



COBOL Analyzer 8.0

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
<http://www.microfocus.com>

© Copyright 2021 Micro Focus or one of its affiliates.

MICRO FOCUS, the Micro Focus logo and are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2022-06-17

Contents

COBOL Analyzer	7
Installation Guide	7
Overview	7
Hardware and Software Requirements	10
CA Performance Optimization Guidelines	13
Installing and Uninstalling COBOL Analyzer	18
Post-Installation Administrative Tasks	18
Troubleshooting the Installation	41
Getting Started	41
Introducing COBOL Analyzer	41
COBOL Analyzer Basics	47
Preparing Projects	73
Registering Source Files	73
Setting Up Projects	82
Verifying Source Files	84
Using Post-Verification Reports	98
Inventorying Applications	101
Identifying Interfaces for Generic API Analysis	108
Analyzing Projects	121
Managing Tags	121
Analyzing Relationship Flows	128
Analyzing Global Data Flow	145
Estimating Complexity and Effort	148
Identifying Classes of Data Items with Change Analyzer	151
Repository Exchange Protocol Syntax	156
Portability Assessment	159
Quality Assessment	159
Analyzing Programs	160
Introducing Interactive Analysis	160
Understanding Program Context	163
Performing Advanced Searches	170
Staging Program Analysis with Code Search	178
Analyzing Impact Traces	186
Analyzing Program Control Flows	194
Setting Up a Glossary	201
Extracting Business Rules	207
Using the Batch Duplicate Finder	225
Creating Components	228
Introducing Logic Analyzer	228
Setting Component Maker Options	235
Extracting Structure-Based Components	241
Extracting Computation-Based Components	245
Extracting Domain-Based Components	248
Eliminating Dead Code	253
Performing Entry Point Isolation	255
Technical Details	256
Using the Batch Refresh Process	261
Using the Batch Refresh Process	261
Using Batch Scripts	277
Using Architecture Modeler	317
Introducing Architecture Modeler	317

Opening Architecture Modeler	317
Understanding the Application-Level Metamodel	318
Defining an Extension with Architecture Modeler	321
Using Galleries	335
CA Web	335
Installing CA Web	335
Deploying CA Web	335
Troubleshooting	336
Getting Started with CA Web	337
CA Web Reports	339
REST API and Jenkins Plugin	339
REST API and Jenkins Plugin	339
Support Notes	345
Supported Languages	345
Supported Features by Language	346
C/C++ Support	352
COBOL Support	352
ACUCOBOL Support	355
Java Support	355
JSP Support	355
.NET Support	356
PL/SQL Support	356
SQL Support	356
VB Support	357
Complexity Metrics	359
Complexity Metrics Overview	359
ADL File	359
Assembler Copybook File	359
Assembler File	360
Assembler Programs	360
BMS Copybook File	361
BMS File	361
C File	362
C++ Class	362
C++ File	363
C++ Function	364
C++ Header File	366
C++ Member Function	366
COBOL Copybook File	368
COBOL File	368
COBOL Program	369
COMS Configuration File	372
Control Cards File	372
Control Language File	372
CSD File	373
DASDL Include File	373
Database Description File	373
Database File	373
DBD File	374
DDL File	374
Device Description File	374
DMSII DASDL File	374
DMSII Database	375
DMSII Dataset	375
DMS DDL File	375
DPS File	376

ECL Addstream	376
ECL File	376
FCT File	376
Fujitsu COBOL Program	377
ICL Form	380
IDMS Schema File	380
IDMS Subschema File	380
Java Class	381
Java File	383
Java Interface	385
Java Method	387
Java Package	389
JCL File	391
JCL Procedure	392
Job	392
Library Description File	393
Macro File	393
Map	393
MFS File	393
MFS Include File	394
Natural Data Area	394
Natural DDM File	394
Natural File	394
Natural Include File	395
Natural Map	395
Natural Program	395
.NET File	398
.NET Method	400
.NET Project	401
.NET Type	403
PCT File	405
PL/I File	406
PL/I Include File	406
PL/I Program	407
PSB Copybook File	409
PSB File	410
RPG Copybook File	410
RPG File	410
System Definition File	410
Unisys 2200 Program	411
VALTAB File	414
VB Class	414
VB File	415
VB Function	415
VB Library	417
VB Method	417
VB Project File	419
WFL File	420
WFL Include File	420
Relationship Projections	420
Relationship Projections Overview	420
Assembler Statements	420
BMS Statements	422
C/C++ Statements	422
CICS Statements	429
Cobol Statements	435

