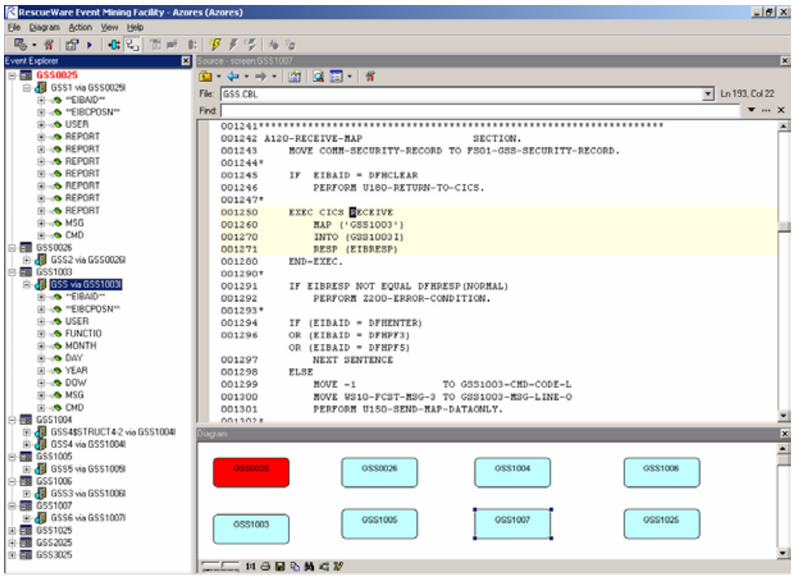
Note: The RECEIVE statement is in the Procedure division. The data is in the working storage.

4 Click the plus sign (+) next to a COBOL program in the hierarchy to display the screen fields.

5 Click on a screen name in the hierarchy. This positions the cursor and highlights the field in the emulated screen (Figure 3-7 on page 3-12).

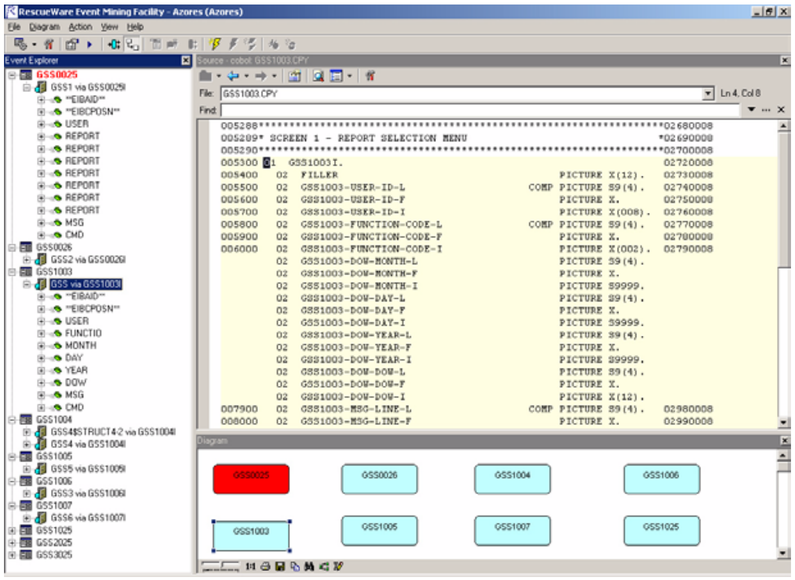
6 Click on a COBOL field in the hierarchy to display the associated source code (Figure 3-8 on page 3-13).

Figure 3-5 *COBOL Program and RECEIVE Statement*



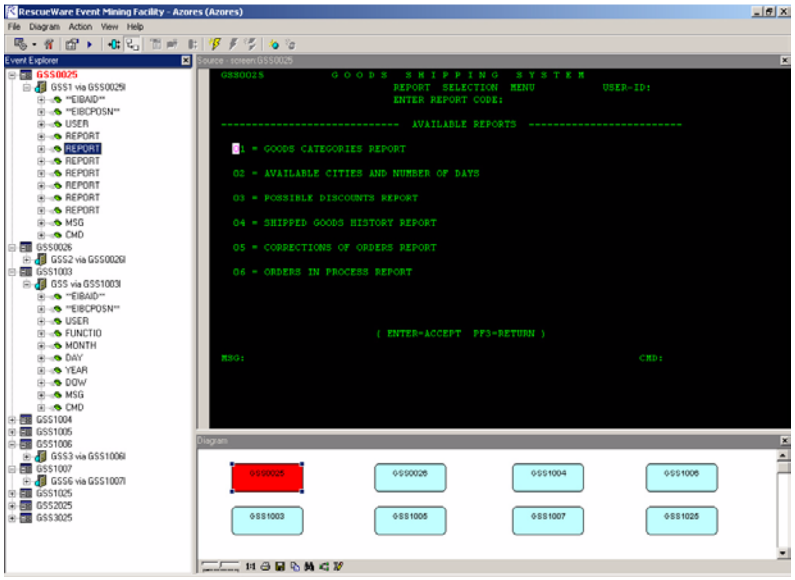
```
001241*****************************************************************
001242 A120-RECEIVE-MAP                          SECTION.
001243     MOVE COMM-SECURITY-RECORD TO FS01-GSS-SECURITY-RECORD.
001244*
001245     IF  EIBAID = DFHCLEAR
001246         PERFORM U180-RETURN-TO-CICS.
001247*
001250     EXEC CICS RECEIVE
001260         MAP ('GSS1003')
001270         INTO (GSS1003I)
001271         RESP (EIBRESP)
001280     END-EXEC.
001290*
001291     IF EIBRESP NOT EQUAL DFHRESP(NORMAL)
001292         PERFORM Z200-ERROR-CONDITION.
001293*
001294     IF (EIBAID = DFHENTER)
001296     OR (EIBAID = DFHPF3)
           OR (EIBAID = DFHPF5)
001297         NEXT SENTENCE
001298     ELSE
001299         MOVE -1               TO GSS1003-CMD-CODE-L
001300         MOVE WS10-FCST-MSG-3 TO GSS1003-MSG-LINE-O
001301         PERFORM U150-SEND-MAP-DATAONLY.
```

Figure 3-6 *COBOL Program and Area where Data Is Received*



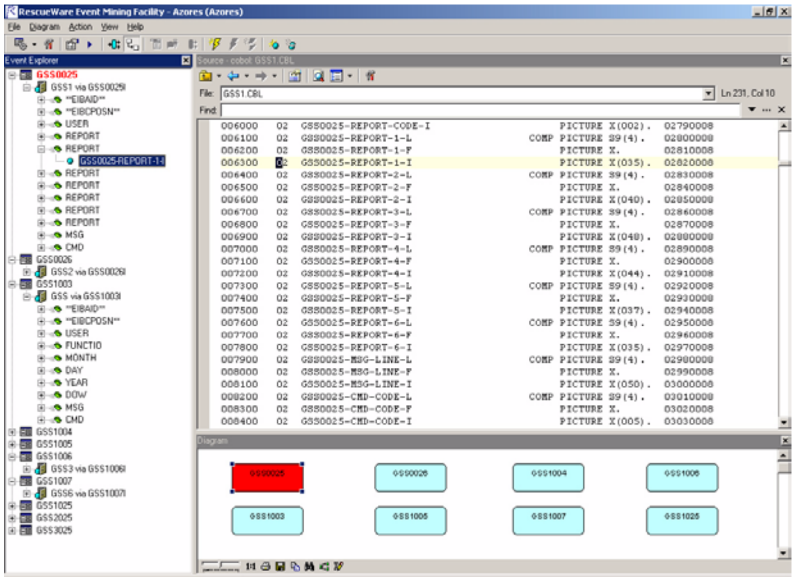
```
005288**********************************************************************02680008
005289* SCREEN 1 - REPORT SELECTION MENU                                   *02690008
005290**********************************************************************02700008
005300 01  GSS1003I.                                                        02720008
005400    02  FILLER                                PICTURE X(12).          02730008
005500    02  GSS1003-USER-ID-L                COMP PICTURE S9(4).          02740008
005600    02  GSS1003-USER-ID-F                     PICTURE X.              02750008
005700    02  GSS1003-USER-ID-I                     PICTURE X(008).         02760008
005800    02  GSS1003-FUNCTION-CODE-L          COMP PICTURE S9(4).          02770008
005900    02  GSS1003-FUNCTION-CODE-F               PICTURE X.              02780008
006000    02  GSS1003-FUNCTION-CODE-I               PICTURE X(002).         02790008
          02  GSS1003-DOW-MONTH-L                   PICTURE S9(4).
          02  GSS1003-DOW-MONTH-F                   PICTURE X.
          02  GSS1003-DOW-MONTH-I                   PICTURE S9999.
          02  GSS1003-DOW-DAY-L                     PICTURE S9(4).
          02  GSS1003-DOW-DAY-F                     PICTURE X.
          02  GSS1003-DOW-DAY-I                     PICTURE S9999.
          02  GSS1003-DOW-YEAR-L                    PICTURE S9(4).
          02  GSS1003-DOW-YEAR-F                    PICTURE X.
          02  GSS1003-DOW-YEAR-I                    PICTURE S9999.
          02  GSS1003-DOW-DOW-L                     PICTURE S9(4).
          02  GSS1003-DOW-DOW-F                     PICTURE X.
          02  GSS1003-DOW-DOW-I                     PICTURE X(12).
007900    02  GSS1003-MSG-LINE-L               COMP PICTURE S9(4).          02980008
008000    02  GSS1003-MSG-LINE-F                    PICTURE X.              02990008
```

Figure 3-7 *COBOL Field and Associated Location in the 3270 Panel*



```
GSS0025          G O O D S   S H I P P I N G   S Y S T E M
                              REPORT  SELECTION  MENU           USER-ID:
                              ENTER REPORT CODE:

----------------------------  AVAILABLE REPORTS  -------------------------

  01 = GOODS CATEGORIES REPORT

  02 = AVAILABLE CITIES AND NUMBER OF DAYS

  03 = POSSIBLE DISCOUNTS REPORT

  04 = SHIPPED GOODS HISTORY REPORT

  05 = CORRECTIONS OF ORDERS REPORT

  06 = ORDERS IN PROCESS REPORT



                          ( ENTER-ACCEPT  PF3-RETURN )

MSG:                                                               CMD:
```

