

Micro Focus[®]

Modernization Workbench[™]

Preparing Projects



Copyright © 2009 Micro Focus (IP) Ltd. All rights reserved.

Micro Focus (IP) Ltd. has made every effort to ensure that this book is correct and accurate, but reserves the right to make changes without notice at its sole discretion at any time. The software described in this document is supplied under a license and may be used or copied only in accordance with the terms of such license, and in particular any warranty of fitness of Micro Focus software products for any particular purpose is expressly excluded and in no event will Micro Focus be liable for any consequential loss.

Micro Focus, the Micro Focus Logo, Micro Focus Server, Micro Focus Studio, Net Express, Net Express Academic Edition, Net Express Personal Edition, Server Express, Mainframe Express, Animator, Application Server, AppMaster Builder, APS, Data Express, Enterprise Server, Enterprise View, EnterpriseLink, Object COBOL Developer Suite, Revolve, Revolve Enterprise Edition, SOA Express, Unlocking the Value of Legacy, and XDB are trademarks or registered trademarks of Micro Focus (IP) Limited in the United Kingdom, the United States and other countries.

IBM®, CICS® and RACF® are registered trademarks, and IMS™ is a trademark, of International Business Machines Corporation.

Copyrights for third party software used in the product:

- The YGrep Search Engine is Copyright (c) 1992-2004 Yves Roumazielles
- Apache web site (<http://www.microfocus.com/docs/links.asp?mfe=apache>)
- Eclipse (<http://www.microfocus.com/docs/links.asp?nx=eclp>)
- Cyrus SASL license
- Open LDAP license

All other trademarks are the property of their respective owners.

No part of this publication, with the exception of the software product user documentation contained on a CD-ROM, may be copied, photocopied, reproduced, transmitted, transcribed, or reduced to any electronic medium or machine-readable form without prior written consent of Micro Focus (IP) Ltd. Contact your Micro Focus representative if you require access to the modified Apache Software Foundation source files.

Licensees may duplicate the software product user documentation contained on a CD-ROM, but only to the extent necessary to support the users authorized access to the software under the license agreement. Any reproduction of the documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification.

U.S. GOVERNMENT RESTRICTED RIGHTS. It is acknowledged that the Software and the Documentation were developed at private expense, that no part is in the public domain, and that the Software and Documentation are Commercial Computer Software provided with RESTRICTED RIGHTS under Federal Acquisition Regulations and agency supplements to them. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFAR 252.227-7013 et. seq. or subparagraphs (c)(1) and (2) of the Commercial Computer Software Restricted Rights at FAR 52.227-19, as applicable. Contractor is Micro Focus (IP) Ltd, 9420 Key West Avenue, Rockville, Maryland 20850. Rights are reserved under copyright laws of the United States with respect to unpublished portions of the Software.

Contents

Preface

<i>Audience</i>	vii
<i>Organization</i>	viii
<i>Conventions</i>	ix
<i>Related Manuals</i>	ix
<i>Online Help</i>	x

1 Overview

<i>Registering Applications</i>	1-1
<i>Verifying Applications</i>	1-2
<i>Inventorying Applications</i>	1-3
<i>Support for Japanese Language Applications</i>	1-4
<i>What's Next?</i>	1-5

2 Setting Up a Workspace and Projects

<i>Registering Source Files</i>	2-1
<i>Creating New Source Files</i>	2-3
<i>Refreshing Source Files</i>	2-3

- Exporting Source Files from a Workspace* 2-4
- Setting Registration Options* 2-4
- Creating Projects* 2-9
- Sharing Projects* 2-10
- Protecting Projects* 2-10
- Moving or Copying Files in Projects* 2-10
- Including Objects in Projects* 2-12
- Deleting a Workspace, Project, or Object* 2-13
- What's Next?* 2-14

3 Setting Verification Options

- Setting Workspace Verification Options* 3-2
 - Specifying Options on the Legacy Dialects Tab* 3-2
 - Specifying Options on the Settings Tab* 3-6
 - Source Type Settings Tab Options* 3-8
- Setting Project Verification Options* 3-16
 - Specifying the Processing Environment* 3-24
 - Optimizing Verification for Advanced Program Analysis* 3-25
- Identifying System Programs* 3-25
- Specifying Boundary Decisions* 3-27
- What's Next?* 3-29

4 Verifying Files and Performing Post-Verification Tasks

- Verifying Source Files* 4-1
- Viewing Verification Reports* 4-3
 - Generating Verification Reports* 4-3
 - Working with Verification Reports* 4-4
 - Setting Verification Report User Preferences* 4-6
 - Filtering Verification Reports* 4-8
 - Exporting Verification Reports* 4-10
- Viewing Inventory Reports* 4-12
- Viewing Executive Reports* 4-13
 - Generating Executive Reports* 4-13
 - Defining Potential Code Anomalies* 4-16
- Viewing Key Object Relationships* 4-17

<i>Performing Post-Verification Program Analysis</i>	4-18
<i>Enabling IMS Port Analysis</i>	4-20
<i>Mapping Root Programs to PSBs in JCL or System</i>	
<i>Definition Files</i>	4-21
<i>Verification Order for IMS Applications</i>	4-22
<i>Reverifying Files in IMS Applications</i>	4-22
<i>Generating Copybooks</i>	4-23
<i>What's Next?</i>	4-27
5 Identifying Missing or Unneeded Program Elements	
<i>Using Reference Reports</i>	5-1
<i>Generating Reference Reports</i>	5-2
<i>Working with Reference Reports</i>	5-3
<i>Detecting System Programs</i>	5-6
<i>Exporting Reference Reports</i>	5-6
<i>Using the Orphan Analysis Tool</i>	5-6
<i>Generating Orphan Analysis Reports</i>	5-7
<i>Exporting Orphan Analysis Reports</i>	5-12
<i>What's Next?</i>	5-13
6 Resolving Decisions	
<i>Understanding Decisions</i>	6-1
<i>Resolving Decisions Manually</i>	6-2
<i>Restoring Manually Resolved Decisions</i>	6-6
<i>Resolving Decisions Automatically</i>	6-7
<i>Exporting Decision Resolution Reports</i>	6-7
<i>What's Next?</i>	6-7
A Identifying Interfaces for Generic API Analysis	
<i>Identifying Unsupported API Calls to the Parser</i>	A-2
<i>Using the <match> Tag</i>	A-4
<i>Using the <flow> Tag</i>	A-5
<i>Using the <vars> Tag</i>	A-6
<i>Using the <rep> Tag</i>	A-8

<i>Using the <hc> Tag</i>	A-11
<i>Using Expressions</i>	A-12
<i>Basic Usage</i>	A-12
<i>Using a Function Call</i>	A-15
<i>Understanding Enumeration Order</i>	A-16
<i>Understanding Decisions</i>	A-17
<i>Understanding Conditions</i>	A-18
<i>Example</i>	A-20
<i>Support for IMS Aliases</i>	A-21

B Cobol Range Overlaps and Range Jumps Detected in the Executive Report

C Recognized File Extensions

Glossary

Index

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,” provides an overview of the preparation process and reviews basic Modernization Workbench concepts.
- Chapter 2, “Setting Up a Workspace and Projects,” describes how to register applications in the repository, and set up workspaces and projects.
- Chapter 3, “Setting Verification Options,” describes how to set options that determine the legacy dialect the parser recognizes, whether to use staged or relaxed parsing, how to treat system programs, and other verification behavior.
- Chapter 4, “Verifying Files and Performing Post-Verification Tasks,” describes how to verify and reverify source files, and how to perform key post-verification tasks.
- Chapter 5, “Identifying Missing or Unneeded Program Elements,” describes how to use reference reports and orphan analysis to identify missing or unneeded application elements.
- Chapter 6, “Resolving Decisions,” describes how to identify and resolve dynamic calls and other relationships that the parser cannot resolve from static sources in Cobol, PL/I, and Natural programs.
- Appendix A, “Identifying Interfaces for Generic API Analysis,” describes how to enable the generic API analysis feature.
- Appendix B, “Cobol Range Overlaps and Range Jumps Detected in the Executive Report,” describes Cobol range overlaps and range jumps listed in the Executive Report.
- Appendix C, “Recognized File Extensions,” lists the file extensions recognized by the workbench registration process.
- 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.
- *Analyzing Projects* describes how to analyze applications at the project level. This guide describes how to create diagrams of applications, how to perform change analysis across applications, and how to estimate application 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 set up an application glossary and how to extract business rules.
- *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 use Application Architect to extract program components from legacy applications.
- *Batch Refresh Process* describes how to use the Modernization Workbench Batch Refresh Process utility to batch-synchronize workbench sources with sources at their original location.
- *Transforming Applications* describes how to generate legacy application components in modern languages.
- *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



Before you can analyze a legacy application in Modernization Workbench, you need to *prepare* it. Preparing an application consists of loading, or *registering*, the application in the workbench, and then *verifying* that the entire application can be understood by the workbench parser. You use reports and other tools to ensure that your application can be parsed in its entirety.

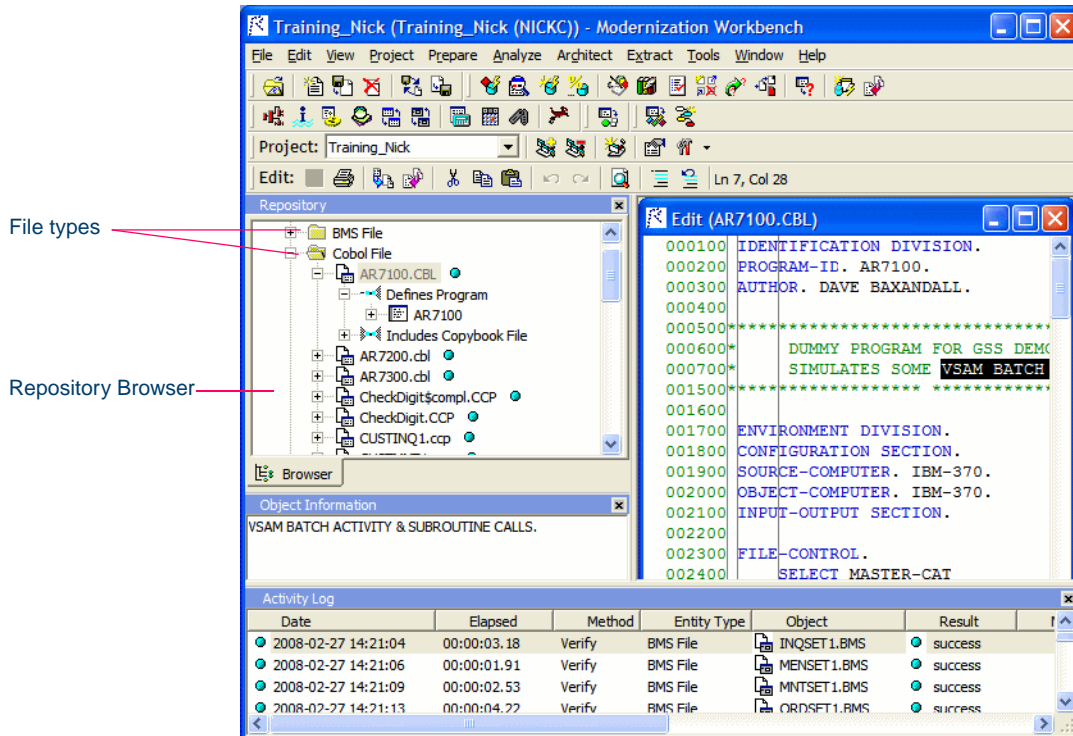
Registering Applications

When you register source files in a workspace, the workbench creates copies of the legacy files on your machine. These are the files you view and edit in the workbench tools. You can restore a file to its original state or update it to its current state as necessary. After registration, the workbench displays the contents of the current workspace in the workbench Repository Browser, organized by file type (Figure 1-1).

Make sure you have assigned appropriate file extensions to legacy files before you register them. You can view and add to the recognized exten-

sions in the Extensions tab of the Workspace Registration options window (Figure 2-1).


Figure 1-1 *Modernization Workbench Repository Browser*



Verifying Applications

Verifying a source file ensures that its contents can be understood by the workbench parser. If you have not verified a file, the workbench displays the file in **bold** type. Since the parser has not generated an object model of the file's contents, only the file itself is displayed.

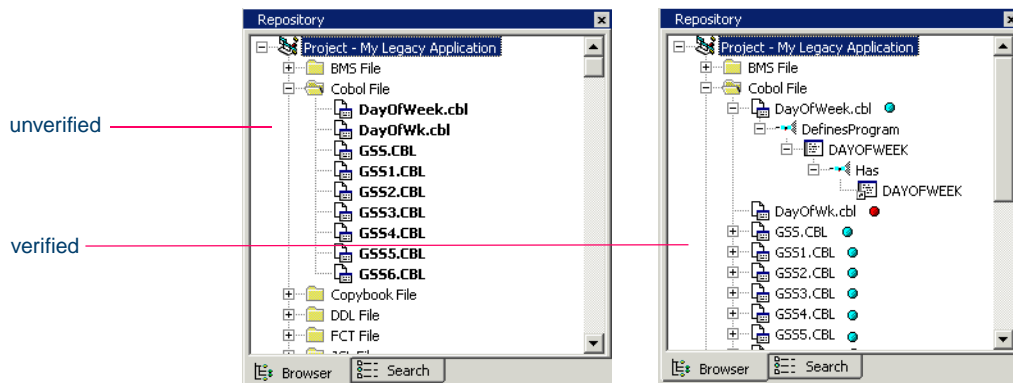
A verified file shows all the objects in the model generated for the file. Verification results are denoted as follows:

- A blue dot  means that the parser has verified the file successfully.

- A red dot ● means that the parser has encountered an error in the file.
- A yellow dot ● means that the parser has encountered an error in the file, but the relaxed parser has verified the file successfully. For more information on the relaxed parser, see [“Enabling Relaxed Parsing”](#) on page 3-14.

Figure 1-2 compares unverified and verified workspaces. Notice in the verified workspace that the parser has built an object model for DayOfWeek.cbl, but not for DayOfWk.cbl, which contained an error. Generally, the parser creates an object model for as much of the file as it understands.

Figure 1-2 *Unverified and Verified Workspaces*



Inventorying Applications

Users often ask why the Modernization Workbench parser encounters errors in working production systems. The reasons usually have to do with the source file delivery mechanism: incorrect versions or copybooks, corruption of special characters because of source code ambiguities, FTP errors, and so forth.

Modernization Workbench offers three related Reference Reports that you can use to ensure that all the parts of an application are available for

analysis. The reports are based on the parser's analysis of references in verified source:

- An Unresolved Report identifies missing program elements.
- An Unreferred Report identifies unreferenced program elements.
- A Cross-Reference Report identifies all application references.

Two other tools let you perform related tasks:

- The Orphan Analysis tool lets you analyze and resolve objects that do not exist in the reference tree for any top-level program object, so-called *orphans*. Orphans can be removed from a system without altering its behavior.
- The Decision Resolution tool identifies and lets you resolve dynamic calls and other relationships that the parser cannot resolve from static sources in Cobol, PL/I, and Natural programs.

Support for Japanese Language Applications

Modernization Workbench provides full support for mainframe-based Cobol or PL/I Japanese language applications. You can register Japanese source files downloaded in text or binary mode:

- Japanese source files downloaded in text, or *workstation*, mode must be in Shift-JIS encoding. If Shift-Out and Shift-In delimiters were replaced with spaces or removed during downloading, Modernization Workbench restores them at registration.

Note: Replacing delimiters with spaces during download generally yields better restoration results than removing them. Preserving delimiters during download is recommended.

- Japanese source files downloaded in binary, or *mainframe*, mode are recoded by Modernization Workbench from EBCDIC to Shift-JIS encoding at registration.

Make sure to set the Windows system and user locales to Japanese before registering source files. For more information on registration options, see [“Specifying Options on the Source Files Tab” on page 2-7](#).

Tip: In all workbench tools that offer search and replace facilities, you can insert Shift-Out and Shift-In delimiters into patterns using Ctrl-Shift-O and Ctrl-Shift-I, respectively. You need only insert the delimiters if you are entering mixed strings.

What's Next?

That's all you need to know before you prepare a Modernization Workbench project. Now let's look at how you register applications in your repository, and set up workspaces and projects.

1-6 Overview
What's Next?

Setting Up a Workspace and Projects



Getting Started in the workbench document set describes how to create and connect to workspaces. This chapter looks at how you register source files in a workspace and how you set up projects.

Registering Source Files

Before you can analyze application source files in Modernization Workbench, you need to load, or *register*, the source files in a workspace. Only a master user can register source files in a multiuser environment.

The workbench assumes that input source files are ASCII files in DOS format. Occasionally, files may be converted incorrectly from other formats to DOS-based ASCII with an extra special character (like “M”) at the end of each line. While Modernization Workbench accepts these files as input, some workbench tools may not work correctly with them. Make sure all source files are in valid ASCII format.

You can register source files in compressed formats (ZIP or RAR), as well as uncompressed formats. Modernization Workbench automatically unpacks the compressed file and registers its contents.

The workbench extracts compressed source files using the command line syntax for archiver versions most widely in use. If you use newer archiver versions, specify the command line syntax in the Archivers tab of the User Preferences window.

Workspace Registration options determine registration behavior. The default values for these options are preset based on your configuration and should be appropriate for most installations. For complete information on the registration options, see [“Setting Registration Options” on page 2-4](#).

Note: Make sure you have assigned appropriate file extensions to legacy files before registering them. You can view and add to the recognized extensions in the Extensions tab of the Workspace Registration options window.

To register source files:

- 1 Set registration options as described in [“Setting Registration Options” on page 2-4](#).
- 2 In the Modernization Workbench Repository pane, create a project for the source files you want to register, or use the default project. To create a project, choose **New Project** in the **Project** menu. The Create Project dialog opens. Enter the name of the new project and click **OK**. The new project is displayed in the Repository pane.
- 3 Select the project in the Repository pane, then drag-and-drop the file or folder for the source files you want to register onto the Repository pane.

You are notified that you have registered the files successfully and are prompted to verify the files. Click **Close**. The Repository pane displays the contents of the new workspace, organized by file type. Verify the files as described in [“Verifying Source Files” on page 4-1](#).

Tip: In the notification dialog, select **Never ask again** if you do not want to be prompted to verify files. In the Environment tab of the User Preferences window, select **Ask user about verification** if you want to be prompted again.

Creating New Source Files

To create a new source file, choose **New** in the **File** menu. A dialog box opens, where you can specify the file name (with extension!) and file type. To create a new source file with the same content as an existing file, select the file and choose **Save As** in the **File** menu. The system automatically registers the created files.

Refreshing Source Files

Use the Modernization Workbench *refresh* feature to update source files to their current state. You can refresh all of the objects in a project or folder, or only selected objects.

The refresh looks for updated legacy source in the original location of the file or, for unresolved source, the location you specified in the Source Files tab of the Workspace Registration options. Once it finds the source, it overwrites the version of the source file maintained by the system. Re-verify the file after the refresh.

Note: If you are licensed to use the batch refresh feature, you can perform the refresh in batch mode. Contact support services for more information.

To refresh source files:

- 1 In the Repository Browser, select the project, folder, or file you want to refresh and choose **Refresh Sources from Disk** in the **File** menu.
- 2 You are prompted to confirm that you want to refresh the selected files. Click **Yes**. The system overwrites the workspace source files.

Exporting Source Files from a Workspace

Export the workspace source for a project or file to a new location by selecting the project or file and clicking **Export Sources** in the **File** menu. The source is copied to the location you specify.

Setting Registration Options

Workspace registration options determine the file extensions the system recognizes, whether it converts source files to workstation encoding, and how it handles trailing enumeration characters and tabulation symbols.

Note: Availability of options depends on your settings in the Modernization Workbench Configuration Manager. For more information, see the installation guide for your product.

Specifying Options on the Extensions Tab

The registration process does not load source files with unknown file extensions. If you already know the extensions that will cause problems, you can add them to the Extensions tab before you register the files, as described in [step 2 on page 2-4](#).

If you don't know the problem file extensions, you can run the registration process anyway. As long as you set the appropriate option ([step 3 on page 2-5](#)), the workbench will generate messages indicating which extensions it did not recognize. You can then add those extensions to the list of recognized extensions and run the registration process again, this time only on the source files that failed to be registered earlier.

To specify options on the Extensions tab:

- 1 In the Modernization Workbench **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the Registration tab, then the Extensions tab (Figure 2-1).
- 2 In the Source Type pane, select the source file type whose extensions you want to view. The extensions for the file type are listed in the Ex-

tensions pane and described in [Appendix C](#). Select each extension you want the system to recognize.

Add an extension by right-clicking in the Extensions pane and choosing **Add** in the pop-up menu. The system displays an empty text field next to a selected check box. Enter the name of the new extension in the field and click outside the field. Make sure to enter the dot (.). Case is irrelevant.

Edit an extension by selecting it and choosing **Edit** in the right-click menu. Delete an extension by selecting it and choosing **Delete** in the right-click menu.

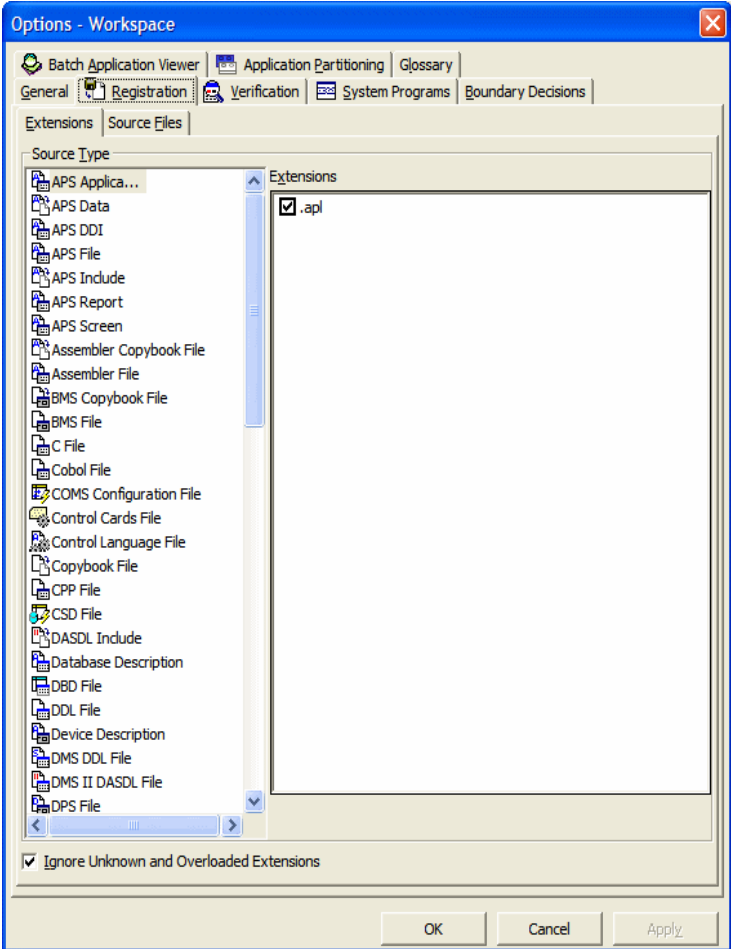
Note: If a source file does not specify an extension when it references an included file, the verification process assumes that the included file has one of the recognized extensions. If multiple included files have the same name but different extensions, the system registers the file with the first extension in the list.