CSD Statements	437
DBD Statements	438
EXEC SQL Statements	439
FCT Statements	440
IDMS DML Statements	441
IDMS Schema Statements	447
Java Statements	447
JCL Statements	454
Natural Statements	462
.NET Statements	465
PCT Statements	467
PL/I Statements	468
PSB Statements	472
RPG Statements	476
SQL DDL Statements	486
System Definition Statements	487
Visual Basic Statements	488

COBOL Analyzer

Installation Guide

Overview

This manual describes how to install and configure Micro Focus COBOL Analyzer (CA), a suite of software products for analyzing and modernizing legacy applications.

COBOL Analyzer provides insight into the technical reality of complex application portfolios, including

- Tools for application and program level understanding, utilizing extensive metrics, reports, diagrammatic views and querying tools to support myriad of business initiatives.
- Quality Assessment with standard code quality queries to serve as a guidance to a code quality practice in both development and maintenance phases.
- Portability Assessment to help you generate various HTML reports to identify points of interest for migrations.
- In depth analysis tools to promote efficiency in the performance of daily tasks such as field changes, understanding of data propagation through a program, and dead code removal.

Business Rule Manager mines business logic from program code and encapsulates the logic in business rules.

While all products are installed with COBOL Analyzer, each product is licensed separately.

CA can be deployed in a multi-user environment or single users with a local database. The multi-user environment gives users access to a common repository of application objects. Repository setup is the responsibility of a master user, leaving team members free to focus on their tasks. The database for the repository is assumed to be the customer's own.

CA installations consist of the following components:

- The CA Server which hosts CA workspace files and related support files.
- CA Clients which host the link files used to connect to workspaces on the server.
- For single user setups, the CA software is installed on the developer's machine and includes an option to install a local database.

This manual also describes how to install and configure the database client used to connect to repositories, for sites at which the database client is not already installed. If you store repositories in Oracle or DB2, a database client must be installed wherever the CA client or CA server is installed.

Installation Tasks

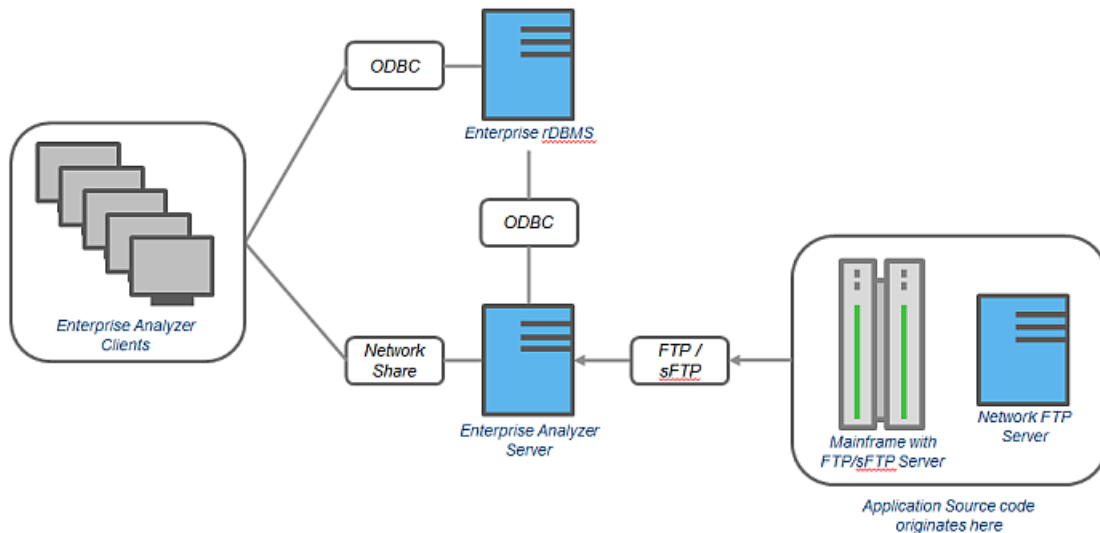
The table below describes the tasks involved in installing COBOL Analyzer and the order in which the tasks must be performed.