Figure 3-8 *COBOL Field and Source Code*

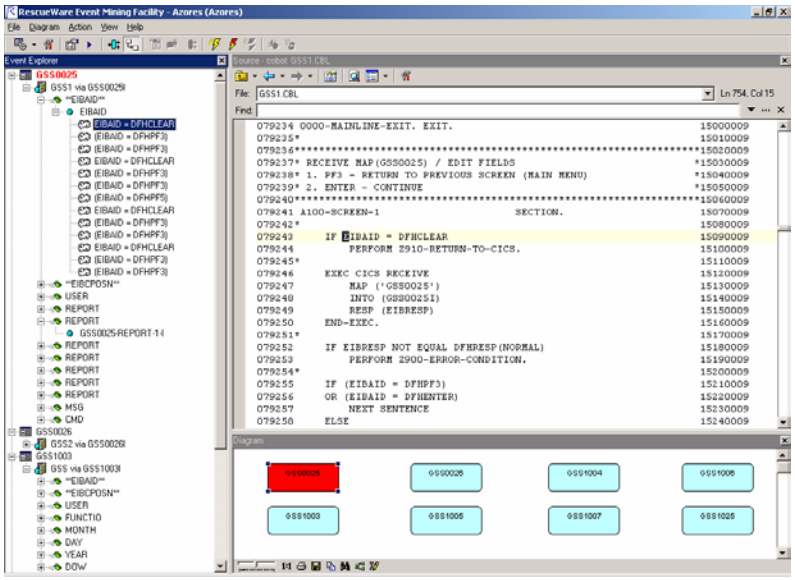


```
006000    02   GSS0025-REPORT-CODE-I                 PICTURE X(002).   02790008
006100    02   GSS0025-REPORT-1-L               COMP PICTURE S9(4).    02800008
006200    02   GSS0025-REPORT-1-F                    PICTURE X.        02810008
006300    02   GSS0025-REPORT-1-I                    PICTURE X(035).   02820008
006400    02   GSS0025-REPORT-2-L               COMP PICTURE S9(4).    02830008
006500    02   GSS0025-REPORT-2-F                    PICTURE X.        02840008
006600    02   GSS0025-REPORT-2-I                    PICTURE X(040).   02850008
006700    02   GSS0025-REPORT-3-L               COMP PICTURE S9(4).    02860008
006800    02   GSS0025-REPORT-3-F                    PICTURE X.        02870008
006900    02   GSS0025-REPORT-3-I                    PICTURE X(048).   02880008
007000    02   GSS0025-REPORT-4-L               COMP PICTURE S9(4).    02890008
007100    02   GSS0025-REPORT-4-F                    PICTURE X.        02900008
007200    02   GSS0025-REPORT-4-I                    PICTURE X(044).   02910008
007300    02   GSS0025-REPORT-5-L               COMP PICTURE S9(4).    02920008
007400    02   GSS0025-REPORT-5-F                    PICTURE X.        02930008
007500    02   GSS0025-REPORT-5-I                    PICTURE X(037).   02940008
007600    02   GSS0025-REPORT-6-L               COMP PICTURE S9(4).    02950008
007700    02   GSS0025-REPORT-6-F                    PICTURE X.        02960008
007800    02   GSS0025-REPORT-6-I                    PICTURE X(035).   02970008
007900    02   GSS0025-MSG-LINE-L               COMP PICTURE S9(4).    02980008
008000    02   GSS0025-MSG-LINE-F                    PICTURE X.        02990008
008100    02   GSS0025-MSG-LINE-I                    PICTURE X(050).   03000008
008200    02   GSS0025-CMD-CODE-L               COMP PICTURE S9(4).    03010008
008300    02   GSS0025-CMD-CODE-F                    PICTURE X.        03020008
008400    02   GSS0025-CMD-CODE-I                    PICTURE X(005).   03030008
```

Some data items have attributes that contain conditions. In general, these conditions are testing for user-initiated events. For convenience, we call these conditions *events*.

7 Click on one of these events to position the cursor in the source where the condition occurs (Figure 3-9).

Figure 3-9 *Event Condition and Source Code where Condition Is Checked*

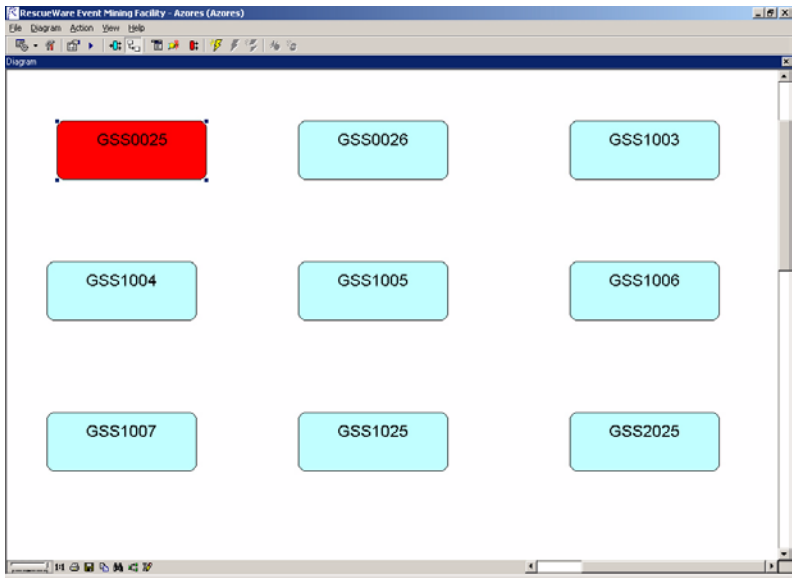


```
079234 0000-MAINLINE-EXIT. EXIT.                                 15000009
079235*                                                          15010009
079236***************************************************************15020009
079237* RECEIVE MAP(GSS0025) / EDIT FIELDS                      *15030009
079238* 1. PF3 - RETURN TO PREVIOUS SCREEN (MAIN MENU)          *15040009
079239* 2. ENTER - CONTINUE                                     *15050009
079240***************************************************************15060009
079241 A100-SCREEN-1                           SECTION.          15070009
079242*                                                          15080009
079243      IF EIBAID = DFHCLEAR                                 15090009
079244          PERFORM 2910-RETURN-TO-CICS.                     15100009
079245*                                                          15110009
079246      EXEC CICS RECEIVE                                    15120009
079247          MAP ('GSS0025')                                  15130009
079248          INTO (GSS0025I)                                  15140009
079249          RESP (EIBRESP)                                   15150009
079250      END-EXEC.                                            15160009
079251*                                                          15170009
079252      IF EIBRESP NOT EQUAL DFHRESP(NORMAL)                 15180009
079253          PERFORM 2900-ERROR-CONDITION.                    15190009
079254*                                                          15200009
079255      IF (EIBAID = DFHPF3)                                 15210009
079256      OR (EIBAID = DFHENTER)                               15220009
079257          NEXT SENTENCE                                    15230009
079258      ELSE                                                 15240009
```

Examples of events are: pressing the **Clear** key, pressing the **F3** key, pressing **Enter**, and so on. The COBOL program has code for these events. Event mining discovers these events and what happens when they occur.

8 To display the window flow for the project, select the **Whole diagram** view of the Diagram pane and then arrange the window flow diagram in that pane as you need. How exactly you can do this, is described in "Diagram pane" on page 3-8. If the diagram is big, then you can switch off the other panes (see page 3-6) and zoom the diagram to fit in the whole User Interface window (Figure 3-10 on page 3-15).

Figure 3-10 *Window Flow Diagram before Editing Flow*


RescueWare Event Mining Facility - Azores (Azores)
File Diagram Action View Help
Diagram
GSS0025
GSS0026
GSS1003
GSS1004
GSS1005
GSS1006
GSS1007
GSS1025
GSS2025

The diagram shows the screens. After you edit the window flow, the new flow (event) relationships in the map are displayed in the diagram, too.

9 To browse the records and data items in the COBOL programs (and included copybooks), select **File:Inspect Data** or click the **View application data** button on the toolbar. The Inspect data window (Figure 3-11 on page 3-16) is displayed.

The Inspect data window can show you the data in two views:

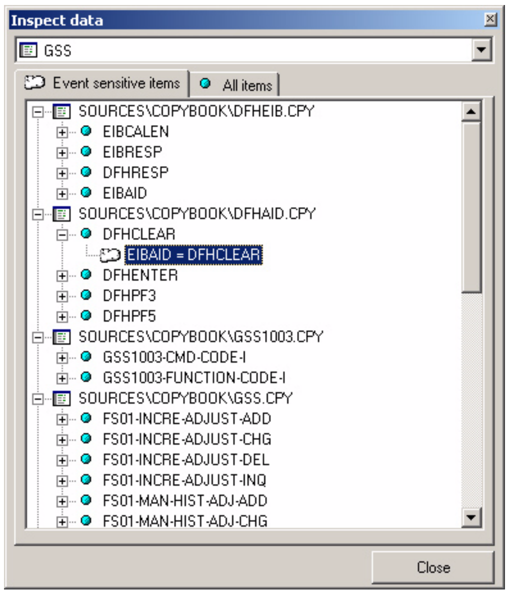
- Event sensitive items

  In this view, the window displays the following three-level hierarchy:

  1. COBOL programs or copybooks
  2. Event-sensitive COBOL fields (variable declarations)
  3. Events

(An event-sensitive field is a node where the program determines what to do with a user request such as an event displaying another window. The COBOL program checks an event-sensitive field to determine what the user wants to do.)

Figure 3-11 *Inspect Data Window*



```
SOURCES\COPYBOOK\DFHEIB.CPY
    EIBCALEN
    EIBRESP
    DFHRESP
    EIBAID
SOURCES\COPYBOOK\DFHAID.CPY
    DFHCLEAR
        EIBAID = DFHCLEAR
    DFHENTER
    DFHPF3
    DFHPF5
SOURCES\COPYBOOK\GSS1003.CPY
    GSS1003-CMD-CODE-I
    GSS1003-FUNCTION-CODE-I
SOURCES\COPYBOOK\GSS.CPY
    FS01-INCRE-ADJUST-ADD
    FS01-INCRE-ADJUST-CHG
    FS01-INCRE-ADJUST-DEL
    FS01-INCRE-ADJUST-INQ
    FS01-MAN-HIST-ADJ-ADD
    FS01-MAN-HIST-ADJ-CHG
```

- All items

  In this view, the window displays a hierarchy, whose upper level lists COBOL programs and copybooks. The branches descending from each upper-level object represent the source COBOL data structure within the object, with all the levels defined for this structure.

The Inspect data window works in tandem with the Source pane of the User Interface window (even though the latter is disabled while the dialog box is on the screen). A selection made in the Inspect data window's hierarchy shows you the related part of the code in the Source pane. The selector is positioned on the beginning of the program or copybook, on the variable declaration of the corresponding level, or on the event, depending on what you selected. To be able to scroll the code in the Source pane or move/resize the User Interface window, exit the Inspect data window.

To browse your data in the Inspect data window:

1. Select the name of a COBOL program in the drop-down list.
2. Depending on the desired view, make sure to have selected one of the two provided tabs, named after the two views available:
   - Event sensitive items
   - All items
3. In the tree, click any boxes with the plus sign (+) to expand the branch(es) you are interested in.

   To collapse any branch, click the corresponding box with the minus sign (–).
4. Select any item to view the related code in the Source pane. Repeat this for all items which you want to see in the code.
5. After browsing the data, click **Close** to exit the Inspect data window.

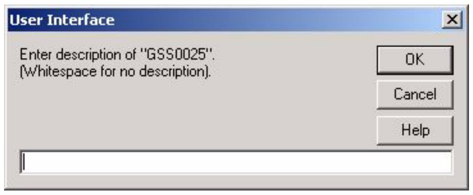
Naming the Screens

Before you begin working with the window flow, give meaningful names to all of the screens in your application. Otherwise, it is difficult to determine the connections among windows.

To name the maps, do the following:

1. *(optional)* We recommend that you base the name of your map on the title of the window.
2. Right-click on the map name in the Event Explorer pane to display the pop-up menu and select **Describe map...**, or select the same-name item in the Action menu, or click the **Describe map** button on the toolbar. A dialog box (Figure 3-12) is displayed.

Figure 3-12 *Pasting or Typing New Name*


User Interface

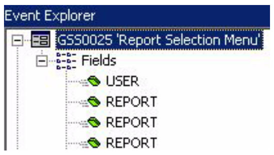
Enter description of "GSS0025".
(Whitespace for no description).

3 Type a description. You can also use the standard Edit pop-up menu, opened by a right-click.

You can change the pasted text. For example, panel titles are often in all uppercase letters, so you can retype the text to make it mixed case.