- 3 Select **Ignore Unknown and Overloaded Extensions** if you do not want the registration process to issue warnings about unrecognized and overloaded extensions. An overloaded extension is one assigned to more than one file type.
- 4 For Cobol programs and copybooks, select **Remove Sequence Numbers** if you want the system to replace preceding enumeration characters, or *sequence numbers*, in source lines with blanks. Sequence numbers are removed only from the source file versions maintained by the workbench.
- 5 For C, C++, or PowerBuilder files, select **Preserve Folder Structure** if you want the folder structure for the application to be preserved in the Repository Browser.

Note: You must select this option if your application uses the same program in multiple folders.

2-6 Setting Up a Workspace and Projects
Setting Registration Options

Figure 2-1 *Extensions Tab*



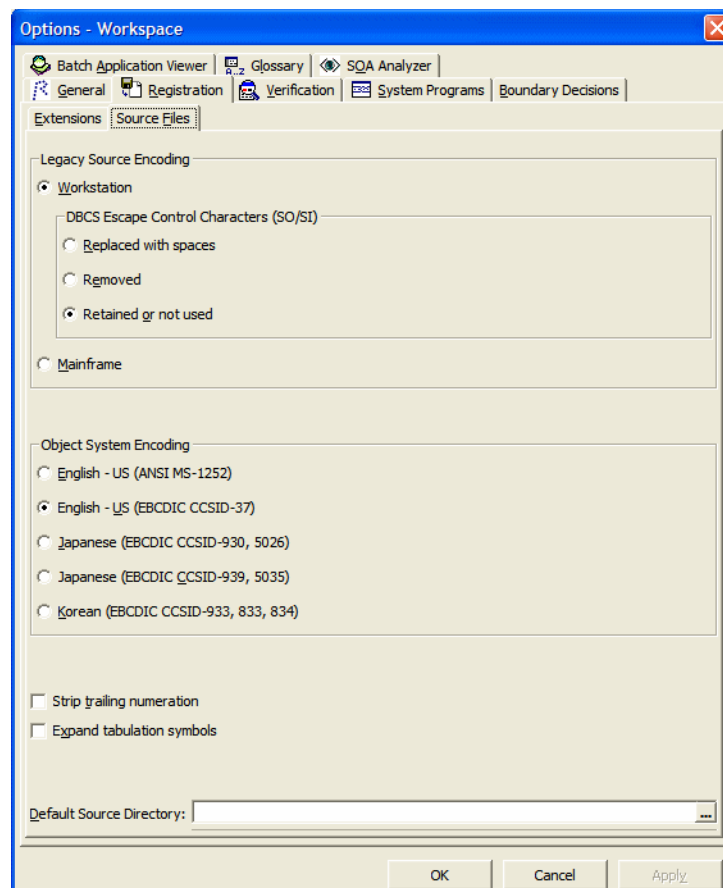
Specifying Options on the Source Files Tab

If the legacy application executes on a mainframe, it's usually best to convert the application source to workstation encoding. If that's not practical, you can have Modernization Workbench convert it for you, as described in [step 2-2 on page 2-7](#).

To specify options on the Source Files tab:

- 1 In the Modernization Workbench **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the Registration tab, then the Source Files tab (Figure 2-2).

Figure 2-2 Source Files Tab



- 2 In the Legacy Source Encoding group box, choose:
 - **Workstation** if the source is workstation-encoded. In the DBCS Escape Control Characters (SO/SI) group box, choose:
 - **Replaced with spaces** if DBCS escape control characters were replaced with spaces.
 - **Removed** if DBCS escape control characters were removed.
 - **Retained or not used** if DBCS escape control characters were left as is or were not used.

Note: Workstation-encoded Japanese source files must use Shift-JIS encoding.

- **Mainframe** if the source is mainframe-encoded. When this option is selected, the registration process automatically converts source files to workstation-encoding. Only the source files maintained by the workbench are converted.
- 3 In the Object System Encoding group box, choose one of the following:
 - **English - US (ANSI MS-1252)** if the original source was U.S. English ANSI-encoded (Unisys 2200 and HP3000 Cobol).
 - **English - US (EBCDIC-CCSID-37)** if the original source was U.S. English EBCDIC-encoded (IBM Cobol).
 - **Japanese (EBCDIC-CCSID-930, 5026)** if the original source was Japanese EBCDIC-encoded, CCSID-930, 5026.
 - **Japanese (EBCDIC-CCSID-939, 5035)** if the original source was Japanese EBCDIC-encoded, CCSID-939, 5035.

During analysis and transformation, hexadecimal literals in Cobol programs and BMS files are translated into character literals according to this setting.

Important: Do not change these settings after source files are registered in a workspace.

- 4 Select the **Strip trailing numeration** check box if you want the system to strip trailing numeration characters (columns 73 through 80) from source lines. Trailing numeration characters are removed only from the source files maintained by the workbench.
- 5 Select the **Expand tabulation symbols** check box if you want the system to replace tabulation symbols with a corresponding number of spaces. Tabulation symbols are replaced only in the source files maintained by the workbench.

Note: You must select this option if you want to view HyperView information for C or C++ programs.

- 6 In the **Default Source Directory** field, enter the root folder on your PC from which the system should refresh unresolved files. You can type over the path in the text box or click the button to the right of the text box to browse for a new location. For more information, see [“Refreshing Source Files” on page 2-3](#).

Creating Projects

When you set up a workspace, the system creates a default project with the same name as the workspace. You can create projects in addition to the default project when you need to analyze subsystems separately or organize source files in more manageable groupings.

To create a project:

- 1 In the Modernization Workbench **Project** menu, choose **New Project**. The Create Project dialog opens (Figure 2-3).

Figure 2-3 *Create Project Dialog*




- 2 Enter the name of the new project and click **OK**. The new project is displayed in the Modernization Workbench window. The project is selected by default.

Sharing Projects

In a multiuser environment, the user who creates a project is referred to as its *owner*. Only the owner can *share* the project with other users.


A shared, or *public*, project is visible to other members of your team. If the project is not *protected*, these team members can delete the project, add source files, or remove source files.

Projects are private by default. Turn on sharing by choosing **Toggle Sharing** in the **Project** menu. Choose **Toggle Sharing** again to turn it off. A  symbol indicates that the project is shared.

Protecting Projects

By default, projects are *unprotected*: any user to whom the project is visible can delete the project, add source files, or remove source files.

The project owner or master user can designate a project as *protected*, in which case *no* user can delete or modify the project, including the project owner or master user: the project is read-only, until the project owner or master user turns protection off.

Turn on protection by selecting the project in the Repository pane and choosing **Toggle Protection** in the **Project** menu. Choose **Toggle Protection** again to turn it off. Look for a symbol like this one  to indicate that a project is protected.

Moving or Copying Files in Projects

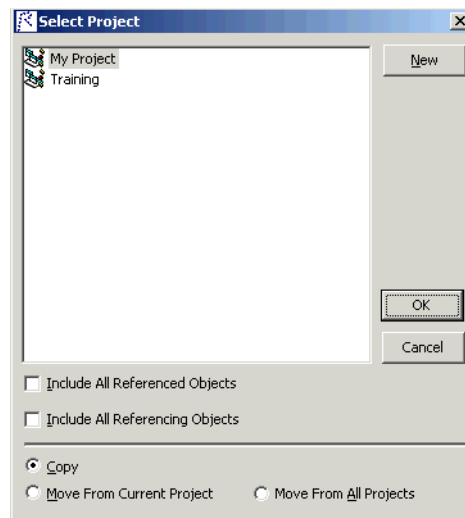
You add source files to a project as described in [“Registering Source Files” on page 2-1](#). You move or copy source files to different projects as described below.

Tip: You can copy a source file or the contents of a project or folder to a different project by selecting it and dragging and dropping the selection, or by using the **Edit** menu choices to cut and paste the selection.

To move or copy objects between projects:

- 1 In the Repository Browser, select the project, folder, or file you want to move or copy. In the **Project** menu, choose **Copy Project Contents** (if you selected a project) or **Include into Project** (if you selected a folder or file). The Select Project window opens (Figure 2-4).

Figure 2-4 *Select Project Window*



- 2 In the Select Project window, select the project you want to move or copy the selection into. Click **New** if you want to create a new project.
- 3 Select the **Include All Referenced Objects** check box if you want to move or copy the objects in the workspace referenced by the selected object (the Cobol copybooks included in a Cobol program file, for example). Select the **Include All Referencing Objects**

2-12 Setting Up a Workspace and Projects *Including Objects in Projects*

check box if you want to move or copy the objects in the workspace that reference the selected object.

Note: This feature is available only for verified files.

- 4 Choose one of the following:
 - **Copy** to copy the selection to the specified project.
 - **Move From Current Project** to move the selection to the specified project.
 - **Move From All Projects** to move the selection from all projects to the selected project.
- 5 Click **OK** to move or copy the selection.

To move source files from a project to the workspace:

- 1 In the Repository Browser, select the project you want to move to the workspace. In the **Project** menu, choose **Empty Project Contents** (if you selected a project) or **Exclude from Project** (if you selected a folder or file). You are prompted to confirm the removal. Click **Yes**.

Including Objects in Projects

After verification, you can include referenced or referencing objects in a project to ensure a closed system. You can include all referencing objects or only “directly referencing” objects: If program A calls program B, and program B calls program C, A is said to directly reference B and indirectly reference C. You can also remove unused support objects.

To include referenced objects in a project:

- 1 To include in a project every object referenced by the objects in the project (including indirectly referenced objects), select the project in the Repository Browser and choose **Include All Referenced Objects** in the **Project** menu.

To include all referencing objects in a project:

- 1 To include in a project every object that references the objects in the project (including indirectly referencing objects), select the project in the Repository Browser and choose **Include All Referencing Objects** in the **Project** menu.

To include directly referencing objects in a project:

- 1 To include in a project every object that directly references the objects in the project, select the project in the Repository Browser and choose **Include Directly Referencing Objects** in the **Project** menu.

To move unused support objects from a project to the workspace:

- 1 To move unused support objects (Cobol copybooks, JCL procedures, PL/I include files, and so forth) from a project to the workspace, select the project in the Repository Browser and choose **Compact Project** in the **Project** menu.

Deleting a Workspace, Project, or Object

Follow the instruction in this section to delete workspaces, projects, or objects.

Deleting a Workspace

Only a master user can delete a workspace in a multiuser environment. Any user can delete a workspace in a single-user environment.

Delete a workspace by choosing **Delete Workspace** in the **File** menu of the workbench Administration tool. A Delete workspace dialog opens, where you can select the workspace you want to delete.

Deleting a Project

You can delete a project or empty it:

- To delete a project from a workspace (without deleting its source files from the workspace), select it and choose either **Delete from Workspace** in the **File** menu or **Delete Project** in the **Project** menu.

2-14 Setting Up a Workspace and Projects *What's Next?*

Note: Only the owner of a project can delete it. For more information, see [“Protecting Projects” on page 2-10](#).

- To leave a project in a workspace but empty out its source files (without deleting the source files from the workspace), select the project and choose **Empty Project Contents** in the **Project** menu.

In all cases, you are prompted to confirm the deletion. Click **Yes**.

Deleting an Object or Folder

Delete an object from a workspace by selecting it and choosing **Delete from Workspace** in the **File** menu.

Delete a folder and all its contents from a workspace by selecting and choosing **Delete Contents from Workspace** in the **File** menu.

In either case, you are prompted to confirm the deletion. Click **Yes**.

What's Next?

Now that you have learned how to set up workspaces and projects and register legacy source files, you are ready to start verifying source files. The next chapter describes the verification options.

Setting Verification Options



Verification options determine the legacy dialect the parser recognizes, whether to use staged or relaxed parsing, how to treat system programs, and other verification behavior. *Workspace verification* options control verification behavior for the current workspace. Project verification options *Project verification* options control verification behavior for the current project.

It's a good idea to become familiar with the options before verifying applications. While your workbench configuration will determine appropriate defaults (see the topic box below), not all the defaults will be suited to your needs. The staged verification feature especially may help you save time by performing only as much of the verification as you need.

How Configuration Manager Settings Affect the Options

Both workspace and project verification options may be affected by your settings in the Modernization Workbench Configuration Manager, as described in the installation guide for your product. If you do not configure the workbench for Natural, for example, you will not see options related to verifying Natural source files.

Setting Workspace Verification Options

Workspace verification options control verification behavior for the current workspace: recognized dialects, whether to perform staged or relaxed parsing, Natural library support, and the like.

Specifying Options on the Legacy Dialects Tab

Use the Legacy Dialects tab on the Workspace Options Verification tab to identify the dialect the application is written in. For information on supported dialects and versions, see the support guide for your product.

To specify options on the Legacy Dialects tab:

- 1 In the Modernization Workbench **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the Verification tab, then the Legacy Dialects tab (Figure 3-1).
- 2 In the Source Type pane, select the source file type whose dialects you want to view. For:
 - Cobol files, go to [step 3](#).
 - PL/I files, go to [step 3-1 on page 3-4](#).
 - Natural files, go to [step 11 on page 3-5](#).
 - Applications that use SQL, go to [step 13 on page 3-5](#).
 - C or C++ files, go to [step 14 on page 3-5](#).
- 3 In the Cobol Dialect pane, choose the Cobol dialect used by the application.
 - For ACUCOBOL-GT[®], select **RM/Cobol compatibility** to ensure proper memory allocation for applications written for Liant RM/COBOL (emulate behavior of -Ds compatibility option). Select **Graphical System** for applications executed on a graphical rather than character-based system.

In the **PERFORM behavior** drop-down, choose:

- **Stack** if the application was compiled with the PERFORM-type option set to allow recursive PERFORMS.

- **All exits active** if the application was compiled with the PERFORM-type option set to not allow recursive PERFORMS.
- For Cobol/390, Enterprise Cobol, choose DBCS or National for the national language behavior of picture symbol N and N-literals in the **Picture clause N-symbol** drop -down (emulate behavior of compiler NSYMBOL option).
- For MicroFocus Cobol, choose the binary storage mode, Word (2, 4, or 8 bytes) or Byte (1 to 8 bytes), in the **Binary Storage Mode** drop-down. Select **Enable MF comments** if the application contains comments in the first position.

In the **PERFORM behavior** drop-down, choose:

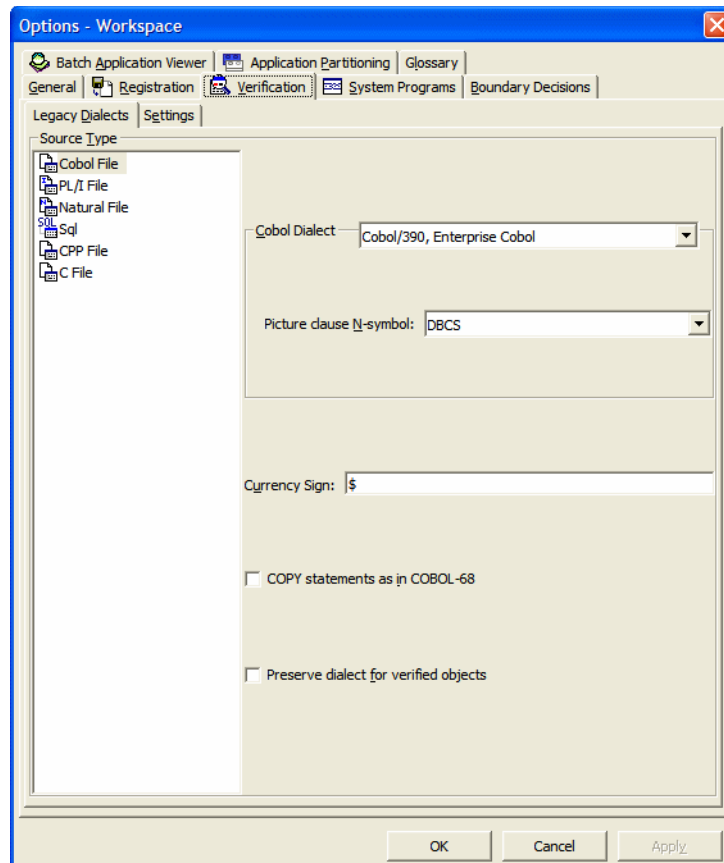
- **Stack** if the application was compiled with the PERFORM-type option set to allow recursive PERFORMS.
- **All exits active** if the application was compiled with the PERFORM-type option set to not allow recursive PERFORMS.

In the **Data File Assignment** drop-down, choose:

- **Dynamic** if the application was compiled with the ASSIGN option set to Dynamic.
 - **External** if the application was compiled with the ASSIGN option set to External.
 - For Unisys 2200 UCS Cobol, select **ASCII compatibility** if you need to ensure consistency with the ASCII version of Unisys Cobol (emulate behavior of compiler COMPAT option).
- 4 In the **Currency Sign** field, enter the currency symbol used by the application.
 - 5 Select **COPY statements as in COBOL-68** if the application was compiled on the mainframe with the OLDCOPY option set.
 - 6 Select **Preserve dialect for verified objects** to ensure that the parser re-verifies successfully verified Cobol files with the same dialect it used when the files were first successfully verified.

3-4 Setting Verification Options
Setting Workspace Verification Options

Figure 3-1 *Legacy Dialects Tab (Cobol File)*



- 7 In the PL/I Dialect pane, choose the PL/I dialect used by the application.
- 8 In the In margins pane, specify the current margins for PL/I source files. In the Out margins pane, specify the margins for the PL/I components to be created with the Modernization Workbench Component Maker tool.
- 9 In the Special symbols pane, add or delete special symbols used in PL/I files.

- 10 In the Logical Operators pane, choose:
 - **Autodetect** if you want the parser to autodetect logical operator characters used in PL/I files.
 - **Characters** if you want to identify the logical operator characters yourself:
 - In the **Not** field, specify the character used for NOT operations.
 - In the **Or** field, specify the character used for OR operations.
- 11 In the Natural Dialect pane, choose the Natural dialect used by the application.
- 12 In the Line Number Step pane, select the line-numbering increment you want the parser to use if you choose to restore line numbers in Natural source files (see [“Restoring Line Numbers in Natural Source”](#)). Choose:
 - **Auto detect** if you want the parser to use a line-numbering increment based on line number references in the source code.
 - **User defined** if you want the parser to use the line-numbering increment you specify. Enter the increment in the **Value** field.
- 13 In the SQL Dialect pane, select the SQL dialect used by the application.
- 14 In the Legacy Dialects pane, select the C or C++ dialect used by the application.

Restoring Line Numbers in Natural Source

Your source file delivery mechanism may have stripped line numbers from Natural source. You can restore stripped line numbers by selecting the Natural source files in the Repository Browser and choosing **Restore Line Numbers in Natural Source** in the **Edit** menu. The parser uses the increment settings you specified in [step 12 on page 3-5](#) when it restores the line numbers.

Specifying Options on the Settings Tab

Use the Settings tab on the Workspace Options Verification tab to enable staged parsing, relaxed parsing, sort card analysis for batch applications, Natural library support, and the like. Set workspace verification options for each type of file listed in the Settings tab.

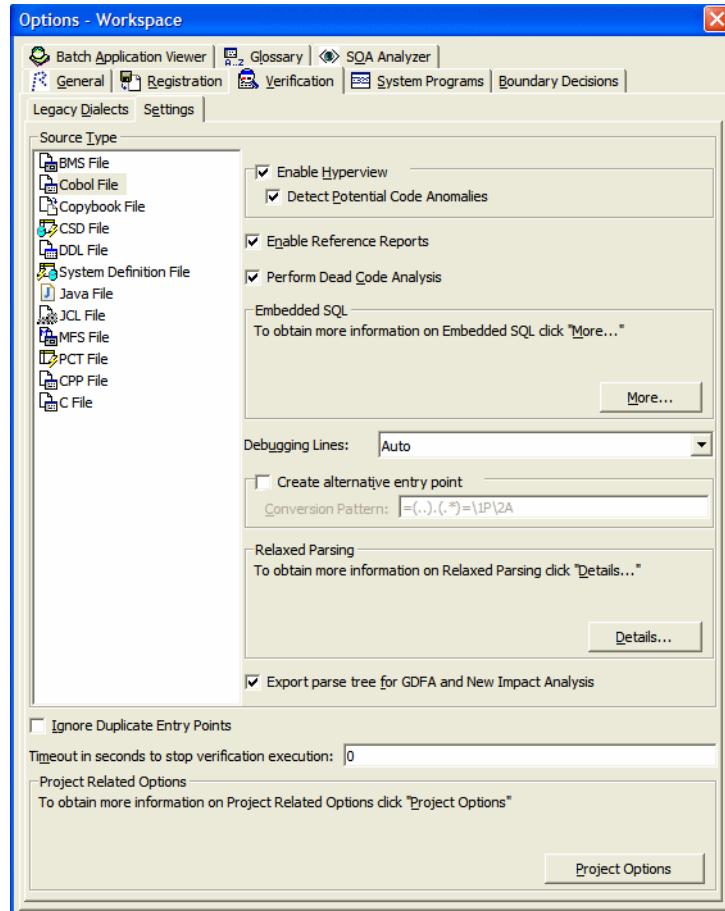
To specify options on the Settings tab:

- 1 In the Modernization Workbench **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the Verification tab, then the Settings tab (Figure 3-2).
- 2 In the Source Type pane, select the source file type whose verification options you want to set, then select the options. For a description of the options, see [“Source Type Settings Tab Options” on page 3-8](#).
- 3 Select **Ignore Duplicate Entry Points** if you want the parser to allow duplicate entry points defined by the Cobol statement ENTRY ‘PROG-ID’ USING A, or its equivalent in other languages. The parser creates an entry point object for the first program in which the entry point was encountered and issues a warning for the second program.

Note: To use this option, you must select **Enable Reference Reports** in the Settings tab, as described in [“Enabling Staged Parsing” on page 3-12](#). You cannot use this option to verify multiple programs with the same program ID.

- 4 In the **Timeout in seconds to stop verification execution**, enter the number of seconds to wait before stopping a stalled verification process.

Figure 3-2 Settings Tab (Cobol File)



Source Type Settings Tab Options

The following table describes verification options for supported source file types.

Table 3-1 *Verification Options for Source File Types*

Option	File Types	Description
Allow Implicit Instream Data	JCL	Inserts a DD * statement before implicit instream data if the statement was omitted from JCL.
Allow Keywords to Be Used as Identifiers	Copybook	Allows Cobol keywords to be used as identifiers.
C/C++ Parser Parameters	C, C++	Specifies the parameters used to compile the application. You can also specify these parameters in project verification options, in which case only the project parameters are used for verification. See step 25 on page 3-23 .
Create Alternative Entry Point	Cobol	Creates an additional entry point with a name based on the specified conversion pattern. Supports systems in which load module names differ from program IDs. For assistance, contact support services.
Debugging Lines	Cobol	Controls parsing of debugging lines. Off means parse lines as comments. On means parse lines as normal statements. Auto means parse lines based on the program debugging mode.
Detect Potential Code Anomalies	Cobol	Enables generation of HyperView information on potential code anomalies. See “Enabling Staged Parsing” on page 3-12 .