Task	Order	Notes
Install CA on Server	1	This is for multi-user environments only.
Install CA on Clients	2	The CA package optionally installs SQL Server Express. This is enough for a small sized repository (up to 10 GB).

Each task is described and explained in the chapters of the *Installation Guide*.

Deployment

The figure below shows the standard CA deployment scenario. The role each machine performs and its key relationships are described in the following sections.



Mainframe and Network Server

The mainframe typically hosts the application to be modeled in CA. Application source files are downloaded from the mainframe (and, if necessary, a network server) to the CA server via FTP or SFTP.


Repository Server

The Repository Server hosts the database for one or more multiuser repositories, one repository per CA workspace. This server provides centralized, network-accessible storage for parsed data and analysis output.

CA Server


The CA Server hosts workspaces, workspace support files (including the copies of application source files CA creates at workspace loading), and workspace output. This server leverages multiple processes to enhance parsing performance in online or batch mode.

Parsed data is sent via ODBC to the repository server. Some analysis output is stored on the CA server as well.

 **Note:** In a multi-user environment, the CA Server cannot reside on the same machine as the CA Repository. The installation program is the same for the CA client and CA server.

CA Client


CA Clients host the link files that let team members connect to workspaces on the CA server. These clients access repository data stored in the repository server via ODBC.

 **Note:** The installation program is the same for the CA client and CA server.

Single User Installation

You can configure COBOL Analyzer to build a workspace on one machine for a single user. You can use Microsoft SQL Server Express - installed by default - to create a workspace on your local machine. CA creates a database for the workspace "on the fly," with no intervention on your part. If you do use SQL

Server Express, bear in mind that the Windows user who creates the workspace must have been configured with appropriate permissions in SQL Server Express. The user who installed SQL Server Express will always have the appropriate permissions. See the SQL Server Express documentation for configuration instructions.

 **Restriction:** The database size limit when using the SQL Server Express option is 10 GB.


Database Setup

If you use Microsoft SQL Server, the DBA must set up a SQL Server database for each workspace repository you plan to create. CA users typically supply Windows credentials to access the repository.


SQL Server Database Setup

If you use Microsoft SQL Server, you must set up a SQL Server database for each workspace repository you plan to create. If CA users connect to the repository through an SQL Server login account, ensure the server authentication property is set to **SQL Server and Windows Authentication mode**, as described below.

The following instructions assume you are using the SQL Server Management Studio tool for SQL Server 2008. The procedure for the SQL Server Management Studio Express tool for SQL Server 2005 is similar.

 **Note:** For Windows XP installations using the Italian locale, you must set the Time format in the Control Panel Regional Language Options to "HH:mm:ss" before attempting to verify an CA workspace with a SQL Server repository. Click **Settings > Control Panel > Regional and Language Options > Customize > Time** and choose "HH:mm:ss" from the **Time format** list.

1. Click **Start > Programs > Microsoft SQL Server 2008 > SQL Server Management Studio**.
2. In the Connect to Server screen, select:
 - Database engine in the **Server type** list.
 - The server on which you want to create the repository database in the **Server name** list.
 - Windows Authentication in the **Authentication** list.
3. Click **Connect**. The Microsoft SQL Server Management Studio window opens. In the **Object Explorer** pane, expand the folder for the server. Click **Databases** and choose **New Database** from the right-click menu. The **New Database** window opens.
4. Select the **General** page. In the **Database name** field, enter the name of the database for the repository. Modify the logical name and initial sizes of the database and log files if needed, then click **OK**. The new database is displayed in the **Object Explorer** pane.

 **Note:** The database collation should be case insensitive. Additionally, check the SQL Server Management Studio documentation for details on other database settings.
5. If you connect to the workspace repository through a Windows user account, skip the remaining steps. If you connect to the workspace repository through an SQL Server login account, click the server name in the **Object Explorer** pane and choose **Properties** from the right-click menu. The **Server Properties** window opens.
6. Select the **Security** page. In the **Server authentication** area, select **SQL Server and Windows Authentication mode**, then click **OK**.
7. In the **Object Explorer** pane, click **Security > Logins** and choose **New Login** from the right-click menu. The **Login - New** window opens.
8. Select the **General** page. In the **Login name** field, enter the database login name, then choose **SQL Server authentication**. Enter the password for the database login in the **Password** and **Confirm password** fields, then click **OK**. The new login is displayed in the **Object Explorer** pane.
9. In the **Object Explorer** pane, expand the database you created for the workspace repository. Click **Security > Users** and choose **New User** from the right-click menu. The **Database User - New** window opens.