4 Click **OK**. The description appears in the first column next to the map (Figure 3-13).

Figure 3-13 *Description Added to a Map*


Event Explorer

GSS0025 'Report Selection Menu'

Fields

USER

REPORT

REPORT

REPORT

5 Repeat this procedure to name all the maps.

Setting the Startup Map

By default, the startup map during emulation is the current map or the first map in the list of maps. To specify which map to use as the startup map:

1 Select a map in the hierarchy on the left of the window.

2 Right-click and select **Declare as startup map** from the pop-up menu, or select the same-name item in the Action menu, or click the **Declare as startup** button on the toolbar.

The name of the new startup map in the Event Explorer hierarchy, as well as the corresponding window flow object in the Diagram pane, turn red.

Editing Events

Modernization Workbench found events during event mining. An event is a place in the code where the program tests a field related to the screen (an IF statement). Based on the test, the program branches one way or another. If the branch causes a different window to be displayed, it is a screen event. For all screen events, you add the links to other windows in the flow.

At the interface level, events are actions that cause a new window to be displayed. Pressing a key such as **F3** or **Enter**, clicking a button, or selecting a menu item are examples of events. When you edit events, you add controls to the window, such as buttons, and you specify which window is displayed (for example, pressing **F3** opens window *X*). These links between windows are displayed in the window flow diagram.

To edit an event:

1 Select an event from the hierarchy.

2 Right-click and select **Edit event** from the pop-up menu, or select the same item in the Action menu, or click the same button on the toolbar. The Event Editor window (Figure 3-14 on page 3-20) is displayed.

The condition for the event appears in the Condition field, and the default name of your New Item in the Caption field. This name also appears selected at the end of the list in the Menu pane, which is on the window's bottom.

In Figure 3-14 on page 3-20, the event occurs on the 3270 panel when the user types `IA`.

3 The fields in this window determine the user actions that display a new window in the new interface you are building. Use the upper fields of the window to specify any of the following controls for the event:

- Menu item (**Caption** text field)

  In this field, type the name (caption) for your new menu item which should activate the event. By default the field contains the standard name “New Item”. As you edit the item selected in the Menu pane, its display changes accordingly.

- Condition (**Condition** text field)

  Edit here the programmatic condition that triggers the event. By default, the original condition for this event is provided.

- Button (**Button** text field)

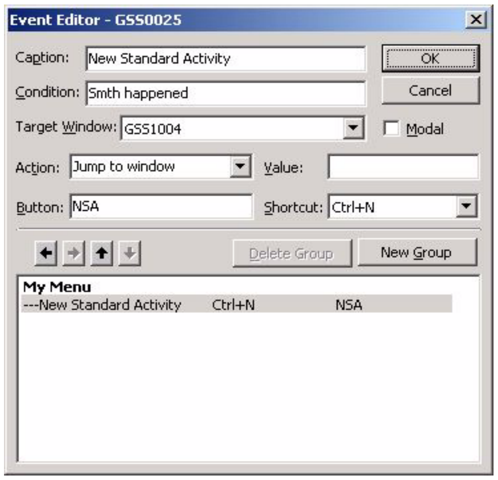
  In the field, type the text to appear on your new button that activates the event.

- Shortcut (**Shortcut** drop-down list)

  In the drop-down list, select the shortcut to trigger this event from the keyboard. By default, “None” (no shortcut) is selected.

In Figure 3-14, the control is a button identified as “New Standard Activity.”

Figure 3-14 *The Event Editor Window*


Event Editor - GSS0025

Caption: New Standard Activity
Condition: Smth happened
Target Window: GSS1004
Modal: ☐
Action: Jump to window
Value:
Button: NSA
Shortcut: Ctrl+N

My Menu
---New Standard Activity Ctrl+N NSA

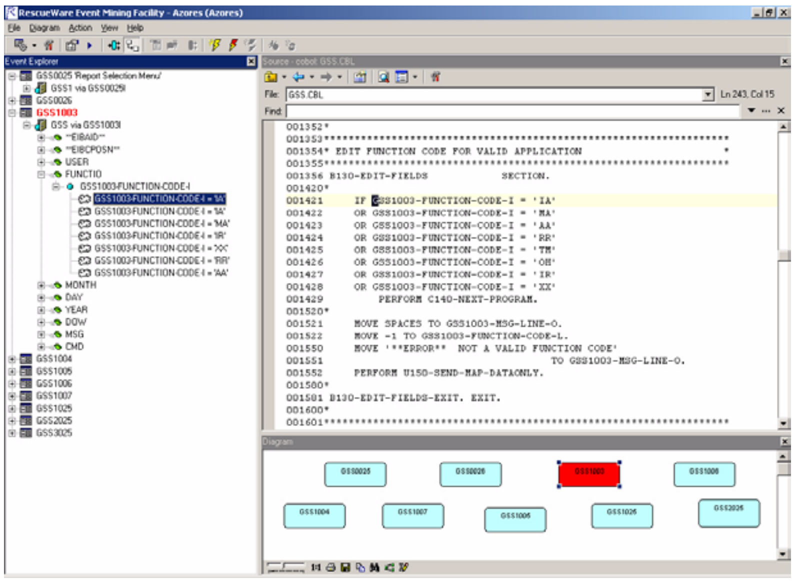
4 In the **Target Window** drop-down list, select the window that is displayed when the event occurs. In Figure 3-14, when the user presses the **NSA** button, the window named GSS1004 is displayed.

In addition to the windows from your legacy application, Modernization Workbench adds (on top of the drop-down list) the following choices for all applications:

- ***END*** exits the entire program,
- ***UNLOAD*** closes the current window and returns to the previous window,
- ***VOID*** is an event that may trigger some internal process, but does not lead to another window, and
- ***HELP*** displays your online help, if there is any.

To determine which window should be displayed, analyze the source code. In particular, look for SEND MAP statements in the COBOL program (Figure 3-15).

Figure 3-15 *Analyzing the Code to Find the Next Window*


RescueWare Event Mining Facility - Azores (Azores)
File Diagram Action View Help
Event Explorer
GSS0025 'Report Selection Menu'
GSS1 via GSS0025I
GSS0026
GSS1003
GSS via GSS1003I
EIBAID
EIBCPOSN
USER
FUNCTIO
GSS1003-FUNCTION-CODE-I
GSS1003-FUNCTION-CODE-I = 'IA'
GSS1003-FUNCTION-CODE-I = 'IA'
GSS1003-FUNCTION-CODE-I = 'MA'
GSS1003-FUNCTION-CODE-I = 'IR'
GSS1003-FUNCTION-CODE-I = 'XX'
GSS1003-FUNCTION-CODE-I = 'RR'
GSS1003-FUNCTION-CODE-I = 'AA'
MONTH
DAY
YEAR
DOW
MSG
CMD
GSS1004
GSS1005
GSS1006
GSS1007
GSS1025
GSS2025
GSS3025
Source - cobol GSS.CBL
File: GSS.CBL
Ln 243, Col 15
Find:
Diagram
GSS0025
GSS0026
GSS1003
GSS1006
GSS1004
GSS1007
GSS1005
GSS1025
GSS2025

```
001352*
001353*****************************************************************
001354* EDIT FUNCTION CODE FOR VALID APPLICATION                       *
001355*****************************************************************
001356 B130-EDIT-FIELDS              SECTION.
001420*
001421      IF GSS1003-FUNCTION-CODE-I = 'IA'
001422      OR GSS1003-FUNCTION-CODE-I = 'MA'
001423      OR GSS1003-FUNCTION-CODE-I = 'AA'
001424      OR GSS1003-FUNCTION-CODE-I = 'RR'
001425      OR GSS1003-FUNCTION-CODE-I = 'TM'
001426      OR GSS1003-FUNCTION-CODE-I = 'OH'
001427      OR GSS1003-FUNCTION-CODE-I = 'IR'
001428      OR GSS1003-FUNCTION-CODE-I = 'XX'
001429          PERFORM C140-NEXT-PROGRAM.
001520*
001521      MOVE SPACES TO GSS1003-MSG-LINE-O.
001522      MOVE -1 TO GSS1003-FUNCTION-CODE-L.
001550      MOVE '**ERROR**  NOT A VALID FUNCTION CODE'
001551                                 TO GSS1003-MSG-LINE-O.
001552      PERFORM U150-SEND-MAP-DATAONLY.
001580*
001581 B130-EDIT-FIELDS-EXIT. EXIT.
001600*
001601*****************************************************************
```

5 Check or uncheck the **Modal** check box to make the displayed window modal or modeless. Modeless windows allow the user to select other windows in the interface while the window is displayed. Use modal windows for error messages and other windows that require

a user action, such as selecting **Cancel**, before the user can select any other windows.

6 In the **Action** field, specify the associated action to be called in response to this event.

7 You can specify a parent menu for your new item. The Menu pane, which is actually a two-level hierarchy (parent menus and independent items on the top level and menu items on the second one), helps you do this in a convenient way.

To create a new menu for this purpose, click the **New Group** button. The default name of your **New Group** (menu) appears in the **Caption** field. It also appears selected above the previously selected item (so that no other menu separates them) in the Menu pane. (The menu names are always bold type.) You can also use an existing menu if you have specified one before for any previous items. To specify a menu as a parent of the item, make sure to position the latter under the former (so that no other menu could separate them) in the Menu pane and to set the item as a second-level one by shifting its name in the pane to the right. You can do all this by selecting corresponding names in the Menu pane and moving them there with the help of the "up"/ "down" and "right" arrow buttons.

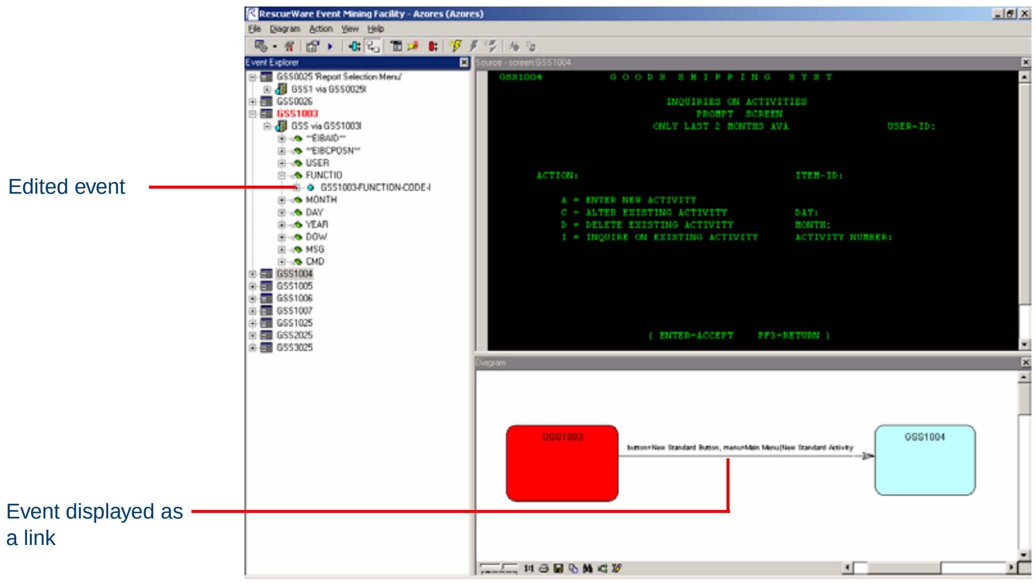
You can always delete any menu by selecting its name in the Menu pane and clicking the **Delete Group** button. You can as well restore any item to the top level by shifting it back to the left with the help of the "left" button (by this you make the item independent of any menu).