Table 3-1 *Verification Options for Source File Types (continued)*

Option	File Types	Description
Enable HyperView	Cobol, Natural, PL/I, RPG	Enables generation of HyperView information. See “Enabling Staged Parsing” on page 3-12.
Enable Reference Reports	Cobol, Control Language, ECL, JCL, Natural, PL/I, RPG, WFL	Enables generation of complete repository information for logical objects. See “Enabling Staged Parsing” on page 3-12.
Enable Quoted SQL Identifiers	Cobol, DDL	Allows quoted SQL identifiers.
Enter classpath to JAR Files and/or path to external Java file root directories	Java	For applications that use external .java files or Java Archive (JAR) files, specifies the location of the external .java files and, if Include Jar/Zip Files From Directories is selected, of the JAR files or ZIP files containing .java files. Right-click in the pane and choose Add in the pop-up menu to specify a file. Right-click and choose Add Folder to specify a folder. Folders are searched recursively.
Ignore Text After Column 72	DDL	Allows trailing enumeration characters (columns 73 through 80) in source lines.
Libraries	PowerBuilder	Specifies the PowerBuilder libraries used by the application. Libraries must be listed in the order they appear in the PBL File folder in the Repository pane. Right-click in the pane and choose Add in the pop-up menu to add a library.

3-10 Setting Verification Options
Setting Workspace Verification Options

Table 3-1 *Verification Options for Source File Types* (continued)

Option	File Types	Description
Libraries support	Natural	Enables Natural library support. See “Enabling Natural Library Support” on page 3-15 .
List of Include Directories	C, C++	Specifies the full path of the folders for include files (either original folders or Repository Browser folders if the include files were registered). Choose a recognized folder in the List of Include Directories pane. Right-click in the pane and choose Add Folder in the pop-up menu to specify a folder. You can also specify these folders in project verification options, in which case the tool looks only for the folders for the project. See step 24 on page 3-23 .
National Characters	System Definition	Specifies the national language characters for currency, number, and at symbols.
Perform Dead Code Analysis	Cobol, PL/I, RPG	Enables collection of dead code statistics. See “Enabling Staged Parsing” on page 3-12 .
Perform DSN Calling Chains Analysis	Control Language, ECL, JCL, WFL	Enables analysis of dataset calling chains. See “Enabling Advanced Data Flow Analysis for Control Language Files” on page 3-15 .
Perform System Calls Analysis	JCL	Enables analysis of system program input data to determine the application program started in a job step.

Table 3-1 *Verification Options for Source File Types* (continued)

Option	File Types	Description
Relaxed Parsing	AS400 Screen, BMS, Cobol, Copybook, CSD, DDL, Device Description, DPS, ECL, MFS, Natural, PL/I	Enables relaxed parsing. See “Enabling Relaxed Parsing” on page 3-14.
Relaxed Parsing for Embedded Statements	Cobol, PL/I	Enables relaxed parsing for embedded SQL, CICS, or DLI statements. See “Enabling Relaxed Parsing” on page 3-14.
Resolve Decisions Automatically	Control Language, WFL	Enables automatic decision resolution. See Chapter 6, “Resolving Decisions.”
Show Macro Generation	C, C++	Specifies whether to display in HyperView statements that derive from macro processing.
Sort Program Aliases	JCL	Enables batch sort card analysis. Choose a recognized sort utility in the Sort Program Aliases pane. Right-click in the pane and choose Add in the pop-up menu to add a sort utility. See “Enabling Sort Card Analysis for Batch Applications” on page 3-15.
SQL Statements Processor	Cobol	Specifies whether the SQL Pre-processor or Coprocessor was used to process embedded SQL statements.
Treat every file with main procedure as a program	C, C++	Specifies whether to treat only files with main functions as programs.

3-12 Setting Verification Options
Setting Workspace Verification Options

Table 3-1 *Verification Options for Source File Types* (continued)

Option	File Types	Description
Truncate Names of Absolute Elements	ECL	Allows the parser to truncate suffixes in the names of Cobol programs called by ECL. Specify a suffix in the adjoining text box. See “Truncating Names of Absolute Elements” on page 3-16.
Use Database Schema	Cobol, PL/I	Specifies whether to associate a program with a database schema. When this option is selected, the parser collects detailed information about SQL ports that cannot be determined from program text (SELECT *). If the schema does not contain the items the SQL statement refers to, an error is generated.

Enabling Staged Parsing

File verification generates repository information in four stages, as described below. You can control which stage the workbench parser performs by setting the *staged parsing* options on the Settings tab for workspace verification options (Figure 3-2). That may save you time verifying very large applications.

Rather than verify the application completely, you can verify it one or two stages at a time, generating only as much information as you need right away. When you are ready to work with a full repository, you can perform the entire verification at once, repeating the stages you’ve already performed and adding the stages you haven’t.

Tip: You can also improve verification performance by postponing program analysis until after verification, as described in [“Performing Post-Verification Program Analysis”](#) on page 4-18.

Basic Repository Information To generate basic repository information only, deselect the staged parsing options on the Settings tab (Figure 3-2). The parser:

- Generates relationships between source files (Cobol program files and copybooks, for example).
- Generates basic logical objects (programs and jobs, for example, but not entry points or screens).
- Generates Defines relationships between source files and logical objects.
- Calculates program complexity.
- Identifies missing support files (Cobol copybooks, JCL procedures, PL/I include files, and so forth).

Note: If you generate only basic repository information when you verify an application, advanced program analysis information is not collected, regardless of your settings in the Project Options Verification tab.

Full Logical Objects Information To generate complete repository information for logical objects, select **Enable Reference Reports** in the Settings tab (Figure 3-2). Set this option to generate Reference and Orphan Analysis reports for logical objects, and to enable non-HyperView analysis tools.

Note: If you select this option, verify all legacy objects in the workspace synchronously to ensure complete repository information.

HyperView Information To generate a HyperView construct model, select **Enable HyperView** in the Settings tab (Figure 3-2). A HyperView construct model defines the relationships between the constructs that comprise the file being verified: its sections, paragraphs, statements, conditions, variables, and so forth.

To generate HyperView information on potential code anomalies, select **Detect Potential Code Anomalies** in the Settings tab. For more information, see [“Viewing Executive Reports” on page 4-13](#).

Note: If you do not generate HyperView information when you verify an application, impact analysis, data flow, and execution flow information is not collected, regardless of your settings in the Project Options Verification tab.

Dead Code Statistics To generate *dead code* statistics, and to set the Dead attribute to True for dead constructs in HyperView, select **Perform Dead Code Analysis** in the Settings tab (Figure 3-2). The statistics comprise:

- Number of *dead statements* in the source file and referenced copybooks. A dead statement is a procedural statement that can never be reached during program execution.
- Number of *dead data elements* in the source file and referenced copybooks. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
- Number of *dead lines* in the source file and referenced copybooks. Dead lines are source lines containing dead statements or dead data elements.

You can view the statistics in the Statistic tab of the Properties window for an object or in the Complexity Metrics tool, as described in *Analyzing Projects* in the workbench documentation set.

Enabling Relaxed Parsing

The *relaxed parsing* option 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.

Use relaxed parsing when you are performing less rigorous analyses that do not need every statement to be modeled (estimating the complexity of an application written in an unsupported dialect, for example).

Note: Relaxed parsing may affect the behavior of other tools. You cannot generate code from legacy application source verified with the relaxed parser.

Enabling Advanced Data Flow Analysis for Control Language Files

Ordinarily, Modernization Workbench data flow analysis tools let you trace the flow of data into or out of a dataset only up to the program actually referenced in the control language file, whether or not that program writes to or reads from the dataset. If you need to trace the flow of data through the entire “calling chain,” that is, not only the referenced program, but also any programs that program calls, and any programs they call in turn:

- Select **Perform DSN Calling Chains Analysis** in the Settings tab for the control language file.
- Verify control language files *after* you verify the source files for the programs they use. If you reverify the source file for a program, you must also reverify the control language file that uses it.

Tip: If you verify an entire project, the workbench parses the files in appropriate order, taking account of the dependencies between control language and program files.

Enabling Sort Card Analysis for Batch Applications

If you use sort utilities in JCL files, you can enable sort card analysis by specifying the names of the sort utilities to the parser in the Settings tab for JCL files. The parser creates an artificial program entity that defines the inputs and outputs for each sort utility invocation. The program has a name of the form *JCLFileName.JobName.StepName.SequenceNumber*, where *SequenceNumber* identifies the order of the step in the job.

Enabling Natural Library Support

If you load Natural programs to a workspace from multiple libraries, and need to prevent library name collisions or want to maintain a list of libraries, select **Libraries support** in the Settings tab for Natural files, then specify the library names for each project in the workspace, as described in [step 20 on page 3-22](#). If you use this feature, the source files themselves must have names of the form *library.program.extension*.

Note: For assistance renaming Natural source files, contact support services.

Truncating Names of Absolute Elements

If you are verifying ECL files for an application in which absolute element names differ from program IDs, you can tell the parser to truncate suffixes in the names of Cobol programs called by ECL. If a Cobol program named CAP13MS.cob, for example, defines the entry point CAP13M, and an ECL program named CAP13M.ecl executes an absolute element called CAP13MA, then setting this option causes the parser to create a reference to the entry point CAP13M rather than CAP13MA.

Setting Project Verification Options

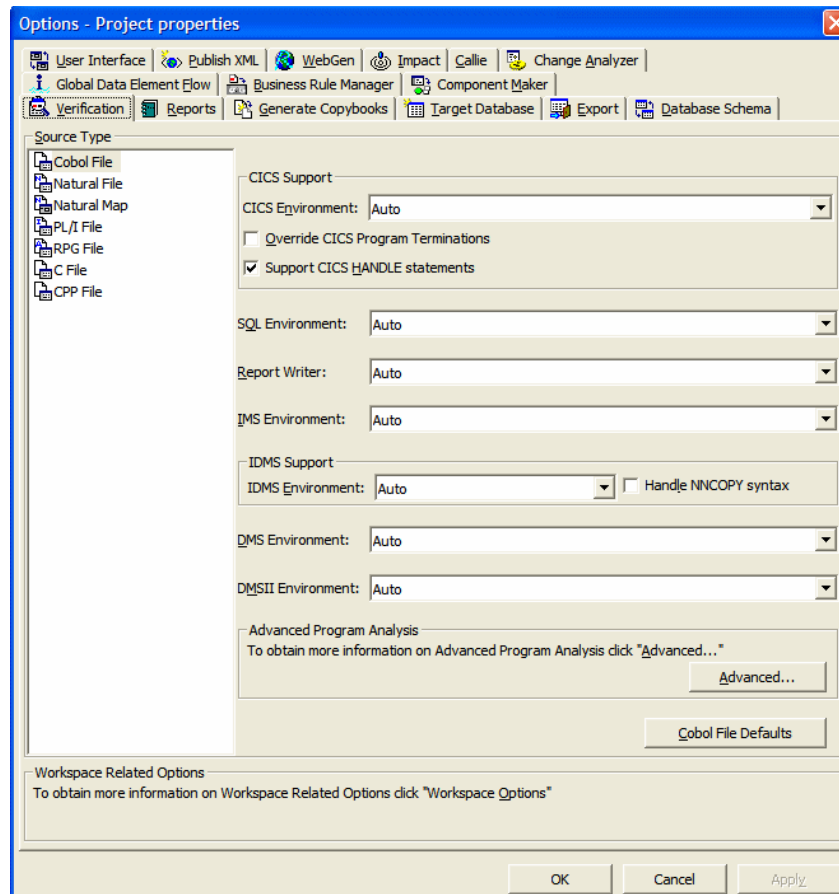
Project verification options control verification behavior for the selected project: the transaction-processing environment, whether to resolve decisions automatically after verification, and the like.

To set project verification options:

- 1 In the Modernization Workbench **Tools** menu, choose **Project Options**. The Project Options window opens. Click the Verification tab. The Verification tab opens (Figure 3-3).
- 2 In the Source Type pane, select the source file type whose project verification options you want to view. For:
 - Cobol files, go to [step 3](#).
 - PL/I files, go to [step 14 on page 3-21](#).
 - Natural files, go to [step 20 on page 3-22](#).
 - Natural Map files, go to [step 23 on page 3-23](#).
 - C or C++ files, go to [step 24 on page 3-23](#).
 - RPG files, [step 27 on page 3-24](#).

Note: For background on the autodetection methods described in steps 3-9 below, see [“Specifying the Processing Environment” on page 3-24](#).

Figure 3-3 Verification Tab (Cobol File)



- 3 In the **CICS Environment** drop-down, specify how you want the parser to interpret CICS-related code in Cobol files.

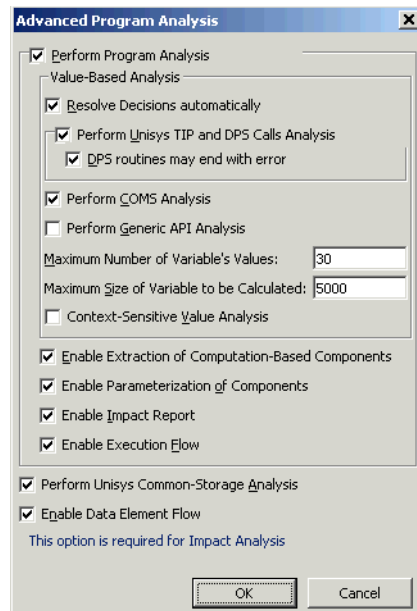
Select **Override CICS Program Terminations** if you want the parser to interpret CICS RETURN, XCTL, and ABEND commands as not terminating program execution. When this option is selected, error-handling code after these statements is either analyzed or treated as dead code.

3-18 Setting Verification Options
Setting Project Verification Options

Select **Support CICS HANDLE statements** if you want the parser to detect dependencies between CICS statements and related error-handling statements.

- 4 In the **SQL Environment** drop-down, specify how you want the parser to interpret SQL-related code in Cobol files. In the **Report Writer Environment** drop-down, specify how you want the parser to interpret Report Writer-related code.
- 5 In the **IMS Environment** drop-down, specify how you want the parser to interpret IMS-related code in Cobol files.
- 6 In the **IDMS Environment** drop-down, specify how you want the parser to interpret IDMS-related code in Cobol files. Select **Handle NNCOPY syntax** if you want the parser to recognize NNCOPY statements in Cobol files.
- 7 In the **DMS Environment** drop-down, specify how you want the parser to interpret Unisys CDML statements in Cobol files.
- 8 In the **DMSII Environment** drop-down, specify how you want the parser to interpret Unisys MCP DMS II statements in Cobol files.
- 9 In the **AIM/DB Environment** drop-down, specify how you want the parser to interpret Fujitsu AIM/DB-related code in Cobol files.
- 10 Click the **Advanced** button. The Advanced Program Analysis window opens (Figure 3-4).

Figure 3-4 Advanced Program Analysis Window (Cobol File)



- 11 Select **Perform Program Analysis** to enable program analysis and component extraction features for Cobol files in the project. If you do not want to enable each feature, you can disable individual features as necessary.

Tip: You can improve verification performance by postponing program analysis until after verification, as described in [“Viewing Key Object Relationships” on page 4-17](#)

Select:

- **Resolve Decisions Automatically** if you want the parser to autoresolve decisions after successfully verifying files. For more information, see [Chapter 6, “Resolving Decisions.”](#)
- **Perform Unisys TIP and DPS Calls Analysis** if you want the parser to perform TIP and DPS calls analysis for Unisys 2200 Cobol files.

- **DPS routines may end with error** if you want the parser to perform call analysis of DPS routines that end in an error. When this option is selected, error-handling code for these routines is either analyzed or treated as dead code.
- **Perform COMS Analysis** if you want the parser to define relationships for Unisys MCP COMS SEND statements.
- **Perform Generic API Analysis** if you want the parser to define relationships with objects passed as parameters in calls to unsupported program interfaces, in addition to relationships with the called programs themselves. For information on how to identify the programs and parameters to the workbench, see [Appendix A, “Identifying Interfaces for Generic API Analysis.”](#)

Tip: You may be able to improve verification performance and avoid out-of-memory problems by manipulating the **Maximum Number of Variable’s Values** and **Maximum Size of Variable to Be Calculated** fields. For more information, see [“Optimizing Verification for Advanced Program Analysis” on page 3-25.](#)

- **Context-Sensitive Value Analysis** if you want the parser to perform context-sensitive automatic decision resolution for Unisys MCP COMS analysis.

Note: Choosing this option may degrade verification performance.

- **Enable Extraction of Computation-Based Components** if you want to enable computation-based componentization.
- **Enable Parameterization of Components** if you want to enable parameterized structure- and computation-based componentization.
- **Enable Impact Report** if you want to enable impact analysis.

Note: You must also set **Enable Data Element Flow** to enable the impact trace tool.

- **Enable Execution Flow** if you want to enable the Execution Path tool.
- 12 Select **Perform Unisys Common-Storage Analysis** if you want the parser to include in the analysis for Unisys Cobol files variables that are not explicitly declared in CALL statements but participate in interprogram communications.

Note: You must set this option to include Unisys Cobol common storage variables in impact traces and global data flow diagrams.

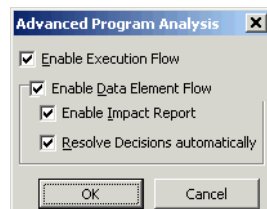
- 13 Select **Enable Data Element Flow** if you want to enable the Global Data Flow, Change Analyzer, and impact trace tools.
- 14 In the Transaction Environment pane, specify how you want the parser to interpret CICS-related code in PL/I files. For more information, see [“Specifying the Processing Environment” on page 3-24](#).

Note: The Auto setting is not available for PL/I.

Select **Override CICS Program Terminations** if you want the parser to interpret CICS RETURN, XCTL, and ABEND commands as not terminating program execution. When this option is selected, error-handling code after these statements is either analyzed or treated as dead code.

- 15 Click the **Advanced** button. The Advanced Program Analysis window opens (Figure 3-5).

Figure 3-5 *Advanced Program Analysis Window (PL/I File)*

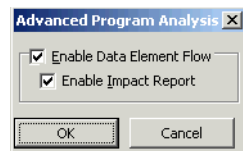


- 16 Select **Enable Execution Flow** if you want to enable the Execution Path tool for PL/I programs.
- 17 Select **Enable Data Element Flow** if you want to enable the Global Data Flow and Change Analyzer tools for PL/I programs.
- 18 Select **Enable Impact Report** if you want to enable impact analysis for PL/I programs.
- 19 Select **Resolve Decisions Automatically** if you want the parser to autoresolve decisions after successfully verifying files. For more information, see [Chapter 6, “Resolving Decisions.”](#)
- 20 If you selected **Libraries support** in the Settings tab for Natural files, select a recognized library in the Libraries pane. Right-click in the Libraries pane and choose **Add** in the pop-up menu to add a library to the list. The system displays an empty text field next to a selected check box. Enter the name of the library in the field and click outside the field.

Edit a library name by selecting it and choosing **Edit** in the right-click menu. Delete a library by selecting it and choosing **Delete** in the right-click menu.

For more information, see “Enabling Natural Library Support” on page 3-15.
- 21 Click the **Advanced** button. The Advanced Program Analysis window opens (Figure 3-6).

Figure 3-6 *Advanced Program Analysis Window (Natural File)*



- 22 Select **Enable Data Element Flow** if you want to enable the Global Data Flow and Change Analyzer tools for Natural programs. Select **Enable Impact Report** if you want to enable impact analysis for Natural programs.

- 23** In the Help routines pane, choose:
- **Programs** if you want the parser to treat help routines in Natural map files as programs.
 - **Helpmaps** if you want the parser to treat help routines in Natural map files as helpmaps.

- 24** In the List of Include Directories pane, choose the folder for include files used by the project (either original folders or Repository Browser folders if the include files were registered). Right-click in the pane and choose **Add** in the pop-up menu to add a folder to the list. Specify the full path of the folder.

Edit a folder by selecting it and choosing **Edit** in the right-click menu. Delete a folder by selecting it and choosing **Delete** in the right-click menu.

Note: If you do not set this option, workspace verification options determine the include folders the parser looks for. For more information, see [“List of Include Directories” on page 3-10](#).

- 25** In the **C/C++ Parser Parameters** field, enter the parameters used to compile the application.

Note: If you do not set this option, workspace verification options control the parameters the parser uses. For more information, see [“C/C++ Parser Parameters” on page 3-8](#).

- 26** Select **Use Precompiled Header File** if you want the parser to use a precompiled header file when it verifies the project. In the adjacent field, enter the full path of the header file. Do not specify the file extension. Using a precompiled header file may improve verification performance significantly.

Note: The content of the header file must appear in both a .c or .cpp file and a .h file. The precompiled header file need not have been used to compile the application.

- 27 In the **SQL Environment** drop-down, specify how you want the parser to interpret SQL-related code in RPG files. For more information, see [“Specifying the Processing Environment” on page 3-24](#).
- 28 In the Advanced Program Analysis pane, select **Enable Data Element Flow** if you want to enable the Global Data Flow and Change Analyzer tools for RPG programs. Select **Enable Impact Report** if you want to enable impact analysis for RPG programs.
- 29 Select **Resolve Decisions Automatically** if you want the parser to autoreresolve decisions after successfully verifying files. For more information, see [Chapter 6, “Resolving Decisions.”](#)

Specifying the Processing Environment

The Modernization Workbench parser autodetects the *environment* file is intended to execute in based on the environment-related code it finds in the file. To ensure correct data flow, it sets up the internal parse tree for the file in a way that emulates the environment on the mainframe.

For Cobol CICS, for example, the parser treats an EXEC CICS statement or DFHCOMMAREA variable as CICS-related and, if necessary, adds the standard CICS copybook DFHEIB to the workspace; declares DFHCOMMAREA in the internal parse tree; and adds the phrase `Procedure Division using DFHEIBLK, DFHCOMMAREA` to the internal parse tree.

Autodetection is not always appropriate, of course. You may want the parser to treat a file as a transaction-processing program even in the absence of CICS- or IMS-related code. For each autodetected environment, select:

- Auto, if you want the parser to autodetect the environment for the file.
- Yes, if you want to force the parser to treat the file as environment-related even in the absence of environment-related code.
- No, if you want to force the parser to treat the file as unrelated to the environment even in the presence of environment-related code. The parser classifies environment-related code as a syntax error.

Optimizing Verification for Advanced Program Analysis

When you set the advanced program analysis options for Cobol projects described in [step 11 on page 3-19](#), the parser calculates constant values for variables at every node in the HyperView parse tree. That's one reason why very large Cobol applications may encounter performance or memory problems during verification.

You may be able to improve verification performance and avoid out-of-memory problems by manipulating advanced program analysis options:

- In the **Maximum Number of Variable's Values** field, enter the maximum number of values to be calculated for each variable during verification for advanced program analysis. Limit is 200.
- In the **Maximum Size of Variable to Be Calculated** field, enter the maximum size in bytes for each variable value to be calculated during verification for advanced program analysis.

The lower the maximums, the better performance and memory usage you can expect. For each setting, you are warned during verification about variables for which the specified maximum is exceeded. It's usually best to increase the overflowed maximum and reverify the application.