10. Select the **General** page. Define the database user:

- In the **User name** field, type the database user name.
- In the **Login name** field, type the database login name. Use the **Browse** button to browse for the login name.



Note: You specify the login name, not the database user name, when you create or connect to a workspace, so it is usually best to make the user name and login name the same.

- In the **Database role membership** pane, check db_owner, then click **OK**. The new user is displayed in the Object Explorer pane.



Tip: You can use the same login and database user for multiple databases or workspaces.

You can configure COBOL Analyzer to build a workspace on one machine for a single user. You can use Microsoft SQL Server Express - installed by default - to create a workspace on your local machine. CA creates a database for the workspace "on the fly," with no intervention on your part. If you do use SQL Server Express, bear in mind that the Windows user who creates the workspace must have been configured with appropriate permissions in SQL Server Express. The user who installed SQL Server Express will always have the appropriate permissions. See the SQL Server Express documentation for configuration instructions.



Restriction: The database size limit when using the SQL Server Express option is 10 GB.



Note: SQL Server database transaction logs can become full, resulting in errors. To avoid this, Micro Focus recommends that you turn off transaction logging or implement automatic log recycling so that normal CA processing is not interrupted. CA does not use the SQL recovery processes so the transaction logging is not required, but it is recommended that the database is regularly backed up.

collation

Hardware and Software Requirements

The following sections describe the hardware, disk space, operating system, and software requirements for CA installations.

Repository Server Hardware Requirements

The table below lists the hard drive storage requirements for the Repository Server. For other hardware recommendations, check with support services.

Type	Requirement	Notes
Hard Drive Storage	Variable	Minimum of 20 GB disk space needed for the software installation of the RDBMS and the CA template database. Plus approximately 60x the size in bytes of the application source code modeled in CA (e.g., 100 MB source = 6 GB).

Repository Server Software Requirements

The table below lists the software requirements for the Repository Server:

Type	Requirement
SQL Server	MS SQL Server 2012, 2014, 2016, 2017 or 2019
PostgreSQL	PostgreSQL 13, 14



CA Server Hardware Requirements

The table below lists the hardware requirements for CA Server installations. Hardware requirements may vary depending on the size of the application you are analyzing.

Type	Requirement	Notes
Processor	2.6 GHz Dual Core or 2x 3.0+ GHz Processors	Dual processing capability with multiple cores in a processor or separate physical processors.
Physical Memory	3 GB RAM	
Virtual Memory	1 GB to 3 GB	
Hard Drive Storage	Variable	For CA workspaces, approximately 40x size in bytes of the application source code modeled in CA (for example, 100 MB source = 4 GB). For CA software, minimum 200 MB.

CA Server Software Requirements

The table below lists the software requirements for CA Server installations:

Type	Requirement
Operating System	<ul style="list-style-type: none"> Microsoft Windows Server 2016, 64-bit Microsoft Windows Server 2019, 64-bit Microsoft Windows 10, 32-bit and 64-bit Microsoft Windows Server 2022, 64-bit Microsoft Windows 11, 64-bit
Pre-Requisite Software	<p> Note: The pre-requisite software is automatically installed by the installer.</p> <ul style="list-style-type: none"> • Sentinel RMS License Manager • .NET Framework 4.0 • .NET Framework 4.6.1 • .NET 6.0.3 • Visual C++ 2012 Redistributable 11.0.61030 • Visual C++ 2017 Redistributable 14.11.25325.0 • SQL Server 2014 Express (optional install)
Web browser	<p>Required to view HTML report outputs or to access the Web client.</p> <ul style="list-style-type: none"> • Internet Explorer 6.0 or higher (optional) • Mozilla Firefox 3.6 or higher • Chrome 6 or higher
Microsoft Office (optional)	<p>Required by CA tools with reporting capabilities to save to Microsoft Office file formats. Excel is required for standard deviation charts in the Executive Report.</p>
Microsoft Visio (optional)	<p>Required to generate output as Microsoft Visio files.</p> <p> Note: When the Visio version is 2013 or higher, the .VSDX format is used instead of .VSD.</p>

Type	Requirement
OpenJDK JRE 8.0 or higher (optional)	Required for Java parsing.