Note: If you delete a menu, you should restore to the top level all its former items. If you restore to the top level of items of some menu, you should delete this menu. If you create a menu, you should include at least one second-level item. If you move any item to the second level, you should include it in at least one menu. It is not possible to leave "orphaned" second-level items and "childless" menus in the pane. This is incorrect, and you will receive an error message after you click **OK**.

8 Click **OK** to save your changes (or **Cancel** to discard them) and exit the Event Editor window.

The edited event is displayed under the COBOL field (data item) in the hierarchy on the left and as a link leaving the window flow diagram at the lower right portion of the window (Figure 3-16 on page 3-23).

Figure 3-16 *Edited Event*

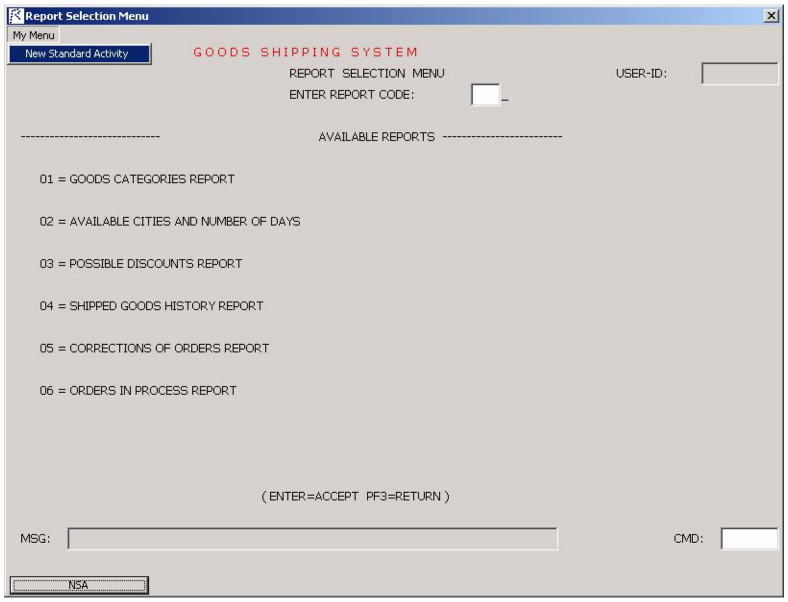


```
GSS1004          G O O D S   S H I P P I N G   S Y S T

                          INQUIRIES ON ACTIVITIES
                               PROMPT  SCREEN
                        ONLY LAST 2 MONTHS AVA                USER-ID:



     ACTION:                                     ITEM-ID:

         A = ENTER NEW ACTIVITY
         C = ALTER EXISTING ACTIVITY             DAY:
         D = DELETE EXISTING ACTIVITY            MONTH:
         I = INQUIRE ON EXISTING ACTIVITY        ACTIVITY NUMBER:




                          ( ENTER=ACCEPT    PF3=RETURN )
```

9 To test the new button on the emulated Visual Basic window, select **Action: Emulate** in the menu or click the **Emulate window flow** button on the toolbar (Figure 3-17 on page 3-24).

10 To see how the emulated windows flow, click the new button. For example, clicking the **New Standard Activity** button in Figure 3-17 on page 3-24 displays window GSS1004 (Figure 3-18 on page 3-24).

Figure 3-17 *New Button on Emulated Window*



GOODS SHIPPING SYSTEM
REPORT SELECTION MENU
USER-ID:
ENTER REPORT CODE:

----------------------------- AVAILABLE REPORTS -------------------------

01 = GOODS CATEGORIES REPORT

02 = AVAILABLE CITIES AND NUMBER OF DAYS

03 = POSSIBLE DISCOUNTS REPORT

04 = SHIPPED GOODS HISTORY REPORT

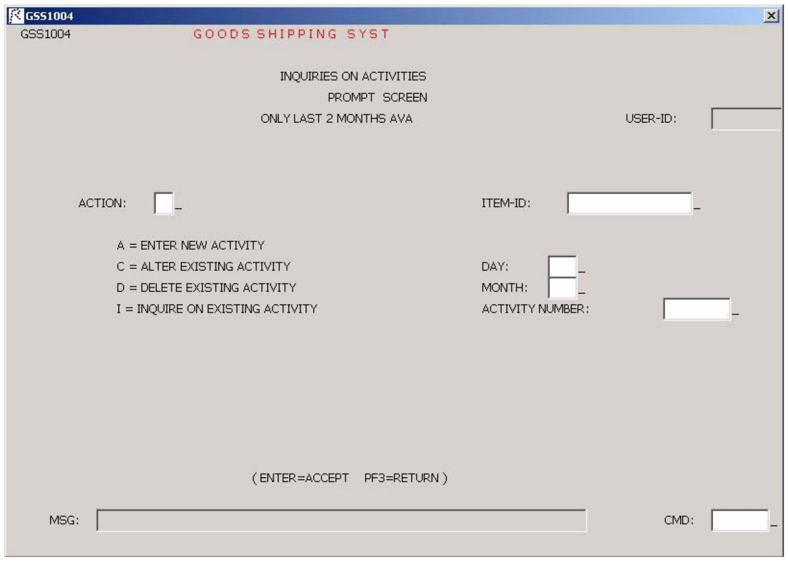
05 = CORRECTIONS OF ORDERS REPORT

06 = ORDERS IN PROCESS REPORT

(ENTER=ACCEPT PF3=RETURN)

MSG:
CMD:

Figure 3-18 *Viewing the Next Window in the Flow*



GSS1004
GOODS SHIPPING SYST

INQUIRIES ON ACTIVITIES
PROMPT SCREEN
ONLY LAST 2 MONTHS AVA
USER-ID:

ACTION:
ITEM-ID:

A = ENTER NEW ACTIVITY
C = ALTER EXISTING ACTIVITY
D = DELETE EXISTING ACTIVITY
I = INQUIRE ON EXISTING ACTIVITY

DAY:
MONTH:
ACTIVITY NUMBER:

(ENTER=ACCEPT PF3=RETURN)

MSG:
CMD:

Editing an Event to Flow Back to the First Window

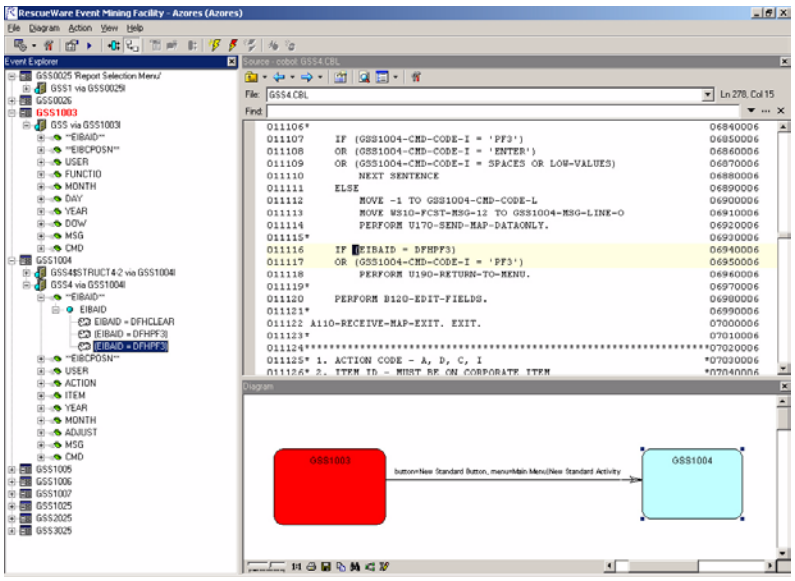
Often, when you edit an event for one window to link to a second window, you want to link back to the first window. For example, you might want the interface to return to the first window after the user exits the second window.

To edit the window flow to return to the first window:

1 In the hierarchy, find the map for the second window. For example, selecting the **New Standard Activity** button on window GSS1003 (Figure 3-17 on page 3-24) causes window GSS1004 to be displayed. Find GSS1004 in the hierarchy.

2 To add a link back to window GSS1003, select the event under GSS1004 that displays the previous window. In Figure 3-19, that event occurs when the user presses the **PF3** key in the 3270 interface.

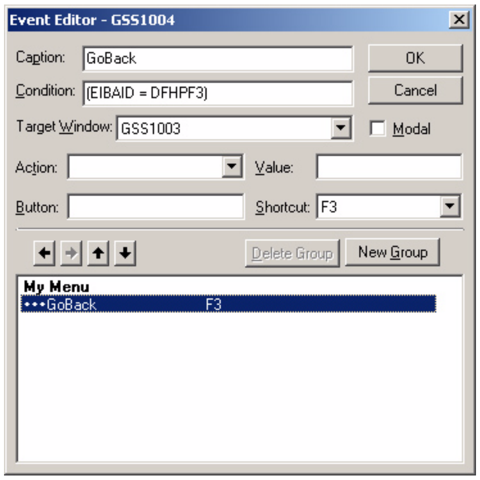
Figure 3-19 *Select the Event that Displays the Previous Window*



```
011106*                                                             06840006
011107     IF (GSS1004-CMD-CODE-I = 'PF3')                          06850006
011108     OR (GSS1004-CMD-CODE-I = 'ENTER')                        06860006
011109     OR (GSS1004-CMD-CODE-I = SPACES OR LOW-VALUES)           06870006
011110         NEXT SENTENCE                                        06880006
011111     ELSE                                                     06890006
011112         MOVE -1 TO GSS1004-CMD-CODE-L                        06900006
011113         MOVE WS10-FCST-MSG-12 TO GSS1004-MSG-LINE-O          06910006
011114         PERFORM U170-SEND-MAP-DATAONLY.                      06920006
011115*                                                             06930006
011116     IF (EIBAID = DFHPF3)                                     06940006
011117     OR (GSS1004-CMD-CODE-I = 'PF3')                          06950006
011118         PERFORM U190-RETURN-TO-MENU.                         06960006
011119*                                                             06970006
011120     PERFORM B120-EDIT-FIELDS.                                06980006
011121*                                                             06990006
011122 A110-RECEIVE-MAP-EXIT. EXIT.                                 07000006
011123*                                                             07010006
011124****************************************************************07020006
011125* 1. ACTION CODE - A, D, C, I                                 *07030006
011126* 2. ITEM ID - MUST BE ON CORPORATE ITEM                      *07040006
```

3 Right-click the event and select **Edit Event** from the pop-up menu. The event from the COBOL program (pressing the **PF3** key) appears in the **Condition** field (Figure 3-20 on page 3-26).

Figure 3-20 *Adding a Shortcut and a Menu Item for an Event*



Event Editor - GSS1004

Caption: GoBack

Condition: (EIBAID = DFHPF3)

Target Window: GSS1003

Modal: ☐

Action:

Value:

Button:

Shortcut: F3

My Menu

•••GoBack F3

4 In the upper fields, choose the options that you want in the new interface. For example, in Figure 3-20, the new window is displayed by pressing **F3**.

5 From the **Target Window** drop-down list, select the name of the window to be displayed when the event occurs. In Figure 3-20, the GSS1003 window will be displayed.

6 In the **Action** field, specify the associated action to be called in response to this event.

7 Click **OK**.

The diagram pane shows that window GSS1004 now has a flow going back to GSS1003 (Figure 3-21).