Identifying System Programs

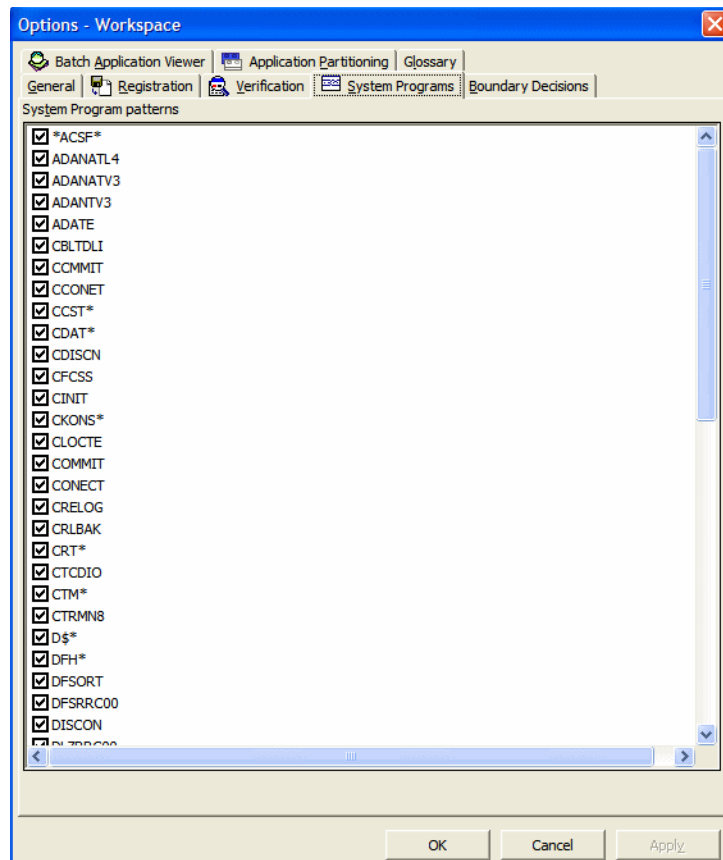
A *system program* is a generic program (a mainframe sort utility, for example) provided by the underlying system and used in unmodified form in the legacy application. You need to identify system programs to the parser so that it can distinguish them from application programs and create relationships for them with their referencing files. Use the System Programs tab in the Workspace Options window to identify system programs.

Note: Contact support services to learn how to identify system programs as drivers, and how to specify simple data flows between input/output datasets of system programs.

To identify system programs:

- 1 In the Modernization Workbench **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the System Programs tab (Figure 3-7).

Figure 3-7 *System Programs Tab*



- 2 In the System Program Patterns pane, select the patterns that match the names of the system programs your application uses.

Add a pattern by right-clicking in the System Program Patterns pane and choosing **Add** in the pop-up menu. The system displays an emp-

ty text field next to a selected check box. Enter the pattern in the field and click outside the field.

Edit a pattern by selecting it and choosing **Edit** in the right-click menu. Delete a pattern by selecting it and choosing **Delete** in the right-click menu.

How to Detect the System Programs Your Application Uses

The most convenient way to detect the system programs an application uses is to run an unresolved report after verification, as described in [Chapter 5](#). Once you learn from the report which system programs are referenced, you can identify them in the System Programs tab and reverify any *one* of their referencing source files.

The reference report tool lets you bring up the System Programs tab while you are in the tool itself. Use the **System Programs** choice in the reference report **View** menu to display the tab, then follow the instructions in [step 2](#) to identify system programs to the parser.

Specifying Boundary Decisions

Specify a *boundary decision* object if your application uses a method call to interface with a database, message queue, or other resource. Suppose the function `f1f()` in the following example writes to a queue named `abc`:

```
int f1f(char*)
{
return 0;
}

int f2f()
{
return f1f("abc");
}
```

As far as the parser is concerned, `f1f("abc")` is a method call like any other method call. There is no indication from the code that the called function is writing to a queue.

When you specify the boundary decisions for a workspace, you tell the parser to create a decision object of a given resource type for each such call. Here is the decision object for the write to the queue:

```
int f2f().InsertsQueue.int f1f(char*)
```

After verification, you can resolve the decisions to the appropriate resources in the Decision Resolution tool.

To set boundary decision options:

- 1 In the **Tools** menu, choose **Workspace Options**. The Workspace Options window opens. Click the Boundary Decisions tab.
- 2 In the Decision Types pane, select the decision types associated with called procedures in your application. For the example above, you would select the Queue decision type.
- 3 In the righthand pane, select each signature of a given method type you want to associate with the selected decision type. For the example above, the method type signature would be `int f1f(char*)`

Add a signature to a method type by right-clicking in the appropriate pane and choosing **Add** in the pop-up menu. The system displays an empty text field next to a selected check box. Enter the name of the new signature in the field and click outside the field.

Edit a signature by selecting it and choosing **Edit** in the right-click menu. Delete a signature by selecting it and choosing **Delete** in the right-click menu.

Note: Do not insert a space between the parentheses in the signature. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications.

Special Handling for C or C++ Signatures

Keep in mind that the signatures of C or C++ functions can contain an asterisk (*) character, as in the example above. So if you specify a signature with a * character, you may receive results containing not only the intended signatures but all signatures matching the wildcard pattern. Delete the unwanted decision objects manually.

What's Next?

That completes our survey of Modernization Workbench verification options. Now let's look at how you use the workbench to verify applications and view inventory reports.

3-30 Setting Verification Options
What's Next?

Verifying Files and Performing Post-Verification Tasks



Verifying a source file ensures that its contents can be understood by the workbench parser. For each successfully verified file, the parser builds an object model that serves as the basis for diagrams, reports, and other documentation. For an unsuccessfully verified file, the parser builds an object model for as much of the file as it understands. The workbench stops parsing the file at the statement at which it encountered an error.

Verifying Source Files

Parsing, or *verifying*, an application source file generates the object model for the file. Only a master user can verify source files in a multiuser environment.

You can verify a single file, a group of files, all the files in a folder, or all the files in a project. If you verify an entire project, the workbench parses the files in appropriate order, taking account of likely dependencies be-

tween file types. Verify Cobol copybooks only if you plan to use the Data Fusion Facility for copybooks not included in a source file.

If your RPG or AS/400 Cobol application uses copy statements that reference Database Description or Device Description files, or if your MCP Cobol application uses copy statements that reference DMSII DASDL files, you need to generate copybooks for the application *before* you verify program files, as described in [“Generating Copybooks” on page 4-23](#).

For distributed languages, consult the support notes in the relevant language support guide for pre-verification requirements. You can find the guides in **Start:All Programs:Micro Focus:Modernization Workbench Documentation**.

To verify source files:

- 1 Set verification options, as described in [Chapter 3, “Setting Verification Options.”](#)
- 2A In the Repository Browser, select the project, folder, or files you want to verify and choose **Verify** in the **Prepare** menu.
- 2B In the Repository Browser, select the Cobol copybooks you want to verify and choose **Verify Copybook** in the **Prepare** menu.

Note: In a multiuser environment, you are prompted to drop repository indexes to improve verification performance. Click **Yes**. You will be prompted to restore the indexes when you analyze the files. For more information, see *Getting Started* in the workbench documentation set.

The parser builds an object model for each successfully verified file. For an unsuccessfully verified file, the parser builds an object model for as much of the file as it understands. Verification results are displayed in the Activity Log.

How the System Refreshes the Repository

When you edit a source file in the Modernization Workbench, the system recursively checks every repository object that may be affected by the edit: *refreshes* the repository. If the edit *invalidates* the object, you need to reverify the source file that contains it. The file with the invalidated object is displayed in **bold** type in the Repository Browser.

Invalidating Objects before Reverification

You can save time reverifying very large applications by invalidating some or all of the source files in them before you reverify. You can invalidate a single file, a group of files, all the files in a folder, or all the files in a project. In the Repository Browser, select the project, folder, or files you want to invalidate and choose **Invalidate Selected Objects** in the **File** menu. Invalidated files are displayed in **bold** type in the Repository Browser.

Viewing Verification Reports

Verification Reports offer a convenient way to analyze verification results for a project. The report displays the verification status and number of verification errors and warnings for each source file in the project, a count of each type of error and warning, and a detailed list of errors and warnings.

You can set options that control the source file types reported on, the amount of program context reported for each message, and the appearance of the display. After generating the report, you can filter out items not of interest to your project.

Generating Verification Reports

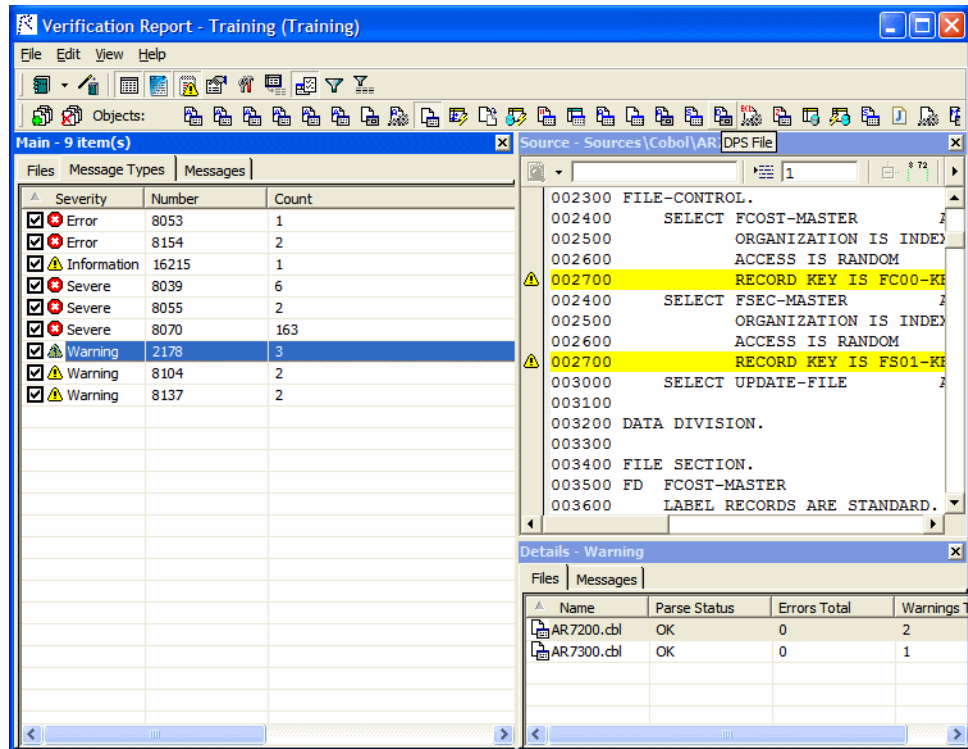
You generate Verification Reports for the current project. Options let you control the types of source files you report on, the amount of program context reported for each message, and the appearance of the display. For more information, see [“Setting Verification Report User Preferences” on page 4-6](#).

To generate a Verification Report:

- 1** In the Repository Browser, select the project for the report and choose **Verification Report** in the **Prepare** menu. The Verification Report window opens on top of the Modernization Workbench main window (Figure 4-1).
- 2** In the **File** menu, choose **Generate report**. The Verification Report window displays the results.

4-4 Verifying Files and Performing Post-Verification Tasks Viewing Verification Reports

Figure 4-1 Verification Report Window





Working with Verification Reports

The Verification Report window consists of a Main pane, Details pane, and Source pane. Select the appropriate choice in the **View** menu to show or hide a pane.

Main Pane

By default, the Main pane displays the Files tab, showing the verification status of each file in the project and a count of the verification errors and warnings issued for each file. Click a file to view the message types and messages issued for it in the Details pane.

The Message Types tab displays a count of each type of verification error and warning issued for the project. It shows the severity and error number of the message type, its generic text, and the number of files for which it was issued. Errors are indicated with a  symbol. Warnings are indicated with a  symbol. Click a message type to view a list of source files for which it was issued and each instance of the message type in the Details pane.

The Messages tab displays each verification error and warning issued for the project. It shows the severity and error number of the message, its detailed text, a count of the number of occurrences of the message in the project, and the number of files for which it was issued. Click a message to view a list of source files for which it was issued in the Details pane.

Sorting Entries Click a column heading to sort the report entries by that column.

Sizing Columns Grab-and-drag the border of a column heading to increase or decrease the width of the column.

Viewing Properties Select a source file and choose **Properties** in the **View** menu to display a set of tabs with file properties. For usage information, see *Getting Started* in the workbench documentation set.


Marking Items Place a check mark next to an item to mark it. To mark all the items in the selected tab, choose **Mark All** in the **Edit** menu. To unmark all the items in the selected tab, choose **Unmark All**. For the relationship between marking and filtering, see [“Filtering Verification Reports” on page 4-8](#).


Copying or Moving Objects between Projects Mark the source files you want to copy or move in the Main pane, then choose **Include Into Project** in the **File** menu. The Copy To Project window opens. Click **Move** if you want to move rather than copy the objects. Click **New** if you want to create a new project. Then choose the project you want to copy or move the objects to and click **OK**.

Copying Entries to the Clipboard Mark the items you want to copy to the clipboard and choose **Copy** in the **Edit** menu.

Details Pane



When the Files tab is selected in the Main pane, the Details pane displays the message types and messages issued for the selected file. Click a message to navigate to the source for the offending code in the Source pane.

When the Message Types tab is selected, the Details pane displays a list of source files for which the selected message type was issued and each instance of the message type. Click a message to navigate to the source for the offending code in the Source pane. Click a file to view its source code in the Source pane. Click the  button to navigate to the first instance of offending code.

When the Messages tab is selected, the Details pane displays a list of source files for which the selected message was issued. Click a file to view its source code in the Source pane. Click the  button to navigate to the first instance of offending code.

Sort and size columns, and view properties, as described in [“Main Pane.”](#)

Source Pane

The Source pane displays view-only source for the file selected in the Main or Details pane. Offending code is highlighted in yellow. Click the  button to navigate to the next instance of offending code. Click the  button to navigate to the previous instance.

Usage is similar to that for the Modernization Workbench HyperView Source pane. For more information, see *Analyzing Programs* in the documentation set.

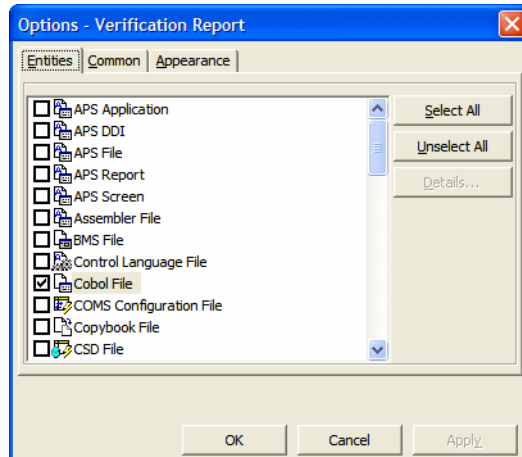
Setting Verification Report User Preferences

Verification Report user preferences control the types of source files you report on and the amount of information you display in the Main and Details panes.

To set Verification Report user preferences:

- 1 In the Verification Report window, choose **Options** in the **View** menu. The Verification Report Options window opens (Figure 4-2).

Figure 4-2 *Verification Report Options Window*



- 2 Click the Entities tab. In the list box, select each type of source file you want to display in the report. Click **Select All** to select all the source file types. Click **Unselect All** to deselect all the source file types.

Tip: You can also click the icons on the Included Entity Types tool bar to select and deselect included types.

- 3 Click the Common tab. Select **Include source context** if you want to display code in the “neighborhood” of the offending code in the Source Code column for a message in the Details pane. In the **Before** and **After** combo boxes, enter the number of lines of code before and after the offending code you want to display.
- 4 Click the Appearance tab. In the **Select Region** drop-down, select the pane you want to set options for. In the **Tables** list box, select the tab you want to set options for. In the **Visible Columns** list box, select each column you want to display in the selected tab.

Filtering Verification Reports

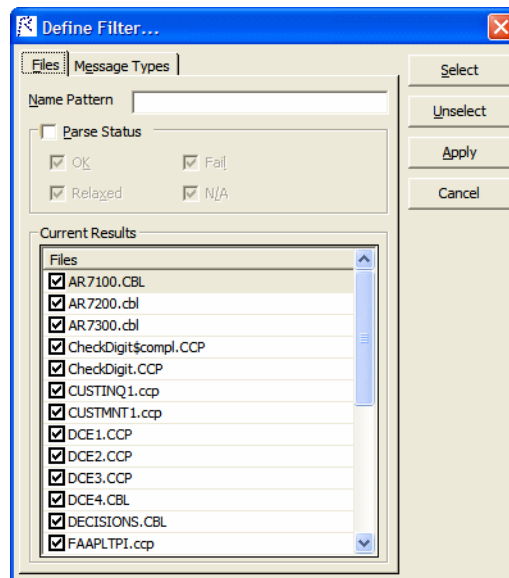
After generating a Verification Report, you can filter out source files, message types, and messages that are not of interest to your project. You filter a Verification Report in two steps:

- Set up the filter. For small projects, you can mark the items you want to appear manually, either in the Verification Report window or the Define Filter window. For larger projects, you'll need to mark items in batch mode, by defining a filter in the Define Filter window.
- Apply the filter. When you apply the filter, only marked items are displayed in the report.

To filter a Verification Report:

- 1 In the Verification Report window, choose **Filter** in the **Edit** menu. The Define Filter window opens (Figure 4-3).

Figure 4-3 *Define Filter Window*



- 2 Click the Files tab. The Current Results pane displays the results for the current files filter.

In the **Name Pattern** field, enter a matching pattern for files you want to display in the report. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Select **Parse Status** to display only source files with a given verification status, then select each verification status you want to report on.

Note: The **Name Pattern** and **Parse Status** fields are ANDED. If a file that matches the name pattern does not have one of the specified statuses, it is not marked.

When you are satisfied with your choices, click **Select** to mark the matched files in the Current Results pane. Click **Unselect** to unmark the matched files. You can also mark files manually.

Click **Apply** to mark the files in the Files tab of the Verification Report window. If **Show Filtered Only** is selected in the **Edit** menu, the Files tab is filtered to show only marked items.

Tip: If no files are marked, clicking **Select** marks them all in the Current Results pane. If all files are marked, clicking **Unselect** unmarks them all in the Current Results pane.

- 3 Click the Message Type tab. The Current Results pane displays the results for the current message types filter.

In the **Name Pattern** field, enter a matching pattern for message types you want to display in the report. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Select **Status** to display only message types with a given status, then select each status you want to report on.

Select **Severity** to display only message types with a given severity, then select each severity, you want to report on.

Note: The **Name Pattern**, **Status**, and **Severity** fields are ANDed. If a message type that matches the name pattern does not have one of the specified statuses or severities, it is not marked.

When you are satisfied with your choices, click **Select** to mark the matched message types in the Current Results pane. Click **Unselect** to unmark the matched message types. You can also mark message types manually.

Click **Apply** to mark the message types in the Message Types tab of the Verification Report window. If **Show Filtered Only** is selected in the **Edit** menu, the Message Types tab is filtered to show only marked items.

Tip: If no message types are marked, clicking **Select** marks them all in the Current Results pane. If all message types are marked, clicking **Unselect** unmarks them all in the Current Results pane.

- 4 Click **Cancel** to dismiss the Define Filter window. In the Verification Report window, choose **Show Filtered Only** in the **Edit** menu. The Files and Message Types tab are filtered to show only marked items. The Messages tab is filtered to show only messages for the marked files or message types.

Note: The filtering results for files and message types are ANDed. If a marked message type was not issued for a marked file, the file does not appear in the report. Of course, this means that source files with 0 errors and warnings are not displayed even if they are marked.

Exporting Verification Reports

You can create printable Verification Reports for the selected tab in the Main or Details pane. You can also use the *Report Wizard* to create printable reports for files and associated message types.

After a report is generated, click **Page Setup** to specify print job options. Click **Print** to send the job to the printer. Click **Save** to save the report to a variety of standard formats.

To create printable reports for the selected tab:

- 1** In the Verification Report window, choose:
 - **Report:Main** in the **File** menu to create a printable report for the selected tab in the Main pane.
 - **Report:Details** in the **File** menu to create a printable report for the selected tab in the Details pane.

The printable report is displayed.

To create printable reports for files and message types:

- 1** In the Verification Report window, choose **Report:Constructor** in the **File** menu. The Report Wizard opens.
- 2A** Deselect **Correct Filtering** if you want to report on all files and message types. Click **Next**. The Edit Columns window opens.
- 2B** Select **Correct Filtering** if you want to edit the report filter. Click **Next**. The Correct Filtering window opens.

In the Files pane, select the files you want to report on. In the Message Types pane, select the message types you want to report on. Select **Auto Complete Filtering** if you want the Report Wizard to select items in the right column based on your selection in the left column, and vice versa. When you are satisfied with your choices, click **Next**. The Edit Columns window opens.

- 3** In the **Select Main Table** drop-down, select Files if you want the report to be organized by the files table, Message Types if you want the report to be organized by the message types table. The **Select Subordered Table** drop-down is modified to reflect your choice.

In the Available Columns pane for each table, select the columns you want to appear in the printed report and click the **>** button to move the column into the Report Columns panes. In the Report Columns pane, select a column and click the **<** button to move the column into the Available Columns pane.

Click the **>>** button to move all columns from the Available Columns pane to the Report Columns pane. Click the **<<** button to move all columns from the Report Columns pane to the Available Columns pane.

4-12 Verifying Files and Performing Post-Verification Tasks *Viewing Inventory Reports*

When you are satisfied with your choices, click **Finish**. The printable report is displayed.

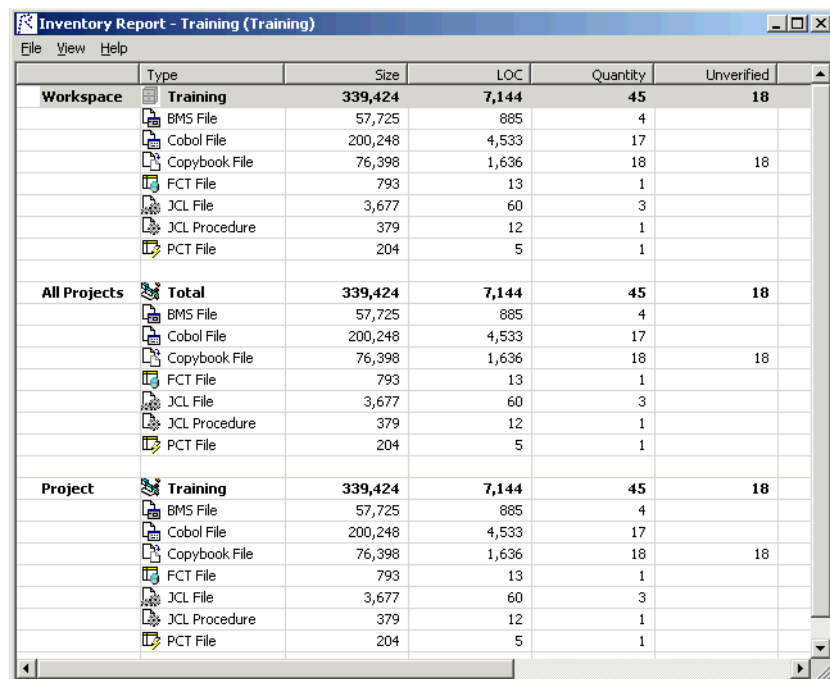
Viewing Inventory Reports

Inventory Reports give high-level statistics for each source file type in the current workspace: size in bytes, number of lines of code, whether verified, and the like.

To generate an Inventory Report:

- 1 In the Repository Browser, select a project and choose **Inventory Report** in the **Prepare** menu. The Inventory Report window opens (Figure 4-4).
- 2 Choose **Save As** in the **File** menu to export the report to HTML, Excel, RTF, Word, or formatted text.

Figure 4-4 *Inventory Report*



The screenshot shows a window titled "Inventory Report - Training (Training)" with a menu bar (File, View, Help) and a table of statistics. The table has columns for Type, Size, LOC, Quantity, and Unverified. It is organized into three sections: Workspace, All Projects, and Project, each showing a total row and a list of file types with their respective statistics.