CA Client Hardware Requirements

The table below lists the hardware requirements for CA Client installations. Hardware requirements may vary depending on the size of the application you are analyzing.

Type	Requirement	Notes
Processor	3.0 GHz Processor	Single processor (single or dual core).
Physical Memory	1GB RAM	
Virtual Memory	1GB to 3GB	
Hard Drive Storage	Variable	For CA software, minimum 200MB.

CA Client Software Requirements

The table below lists the software requirements for CA Client installations:

Type	Requirement
Operating System	Microsoft Windows Server 2016, 64-bit Microsoft Windows Server 2019, 64-bit Microsoft Windows 10, 32-bit and 64-bit Microsoft Windows Server 2022, 64-bit Microsoft Windows 11, 64-bit
Pre-Requisite Software	 Note: The pre-requisite software is automatically installed by the installer. <ul style="list-style-type: none"> • Sentinel RMS License Manager • .NET Framework 4.0 • .NET Framework 4.6.1 • .NET 6.0.3 • Visual C++ 2012 Redistributable 11.0.61030 • Visual C++ 2017 Redistributable 14.11.25325.0 • SQL Server 2014 Express (optional install)
Web browser	Required to view HTML report outputs or to access the Web client. <ul style="list-style-type: none"> • Internet Explorer 6.0 or higher (optional) • Mozilla Firefox 3.6 or higher • Chrome 6 or higher
Microsoft Office (optional)	Required by CA tools with reporting capabilities to save to Microsoft Office file formats. Excel is required for standard deviation charts in the Executive Report.
Microsoft Visio (optional)	Required to generate output as Microsoft Visio files.  Note: When the Visio version is 2013 or higher, the .VSDX format is used instead of .VSD.
OpenJDK JRE 8.0 or higher (optional)	Required for Java parsing.

CA Performance Optimization Guidelines

This section describes the optimal performance environments in which to run COBOL Analyzer, including selecting the right hardware configuration for specific types of usage, and optimizing the CA configuration for the selected configuration. The suggestions for performance improvements are focused on the repository build phase of the workspace.

The most time-consuming and hardware-intensive aspects of running COBOL Analyzer are the source verification and database loading during the workspace build phase. The guidelines include suggestions for improving performance by using multiple Queue Processors and taking advantage of multiple CPU cores, using the parallel verification option which determines the number of used Queue Processors, and guidelines for defining the number of Queue Processors needed for a given configuration of available machines and database server power.

There are two deployment scenarios for COBOL Analyzer:

- Single User - this is typically a mobile user, such as a consultant on a laptop, who uses the repository for demonstration or assessment purposes.
- Enterprise Installations - several computers are used (one for a database server, one for workspace and multiple user machines).

Choosing Hardware Configuration

Single User (One Machine)

Minimum Hardware Requirements

Type	Requirement	Notes
Processor	1.8 GHz, minimum dual core	
Physical Memory	2 GB RAM	
Hard Drive Storage	HDD with at least 20 GB free space	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

With this minimum hardware configuration you can run COBOL Analyzer and analyze sources.

Suggested Upgrades

The analysis time increases when the number of analyzed sources rises. To reduce the analysis time we recommend you use one of the suggested upgrades:

1. A high speed Solid State Drive (SSD) for storing database tables. This can be added as an external drive; however USB 3.0 or eSATA must be used. This will improve the verification performance by 25% over the minimum configuration.
2. Use a quad core processor such as Intel Core i7 2670QM or any other quad core QE/QM processor. This improves the verification performance by 50% over the minimum system configuration.



Note: You must run additional Queue Processors to benefit from the additional cores. See the *Choosing the Optimal Amount Of Queue Processors* section for details.



Note: Improved verification performance means that if the project verification takes 100 minutes, and you get 33% performance improvement, the verification time is reduced to 67 minutes.

Recommended Configuration 1

This configuration gives 33% improved verification performance. The changes from the minimum configuration are marked in bold.

Type	Requirement	Notes
Processor	2.0 GHz, 2 cores	
Physical Memory	4 GB	
Hard Disk Drive	SSD with at least 20 GB free space	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

Recommended Configuration 2

Recommended Configuration 2 gives 50% improved performance of verification. The changes from the minimum configuration are marked in bold.