Figure 3-21 *Bidirectional Window Flow*

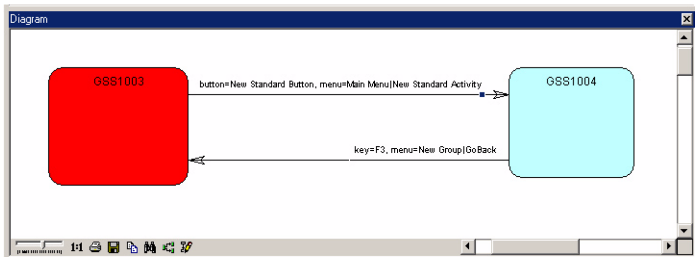
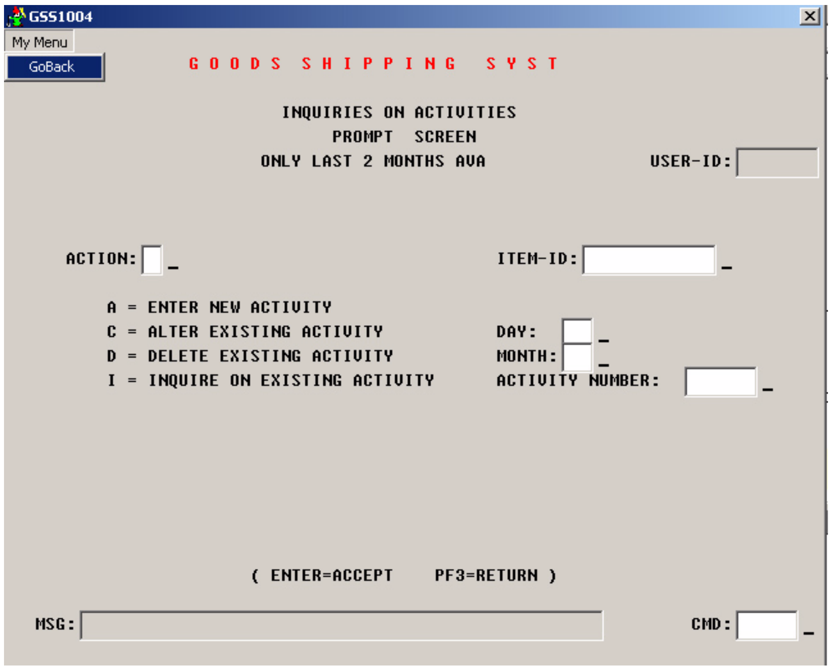


Diagram
GSS1003
button=New Standard Button, menu=Main Menu|New Standard Activity
GSS1004
key=F3, menu=New Group|GoBack

8 Close the Window Flow Diagram window.

9 Select **Action:Emulate** in the menu or click the **Emulate window flow** button on the toolbar. The emulated Visual Basic window (Figure 3-22) is displayed.

Figure 3-22 *Emulated Menu and Function Key on GSS1004 Window*



```
                    G O O D S   S H I P P I N G   S Y S T

                         INQUIRIES ON ACTIVITIES
                              PROMPT  SCREEN
                       ONLY LAST 2 MONTHS AVA                    USER-ID:


   ACTION: _                                     ITEM-ID:            _

       A = ENTER NEW ACTIVITY
       C = ALTER EXISTING ACTIVITY               DAY:   _
       D = DELETE EXISTING ACTIVITY              MONTH: _
       I = INQUIRE ON EXISTING ACTIVITY          ACTIVITY NUMBER:      _



                     ( ENTER=ACCEPT    PF3=RETURN )

  MSG:                                                           CMD:    _
```

10 To test the flow, select **My Menu:GoBack** or select the emulated **F3** key. The GSS1003 window is displayed.

Adding an Event-Sensitive Data Item

An event-sensitive field is a node where the program determines what to do with a user request such as an event displaying another window. The COBOL program checks an event-sensitive field to determine what the user wants to do. For example, suppose your application has a panel where users enter customer information. There are two fields:

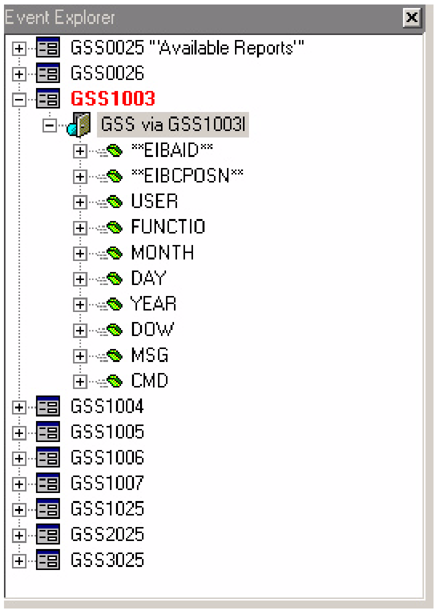
- In first field, users enter the customer name.
- In the second, users enter A to add a customer or B to delete a customer.

The field where users enter the customer name is not an event-sensitive field. Users enter some data there when adding or deleting a name. Only the field that adds or deletes the customer is event-sensitive.

In some cases, Modernization Workbench does not find all the events for a field because it found no conditional test (IF statement). If you know that a field is event-sensitive or you want to define an event for a field, evaluate the COBOL code to determine which data items you need to add. The data items you add must have associated events that result in a different panel being displayed.

There can be event-sensitive fields that are hidden. A list of fields in a hierarchy is shown in Figure 3-23.

Figure 3-23 *Fields in the GSS COBOL Program*



```
GSS0025 "Available Reports'"
GSS0026
GSS1003
  GSS via GSS1003I
    **EIBAID**
    **EIBCPOSN**
    USER
    FUNCTIO
    MONTH
    DAY
    YEAR
    DOW
    MSG
    CMD
GSS1004
GSS1005
GSS1006
GSS1007
GSS1025
GSS2025
GSS3025
```

Notice that most of the fields in Figure 3-23 have names like USER, FUNCTION, and so on. There are also two special fields that are enclosed in quotation marks:

- **EIBAID** receives the value of the key that the user presses when the map is displayed on the screen.
- **EIBCPOSN** receives coordinates of the cursor on screen when the user presses a key while the BMS map is displayed on the screen. This field is an event-sensitive field in all BMS maps.

Unlike the other fields, these two fields do not appear in the emulated 3270 panel. They appear only in the COBOL program. However, these fields are very important for the interaction between the map and the program. Modernization Workbench automatically inserts these two fields.

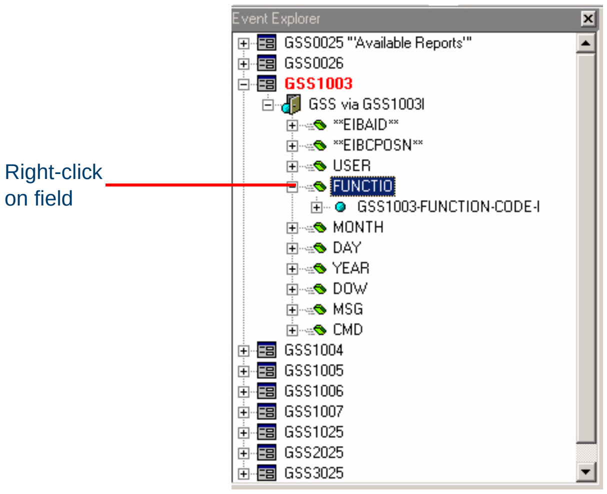
All BMS maps use the EIBAID field. If a plus sign (+) is displayed beside EIBAID, then Modernization Workbench found events that are associated with this field. However, if there is no plus sign, then Modernization Workbench did not find events for this field. That occurs when the program moves EIBAID to a field and then checks that field instead of checking EIBAID directly. So, when Modernization Workbench checks which conditions are tested on “EIBAID,” nothing is discovered. The check is being done on another field. Such fields are, in effect, hidden event-sensitive fields. When this occurs in the COBOL program, EIBAID has no plus sign beside it (no events). You must determine what the events should be and add them to declare that field as event-sensitive. To determine which field the COBOL program checks for the cursor position, look in the source code to see where the program moves “EIBAID.”

To add an event-sensitive data item:

1. Right-click on the field that is missing data items (Figure 3-24 on page 3-31) to display the pop-up menu.
2. Select **Add Event Sensitive Data Item** from the pop-up menu. The Add event sensitive data item window (Figure 3-25 on page 3-31) is displayed.
3. Select the data item you want to add to the field and click the **Select** button.

The new event appears in the hierarchy (Figure 3-26 on page 3-32).

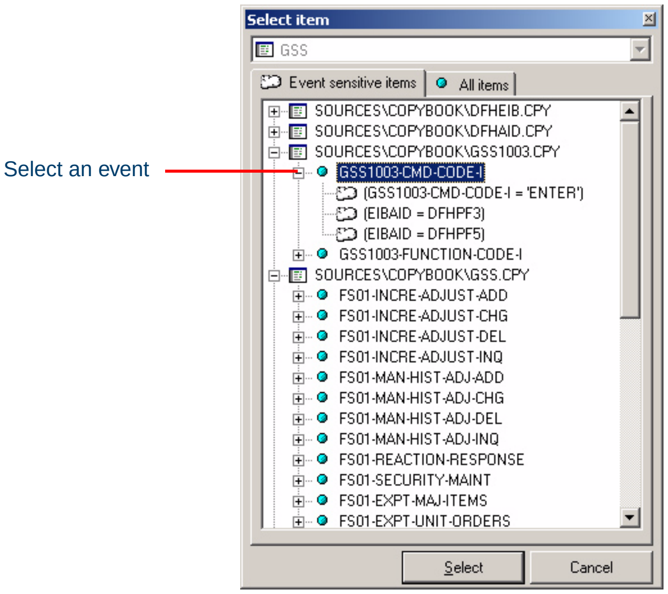
Figure 3-24 *Select a Field*



Event Explorer

```
GSS0025 "Available Reports'"
GSS0026
GSS1003
  GSS via GSS1003I
    **EIBAID**
    **EIBCPOSN**
    USER
    FUNCTIO
      GSS1003-FUNCTION-CODE-I
    MONTH
    DAY
    YEAR
    DOW
    MSG
    CMD
GSS1004
GSS1005
GSS1006
GSS1007
GSS1025
GSS2025
GSS3025
```

Figure 3-25 *Select Event Sensitive Data Item Window*



Select item

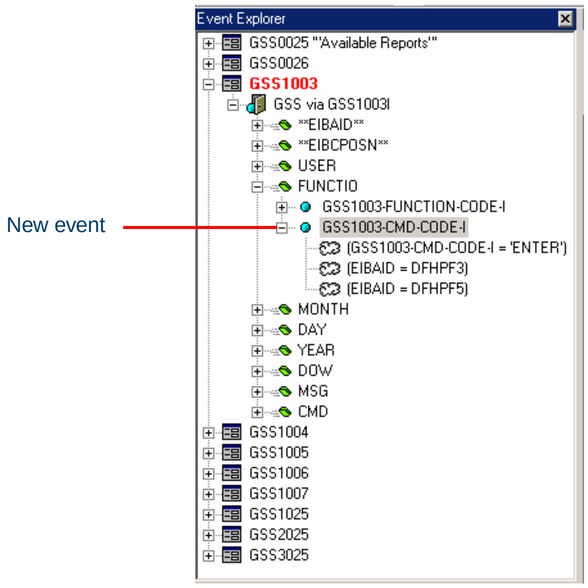
GSS

Event sensitive items | All items

```
SOURCES\COPYBOOK\DFHEIB.CPY
SOURCES\COPYBOOK\DFHAID.CPY
SOURCES\COPYBOOK\GSS1003.CPY
  GSS1003-CMD-CODE-I
    (GSS1003-CMD-CODE-I = 'ENTER')
    (EIBAID = DFHPF3)
    (EIBAID = DFHPF5)
  GSS1003-FUNCTION-CODE-I
SOURCES\COPYBOOK\GSS.CPY
  FS01-INCRE-ADJUST-ADD
  FS01-INCRE-ADJUST-CHG
  FS01-INCRE-ADJUST-DEL
  FS01-INCRE-ADJUST-INQ
  FS01-MAN-HIST-ADJ-ADD
  FS01-MAN-HIST-ADJ-CHG
  FS01-MAN-HIST-ADJ-DEL
  FS01-MAN-HIST-ADJ-INQ
  FS01-REACTION-RESPONSE
  FS01-SECURITY-MAINT
  FS01-EXPT-MAJ-ITEMS
  FS01-EXPT-UNIT-ORDERS
```