	Type	Size	LOC	Quantity	Unverified
Workspace	Training	339,424	7,144	45	18
	BMS File	57,725	885	4	
	Cobol File	200,248	4,533	17	
	Copybook File	76,398	1,636	18	18
	FCT File	793	13	1	
	JCL File	3,677	60	3	
	JCL Procedure	379	12	1	
	PCT File	204	5	1	
All Projects	Total	339,424	7,144	45	18
	BMS File	57,725	885	4	
	Cobol File	200,248	4,533	17	
	Copybook File	76,398	1,636	18	18
	FCT File	793	13	1	
	JCL File	3,677	60	3	
	JCL Procedure	379	12	1	
	PCT File	204	5	1	
Project	Training	339,424	7,144	45	18
	BMS File	57,725	885	4	
	Cobol File	200,248	4,533	17	
	Copybook File	76,398	1,636	18	18
	FCT File	793	13	1	
	JCL File	3,677	60	3	
	JCL Procedure	379	12	1	
	PCT File	204	5	1	

Viewing Executive Reports

Executive Reports offer HTML views of application inventories that a manager can use to assess the risks and costs of supporting the application:

- The Application Summary view gives statistics for industry-standard metrics such as program volume, maintainability, cyclomatic complexity, and number of defects.
- The Potential Code Anomalies view gives statistics for potential anomalies that may mark programs as candidates for re-engineering: GOTO non-exits, range overlaps, and the like. You can customize potential code anomalies as described in [“Defining Potential Code Anomalies” on page 4-16](#).

Note: Set **Detect Potential Code Anomalies** in the Settings tab of the workspace verification options to generate these statistics.

- The Repository Statistics view gives statistics for Modernization Workbench verification results and unresolved or unreferenced application elements.
- The Standard Deviations view displays graphs that plot the deviation of the programs in the application from the means for six key industry-standard metrics.

The top page in each view displays the available statistics and graphs. Click the links to view the detail for each type of statistic or graph. In the statistic or graph detail page, click the link for a program to view the detail for that program.

Generating Executive Reports

Generate an Executive report as described in this section.

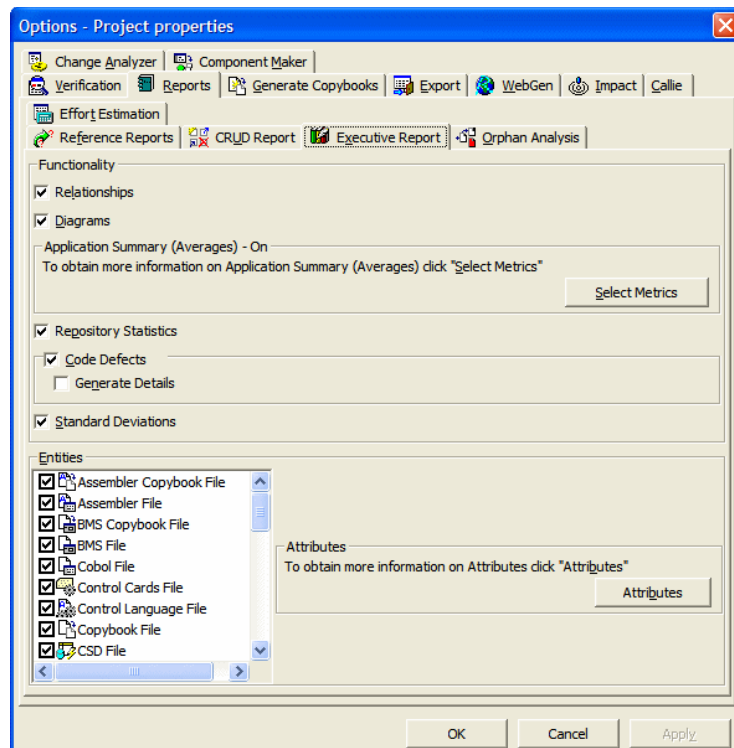
Tip: After generating an Executive Report, use the Executive Report category in the HyperView Clipper tool to view potential code anomalies in program context. For HyperView usage, see *Analyzing Programs* in the workbench documentation set.

4-14 Verifying Files and Performing Post-Verification Tasks
Viewing Executive Reports

To generate an Executive Report:

- 1 In the Repository Browser, select a project and choose **Project Options** in the **Tools** menu. The Project Options window opens. Click the **Reports** tab, then the **Executive Report** tab (Figure 4-5).

Figure 4-5 Project Options Executive Report Tab



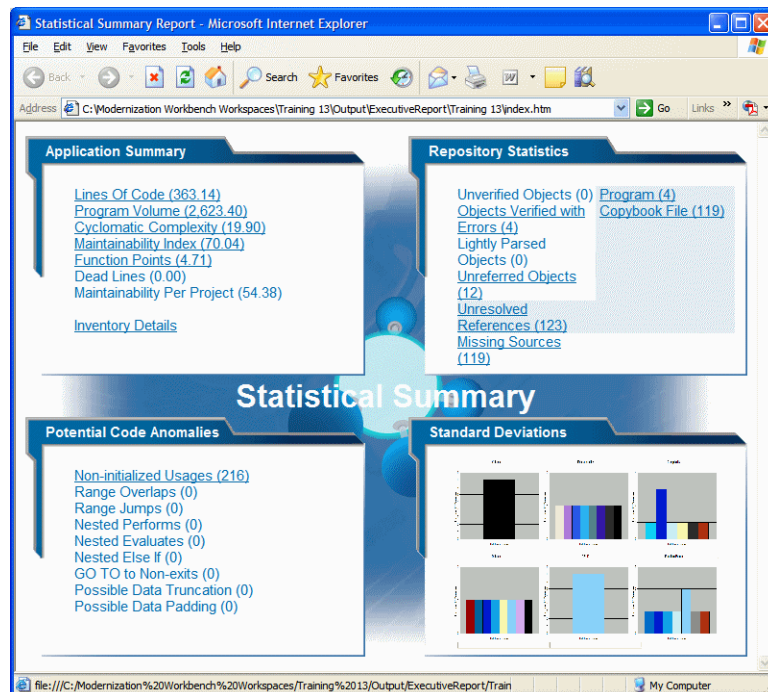
- 2 In the Executive Report tab:
 - Select each report function you want to enable. For the Application Summary function, click **Select Metrics**. In the Application Summary (Averages) window, select **Application Summary (Averages)** to enable the function, then choose each metric you want to include in the report.
 - Select each type of entity you want to include in the report. To edit the attributes included in the report for the entity type, click

Attributes. In the Attributes window, select each attribute of the selected entity type to include in the report.

Generally, the fewer entities and functions you choose, the better performance you can expect. You should especially consider not reporting on:

- Detail for code anomalies. Leave **Generate Details** unchecked.
 - Data stores. Leave data store entities unchecked.
 - Relationships, if you do not need cross-reference information. Leave **Relationships** unchecked
- 3** In the Repository Browser, choose **Executive Report** in the **Prepare** menu. You are prompted to display the report now. Click **Yes** if you want to view the report immediately. The Executive Report opens in a browser (Figure 4-6). The report is stored in `\Workspace\Output\Executive Report\Project\index.htm`.

Figure 4-6 *Executive Report*



Defining Potential Code Anomalies

You can view and modify existing definitions of potential code anomalies (other than range overlaps and range jumps) in the HyperView advanced search criteria in the Coding Standards folder.

To define a new code anomaly, you must define an advanced search criterion for the anomaly and a matching entry in the file *workbench\home\Data\CodeDefects.xml*. The entry has the form:

```
<DEFECT Id="name"  
        Internal="True|False"  
        Enabled="True|False"  
        Caption="display name"  
        ListName="list name"  
        Criterion="path of criterion"  
>
```

where:

- `Id` is a unique name identifying the code anomaly in the workbench.
- `Internal` specifies whether the anomaly is implemented internally in program code (True), or externally in an advanced search criterion (False).

Note: You must specify False. Code anomalies with an Internal value of True cannot be modified.

- `Enabled` specifies whether the code anomaly is displayed in the Executive Report.
- `Caption` is the display name for the anomaly in the Executive Report.
- `ListName` is the name of the list of anomalous code constructs displayed in the Executive Report category of the HyperView Clipper tool.
- `Criterion` is the full path name of the criterion in the HyperView Advanced Search tool, including the tab name (General) and any folder names. For example, General: Coding Standards\MOVE Statements\Possible Data Padding.

You can display the anomaly caption in Japanese or Korean in the Executive Report by creating an entry for the anomaly in the file *workbench home\Language\[Jpn|Kor]\CodeDefects.xrc*. The entry has the form:

```
<String name="name"  
        listname="list name"  
        caption="translated display name"  
        description="description"  
>
```

where:

- `name` is the unique name of the code anomaly in the workbench (ID attribute of CodeDefects.xml entry).
- `listname` is the name of the list of anomalous code constructs displayed in the Executive Report category of the HyperView Clipper tool (ListName attribute of CodeDefects.xml entry).
- `caption` is the translated display name for the anomaly in the Executive Report.
- `description` contains a description of the entry.

Viewing Key Object Relationships

The Query Repository feature lets you hone in on key relationships of an object: all the programs that call another program, or all the programs that update a data store, for example. From the results, you can launch further queries without having to return to the Repository Browser. You can create printable relationship reports and export them to a variety of standard formats.

Use this feature to view:

- Dependent sources: all the sources files that depend on another file.
- Direct references: all the objects that another object references (outgoing) or is referenced by (incoming).
- Calls: all the objects that call another object.
- CRUD operations: all the data stores another object creates, reads, updates, or deletes.

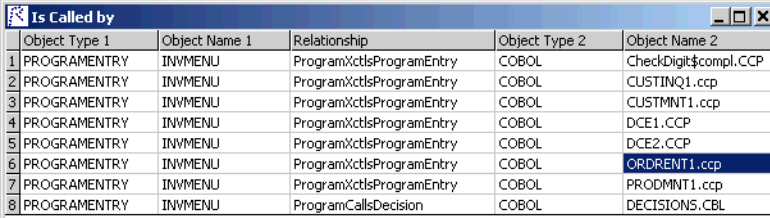
4-18 Verifying Files and Performing Post-Verification Tasks
Performing Post-Verification Program Analysis

- Used data stores: all the data stores used by another file.
- Used source files: all the source files (such as copybooks) used by another file.

To query object relationships:

- 1 In the Repository Browser, select the workspace, project, folder, files, or objects you want to view relationships for and choose **Query Repository:Relationship Type** in the **Edit** menu. A window similar to the one shown in Figure 4-7 appears.

Figure 4-7 *Query Repository Results Window*



	Object Type 1	Object Name 1	Relationship	Object Type 2	Object Name 2
1	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	CheckDigit\$compl.CCP
2	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	CUSTOMNT1.ccp
3	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	CUSTOMNT1.ccp
4	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	DCE1.CCP
5	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	DCE2.CCP
6	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	ORDRENT1.ccp
7	PROGRAMENTRY	INVMENU	ProgramXcclsProgramEntry	COBOL	PRODMNT1.ccp
8	PROGRAMENTRY	INVMENU	ProgramCallsDecision	COBOL	DECISIONS.CBL

- 2 To launch a query from the results, select one or more objects in the righthand column and choose **Query Repository:Relationship Type** in the **Edit** menu.
- 3 To create a printable relationship report, select the report you want to print and choose **Print** in the **File** menu. The printable report is displayed. In the printable report window, click **Page Setup** to specify print job options. Click **Print** to send the job to the printer. Click **Save** to save the report to a variety of standard formats.

Performing Post-Verification Program Analysis

Much of the performance cost of program verification for Cobol projects is incurred by the advanced program analysis options described in [step 11 on page 3-19](#). These features enable impact analysis, data flow analysis, and similar tasks.

You can improve verification performance by postponing some or all of advanced program analysis until after verification. Use the *post-verification program analysis* feature to collect the remaining program analysis information without having to reverify your entire legacy program.

To perform post-verification program analysis, select the project verification options for each program analysis feature you want to enable (Figure 3-4). In the Repository Browser, select the programs you want to analyze (or the entire project) and choose **Analyze Program** in the **Prepare** menu.

Note: Source files must have been verified with the **Enable Reference Reports** and **Enable HyperView** options selected in the Settings tab for workspace verification options, as described in [“Enabling Staged Parsing” on page 3-12](#).

The system collects the required information for each analysis feature you select. And it does so incrementally: if you verify a Cobol source file with the **Enable Data Element Flow** option selected, and then perform post-verification analysis with both that option and the **Enable Impact Analysis** option selected, only impact analysis information will be collected.

The same is true for information collected in a previous post-verification analysis. In fact, if all advanced analysis information has been collected for a program, the post-verification analysis feature simply will not start. In that case, you can only generate the analysis information again by reverifying the program.

Restrictions

There are a few cases in which analysis information is not collected incrementally:

- For PL/I programs, selecting **Resolve Decisions Automatically** causes information for **Enable Data Element Flow** also to be collected, whether or not it already has been collected. Select these options together when you perform program analysis.
- For Cobol programs, selecting *any* of the options dependent on the **Perform Program Analysis** option, whether during a previous ver-

ification or a previous program analysis, results in *none* of the information for those options being collected in a subsequent post-verification program analysis, with two exceptions: information *is* collected for **Enable Impact Report** and **Enable Execution Flow**, as long as you have not selected the options previously.

So if you verify a program with the **Resolve Decisions Automatically** option selected, then perform a subsequent program analysis with the **Perform Generic API Analysis** option selected, API analysis information is not collected. Whereas if you perform the subsequent program analysis with the **Enable Impact Report** option selected, impact analysis information *is* collected.

Similarly, if you perform program analysis with the **Enable Impact Report** option selected, then perform a subsequent program analysis with the **Enable Parameterization of Components** option selected, no parameterization information is collected. Whereas if you perform the subsequent program analysis with the **Enable Execution Flow** option selected, execution flow information is collected.

What this suggests is that, with the exception of **Enable Impact Report** and **Enable Execution Flow**, you should select all of the **Perform Program Analysis** options you are going to need for program analysis the *first* time you collect analysis information, whether during verification or subsequent post-verification analysis.

Enabling IMS Port Analysis

It is virtually impossible to determine from program code the database segments or screens an IMS program operates on. Only an application-wide analysis can trace PSB usage through the entire application call sequence.

To determine the types of database operation (insert, read, update, or delete) IMS programs perform, and to list in the browser each of the database segments or screens the operations are performed on, select the project or the individual source files for the programs and choose **IMS Analysis** in the **Prepare** menu.


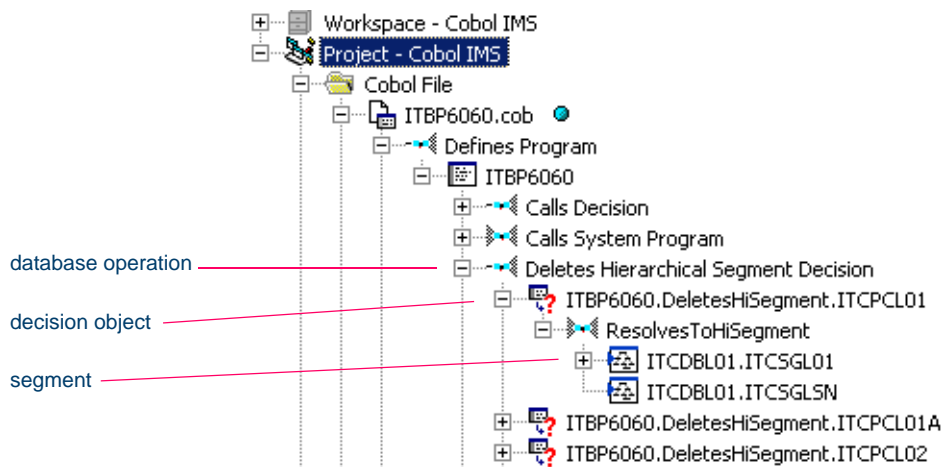
Figure 4-8 shows typical results. The objects marked with the  icon are abstract *decision objects*, indicating that the database operation, in this case, Deletes, has been resolved to multiple segments.

Figure 4-8 *IMS Port Analysis Results*



Mapping Root Programs to PSBs in JCL or System Definition Files

You must identify IMS “root programs” and corresponding PSBs in a JCL file for a batch application or a System Definition file for an online application. A root program is directly invoked by IMS with a list of PCBs as parameters. It can pass these PCBs as parameters in calls to other programs.

If you do not have actual JCL or System Definition files, you must create dummy ones. Analyzing the application without these files does nothing. Sample JCL and System Definition files follow:

Sample JCL file:

```
//imsbatch JOB
//S1 EXEC PGM=DFSRR00,REGION=2048K,
//
//
//PARM=(DLI,programe,psbname,7,0000,,0,,N,0,T,0,,N,,,N)
//
```

4-22 Verifying Files and Performing Post-Verification Tasks *Enabling IMS Port Analysis*

Sample System Definition file:

```
APPLCTN PSB=progrname  
TRANSACTION CODE=trnname
```

Verification Order for IMS Applications

If you verify an entire project for an IMS application, the workbench parses the source files in appropriate order, taking account of the dependencies between file types. Otherwise, verify source files in the following order:

- DBD files (for GSAM databases)
- MFS files
- PSB files
- Cobol or PL/I files

Note: For Cobol files, set the **Perform Program Analysis** and **Enable Data Element Flow** project verification options. For PL/I files, set the **Enable Data Element Flow** project verification option.

- JCL or System Definition files.

Reverifying Files in IMS Applications

If you reverify a root program, the JCL or System Definition file that maps the program to a PSB will be invalidated. If you reverify non-root programs, all call chains leading to them will be analyzed and any JCLs or System Definition files that start corresponding root programs will be invalidated. Make sure to reverify invalidated files.

Conversely, if you change a program-to-PSB mapping inside a JCL or System Definition file, or change the PSB file itself, make sure to reverify the mapped program before reverifying the JCL or System Definition file.

When you rerun IMS Analysis, it will process all complete call chains, starting from all reverified JCLs and System Definition files. You can limit the number of root programs that are re-analyzed in subsequent runs of IMS calls analysis by setting up System Definition files so that they reference one transaction only.

Note: If you rerun IMS Analysis without changing anything in the project, it will end with the warning “No information to perform IMS Analysis.” If you receive this message on the first run of IMS Analysis, make sure that all JCLs, System Definition files, and corresponding root programs have been verified, and that you have a call chain from root to every IMS-relevant program in the project (check for the strings “+IMSC” or “+IMSE” in the Environment attribute on the System tab of the properties for the program).

Generating Copybooks

RPG programs and Cobol programs that execute in the AS/400 environment often use copy statements that reference Database Description or Device Description files rather than copybooks. MCP Cobol programs use copy statements that reference DMSII DASDL files. If your application uses copy statements to reference these types of files, you need to verify the files and generate copybooks for the application before you verify program files.

Copybook generation takes place in two steps:

- For each database and device file object generated at verification, the system creates a *target copybook* object.
- For each target copybook object, the system creates one or more *physical copybooks*.

Options let you combine the steps or perform them separately.

To generate copybooks for RPG or AS/400 Cobol programs:

- 1 Verify Database Description and Device Description files, as described in [“Verifying Source Files” on page 4-1](#).
 - For each Database Description file, the system creates a *database file* object with the same name as the Database Description file.
 - For each Device Description file, the system creates a *device file* object with the same name as the Device Description file.

- 2 Before converting target copybooks to physical copybooks, set project options that control conversion behavior. In the Repository Browser, select the project and choose **Project Options** in the **Tools** menu. The Project Options window opens. Click the Generate Copybooks tab.

In the Generate Copybooks tab:

- Select **Convert Target Copybooks to Legacy Objects** if you want to generate target copybooks and convert them to physical copybooks in the same step (see [step 3](#)).
 - In the Target Copybooks Conversion pane, select **Assign Converted Files to the Current Project** if you want the system to create physical copybooks in the current project.
 - In the Conversion Conflicts pane, choose:
 - **Keep Old Legacy Objects** if you want the system not to overwrite existing physical copybooks.
 - **Replace Old Legacy Objects** if you want the system to overwrite existing physical copybooks.
 - Select **Remove Target Copybooks After Successful Conversion** if you want the system to remove target copybooks from the current project after physical copybooks are generated.
- 3 In the Repository Browser, select the project and choose:
 - **Generate Copybooks for Project** in the **Prepare** menu for AS/400 Cobol.
 - **Generate RPG Copybooks for Project** in the **Prepare** menu for RPG.

The system creates a *target copybook* object for each database and device file object, with a name of the form *database file.DBCOPY-BOOK* or *device file.DVCOPYBOOK*, in the Target Copybooks folder.

If you selected the **Convert Target Copybooks to Legacy Objects** option in [step 2](#), the system converts the target copybooks to physical copybooks, with names of the form *DD_OF_database file.CPY* or *DV_OF_device file.CPY*.

Note: To generate copybooks for given database or device file objects, select the objects and choose **Generate Copybooks** or **Generate RPG Copybooks**, as appropriate, in the **Prepare** menu.

Keep in mind that when you generate copybooks for an entire project, the system processes objects in the appropriate order, taking account of the dependencies between them. That is not the case when you generate copybooks for given objects.

Copybooks for database file objects must be generated before copybooks for device file objects. Copybooks for referenced objects must be generated before copybooks for referencing objects.

- 4 If you did not select the **Convert Target Copybooks to Legacy Objects** option in [step 2](#), select the target copybooks in the Repository Browser and choose **Convert to Legacy** in the **Prepare** menu. The system converts the target copybooks to physical copybooks, with names of the form `DD_OF_database_file.CPY` or `DV_OF_device_file.CPY`.

Note: After generating copybooks, you can generate screens for AS/400 Cobol device file objects by selecting the objects in the Repository Browser and choosing **Generate Screens** in the **Prepare** menu.

To generate copybooks for MCP Cobol programs:

- 1 Before converting target copybooks to physical copybooks, set project options that control conversion behavior. In the Repository Browser, select the project and choose **Project Options** in the **Tools** menu. The Project Options window opens. Click the Generate Copybooks tab.

In the Generate Copybooks tab:

- Select **Generate After Successful Verification** if you want to generate target copybooks automatically on verification of DM-SII DASDL files.

- Select **Convert Target Copybooks to Legacy Objects** if you want to generate target copybooks and convert them to physical copybooks in the same step.
 - In the Target Copybooks Conversion pane, select **Assign Converted Files to the Current Project** if you want the system to create physical copybooks in the current project.
 - In the Conversion Conflicts pane, choose:
 - **Keep Old Legacy Objects** if you want the system not to overwrite existing physical copybooks.
 - **Replace Old Legacy Objects** if you want the system to overwrite existing physical copybooks.
 - Select **Remove Target Copybooks After Successful Conversion** if you want the system to remove target copybooks from the current project after physical copybooks are generated.
- 2 Verify DMSII DASDL files, as described in [“Verifying Source Files” on page 4-1](#). For each DMSII DASDL file, the system creates a *DMSII database file* object with the same name as the DMSII DASDL file.

If you selected the **Generate After Successful Verification** option in [step 1](#), the system creates a *target copybook* object for each database file object, with a name of the form *database file.DBCOPYBOOK*, in the Target Copybooks folder. If you also selected the **Convert Target Copybooks to Legacy Objects** option in [step 1](#), the system converts the target copybooks to physical copybooks, with names of the form *DV_OF_device file.CPY*.

- 3 If you did not select the **Generate After Successful Verification** option in [step 1](#), select the project in the Repository Browser and choose **Generate Copybooks for Project** in the **Prepare** menu.

The system creates a *target copybook* object for each database file object, with a name of the form *database file.DBCOPYBOOK*, in the Target Copybooks folder.

If you selected the **Convert Target Copybooks to Legacy Objects** option in [step 1](#), the system converts the target copybooks to physical copybooks, with names of the form `DD_OF_database file.CPY`.

Note: To generate copybooks for given database file objects, select the objects and choose **Generate Copybooks** in the **Prepare** menu.

Keep in mind that when you generate copybooks for an entire project, the system processes objects in the appropriate order, taking account of the dependencies between them. That is not the case when you generate copybooks for given objects. Copybooks for referenced objects must be generated before copybooks for referencing objects.

- 4 If you did not select the **Convert Target Copybooks to Legacy Objects** option in [step 1](#), select the target copybooks in the Repository Browser and choose **Convert to Legacy** in the **Prepare** menu. The system converts the target copybooks to physical copybooks with names of the form `DD_OF_database file.CPY`.

What's Next?

You need to correct any errors in your application before the parser can generate a complete object model. You can correct coding errors in the system editor, as described in *Getting Started*. Use the tools described in the next chapter to identify missing or unneeded program elements.

4-28 Verifying Files and Performing Post-Verification Tasks
What's Next?

Identifying Missing or Unneeded Program Elements



Reference Reports identify missing or unneeded files or objects in legacy applications. The reports are based on the parser's analysis of references in verified source:


- An Unresolved Report identifies missing application elements.
- An Unreferred Report identifies unreferenced application elements.
- A Cross-reference Report identifies all application references.

The Orphan Analysis tool lets you analyze and resolve objects that do not exist in the reference tree for any top-level object, so-called *orphans*. Orphans can be removed from a system without altering its behavior.

Using Reference Reports

When you verify a legacy application, the parser generates a model of the application that describes the objects in the application and how they interact. If a Cobol source file contains a COPY statement, for example, the system creates a relationship between the file and the Cobol copy-

5-2 Identifying Missing or Unneeded Program Elements
Using Reference Reports

book referenced by the statement. If the copybook doesn't exist in the repository, the system flags it as missing by listing it with a  symbol in the tree view of the Repository Browser. Reference Reports let you track these kinds of referential dependencies in verified source.

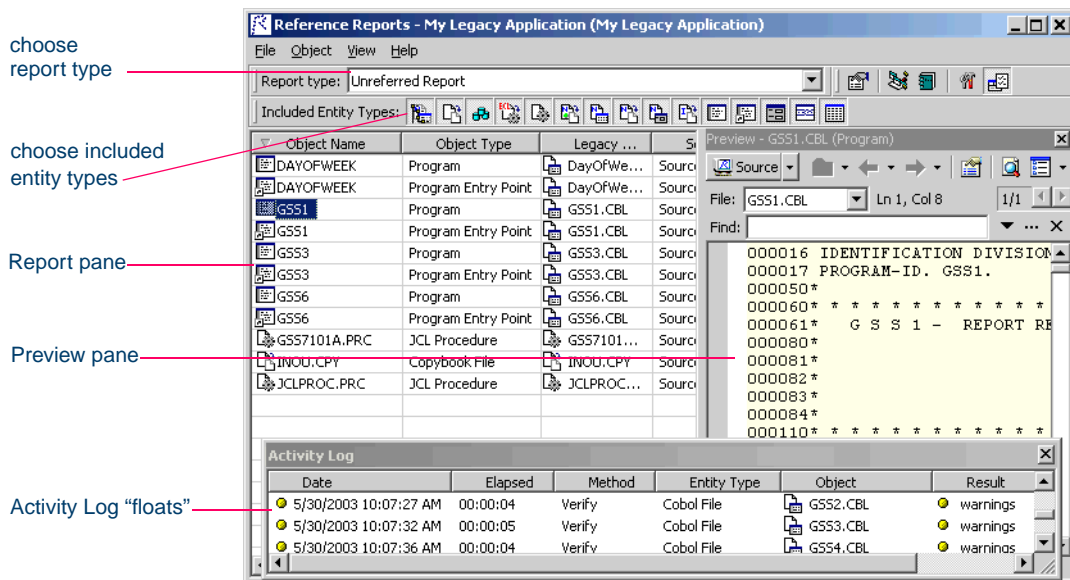
Generating Reference Reports

You generate Reference Reports for the current project. You can restrict the report to that project, or include references from other projects.

To generate reference reports:

- 1 In the Repository Browser, select the project for the Reference Report and choose **Reference Reports** in the **Prepare** menu. An empty Reference Reports window opens on top of the Modernization Workbench main window.
- 2 In the **Report type** drop-down, choose the Reference Report type. Figure 5-1 shows an Unreferred Report window. The windows for the other reports are similar.

Figure 5-1 Unreferred Report Window



Working with Reference Reports

The Reference Report window consists of a Report pane, Preview pane, and Activity Log. The Report pane is always displayed. Select the appropriate choice in the **View** menu to show/hide the Preview pane and Activity Log.

Report Pane

The Report pane displays the objects in the Reference Report and their relationships. Table 5-1 describes the columns in the Report pane.

Table 5-1 *Reference Report Columns*

Column Name	Report Types	Description
Object Name	All	The name of the unresolved, unreferenced, or cross-referenced object.
Object Type	All	The entity type of the unresolved, unreferenced, or cross-referenced object.
Legacy Object	Unreferred Report, Cross-reference Report	The source file that contains the unreferenced or cross-referenced object.
Source	Unreferred Report, Cross-reference Report	The location in the workspace folder of the source file that contains the unreferenced or cross-referenced object.
Referred by	Unresolved Report, Cross-reference Report	The name of the referring object.
Referring Object Type	Unresolved Report, Cross-reference Report	The entity type of the referring object.

5-4 Identifying Missing or Unneeded Program Elements
Using Reference Reports

Table 5-1 *Reference Report Columns* (continued)

Column Name	Report Types	Description
Relationship	Unresolved Report, Cross-reference Report	The relationship between the unresolved or cross-referenced object and the referring object.
Object Description	All	The description of the unresolved, unreferenced, or cross-referenced object entered by the user on the Description tab of the Object Properties window.

Sorting Entries Click a column heading in the Report pane to sort the report entries by that column.

Sizing Columns Grab-and-drag the border of a column heading to increase or decrease the width of the column.

Viewing Properties Select an object in the Report pane and choose **Properties** in the **Object** menu to display a set of tabs with object properties. For usage information, see *Getting Started* in the workbench documentation set.

Restricting the Types of Objects Included in the Report Choose **Options** in the **View** menu to open the Reference Reports Options window, where you can select the types of entities included in the report.

Tip: You can also click the icons on the Included Entity Types toolbar to select and deselect included types. You can hide the toolbar by deselecting **Selection Toolbar** in the **View** menu.

Restricting References to the Current Project Choose **Restrict References to Project** in the **View** menu to limit the report to references in the current project.

Preview Pane

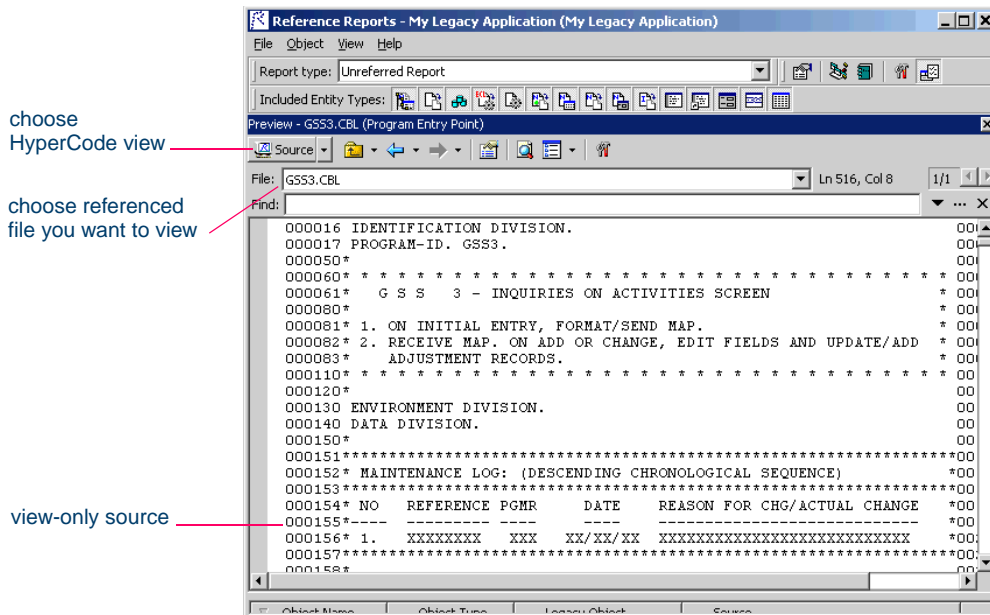
The Preview pane lets you browse information about the object selected in the lefthand column of the Report pane. The information available depends on the type of object selected. You see only source code for a copybook, for example, but full HyperCode for a program.

Note: “HyperCode” is shorthand for the information displayed in the Modernization Workbench HyperView tool. For HyperView usage information, see *Analyzing Programs* in the workbench documentation set.

Choose **Preview** in the **View** menu to open the Preview pane (Figure 5-2). Select an object in the Object Name column of the Report pane to view it in the Preview pane. Choose the information you want to view for the object from the **Source** drop-down. Use the search facilities to navigate to the source you want to view.

Note: The Preview pane shows a view-only copy of application source.

Figure 5-2 Preview Pane for Program Object



Detecting System Programs

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. You need to identify system programs to the parser so that it can create relationships for them with their referencing files.

The most convenient way to detect the system programs an application uses is to run an Unresolved Report after verification. Once you learn from the report which system programs are referenced, you can identify them to the parser in the System Programs tab (Figure 3-7) and reverify any *one* of their referencing source files.

The Reference Report tool lets you bring up the System Programs tab while you are in the tool itself. Use the **System Programs** choice in the **View** menu to display the tab, then follow the instructions in [step 2 on page 3-26](#) to identify system programs to the parser.

Exporting Reference Reports

Choose **Report** in the **File** menu to display a printable Reference Report. In the printable report, click **Print** to print the report. Click **Save** to export the report to HTML, Excel, RTF, Word, or formatted text.

Using the Orphan Analysis Tool

Use the Orphan Analysis tool to determine whether an object exists in the reference tree for a top-level program object. An object that does not exist in the reference tree for *any* top-level object is called an *orphan*. Orphans can be removed from a system without altering its behavior.

What's the difference between an orphan and an unreferenced object?

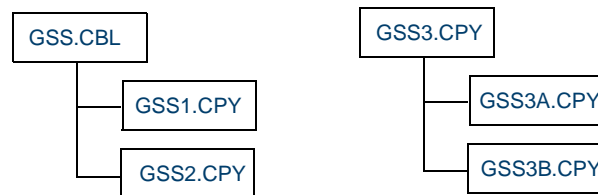
- All unreferenced objects are orphans.
- Not every orphan is unreferenced.

Consider the example shown in Figure 5-3 below. An unreferred report shows that the copybook GSS3.CPY is not referenced by any object in the project. Meanwhile, a cross-reference report shows that GSS3.CPY

references GSS3A.CPY and GSS3B.CPY. These copybooks do *not* appear in the unreferred report because they *are* referenced by GSS3.CPY.

Only orphan analysis will show that the two copybooks are not in the reference tree for the GSS program and, therefore, can be safely removed from the project.

Figure 5-3 *Unreferenced and Orphan Objects*



GSS3.CPY is unreferenced, while GSS3A.CPY and GSS3B.CPY are referenced. But all three copybooks are orphans!

Generating Orphan Analysis Reports

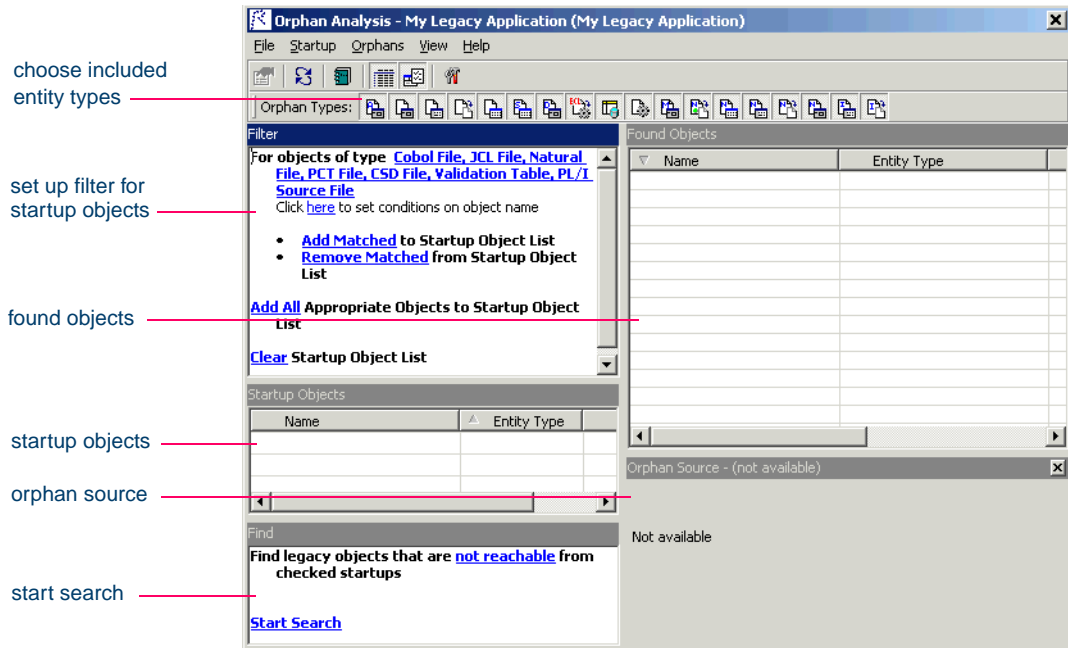
Orphan Analysis reports show whether an object exists in the reference tree for a top-level object: whether it can be *reached* in the tree for one or more *startup* objects. You generate Orphan Analysis reports for the current project. Unlike Reference Reports, the report cannot include references from other projects.

To generate orphan analysis reports:

- 1 In the Repository Browser, select the project for orphan analysis and choose **Orphan Analysis** in the **Prepare** menu. An empty Orphan Analysis window opens on top of the Modernization Workbench main window (Figure 5-4).

5-8 Identifying Missing or Unneeded Program Elements
Using the Orphan Analysis Tool

Figure 5-4 Orphan Analysis Window (Empty)

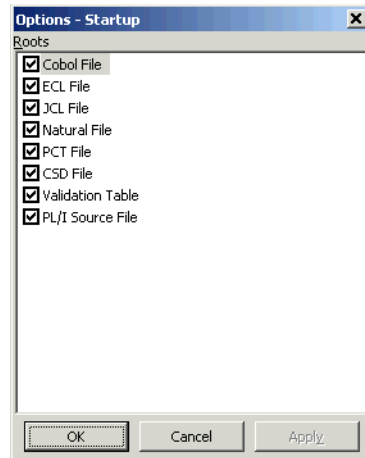


- 2 In the **Orphan Types** tool bar, choose the types of entities to include in the report by clicking the appropriate icon. Place your cursor over an icon for a moment to display a tool tip that describes the icon.

Tip: You can also set up the analysis in the Options window. Choose **Options** in the **View** menu to open the Options window.

- 3 In the Filter pane, set up a search filter for the startup objects in the analysis. You can filter on entity type, entity name, or both:
 - To filter on entity type, click on the link for an entity type in the **For objects of type** field, or if no entities are displayed, click on the **[Entity Types]** link. The Startup Options window opens (Figure 5-5). Select the types of entities you want to filter on and click **OK**.

Figure 5-5 *Startup Options Window*



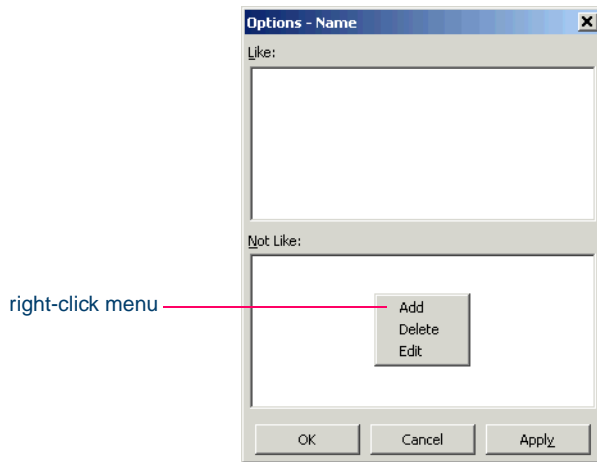
- To filter on entity name, click the [here](#) link. The Name Options window opens (Figure 5-6). The recognized name matching patterns for orphan analysis are listed in the **Like** and **Unlike** fields. Select the patterns for the entities you want to filter on and click **OK**

Add a pattern by right-clicking in the **Like** or **Unlike** field and choosing **Add** in the pop-up menu. The system displays an empty text field next to a selected check box. Enter the text for the pattern and click outside the field. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Edit a pattern by selecting it and choosing **Edit** in the right-click menu. Delete a pattern by selecting it and choosing **Delete** in the right-click menu.

5-10 Identifying Missing or Unneeded Program Elements
Using the Orphan Analysis Tool

Figure 5-6 *Name Options Window*



- 4 Click the **Add Matched** link to apply the filter. The matched objects are displayed in the **Startup Objects** pane. Place a check mark next to each startup object you want to use in the analysis.

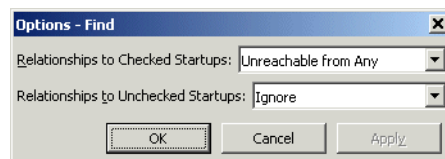
Tip: To place a check mark next to all the matched objects, choose **Select All** in the **Startup** menu, then **Mark as Startup** in the **Startup** menu. To uncheck all the matched objects, choose **Unmark** in the **Startup** menu.

To select (but not mark as startup) all objects of a given type, choose **Select: Object Type** in the **Startup** menu. You can remove a matched object from the list by selecting it and choosing **Remove from List** in the **Startup** menu.

To add all the objects in the project that would be appropriate startup objects to the list, click the **Add All** link in the Filter pane. You can filter this list by clicking the **Remove Matched** link in the Filter pane. To clear the list, click **Clear** in the Filter pane.

- 5 Click the link in the **Find legacy objects...** text at the top of the pane. The Find Options dialog opens (Figure 5-7). Define the terms of the orphan search by selecting the appropriate choice in the **Relationships to Checked Startups** drop-down, the **Relationships to Unchecked Startups** drop-down, or both. Click **OK** when you are done.

Figure 5-7 *Find Options Dialog*

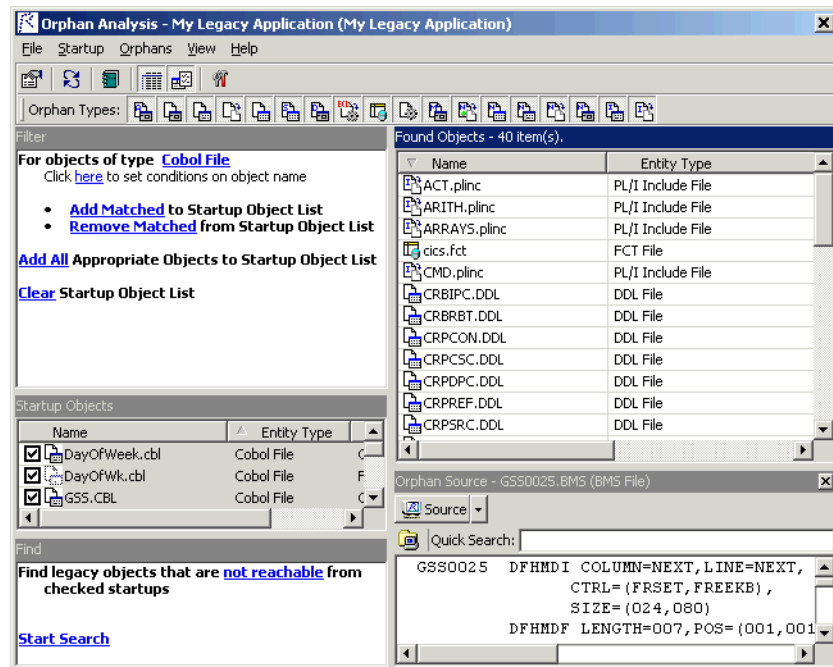


- 6 Click the **Start Search** link in the Startup pane to start the search. The system shows the results of the search in the Found Objects pane (Figure 5-8).
 - 7 Choose **Report View** in the **View** menu to show the entity type and source file of the orphan in the Found Objects pane. Deselect **Report View** to show the list of orphans only.
 - 8 Select an orphan to show its source code in the Orphan Source pane. The Orphan Source pane is similar to the Reference Report window Preview pane. For more information, see [“Preview Pane” on page 5-5](#).
- Tip:** Select a startup object or orphan and choose **Properties** in the right-click menu to display a set of tabs with object properties. For usage information, see *Getting Started* in the Modernization Workbench document set.
- 9 Select an orphan in the Found Objects pane and choose **Exclude from Project** to remove the orphan from the project but not the workspace. Choose **Delete from Workspace** to remove the orphan from the workspace.

5-12 Identifying Missing or Unneeded Program Elements
Using the Orphan Analysis Tool

Tip: You can hide the Orphan Source pane and Orphan Types tool bar by selecting the appropriate choice in the **View** menu. You can display the Activity Log for the Modernization Workbench session by selecting **Activity Log** in the **View** menu.

Figure 5-8 Orphan Analysis Window (Populated)



Exporting Orphan Analysis Reports

Choose **Save Report As** in the **File** menu to display a printable Orphan Analysis report. In the printable report, click **Print** to print the report. Click **Save** to export the report to HTML, Excel, RTF, Word, or formatted text.

What's Next?

Now that you have identified missing program elements in your project and removed any orphans, you can drill down deeper into the application to locate potential problems. The tool described in the next chapter lets you identify and resolve dynamic calls and other relationships that the parser cannot resolve from static sources.

5-14 Identifying Missing or Unneeded Program Elements

What's Next?

Resolving Decisions



You need to have a complete picture of the control and data flows in a legacy application before you can diagram and analyze the application. The parser models the control and data transfers it can resolve from static sources. Some transfers, however, are not resolved until run time. *Decision resolution* lets you identify and resolve dynamic calls and other relationships that the parser cannot resolve from static sources.

Understanding Decisions


A *decision* is a reference to another object, a program or screen, for example, that is not resolved until run time. Consider a Cobol program that contains the following statement:

```
CALL 'NEXTPROG' .
```

The Modernization Workbench parser models the transfer of control to program NEXTPROG by creating a Calls relationship between the original program and NEXTPROG.

But what if the statement read this way instead:

```
CALL NEXT.
```

where NEXT is a field whose value is only determined at run time. In this case, the parser creates a Calls relationship between the program and an abstract *decision object* called *PROG.CALL.NEXT*, and lists the decision object with a  icon in the tree view of the Repository Browser.

Manually Resolving Decisions The Decision Resolution tool creates a list of such decisions and helps you navigate to the program source code that indicates how the decision should be resolved. You may learn from a declaration or MOVE statement, for example, that the NEXT field takes either the value NEXTPROG or ENDPROG at run time. In that case, you would resolve the decision manually by telling the system to create *resolves to* relationships between the decision and the programs these literals reference.

Automatically Resolving Decisions Of course, where there are hundreds or even thousands of such decisions in an application, it may not be practical to resolve each decision manually. In these situations, you can use the *autoresolve* feature to resolve decisions automatically.