Select | Cancel

Figure 3-26 *Event-Sensitive Data Item Added to Field*



```
Event Explorer
+ GSS0025 '"Available Reports'"
+ GSS0026
- GSS1003
  - GSS via GSS1003I
    + **EIBAID**
    + **EIBCPOSN**
    + USER
    - FUNCTIO
      + GSS1003-FUNCTION-CODE-I
      - GSS1003-CMD-CODE-I
          (GSS1003-CMD-CODE-I = 'ENTER')
          (EIBAID = DFHPF3)
          (EIBAID = DFHPF5)
    + MONTH
    + DAY
    + YEAR
    + DOW
    + MSG
    + CMD
+ GSS1004
+ GSS1005
+ GSS1006
+ GSS1007
+ GSS1025
+ GSS2025
+ GSS3025
```

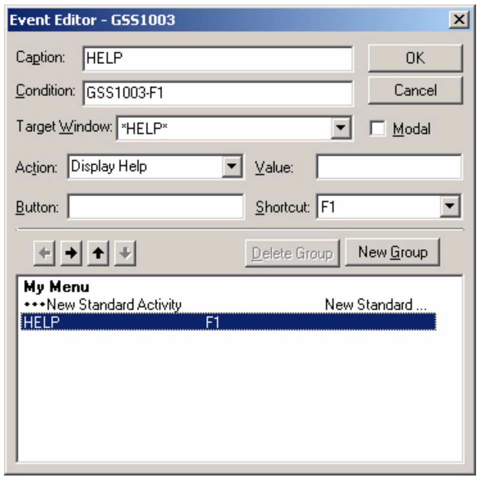
Adding a New Event to a Window

In some cases, you may want to add events to your new interface that did not exist in the legacy system. For example, you may want to add new events for your windows to display online help windows.

To add a new event:

1. Select a map from the hierarchy.
2. Right-click to display the pop-up menu and select **Add Event**. The Add event window (Figure 3-27 on page 3-33) is displayed.
3. Type a description of the event in the **Condition** field. This description is displayed in the hierarchy at the left of the User Interface window after clicking **OK**.
4. Select the key, button, and menu controls you want to use. For example, to display a help window when the user presses **F1**, select the **Shortcut** drop-down list (Figure 3-27 on page 3-33) and choose **F1**.

Figure 3-27 *Adding an Event to Display a Help Window*


Event Editor - GSS1003

Caption: HELP

Condition: GSS1003-F1

Target Window: *HELP*

Modal: ☐

Action: Display Help

Value:

Button:

Shortcut: F1

```
My Menu
•••New Standard Activity                New Standard ...
HELP                F1
```

5 Select a window from the **Target Window** drop-down list. In addition to the windows from your legacy application, Modernization Workbench adds the following choices for all applications:

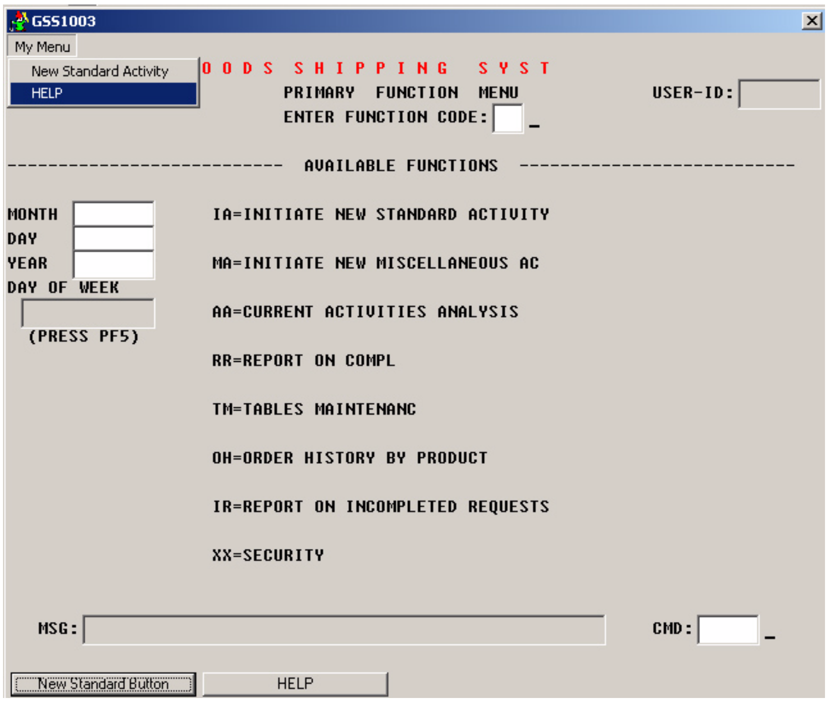
- ***END*** exits the entire program.
- ***UNLOAD*** closes the current window and returns to the previous window.
- ***VOID*** is an event that may trigger some internal process, but does not lead to another window.
- ***HELP*** displays online help.

6 In the **Action** field, specify the associated action to be called in response to the new event.

7 Click **OK**. The new event is displayed in the hierarchy.

8 Select **File:Emulate** to see the changes to the interface resulting from adding a new event. An emulated **F1** key is shown in Figure 3-28 on page 3-34.

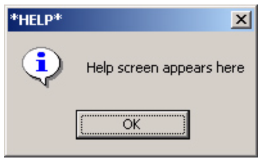
Figure 3-28 *Emulated F1 Key on Window*



```
O O D S   S H I P P I N G   S Y S T
          PRIMARY  FUNCTION  MENU                    USER-ID:
          ENTER FUNCTION CODE:    _

--------------------------  AVAILABLE FUNCTIONS  --------------------------

MONTH                IA=INITIATE NEW STANDARD ACTIVITY
DAY
YEAR                 MA=INITIATE NEW MISCELLANEOUS AC
DAY OF WEEK
                     AA=CURRENT ACTIVITIES ANALYSIS
  (PRESS PF5)
                     RR=REPORT ON COMPL

                     TM=TABLES MAINTENANC

                     OH=ORDER HISTORY BY PRODUCT

                     IR=REPORT ON INCOMPLETED REQUESTS

                     XX=SECURITY

   MSG:                                                  CMD:      _
```

If you select the emulated **F1** key, a window representing online help (Figure 3-29) is displayed.

Figure 3-29 *Emulated Help Window*



Help screen appears here

Deleting a Data Item

If you want to delete a data item from the window hierarchy (for example, if you added an incorrect event-sensitive data item), right-click on the data item you want to delete. Select **Delete dataitem** from the pop-up menu that is displayed or click the **Delete data item** button on the toolbar.

Generating Target Code

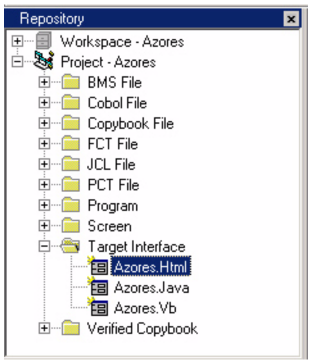
Use the specification of the window flow diagram to generate a *target interface* which is a group of generated files for a single kind of output, such as HTML. You can generate the following types of output:

- HTML
- Java

To generate the target code Select **File:Generate:Generate <code type>** from the menu or click the triangle next to the **Generate** button and select the code type from the drop down menu.

Modernization Workbench generates the new code for your application and adds a new object in the current project's Target Interface folder. The object's name is the name of the current project with a suffix indicating the target code type (Figure 3-30).

Figure 3-30 *Target Interface Model*



To view the code, right-click on a target interface and select **View** from the pop-up menu or double-click. This displays a window that lists all the generated source. To view the source file, double-click on it or right-click and select **Open** from the pop-up menu.

Note: To view a transformed window, a third party application will have to be installed on the target machine.

Generating Java Servlets for HTML

To generate Java servlets for screens in a window flow, which is being converted to HTML, execute the **Generate HTML** action with the Generate Servlet option set to **on**, as described above. This results in generating for each BMS\AS400 screen a pair of files:

- *<Screen name>.html* (target HTML screen);
- *<Screen name>.java* (servlet file).

Note: Some actions that you have added to the window flow when editing it (like calling other screens from a given one) may not be recognized by the corresponding servlet, if you do not make proper changes in the HTML code.

The servlets can then be installed on your server and provide user's access to target HTML screens, showing them via Internet. This is all that the generated servlets can do originally: they are templates for real servlet application and one has to add an additional servlet code to meet any particular needs.

To enable a servlet, it is necessary to add to the *servlet.properties* file the following line of code:

```
'servlet.<ServletName>.initArgs=htmlLocation=<directory
for html file>'.
```

The directory pathname should end with slash (‘\’).

Note: All slash (‘\’) symbols in the directory pathname should be doubled (‘\\’) to avoid confusing them with the line continuation symbol.

An example of Modernization Workbench-generated servlet code for the screen named NCPSRTSM is as follows:

```
// Generated on Wed Dec 19 17:21:22 2001

// File: GSS1003.java
```

```
import javax.servlet.http.*;
import javax.servlet.*;
import com.relativity.rescuent.servlet.*;
import java.io.*;

public class GSS1003 extends FileServlet {
   public void service
      (HttpServletRequest req, HttpServletResponse res)
   throws IOException
   {sendHtml(res,null);
}
```

Exporting Source Files

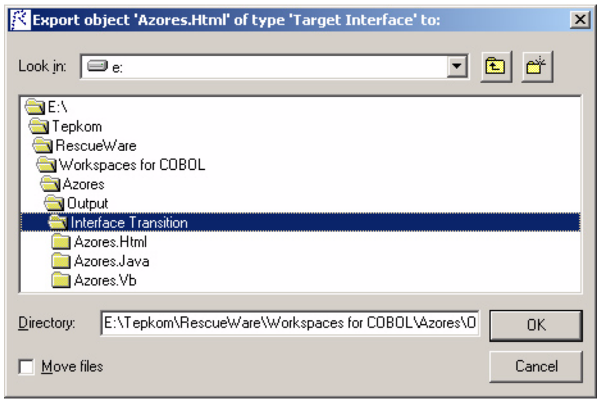
After you generate the code, you export the files to an external directory.

To export the target interface model:

1. Right-click the target interface you want to export.
2. Select **Export** from the pop-up menu or click the **Export** button on the **Target View** toolbar. The **Export object** window (Figure 3-31 on page 3-38) is displayed.
3. Select a directory.