The Decision Resolution tool analyzes declarations and MOVE statements, and any other means of populating a decision point, to determine the target of the control or data transfer. The tool may not be able to auto-resolve every decision, or even every decision completely, but it should get you to a point where you can complete decision resolution manually.

Resolving Decisions Manually

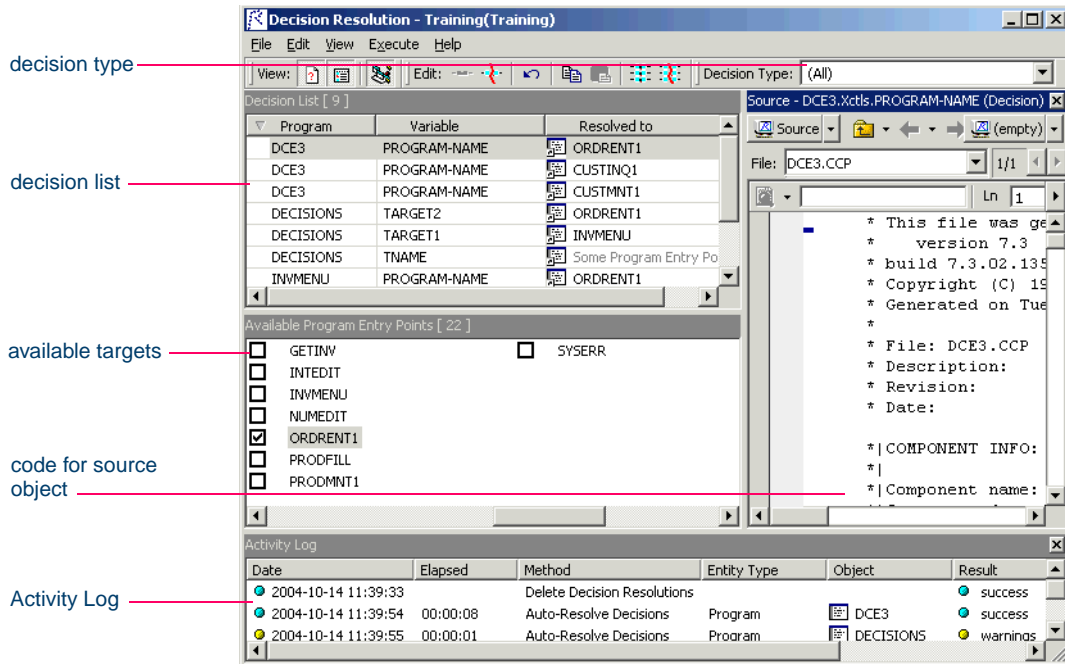
The Decision Resolution tool generates decisions for programs in the current project. You can limit the targets for decision resolution to objects in the current project, or include objects in other projects as potential targets.

To resolve decisions manually:

- 1 In the Repository Browser, select the project for which you want to resolve decisions and choose **Resolve Decisions** in the **Prepare**

menu. The Decision Resolution tool window opens on top of the Modernization Workbench main window (Figure 6-1).

Figure 6-1 *Decision Resolution Tool Window*



2 In the **Decision Type** drop-down, choose the type of decision you are interested in. A list of decisions of that type is displayed in the Decision List pane:

- The Program column lists the names of programs that contain decisions.
- The Variable column lists the program variables that require decisions.
- The Completed column indicates whether the decision has been resolved. A plus sign (+) means Yes.
- The Unreachable column lists decisions in dead code.

- The Resolved to column lists the target objects each variable resolves to: one or more entry points, for example. An unresolved decision contains the grayed-out text *Some Object*.

Tip: Click a column heading in the Decision List pane to sort the report entries by that column. Grab-and-drag the border of a column heading to increase or decrease the width of the column.

- 3 The Available *Targets* pane lists the targets in the workspace for the selected decision type. In the **View** menu, choose **Restrict to Current Project** to limit the targets to objects in the current project.
- 4 Select an entry in the Decision List pane to navigate to the decision in the Source pane. The Source pane is similar to the Reference Report window Preview pane. For more information, see [“Preview Pane” on page 5-5](#).
- 5A To resolve decisions to available targets, select one or more entries in the Decision List pane and place a check mark next to one or more target objects in the Available *Targets* pane.

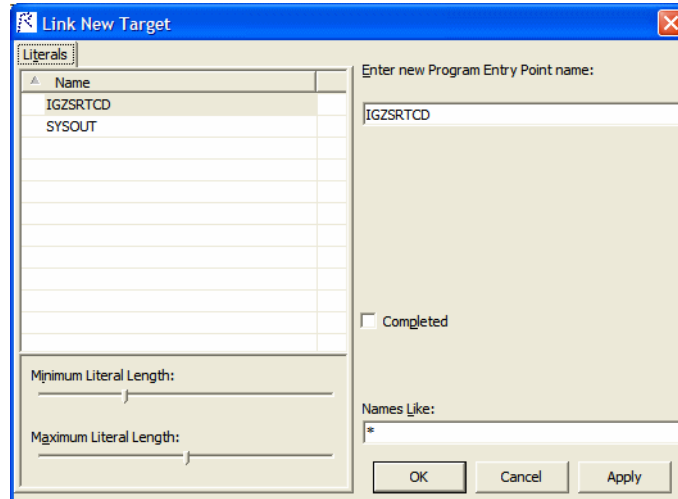
Tip: To select a range of entries, hold down the Shift key, click the first item in the range, then click the last item in the range. To select entries that are not in a range, hold down the Control key, then click each entry you want to select.

If you link an entry to multiple targets, the Decision Resolution tool creates as many entries as there are targets.

Tip: If you are linking an entry to multiple targets, you can save time by selecting the targets and choosing **Link Selected Targets** in the **Edit** menu. Use the **Copy** choice in the **Edit** menu to copy selected targets to the clipboard, then the **Paste** choice to link the targets to an entry.

- 5B To resolve decisions to targets not in the workspace, select one or more entries in the Decision List pane and choose **Link New Target** in the **Edit** menu. The Link New Target window opens (Figure 6-2).

Figure 6-2 *Link New Target Window*



In the Link New Target window, enter the name of the new target in the field on the righthand side of the window, or populate the field by clicking a literal in the list of program literals on the Literals tab. You can filter the list by using:

- The **Minimum Literal Length** slider to specify the minimum number of characters the literal can contain.
- The **Maximum Literal Length** slider to specify the maximum number of characters the literal can contain.
- The **Names Like** field to enter a matching pattern for the literal. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Place a check mark next to **Completed** if you want the resolution to be marked as completed. When you are satisfied with your entry, click **OK**.

Tip: Before saving, you can undo changes by choosing **Undo all changes** in the **Edit** menu.

- 6 To delete a decision resolution, remove the check mark next to the target object.
- 7 In the **File** menu, choose **Save** to save the decision resolutions in the repository.

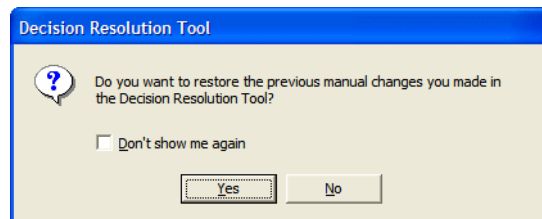
Tip: You can hide the Source pane or Activity Log window by clicking the close box in the upper righthand corner. Select the appropriate choice in the **View** menu to show the Source pane or Activity Log window again.

Restoring Manually Resolved Decisions

Reverifying a file invalidates all of its objects, including its manually resolved decisions. The *decision persistence* feature lets you restore manually resolved decisions when you return to the Decision Resolution tool.

After reverifying a file for which you have manually resolved decisions, reopen the Decision Resolution tool. A dialog box prompts you to restore manually resolved decisions (Figure 6-3). Click **Yes** if you want to restore the decisions. Click **No** otherwise.

Figure 6-3 *Restore Decisions Dialog Box*



Tip: Place a check mark next to **Don't show me again** if you want the Decision Resolution tool to open without prompting you to restore manually resolved decisions. In the Decision Resolution Tool tab of the User Preferences window, place a check mark next to **Ask before restoring previous manual changes** if you want to be prompted again.

Resolving Decisions Automatically

You can autoresolve decisions during verification by setting the **Resolve decisions automatically** option in the project verification options, as described in [“Setting Project Verification Options” on page 3-16](#). For program objects, you can also autoresolve decisions after verification, in the Modernization Workbench main window, as described below. Only a master user can autoresolve decisions in a multiuser environment.

To resolve decisions automatically:

- 1 In the Modernization Workbench Repository pane, select the project, folder, or files you want to autoresolve decisions in and choose **AutoResolve Decisions** in the **Prepare** menu.

Note: The Decision Resolution tool cannot autoresolve every decision. The target name may be read from a data file, for example. The Autoresolve feature is available only for programs. It is not available for Natural programs.

Exporting Decision Resolution Reports

Choose **Report** in the **File** menu to display a printable Decision Resolution report. In the printable report, click **Print** to print the report. Click **Save** to export the report to HTML, Excel, RTF, Word, or formatted text.

What’s Next?

You should now have a completely verified and resolved workspace. That means you’re ready to start analyzing your application, as described in *Analyzing Projects* in the workbench documentation set.

6-8 Resolving Decisions
What's Next?

Identifying Interfaces for Generic API Analysis



Use the Generic API Analysis feature if your legacy program calls an unsupported API to interface with a database manager, transaction manager, or similar external facility. In this call, for example:

```
CALL 'XREAD' using X
```

where X evaluates to a table name, the call to XREAD is of less interest than its parameter, the table the called program reads from. But because the parser does not recognize XREAD, only the call is modeled in the workbench repository.

You enable the Generic API Analysis feature by identifying unsupported APIs and their parameters in the file *workbench home\Data\Legacy.xml*. When you verify the application, select **Perform Generic API Analysis** in the project verification options ([step 11 on page 3-19](#)) to instruct the parser to define relationships with the objects passed as parameters in the calls, in addition to relationships with the unsupported APIs themselves.

This appendix shows you how to identify the programs and parameters to the parser before verifying your application. You can specify both ob-

ject and construct model information, and create different relationships or entities for the same parameter in a call.

The specification requires a thorough understanding of the Modernization Workbench repository models. Only the predefined definitions described below are guaranteed to provide consistent data to workbench databases. For background on the repository models, see the *Software Development Kit*, available from support services.

Identifying Unsupported API Calls to the Parser

Follow the instructions in this section to identify unsupported API calls to the parser. For each call, you need to define an entry in *workbench home*\Data\Legacy.xml that specifies, at a minimum:

- The name of the called program and the method of invocation in the <match> tag.
- The program control flow in the <flow> tag, and the direction of the data flow through the parameters of interest in the <param> subtags.
- How to represent the call in the object model repository in the <rep> tag, and in the construct model repository in the <hc> tag.

Use the optional <vars> tag to extract values of a specified type, size, and offset from a parameter for use in a <rep> or <hc> definition.

Most repository entities can be represented in a <rep> or <hc> definition with the predefined patterns in *workbench home*\Data\Legacy.xml.api. These patterns do virtually all of the work of the specification for you, supplying the relationship of the entity to the called program, its internal name, and so forth.

[“Using Predefined Patterns” on page A-8](#) describes the syntax for specifying predefined patterns in a <rep> or <hc> definition. Consult Legacy.xml.api for supported patterns and for required parameters and values.

Note: You can find samples of Generic API usage in *workbench home*\Data\Legacy.xml. Consult support services for additional assistance.

To identify unsupported API calls to the parser:

- 1 Open the file *workbench home*\Data\Legacy.xml in an editor.
- 2 Locate the <GenericAPI> section for the language and dialect you use.
- 3 Create entries for each unsupported API call, of the form:

```
<APIEntry name='entry name'>
  <match stmt='method of invocation'>
    <name value='program name to be matched' />
    <name value='alternate program name' />
  </match>
  <flow halts='yes|no'>
    <param index='index of parameter'
      usage='r|w|rw' />
  </flow>
  <vars>
    <arg var='variable name'
      param='index of parameter'
      type='variable type'
      offset='offset of field in bytes'
      bitoffset='offset of field in bits'
      size='size of field in bytes'
      bitsize='size of field in bits' />
    <!-- PL/I-specific -->
    len='size of char or bit string field'
    mode='binary|decimal' [numeric field]
    scale='scale of fixed point field'
    prec='precision of numeric field'
    varying='yes|no' />
  </vars>
  <rep>
    <entity type='type of entity'
      name='name of entity'
      produced='yes|no' />
    <attr name='name of attribute'
      value='value of attribute'
      join='delimiter' />
    <cond if-cond='expression'
      value='expression' />
  </entity>
```

A-4 Identifying Interfaces for Generic API Analysis
Identifying Unsupported API Calls to the Parser

```
<rel name='name of relationship'  
      decision='yes|no' />  
  <target type='entity type of right end'  
        name='name of left end'  
        produced='yes|no' />  
    <attr name='name of attribute'  
        value='value of attribute'  
        join='delimiter' />  
  <source type='entity type of left end'  
        name='name of left end'  
        produced='yes|no' />  
    <attr name='name of attribute'  
        value='value of attribute'  
        join='delimiter' />  
  <cond if-cond='expression'  
        value='expression' />  
</rel>  
</rep>  
<hc>  
  <attr name='name of attribute'  
        value='value of attribute'  
        join='delimiter' />  
</hc>  
</APIEntry>
```

where the *name* attribute of the <API Entry> tag is the name of the entry (used for error diagnostics only) and the remaining tags are as described in the following sections.

- 4 Select **Perform Generic API Analysis** in the Verification tab of the Project Options window ([step 11 on page 3-19](#)).
- 5 Verify the project.

Using the <match> Tag

The *stmt* attribute of the <match> tag identifies the method of invocation: a CALL, LINK, or XCTL statement. The *value* attribute of the <name> subtag identifies the name of the program to be matched. It can also be used to specify an alternative name for the entry.

Note: The name of the program to be matched must be unique in the <GenericAPI> section. If names are not unique, the parser uses the last entry in which the name appears.

Example:

```
<match stmt="CALL">  
  <name value="XREAD"/>  
</match>
```

Using the <flow> Tag

The <flow> tag characterizes the program control flow. The *halts* attribute of the <flow> tag specifies whether the calling program terminates after the call:

- yes, if control is not received back from the API.
- no (the default), if the calling program returns normally.

The <param> subtag identifies characteristics of the call parameters. Attributes are:

- *index* is the index of the parameter that references the item of interest, beginning with 1. Use an asterisk (*) to specify all parameters not specified directly.
- *usage* specifies the direction of the data flow through the parameter: r for input, w for output, rw for input/output. Unspecified parameters are assumed to be input/output parameters.

Note: *halts* is supported only for call statements. For PL/I, input parameters are treated as input/output parameters.

Example:

```
<flow halts='no'>  
  <param index='1' usage='r' />  
  <param index='2' usage='r' />  
  <param index='3' usage='rw' />  
  <param index='*' usage='rw' />  
</flow>
```

Using the <vars> Tag

Use the <vars> tag to extract values of a specified type, size, and offset from a call parameter. You can then refer to the extracted values in the <rep> and <hc> tags using the %*var_name* notation. For more information, see [“Using Expressions” on page A-12](#).

The <arg> subtag characterizes the argument of interest. Attributes are:

- *var* specifies the variable name.
- *param* specifies the index of the parameter.
- *type* specifies the variable type.
- *offset* specifies the offset of the field in bytes.
- *bitoffset* specifies the offset of the field in bits.
- *size* specifies the size of the field in bytes.
- *bitsize* specifies the size of the field in bits.

Additional attributes for PL/I are:

- *len* specifies the size of a character or bit string field.
- *mode* specifies the binary or decimal mode for a numeric field.
- *scale* specifies the scale of a fixed-point numeric field.
- *prec* specifies the precision of a fixed-point or floating-point numeric field.
- *varying* specifies whether a bit string variable is encoded as a varying-length string in the structure (yes or no, the default).

Supported data types are described in the language-specific sections below.

Example:

Suppose a call to a database-entry API looks like this:

```
CALL 'DBENTRY' USING DB-USER-ID
                    DB-XYZ-REQUEST-AREA
                    XYZ01-RECORD
                    DB-XYZ-ELEMENT-LIST.
```

If the second parameter contains a 3-character table name in bytes 6-8, the following definition extracts the name for use as the right end of a relationship:

```
<vars>
  <arg var='TableName'
        param='2'
        type='auto'
        offset='5'
        size='3' />
</vars>
<rep>
  <rel>
    <target type='TABLE'
            name='%TableName' />
    .
    .
    .
  </rel>
</rep>
```

Cobol-Specific Usage

For Cobol, use the following data types in the <vars> tag:

- *data* extracts a subarea of the parameter as raw byte data. You must specify the size and offset.
- *auto* automatically determines the type of the variable, using the offset. If that is not possible, *auto* looks for a matching variable declaration and uses its type. You must specify the offset.
- *int* behaves as *auto*, additionally checking that the resulting value is a valid integer and converting it to the canonical form. Offset defaults to 0.

Note: *bitoffset* and *bitsize* are currently not supported. *auto* is not always reliable. Use *data* whenever possible.

PL/I-Specific Usage

For PL/I, use the following data types in the <vars> tag:

- *data* extracts a subarea of the parameter as raw byte data. You must specify the size and offset.
- *char* specifies a character variable, with attribute *varying* if the string is encoded as a varying-length string in the structure. Offset defaults to 0, and size is specified via the required *len* attribute, which specifies the string length.
- *bit* specifies a bit string variable, with attribute *varying* if the string is encoded as a varying-length string in the structure. Offset defaults to 0, and size is specified via the required *len* attribute, which specifies the string length in bits.
- *fixed* specifies a fixed-point numeric variable, with attributes *mode* (binary or decimal, the default), *scale* (default 0), and *prec* (precision, default 5). Offset defaults to 0, and size is overridden with a value calculated from the type.
- *float* specifies a floating-point numeric variable, with attributes *mode* (binary or decimal, the default) and *prec* (precision, default 5). Offset defaults to 0, and size is overridden with a value calculated from the type.

Note: Do not use *bitoffset* and *bitsize* for types other than bit string.

Using the <rep> Tag

Use the <rep> tag to represent the API call in the object model repository. You can use predefined or custom patterns to specify the relationship of interest. *Expressions* let you extract parameter values and context information for use in specifications of entity or relationship characteristics, as described in [“Using Expressions” on page A-12](#).

Using Predefined Patterns

Most repository entities can be represented with the predefined patterns in *workbench home\Data\Legacy.xml.api*. These patterns do virtually all of the work of the specification for you, supplying the relationship of the entity to the called program, its internal name, and so forth. They are guaranteed to provide consistent data to workbench databases.

To specify a predefined pattern, use the pattern name as a tag (<tip-file>, for example) anywhere you might use a <rel> tag. If the predefined pat-

tern is specified at the top level of the entry, the parser creates a relationship with the calling program. If the predefined pattern is nested in an entity specification, the parser creates a relationship with the parent entity.

Each pattern has parameters that you can code as XML attributes or as subtags. So:

```
<transaction name='%2' params='' hc-kind='dpsSETRX' />
```

Is equivalent to:

```
<transaction params=''>
  <name value='%2' />
  <hc-kind value='dpsSETRX' />
</transaction>
```

Use the subtag method when a parameter can have multiple values:

```
<file filename=' %2' data-record='%3'>
  <action switch-var='%op'>
    <case eq='1' value='Reads' />
    <case eq='2' value='Reads' />
    <case eq='4' value='Updates' />
    <case eq='28' value='Inserts' />
  </action>
  <hc-kind switch-var='%op'>
    <case eq='1' value='fcssRR' />
    <case eq='2' value='fcssRL' />
    <case eq='4' value='fcssWR' />
    <case eq='28' value='fcssAW' />
  </hc-kind>
</file>
```

Check Legacy.xml.api for further details of predefined pattern usage and for required parameters and values.

Using Custom Patterns

Use custom patterns only when a predefined pattern is not available. Custom patterns are not guaranteed to provide consistent data to workbench databases.

Using the <entity> Subtag The <entity> subtag represents an entity in the object model repository. Attributes are:

A-10 Identifying Interfaces for Generic API Analysis
Identifying Unsupported API Calls to the Parser

- *type* specifies the entity type.
- *name* specifies the entity name.
- *produced* optionally indicates whether the entity is *extracted*, in which case it is deleted from the repository when the source file is invalidated (yes or no, the default).

Use the <attr> subtag to specify entity attributes. Attributes of the subtag are:

- *name* specifies the attribute name.
- *value* contains an expression that defines the attribute value.
- *join* specifies the delimiter to use if all possible variable values are to be joined in a single value.

Use the <cond> subtag to specify a condition, as described in [“Understanding Conditions” on page A-18](#).

Using the <rel> Subtag The <rel> subtag represents a relationship in the object model repository. Attributes are:

- *name* specifies the relationship end name, which can be unrolled into a separate tag like the name or type of an entity.
- *decision* specifies a decision, as described in [“Understanding Decisions” on page A-17](#).

The <target> and <source> subtags represent, respectively, the right and left ends of the relationship. These subtags are equivalent in function and syntax to the <entity> tag. Use the <cond> subtag to specify a condition, as described in [“Understanding Conditions” on page A-18](#).

Tip: As a practical matter, you will almost never have occasion to use the <entity> subtag.

If the <rel> subtag is specified at the top level of the entry, and no <source> tag is specified, the parser creates the relationship with the calling program; otherwise, it creates the relationship between the <source> and <target> entities. If the <rel> subtag is nested in an entity specification, the parser creates the relationship with the parent entity.

Example:

Let's assume now that we know that the second parameter in the API call on [page A-6](#) contains a variable in bytes 1-3 that specifies the CRUD operation, in addition to the variable in bytes 6-8 specifying the table name. The following definition extracts the CRUD operation and table name:

```
<vars>
  <arg var='OpName'
    param='2'
    type='data'
    offset='0'
    size='3' />
  <arg var='TableName'
    param='2'
    type='auto'
    offset='5'
    size='3' />
</vars>
<rep>
  <rel>
    <target type='TABLE'
      name='%TableName' />
    <name switch-var='OpName'>
      <case eq='RED' value='ReadsTable' />
      <case eq='UPD' value='UpdatesTable' />
      <case eq='ADD' value='InsertsTable' />
      <case eq='DEL' value='DeletesTable' />
    </name>
  </rel>
</rep>
```

Using the <hc> Tag

Use the <attr> subtag of the <hc> tag to represent the construct model, or *HyperCode*, attributes of entities defined in the call. Syntax is identical to that described for object model attributes in [“Using the <rep> Tag” on page A-8](#).

Using Expressions

Expressions let you extract parameter values and context information for specifications of entity or relationship characteristics. You can use simple variable names in expressions, or apply a primitive function call to a variable.

Basic Usage

Use the `%var_name` or `%parameter_number` notation to define variables for parameter values. The number corresponds to the index of the parameter; parameters are indexed beginning with 1. Negative numbers index from the last parameter to the first.

Variables with names beginning with an underscore are reserved for special values. They generally have only one value. Table A-1 describes the reserved variable names.

Table A-1 *Reserved Variable Names*

Name	Description
<code>_line, _col</code>	Line and column numbers of the call in source code.
<code>_file</code>	Index of the file in the file table.
<code>_uid</code>	UID of the node of the call in the syntax tree.
<code>_fail</code>	A permanently undefined variable. Use it to cause explicit failure.
<code>_yes</code>	A non-empty string for use as a true value.
<code>_no</code>	An empty string for use as a false value.
<code>_pgmname</code>	Name of the calling program.
<code>_hcid</code>	HyperCode ID of the call node.
<code>_varname nn</code>	If parameter number <i>nn</i> is passed using a simple variable reference (not a constant or an expression), this substitution variable contains its name. Otherwise, it is undefined.

Simple Notation

The simplest way to use a variable is to include it in an attribute value, prefixed with the percent character (%). (%% denotes the character itself.) If the character directly after the % is a digit or a minus sign (-), the end of the variable name is considered to be the first non-digit character. Otherwise, the end of the name is considered to be the first non-alphanumeric, non-underscore character. In:

```
'%abc.%2def'
```

the first variable name is `abc` and the second is `2`. It is also possible to specify the end of the variable name explicitly by enclosing the name in curly brackets:

```
'%{abc}.%{2}def'
```

When evaluated, a compound string like this produces a string value that concatenates variable values and simple text fragments, or fails if any of the variables is undefined.

Switch Usage

Use a *switch-var* attribute instead of the *value* attribute when a tag expects a value with a compound string expression. The *switch-var* attribute contains a single variable name (which may be prefixed by %, but cannot be enclosed in curly brackets). Use `<case>`, `<undef>`, or `<default>` subtags to specify switch cases. These tags also expect the *value* attribute, so switches can be nested:

```
<name switch-var='var'>
  <case eq='value1' value='...'/>
  <case eq='value2' switch-var='%var2'>
    <undef value='...'/>
  </case>
  <undef value='...'/>
  <default value='...'/>
</name>
```

When a switch is evaluated, the value of the variable specified using the *switch-var* attribute is matched against the literal specified in the `<case>` tags. The literal must be the same size as the variable. (The literals `value1` and `value2` in the example assume that `var` is defined as

having six bytes.) If an appropriate case is found, the corresponding case value is evaluated. If the variable is undefined, and the `<undef>` tag is specified, its value is used; if not, the switch fails. Otherwise, if the `<default>` case is specified, it is used; if not, the switch fails.

Fallback Chain Usage

Whenever multiple tags specifying a single attribute are presented in a `<name>`, `<type>`, or `<case>/<undef>/<default>` specification, those tags are joined into a *fallback chain*. If an entry in the chain fails, evaluation proceeds to the next entry. Only when the last entry of the chain fails is the failure propagated upward:

```
<name value='%a' />  
<name value='%b' />  
<name value='UNKNOWN' />
```

If `%a` is defined, the name is its value. Otherwise, if `%b` is defined, the name is `%b`. Finally, if both are undefined, the name is `UNKNOWN`.

Fallback Semantics for Attributes To determine the value of an attribute, the `<attr>` definitions for that attribute are processed one by one in order of appearance within the parent tag. For each definition, all combinations of variables used within it are enumerated, and all non-fail values produced are collected into a set:

- If the set contains exactly one value, it is taken as the value of the attribute.
- If the set contains multiple values, and the `<attr>` tag has a *join* attribute specified, the values are concatenated using the value of the *join* attribute as a delimiter, and the resulting string is used as the value for the repository attribute.
- Otherwise, the definition fails, and the next definition in the sequence is processed. If there are no definitions left, the attribute is left unspecified.

This behavior provides a way to determine if the variable has a specific value in its value set. The following example sets the attribute to `False` if the first parameter can be undefined, to `True` otherwise:

```
<attr name='Completed' switch-var='1'>
  <undef value='False' />
</attr>
<attr name='Completed' value='True' />
```

Using a Function Call

When a variable name contains commas, it is split into a sequence of fragments at their boundaries, and then interpreted as a sequence of function names and their parameters. In the following example:

```
%{substr,0,4,myvar}
```

the `substr` function extracts the first four characters from the value of `%myvar`. Table A-2 describes the available functions.

Functions can be nested by specifying additional commas in the last argument of the preceding function. In the following example:

```
%{int,substr,0,4,map}
switch-var='trim,substr,4,,map'
```

the first line takes the first four characters of the variable and converts them to a canonical integer, the second line takes the remainder, removes leading and trailing spaces, and uses the result in a switch, and so forth.

Table A-2 *Generic API Analysis Functions*

Function	Description
<code>substr,<start>,<size>,<variable></code>	Extracts a substring from the value of the variable. The substring begins at position <code><start></code> (counted from 0), and is <code><size></code> characters long. If <code><size></code> is an empty string, the substring extends up to the end of the value string.
<code>int,<variable></code>	Interprets the value of the variable as an integer and formats it in canonical form, without preceding zeroes or explicit plus (+) sign. If the value is not an integer, the function fails.

Table A-2 *Generic API Analysis Functions* (continued)

Function	Description
trim,<variable>	Removes leading and trailing spaces from a string value of the variable.
const,<string> or =,<string>	Returns the string as the function result.
warning,<id-num>[,<variable>]	Produces the warning specified by <id-num>, a numeric code that identifies the warning in the backend.msg file, and returns the value of the variable. If the variable is not specified, the function fails. So % {warning,12345} is equivalent to % {warning,12345,_fail}.

Understanding Enumeration Order

If the definition of the name of a relationship or the name or type of an entity contains substitution variables that have several possible values, the parser enumerates the possible combinations. The loops are performed at the boundary of the innermost <entity> or <rel> tag that contains the reference. (Loops for the target or source are raised to the <rel> level.)

Once the value for a variable has been established at a loop insertion point, it is propagated unchanged to the tags within the loop tag. So an entity attribute specification that refers to a variable used in the name of the entity will always use the exact single value that was used in the name.

If the expression for a name or type fails, the specified entity or relationship is locked out from processing for the particular combination of values that caused it to fail. This behavior can be used to completely block whole branches of entity/relationship definition tags:

```
<entity ...>  
  <type switch-var='a'>  
    <case eq='1' value='TABLE' />  
  </type>
```

```

    <rel name='InsertsTable' ..../>
</entity>
<entity ...>
  <type switch-var='a'>
    <case eq='2' value='MAP' />
  </type>
  <rel name='Sends'..../>
</entity>

```

If %a is 1, the first declaration tree is used, and the table relationship is generated; the second declaration is blocked. If %a is 2, the second declaration tree is used, and the map relationship is generated; the first declaration is blocked.

Note: These enumeration rules require that the value of a repository entity attribute not depend on variables used in the name of an enclosing <rel> tag, unless that variable is also used in the name of the entity itself. Otherwise, the behavior is undefined.

Understanding Decisions

A *decision* is a reference to another object (a program or screen, for example) that is not resolved until run time. If there are multiple possible combinations of values of variables used in the name of the target entity, or if some of the variables are undefined, the parser creates a decision entity, replacing the named relationship with a relationship to the decision and a set of relationships from the decision to each instance of the target entity.

When you use the <rel> tag at the top level of the repository definition, you can specify a *decision* attribute that tells the parser to create a decision regardless of the number of possible values:

- yes means that a decision is created regardless of the number of possible values.
- no means that a decision is never created (multiple values results in multiple direct relationships).
- auto means that a decision is created if more than one possible value exists, and is not created if there is exactly one possible value.

Both the relationship name and the type of the target entity must be specified as plain static strings, without any variable substitutions or switches:

```
<rep>
  <rel name='ReadsDataport' decision='yes'>
    <target type='DATAPORT' name='%_pgmname.%x' />
  </rel>
</rep>
```

Understanding Conditions

The `<cond>` subtag specifies a condition that governs the evaluation of declarations in its parent `<entity>` or `<relationship>` tag. The evaluation semantics of the tag follow the semantics for the `<attr>` tag: a non-empty string as a result indicates that the condition is true, an empty string or a failure indicates that the condition is false. Multiple `<cond>` tags can be specified, creating a fallback chain with `<attr>`-style fallback semantics.

Notice in the example given in the previous section that the parser creates a decision entity even when the name of the target resolves to a single value. Use a `<cond>` subtag in the relationship definition to avoid that:

```
<rel name='ReadsDataportDecision'>
  <cond if-multi='%x' value='%_yes' />
  <target type='DECISION'>
    <attr name='HCID' value='%_hcid' />
    <attr name='DecisionType' value='DATAPORT' />
    <attr name='AKA'
value='%_pgmname.ReadsDataport.%_varname1' />
    <attr name='AKA'
value='%_pgmname.ReadsDataport.' />
    <attr name='VariableName' value='%_varname1' />
    <attr name='Completed' if-closed='%x'
value='True' />
    <rel name='ResolvesToDATAPORT'>
      <target type='DATAPORT'
name='%_pgmname.%x' />
    </rel>
  </target>
</rel>
<rel name='ReadsDataport'>
```



```

<cond if-single='%x' value='%_yes' />
  <target type='DATAPORT' name='%_pgmname.%x' />
</rel>

```

This repository definition produces the same result as the example in the previous section, except that no decision is created when the name of the target resolves to a single value.

`_yes` and `_no` are predefined variables that evaluate, respectively, to a non-empty and empty string, as described in Table A-1. The *if-single* attribute means that the `<cond>` tag should be interpreted only if the specified variable has a single defined value. The *if-multi* attribute means that the `<cond>` tag should be interpreted if the variable has multiple values, none, or can be undefined. The *if-closed* attribute blocks the `<cond>` tag if the variable has an undefined value.

Note: *if-single*, *if-multi*, and *if-closed* can also be used with the `<attr>` tag.

Conditions have *join* set to an empty string by default, resulting in a `_yes` outcome if any combination of values of the variables used in switches within causes it to evaluate to `_yes`. If a particular condition definition should fail when some of the values evaluate to `_no` and others to `_yes`, use a *yes-only*='yes' attribute specification. That causes *join* to be unset, and the condition to give a non-fail outcome only when all values evaluate to `_yes`.

In a relationship definition, `<cond>` determines whether the relationship is generated. For a decision relationship, it also determines whether the decision entity should be generated.

In an entity definition, `<cond>` governs all attribute and subrelationship definitions in the tag, and the creation of the entity in case of a standalone entity. For an entity specified in a `<target>` or `<source>` tag, instantiation of the relationship automatically spawns the corresponding entity, meaning that a false condition on the source or target of a relationship does not prevent creation of corresponding entities.

Example

The following example illustrates use of the Generic API Analysis feature:

```
<APIEntry name='Call another program'>
  <match stmt="CALL">
    <name value="INVOKEPGM"/>
  </match>
  <flow halts='no'>
    <param index='1' usage='r' />
    <param index='*' usage='w' />
  </flow>
  <vars>
    <arg var='a' param='2' type='bit' len='5' />
  </vars>
  <rep>
    <rel name='CallsDecision'>
      <target type='DECISION'>
        <attr name='AKA'
          value='%_pgmname.
Calls.INVOKEPGM(%_varname1)'/>
        <attr name='AKA'
          value='%_pgmname.
Calls.INVOKEPGM' />

        <attr name='DecisionType'
          value='PROGRAMENTRY' />
        <attr name='HCID' value='%_hcid' />
        <attr name='VariableName'
          value='%_varname1' />

        <attr name='Completed' switch-var='1'>
          <undef value='False' />
        </attr>
        <attr name='Completed' value='True' />

        <rel name='ResolvesToProgramEntry'>
          <target type='PROGRAMENTRY'
            name='%1' />
        </rel>
      </target>
    </rel>
  </rep>
</APIEntry>
```

```

<hc>
  <attr name='test' switch-var='a' join=','
    <case eq='00101' value='X' />
    <undef value='?' />
    <default value='%a' />
  </attr>
</hc>
</APIEntry>

```

Support for IMS Aliases

The <IMSC> subtag in the <Auxiliary> section of Legacy.xml contains definitions for the standard CBLTDLI or PLITDLI programs. You can also use it to define aliases for non-standard IMS batch interfaces.

If the order of parameters in the alias program is the same as the order of parameters in the standard program, simply enter the alias name in the <Detect> and <APIEntry> tags, as follows:

```

<IMSC>
  <Cobol>
    <Detect>
      <item> 'CBLTDLI' </item>
      <item> 'MYCBLTDLI' </item>
    </Detect>
    ...
    <Process>
      <APIEntry name='IMS call'>
        <match stmt="CALL">
          <name value="CBLTDLI" />
          <name value="MYCBLTDLI" />
        </match>
        ...
      </Process>
    </Cobol>
  </IMSC>

```

If the order of parameters in the alias program differs from the order in the standard program, you also need to specify a full API entry, using the:

- `<match>` tag to define the alias name and method of invocation.
- `<flow>` tag to characterize the program control flow.
- `<ims-call>` tag to specify the call parameters.

Use the definitions for CBLTDLI or PLITDLI as examples.

Attributes of `<ims-call>` are:

- *count* specifies the index of the parameter that contains the argument count.
- *opcode* specifies the index of the parameter that contains the operation code.
- *pcb* specifies the index of the parameter that contains the Program Control Block (PCB) pointer.
- *arg-base* specifies the index of the first data parameter, usually io-area.

Note: Alternative parameter order is allowed only for the *params-num*, *function-code*, and *pcb* parameters. All other parameters (*io-area* and *ssa*) must appear in the same order as they do in the standard IMS call, at the end of the parameter list.

Skip Type Usage

Use the *skip-type* attribute of the `<param>` subtag in the `<halts>` section to ensure that the optional first parameter of a Cobol IMS CALL is parsed only if necessary. If the actual parameter passed by the program in the first position has the type specified by the regular expression in *skip-type*, the parameter is filled with a dummy value and the actual value is used in the next parameter.

Note: Skip definitions are also available for use in non-IMS generic API entries.

Example:

If the first parameter in a call is a 4-character picture, the following definition inserts a dummy value in the first position and treats the actual value as that of the second parameter:

```
<param index='1'  
      usage='r'  
      skip-type='PIC:(X\ (4\)' />
```

Note: Skip definitions are currently limited to declarations having picture clauses. Use regular expression syntax to specify normalized picture strings.

A-24 Identifying Interfaces for Generic API Analysis
Support for IMS Aliases

Cobol Range Overlaps and Range Jumps Detected in the Executive Report



This appendix lists Cobol range overlaps and range jumps detected in the Executive Report. S* defects appear in the report under “Range Overlaps” as the sum of all defects S1+S2+S3+S4+S5+S6. G* defects appear in the report under “Range Jumps” as the sum of all defects G1+G2+G3+G5+G6+G7.

S0. No defects

```
        Perform A1 thru A2.  
        Perform B1 thru B2.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      A2.  
\--- StatementsA2.  
        ...
```

B-2 Cobol Range Overlaps and Range Jumps Detected in the Executive Report

```
.--B1.  
|   StatementsB1.  
|   ...  
|   B2.  
\--- StatementsB2.
```

S1. Overlapped sections

```
    Perform A1 thru A2.  
    Perform B1 thru B2.  
    ...  
.--A1.  
|   StatementsA1.  
|   ...  
|   B1.           --.  
|   StatementsB1. |  
|   ...           |  
|   A2.           |  
\--- StatementsA2. |  
    ...           |  
    B2.           |  
    StatementsB2. --'
```

S2. Overlapped sections

```
    Perform A1 thru A2.  
    Perform B1 thru B2.  
    ...  
.--B1.  
|   StatementsB1.  
|   ...  
|   A1.           --.  
|   StatementsA1. |  
|   ...           |
```



```

| B2. |
\--- StatementsB2. |
... |
A2. |
StatementsA2. --'

```

S3. Overlapped sections

```

Perform A1 thru A2.
Perform B1 thru B2.
...
.--A1.
| StatementsA1.
| ...
| B1. --.
| StatementsB1. |
| ... |
| B2. |
| StatementsB2. --'
| ...
| A2.
\--- StatementsA2.

```

S4. Overlapped sections

```

Perform A1 thru A2.
Perform A1 thru B2.
...
.--A1. --.
| StatementsA1. |
| ... |
| B2. |
| StatementsB2. --'
| ...

```

B-4 Cobol Range Overlaps and Range Jumps Detected in the Executive Report

```
| A2.  
\--- StatementsA2.
```

S5. Overlapped sections

```
    Perform A1 thru A2.  
    Perform B1 thru A2.  
    ...  
.--A1.  
|   StatementsA1.  
|   ...  
| B1.                --.  
|   StatementsB1.   |  
|   ...             |  
| A2.                |  
\--- StatementsA2.  --'
```

S6. Overlapped sections

```
    Perform A1 thru A2.  
    ...  
.--A1.  
|   StatementsA1.  
|   ...  
|   Perform B1 thru B2.  
|   ...  
| B1.                --.  
|   StatementsB1.   |  
|   ...             |  
| B2.                |  
|   StatementsB2.  --'  
|   ...  
| A2.  
\--- StatementsA2.
```

G0. No defects

```
        Perform A1 thru A2.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      goto B1.  
|      ...  
| B1.  
|      StatementsB1.  
|      ...  
| A2.  
\--- StatementsA2.
```

G0. No defects

```
        Perform A1 thru A2.  
        goto B1.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
| A2.  
\--- StatementsA2.  
        ...  
        B1.  
        StatementsB1.
```

G0. No defects

```
        Perform A1 thru A2.  
        goto B1.
```

B-6 Cobol Range Overlaps and Range Jumps Detected in the Executive Report

```
    ...  
    B1.  
        StatementsB1.  
    ...  
.--A1.  
|    StatementsA1.  
|    ...  
|    A2.  
\--- StatementsA2.
```

G0. No defects

```
        Perform A1 thru A2.  
.--A1.  
|    StatementsA1.  
|    ...  
|    A2.  
\--- StatementsA2.  
    ...  
        goto B1.  
    ...  
    B1.  
        StatementsB1.
```

G1. Break-in goto

```
        Perform A1 thru A2.  
    ...  
        goto B1.  
    ...  
.--A1.  
|    StatementsA1.  
|    ...  
|    B1.
```

```
|      StatementsB1.  
|      ...  
|      A2.  
\--- StatementsA2.
```

G2. Break-in goto

```
      Perform A1 thru A2.  
      ...  
      goto A1.  
      ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      A2.  
\--- StatementsA2.
```

G3. Break-in goto

```
      Perform A1 thru A2.  
      ...  
      goto A2.  
      ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      A2.  
\--- StatementsA2.
```

S3G4=G1. Overlapped sections, break-in goto

```
        Perform A1 thru A2.
        ...
.--A1.
|      StatementsA1.
|      ...
|      Perform B1 thru B2.
|      ...
|      goto C1.
|  B1.                --.
|      StatementsB1.  |
|      ...            |
|  C1.                |
|      ...            |
|  B2.                |
|      StatementsB2.  --'
|      ...
|  A2.
\--- StatementsA2.
```

G5. Break-out goto

```
        Perform A1 thru A2.
        ...
.--A1.
|      StatementsA1.
|      ...
|      goto B1.
|      ...
|  A2.
\--- StatementsA2.
        ...
        B1.
          StatementsB1.
```

G6. Break-out goto

```
        Perform A1 thru A2.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      goto A1.  
|      ...  
|      A2.  
\--- StatementsA2.
```

G7. Break-out goto

```
        Perform A1 thru A2.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      goto A2.  
|      ...  
|      A2.  
\--- StatementsA2.
```

G8. No Defect s

```
        Perform A1 thru A2.  
        ...  
.--A1.  
|      StatementsA1.  
|      ...  
|      goto A2.  
|      ...
```


Recognized File Extensions



The table in this appendix section lists the file extensions recognized by the workbench registration process.

Table C-1 *Recognized File Extensions*

File Type	Extensions
BMS Copybook File	.bmscopy
BMS File	.bms
C File	.c
Cobol File	.cbl, .cob, .ccp
Control Cards File	.crd, .srt
Copybook File	.cpy, .dcl

Table C-1 *Recognized File Extensions* (continued)

File Type	Extensions
CPP File	.cpp
CSD File	.csd
DBD File	.dbd
DDL File	.ddl
EasyTrieve File	.ezt, .esy
EasyTrieve Macro	.ezm, .z
FCT File	.fct
Header File	.h, .hpp, .hxx, .tlh, .inl
IDMS Schema File	.idms
Java File	.java
JCL File	.jcl
JCL Procedure	.prc
MFS File	.mfs
MFS Include File	.mfi
PCT File	.pct
PSB Copybook File	.psbcpy
PSB File	.psb

Table C-1 *Recognized File Extensions* (continued)

File Type	Extensions
System Definition File	.ims
User Document	.doc

C-4 Recognized File Extensions

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 main-

frame 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

A

API analysis A-1
autoresolving decisions 3-19, 3-22, 3-24,
6-2, 6-7

B

batch sort card analysis 3-11, 3-15
boundary decisions 3-27

C

calling chains analysis for JCL 3-10
COBOL-68 3-3
COMS analysis 3-20
copybooks
 verifying 4-2
copybooks, generating 4-23
cross-reference report
 exporting 5-6
 generating 4-3, 5-2
 overview 5-1
currency sign 3-3

D

data operations 4-20
dead code analysis 3-10, 3-14
decision resolution
 overview 6-1
 resolving decision automatically 6-2,
6-7
 resolving decision manually 6-2
 restoring manually resolved decisions
6-6
dialects 3-2

E

extensions 2-4

F

file extensions iii-viii, 2-4, C-1

G

generic API analysis 3-20, A-1, A-4

H

HyperView, enabling 3-9, 3-13

I

IMS port analysis 4-20

invalidating objects 4-3

Inventory Report 4-12

J

Japanese source files 2-8

JCL

- verification requirements for data flow analysis 3-10

- verification requirements for IMS port analysis 4-21

L

legacy dialects 3-2

loading source files 2-2

M

mainframe encoding 2-7

N

Natural library support 3-10, 3-15

Natural line numbers 3-5

O

orphan analysis

- exporting reports 5-12

- generating reports 5-7

- overview 5-6

P

PERFORM behavior 3-2, 3-3

post-verification tasks

- enabling IMS port analysis 4-20

- program analysis 4-18

- viewing Executive Reports 4-13

- viewing Inventory Reports 4-3

project

- copying files 2-10

- creating 2-9

- including objects 2-12

- moving files 2-10

- verifying files 4-1

Q

query repository feature 4-17

R

reference reports

- enabling in verification options 3-6, 3-9, 3-13

- exporting 5-6

- generating 4-3, 5-2

- overview 5-1

refreshing source files 2-3

refreshing the repository 4-2

registering applications

- loading source files 2-2

- overview 1-1

registering files 2-1

relaxed parsing

- discussed 3-14

- selecting 3-11

repository

- querying 4-17

- refreshing 4-2

resolving decisions 6-1

S

- sharing projects 2-10
- Shift-JIS encoding 2-8
- sort card analysis 3-11, 3-15
- source files
 - encoding 2-7
 - exporting 2-4
 - refreshing 2-3
 - registering 2-2
 - verifying 4-1
- staged parsing 3-12
- system programs 3-25, 5-6

T

- transaction-processing verification option
 - 3-17

U

- unreferred report
 - exporting 5-6
 - generating 4-3, 5-2
 - overview 5-1
- unresolved report
 - exporting 5-6
 - generating 4-3, 5-2
 - overview 5-1

V

- verification options
 - and Configuration Manager 3-1
 - discussed 3-1
 - Legacy Dialects tab 3-2
 - project 3-16
 - Settings tab 3-6
 - staged parsing 3-12
 - workspace 3-2

- Verification Report 4-3

- verifying applications
 - invalidating objects 4-3
 - overview 1-2
 - verifying source files 4-1
- verifying files 4-1

W

- workspace
 - deleting 2-13
 - exporting files 2-4
 - registering files 2-1
 - verifying files 4-1
- workstation encoding 2-7

Index-4