   You can either copy or move the files to the directory. By default, Modernization Workbench copies the files. If you want to move the files (copy them and then delete them from your project's directory), select the **Move** check box.
4. Click **OK**

Figure 3-31 *Selecting Directory for Export*



Export object 'Azores.Html' of type 'Target Interface' to:

Look in: e:

E:\
Tepkom
RescueWare
Workspaces for COBOL
Azores
Output
Interface Transition
Azores.Html
Azores.Java
Azores.Vb

Directory: E:\Tepkom\RescueWare\Workspaces for COBOL\Azores\O

Move files

Data Access Object Support (CRUD Runtime Facilities)

A

The Modernization Workbench provides Data Access Object (DAO) support for Java.

DATAX Modernization Workbench: ActiveX Interface Description

Interface ITable: IDispatch

Properties:

- **DSN**: the ODBC data source name
- **TableName**: the name of table in the data source
- **User**: user name
- **Password**: user password
- **DBMS:** target DBMS (Data Base Management System)
- **Debug** – to enable/disable debug output in console

- **LastError** – string representation of last error

Methods:

- **Init:** initialization
- **GetEmptyRow**—creates new object to handle table record, returns its IRow interface pointer
- **Insert:** insert record
- **Update:** update records
- **Delete:** delete records
- **Select:** single class selection
- **OpenCursor:** open cursor for the table
- **OpenCursorEx:** open cursor for the record set

Interface IRow : IDispatch

Properties:

- **SchemaPtr:** interface to the external database schema ISchema interface (Modernization Workbench ODBC Schema component)
- **TableName**: table which owns the row
- **FieldsCount:** count of fields in the row
- **EmptyFlag:** flag to specify empty rows

Methods:

- **Init:** initialization
- **Clear:** clear contents
- **Copy:** copy the contents from another row
- **Clone:** create a copy of the row
- **SetFieldValue:** set value for the specified field
- **GetFieldValue:** get value from the specified field

Interface IScrollCursor: IDispatch

Properties:

- There are no properties for this interface.

Methods:

- **Scroll:** fetch the next row in the record set
- **GetCurrentRow:** get the row at which the cursor is positioned
- **DeleteCurrent:** delete the records in the table, as does COBOL DELETE WHERE CURRENT OF
- **DeleteCurrent:** update the records in the table, as does COBOL UPDATE WHERE CURRENT OF

Java Support for Data Access Objects

Java support for Data Access Objects (DAO) is implemented by the Java CRUD Runtime library.The library consists of several classes designed to simplify the work with such data structures as database tables, cursors, and others. All classes are implemented in the Java package by their DAO names.

The following classes are in the DAO package:

- CRUDCell
- CRUDColumn
- CRUDRow
- CRUDTable
- CRUDCursor
- CRUDTypes
- CRUDException
- CRUDConstants
- CRUDMessages

In the target classes generated for a particular SQL table, you would not see all of the methods. There are inherited classes, specific for the table. A more detailed description of the base DAO classes is given below.

CRUDCell Class

The Cell class corresponds to the abstract notion of a table cell and has the suitable members and methods.

Members:

There are not the public members for this class.The only means to get/set the data is by a method call. For this reason the data members are described with their get/set methods.

- **String name:** the name of the table column for which the Cell is created
- **String getName():** the method to get String name’s value
- **int type:** the SQL type of the corresponding column, in terms of the constants described in the CRUDTypes class
- **int getType():** the method to get type
- **boolean empty:** the ‘empty flag’ shows if there is any value in the inner buffer of the Cell. Modernization Workbench checks whether the Cell instance contains any value.
- **boolean isEmpty():** allows Modernization Workbench to look at the ‘empty’ from outside
- **boolean nullable:** the ‘nullable flag’ shows if the corresponding table column can contain the NULL value. This flag is being checked, for example, when you want to submit an INSERT statement with values for some columns unspecified.
- **boolean isNullable():** allows Modernization Workbench to look at the ‘nullable’ from outside
- **boolean blank:** the ‘blank flag’ shows if the cell contains the NULL value. Because of the NULL value being a specific value, this flag is different from the ‘empty flag’.

- **boolean isNull():** allows Modernization Workbench to look at the ‘blank’ from outside

Constructors:

- **CRUDCell(String name, int type, boolean nullable):** constructs the cell for the column with name and type transmitted, and the ‘null’ parameter, which shows if the cell allows the NULL value.

Methods:

- **void setString(String val):** method to fill the buffer with a non-null string value
- **String getString()**: method to get the string representation of the buffer contents. Empty string will be return, if buffer is empty.
- **void load(String val, boolean blank)**: method that loads a particular value into the inner Cell buffer. The parameter ‘blank’ allows Modernization Workbench to load the NULL value. Because the Java language does not have the NULL value for all its types, such as the ‘int’ type, Modernization Workbench uses a new parameter to resolve this issue.
- **void clear():** sets the cell contents to the empty state
- **void setNull()**: sets the cell contents to the null state
- **void setInt(int i)**: sets the value from an integer
- **setShort(short s)**: sets the value from an integer
- **void setDouble(double d)**: sets the cell from a Double value
- **void setFloat(float f)**: sets the cell from a Float
- **void setBoolean(boolean b)**: sets the cell from Boolean
- **void setByte(byte b)**: sets the cell value from Byte
- **void setLong(long l)**: sets the cell from Long value
- **int getInt()**: gets the cell value as integer
- **short getShort()**: gets the cell value as short
- **double getDouble()**: gets the cell value as double

- **float getFloat()**: gets the cell value as float
- **boolean getBoolean()**: gets the cell value as a boolean
- **byte getByte()**: gets the cell value as a byte
- **long getLong()**: gets the cell value as a long

Class CRUDTypes

This class serves as the container for the ‘type’ constants necessary for DAO. It was designed in the JDBC style, because the SQL ‘type’ constants are declared in a similar way.

Members:

Public (final static) members in the class implementation are constituted by the following constants:

- int BIT
- int TINYINT
- int SMALLINT
- int INTEGER
- int BIGINT
- int FLOAT
- int REAL
- int DOUBLE
- int NUMERIC
- int DECIMAL
- int CHAR
- int VARCHAR
- int LONGVARCHAR
- int DATE
- int TIME

- int TIMESTAMP
- int BINARY
- int VARBINARY
- int LONGVARBINARY
- int NULL
- int OTHER
- int DATE_TYPE
- int TIME_TYPE
- int TIMESTAMP_TYPE

Note: It is presumed that all the SQL types available are covered by this types set. This class does not have instances.

Class CRUDConstants

This class serves as the container for others constants necessary for DAO.

Class CRUDMessages

This class serves as the container for the CRUD messages.

Class CRUDException

This class extends the standard SQLException class. This class is necessary because all SQL exceptions are also exceptions according to CRUD, but not all exceptions, which appear as such from the CRUD point of view, are SQL exceptions. More exactly, the CRUD routines have a higher level than SQL, CRUD being a superstructure for JDBC. Because users would usually require the high-level error messages, CRUDException class is designed to work at the CRUD level.

Members:

This class has no data members.

Constructors:

- CRUDException(String reason)

These constructor are not smart and just recall the corresponding constructors from the super class.

Methods:

- **void handler(CRUDException ex)**: a default handler that reads the whole exceptions chain and prints the detailed messages from each exception. Use this handler method to debug your application or as a default handler in the ‘catch’ part.

Class CRUDRow

This class represents a real SQL table row. This class hides the buffers for each column that are necessary to store one table row. Therefore, an instance of the CRUDRow class can be used to insert a new row in the table, to get one row from the table, and so on. An instance of this class must be tuned for a particular table, therefore the most frequent way to create it is by using the ‘createRow’ method from the CRUDTable class.

Members:

There are not the public data members for this class.

Constructors:

- **CRUDRow()**: initializes the inner members by nulls. To create an instance of the CRUDRow class, execute the CRUDTable method ‘createRow’.

Methods:

- **int getCount() {return cnt;}**: returns the result of the columns count for the row
- **boolean isEmpty()**: shows whether all cells in this row are in the empty state
- **boolean isEmpty(String name)**: shows whether a specified cells in this row are in the empty state

- **void assign(CRUDRow row):** assigns this row to the parameter received
- **boolean isNull(String name) throws CRUDException:** shows if the row cell corresponding to the column with the name specified has the NULL value (in SQL terms)
- **void reset() throws CRUDException**: clears all cells in the row
- **void copy(CRUDRow row) throws CRUDException:** copies this row to the row specified
- **public String getString(String col) throws CRUDException:** gets the string representation of the cell value for the cell associated with the column with the name specified
- **void setString(String col, String val) throws CRUDException:** sets the value for the cell linked to the specified column
- **void setNull(String col) throws CRUDException:** attempts to set the Null value and throws an exception if this can’t be done
- **void setEmpty(String col) throws CRUDException:** attempts to set the specified cell into the empty state and throws an exception if this can’t be done
- **void setByte(String col, byte val) throws CRUDException:** attempts to set byte and throws an exception if the data type does not match its expected format
- **byte getByte(String col) throws CRUDException:** attempts to get byte and throws an exception if the data type does not match its expected format
- **void setInt(String col, int val) throws CRUDException:** attempts to set integer and throws an exception if the data type does not match its expected format
- **int getInt(String col) throws CRUDException:** attempts to get integer and throws an exception if the data type does not match its expected format

- **void setShort(String col, short val) throws CRUDException:** attempts to set short integer and throws an exception if the data type does not match its expected format
- **short getShort(String col) throws CRUDException:** attempts to get short integer and throws an exception if the data type does not match its expected format
- **void setLong(String col, long val) throws CRUDException:** attempts to set long and throws an exception if the data type does not match its expected format
- **short getLong(String col) throws CRUDException:** attempts to get long and throws an exception if the data type does not match its expected format
- **void setDouble(String col, double val) throws CRUDException:** attempts to set double and throws an exception if the data type does not match its expected format
- **short getDouble (String col) throws CRUDException:** attempts to get double and throws an exception if the data type does not match its expected format
- **void setFloat(String col, float val) throws CRUDException:** attempts to set float and throws an exception if the data type does not match its expected format
- **short getFloat (String col) throws CRUDException:** attempts to get float and throws an exception if the data type does not match its expected format
- **void setBoolean(String col, boolean val) throws CRUDException:** attempts to set boolean and throws an exception if the data type does not match its expected format
- **short getBoolean(String col) throws CRUDException:** attempts to get boolean and throws an exception if the data type does not match its expected format
- **CRUDCell getCell(int num) throws CRUDException**: finds the cell with number specified

- **CRUDCell getCell(String name) throws CRUDException**: finds the cell with name specified

Class CRUDColumn

This class stores a real SQL column structure. During the CRUDTable instance initialization, Modernization Workbench discovers all the column types, sizes, scales; so it must have a data structure that can keep the column parameters. This data structure creates a CRUDRow instance. The CRUDColumn class extends the base Java class Object and uses Java growable arrays filled by CRUDColumns.

Members:

There are no public data members for this class.

Constructors:

- **CRUDColumn(String name, int type, boolean nullable)**: creates the CRUDColumn object for the column with the specified name, type and Boolean flag of nullability which shows if the column can contain the NULL value.

Methods:

- **CRUDCell createCell() throws CRUDException**: method to create the CRUDCell instance corresponding to this column

Class CRUDTable

This class represents a real SQL table. It contains all the necessary CRUD (Create, Update, Delete) routines.

Members:

- There are no public data members for this class.

Constructors:

- CRUDTable(String dsn, String table) throws CRUDException
- **CRUDTable(String dsn, String table, String user, String password) throws CRUDException**: creates new CRUDTable variable

for a real table in the registered database with DSN specified. During initialization, a connection with the database is established with the corresponding variable of java.sql. The connection type is stored with private data members of the class. During initialization, Modernization Workbench explores the table structure by runtime JDBC(ODBC) calls and fills its inner structures with search results.

Destructors:

- **void finalize() throws CRUDException:** as a destructor for CRUDTable, it must close the data-base connection and free all captured inner resources (such as open SQL statements)

Methods:

- **done() throws CRUDException:** disconnects from the database and frees the CRUDTable instance from links with any table.
- **String getName():** returns the name of the table associated with this specimen of the CRUD-Table class
- **getCount():** returns the result of the columns count in the associated table
- **CRUDRow createRow() throws CRUDException:** the main method for creating CRUDRow, initialized for a particular table
- **String composeInsertQuery(CRUDRow row) throws SQLException**: constructs SQL query for INSERT method of CRUD
- **String composeDeleteQuery (CRUDRow row) throws SQLException**: constructs SQL query for DELETE method of CRUD
- **String composeUpdateQuery (CRUDRow row) throws CRUDException**: constructs SQL query for UPDATE method of CRUD
- **String composeSelectQuery (CRUDRow row) throws CRUDException**: constructs SQL query for SELECT method of CRUD
- **void insert(CRUDRow row) throws CRUDException:** inserts new row in the table. The row must be filled with the help of the CRUDRow class. It requires that some of the fields, which are not nullable, should be filled; otherwise an exception (‘Nothing to insert

specified') is thrown because an empty row cannot be inserted in the table. This condition is checked by the driver at execution time.

- **void delete(CRUDRow row) throws CRUDException:** deletes specified row from the table. It requires that some of the fields should be filled; otherwise an exception ('No row to delete') is thrown. This method considers the parameter row as a set of search criteria to delete rows. For example, the parameter row is filled by values (<value1>, <not filled>, <value3>, <value4>), so the "delete" method will delete all rows that match the condition: <column1> = <value1> AND <column3> = <value3> AND <column4> = <value4>. In order to delete one row the user should specify enough values to form a primary key tuple. The case of no value specification is disabled because such a call would delete all table rows. Under such a condition, no search criteria is specified, therefore all rows fit to be deleted. Such a situation is dangerous and should be cautiously handled.
- **void update(CRUDRow row1, CRUDRow row2) throws CRUDException:** updates a row in the table. The first parameter is this row with its old values, and the second one is the row with the values to set. Neither of the two specified rows can be empty, otherwise an exception ('No row to update specified' / 'No values to update with specified') is thrown.
- **CRUDRow select(CRUDRow row) throws CRUDException:** a singleton select (i.e., searching for a record by primary key) method. It returns one row or null as the result of a selection (search). The parameter row is used as a filter. The rows with the same values as in the parameter are selected. To ignore some columns during the selection, set empty values for them in the parameter row. If more than one row has been selected (found), then only the first of the resulting rows is returned. If all fields in the parameter row are empty, then all the table rows participate in the result set, and only the first is returned. If no results are produced, then this method returns NULL.
- **CRUDCursor openCursor(CRUDRow row) throws CRUDException:** selection method for opening and returning the cursor. The principle is the same as for the singleton select method immediately above, but here the number of the resulting rows is not limited. Some

of the fields in the parameter row must be filled or all the table rows are returned. Also, you receive the created cursor variable as the result, even if no row has been selected.

Class CRUDCursor

Because some CRUD routines, such as SELECT, can generate a result set with more than one row, Modernization Workbench implements a mechanism to work with results of this kind as the CRUDCursor class. Usually a CRUDCursor instance is created by the CRUDTable.method ‘openCursor’.

Members:

- There are no public data members for this class.

Constructors:

- **CRUDCursor():** creates new cursor and sets ‘rs’ equal to null
- **CRUDCursor(Connection con, ResultSet rs):** initializes private variables ‘rs’ and ‘con’ by the parameter value. It is used in the ‘openCursor’ method of the CRUDTable.

Methods:

- **void assign(CRUDCursor cur):** assigns one cursor to another. This method hides details of cursor class initialization in the generated text.
- **boolean next() throws CRUDException:** the most important method for the CRUDCursor class. This method moves the cursor to the next row in the result set, if possible, or returns FALSE and stays at the current row.
- **void read(CRUDRow row) throws CRUDException:** reads the current cursor row into the parameter. This method is the only way to get data from the cursor. With the next() method, it provides a mechanism for going through the cursor and fetching all the rows needed.
- **void close()**: closes cursor.

Glossary

ADABAS

ADABAS is a Software AG relational DBMS for large, mission-critical applications.

API

API stands for application programming interface, a set of routines, protocols, and tools for building software applications.

applet

See Java applet.

AS/400

The AS/400 is a midrange server designed for small businesses and departments in large enterprises.

BMS

BMS stands for Basic Mapping Support, an interface between application formats and CICS that formats input and output display data.

BSTR

BSTR is a Microsoft format for transferring binary strings.

CDML

CDML stands for Cobol Data Manipulation Language, an extension of the Cobol programming language that enables applications programmers to code special instructions to manipulate data in a DMS database and to compile those instructions for execution.

CICS

CICS stands for Customer Information Control System, a program that allows concurrent processing of transactions from multiple terminals.

Cobol

Cobol stands for Common Business-Oriented Language, a high-level programming language used for business applications.

COM

COM stands for Component Object Model, a software architecture developed by Microsoft to build component-based applications. COM objects are discrete components, each with a unique identity, which expose interfaces that allow applications and other components to access their features.

complexity

An application's complexity is an estimate of how difficult it is to maintain, analyze, transform, and so forth.

component

A component is a self-contained program that can be reused with other programs in modular fashion.

construct

A construct is an item in the parse tree for a source file — a section, statement, condition, variable, or the like. A variable, for example, can be related in the parse tree to any of three other constructs — a declaration, a dataport, or a condition.

copybook

A copybook is a common piece of source code to be copied into many Cobol source programs. Copybooks are functionally equivalent to C and C++ include files.

CORBA

CORBA stands for Common Object Request Broker Architecture, an architecture that enables distributed objects to communicate with one another regardless of the programming language they were written in or the operating system they are running on.

CSD file

CSD stands for CICS System Definition. A CSD file is a VSAM data set containing a resource definition record for every resource defined to CICS.

database schema

A database schema is the structure of a database system, described in a formal language supported by the DBMS. In a relational database, the schema defines the tables, the fields in each table, and the relationships between fields and tables.

dataport

A dataport is an input/output statement or a call to or from another program.

DB/2

DB/2 stands for Database 2, an IBM system for managing relational databases.

DBCS

DBCS stands for double-byte character string, a character set that uses two-byte (16-bit) characters rather than one-byte (8-bit) characters.

DBMS

DBMS stands for database management system, a collection of programs that enable you to store, modify, and extract information from a database.

DDL

DDL stands for Data Description Language (DDL), a language that describes the structure of data in a database.

decision resolution

Decision resolution lets you identify and resolve dynamic calls and other relationships that the parser cannot resolve from static sources.

DMS

DMS stands for Data Management System, a Unisys database management software product that conforms to the CODASYL (network) data model and enables data definition, manipulation, and maintenance in mass storage database files.

DPS

DPS stands for Display Processing System, a Unisys product that enables users to define forms on a terminal.

ECL

ECL stands for Executive Control Language, the operating system language for Unisys OS 2200 systems.

effort

Effort is an estimate of the time it will take to complete a task related to an application, based on weighted values for selected complexity metrics.

EJB

EJB stands for Enterprise JavaBeans, a Java API developed by Sun Microsystems that defines a component architecture for multi-tier client/server systems.

EMF

EMF stands for Enhanced MetaFile, a Windows format for graphic images.

entity

An entity is an object in the repository model for a legacy application. The relationships between entities describe the ways in which the elements of the application interact.

FCT

FCT stands for File Control Table (FCT), a CICS table that contains processing requirements for output data streams received via a remote job entry session from a host system. Compare PCT.

HTML

HTML stands for HyperText Markup Language, the authoring language used to create documents on the World Wide Web.

IDL

IDL stands for Interface Definition Language (IDL), a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

IDMS

IDMS stands for Integrated Database Management System, a Computer Associates database management system for the IBM mainframe and compatible environments.

IMS

IMS stands for Information Management System, an IBM program product that provides transaction management and database management functions for large commercial application systems.

Java

Java is a high-level object-oriented programming language developed by Sun Microsystems.

Java applet

A Java applet is a program that can be sent with a Web page. Java applets perform interactive animations, immediate calculations, and other simple tasks without having to send a user request back to the server.

JavaBeans

JavaBeans is a specification developed by Sun Microsystems that defines how Java objects interact. An object that conforms to this specification is called a JavaBean.

JCL

JCL stands for Job Control Language, a language for identifying a job to OS/390 and for describing the job's requirements.

JDBC

JDBC stands for Java Database Connectivity, a standard for accessing diverse database systems using the Java programming language.

job

A job is the unit of work that a computer operator or a program called a *job scheduler* gives to the operating system. In IBM mainframe operating systems, a job is described with job control language (JCL).

logical component
A logical component is an abstract repository object that gives you access to the source files that comprise a component.

MFS
MFS stands for Message Format Service, a method of processing IMS input and output messages.

Natural
Natural is a programming language developed and marketed by Software AG for the enterprise environment.

object model
An object model is a representation of an application and its encapsulated data.

object-oriented programming
Object-oriented programming organizes programs in terms of objects rather than actions, and data rather than logic.

ODBC
ODBC stands for Open Database Connectivity, a standard for accessing diverse database systems.

orphan
An orphan is an object that does not exist in the reference tree for any startup object. Orphans can be removed from a system without altering its behavior.

parser
The parser defines the object model and parse tree for a legacy application.

parse tree
A parse tree defines the relationships between the constructs that comprise a source file — its sections, paragraphs, statements, conditions, variables, and so forth.

PCT
PCT stands for Program Control Table, a CICS table that defines the transactions that the CICS system can process. Compare FCT.

PL/I

PL/I stands for Programming Language One, a third-generation programming language developed in the early 1960s as an alternative to assembler language, Cobol, and FORTRAN.

profile

Profiles are HTML views into a repository that show all of the analysis you have done on an application. Profiles are convenient ways to share information about legacy applications across your organization.

QSAM

QSAM stands for Queued Sequential Access Method, a type of processing that uses a queue of data records—either input records awaiting processing or output records that have been processed and are ready for transfer to storage or an output device.

relationship

The relationships between entities in the repository model for a legacy application describe the ways in which the elements of the application interact.

relaxed parsing

Relaxed parsing lets you verify a source file despite errors. Ordinarily, the parser stops at a statement when it encounters an error. Relaxed parsing tells the parser to continue to the next statement.

repository

A repository is a database of program objects that comprise the model for an application.

schema

See database schema.

SQL

SQL stands for Structured Query Language, a standard language for relational database operations

system program

A system program is a generic program — a mainframe sort utility, for example — provided by the underlying system and used in unmodified form in the legacy application.

TIP
: TIP stands for Transaction Processing, the Unisys real-time system for processing transactions under Exec control.

transaction
: A transaction is a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity.

VALTAB
: VALTAB stands for Validation Table, which contains the information the system needs to locate, load, and execute transaction programs. See also TIP.

VSAM
: VSAM stands for Virtual Storage Access Method, an IBM program that controls communication and the flow of data in a Systems Network Architecture network.

XML
: XML stands for Extensible Markup Language, a specification for creating common information formats.

Index