Type	Requirement	Notes
Processor	2.0 GHz, 4 cores	
Physical Memory	4 GB	
Hard Disk Drive	Any type with at least 20 GB free space	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

Performance Configuration

The Performance Configuration is a combination of all upgrades. It gives 66% improved verification performance. The changes from the minimum configuration are marked in bold.

Type	Requirement	Notes
Processor	2.0 GHz, 4 cores	
Physical Memory	4 GB	
Hard Disk Drive	SSD with at least 20 GB free space	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

Enterprise Installation (Multiple Users Using Separate Database Server)

A typical environment where large projects will be verified on a regular basis usually consists of 3 types of components:

1. Database server (MS SQL Server).
2. CA server with the workspace files. This is the machine where CA is installed and where the CA workspace folder is located.



Note: There can be more than one workspace.

3. Processing node(s). This is one or more computers running Queue Processors, or hosting the users of the tool.

Minimum hardware requirements

Type	Requirement	Notes
Processor	2.6 GHz Dual Core or 2x 3.0+ GHz Processors	Dual processing capability with multiple cores in a processor or separate physical processors.
Physical Memory	3 GB RAM	
Hard Disk Drive		For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

The minimum configuration can be up to four concurrent Queue Processors while providing reasonable response time. To improve performance, one of the following upgrades should be used:

1. Use faster storage, for example use SSD or storage arrays in RAID0. SSD improves the performance of verification by 33%. 2x SCSI 15000 rpm gives a similar performance improvement.
2. Use 8 GB of RAM. Increasing the amount of RAM improves the performance of verification by 15% when using more Queue Processors.
3. Network latency between database server and repository server and users should be less than 1ms.

Recommended hardware configuration


This is the recommended hardware configuration. The changes from the minimum configuration are marked in bold.

Type	Requirement	Note
Processor	2.6 GHz Dual Core or 2x 3.0+ GHz Processors	Dual processing capability with multiple cores in a processor or separate physical processors.
Physical Memory	8 GB RAM	
Hard Disk Drive	SSD or SCSI 15000 rpm in RAID0	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB).

Performance Configuration

You get 33% improved performance of verification.

Type	Requirement	Note
Processor	Xeon CPU E5310 or Core i7 3770	
Physical Memory	8 GB	
Hard Disk Drive	2x SCSI 15000 rpm RAID0 or 240GB SSD	

 **Note:** Performance configuration can support up to 16 concurrent Queue Processors. Note that user activity through the online tool should be counted as an active Queue Processor.

CA and Workspace Server

The repository server stores the COBOL Analyzer workspace folder and files. It is shared on the network so that other users can access it.



Warning: Micro Focus does not recommend combining the CA server and database server (and even processing node) for machines with limited resources.

Minimum hardware requirements for Repository Server are:

Type	Requirement	Note
Hard Disk Drive	Minimum of 20 GB disk space needed for the software installation of the database software.	For CA workspaces, approximately 60x size in bytes of the application source code modeled in CA (for example, 100 MB source = 6 GB). Storing your workspace on an SSD drive improves verification performance by 5%-15%, depending on your project.

Processing Node/Nodes

This is the machine that runs the Queue Processors and the processes that they kick off. The minimum configuration lets you run a single Queue Processor.

Minimum System Requirements

Type	Requirement
Processor	3.0 GHz single core or 2 GHz dual core
Physical Memory	2 GB
Virtual Memory	1 GB to 3 GB
Hard Disk Drive	1 GB


The minimum system lets you analyze small volumes of sources and use the basic functionality of the tool. To determine the optimal amount of Queue Processors for your Processing node see *Choosing the Optimal Amount of Queue Processors*. To improve the performance, some of the following upgrades can be used:

1. Use more CPUs or CPU cores. COBOL Analyzer scales very well so the performance improvement matches the CPU cores added - verification performance is improved by 90% when CPU cores are doubled. The limit is eight physical cores that exclude hyper threading, with 16 queue processors. Each concurrent Queue Processor instance needs up to an additional 2GB of RAM. Make sure your database server can support the additional Queue Processors, see *Database Server*.
2. Use more computers as processing nodes.

Recommended Configuration (66% improved performance of verification)

Type	Requirement
Processor	3 GHz quad core CPU
Physical Memory	4 GB
Hard Disk Drive	4 GB

As these processing nodes communicate extensively with the database server, a low-latency network is required between processing node(s) and database server.

 **Note:** For best results, the computers must be on the same local network. You can use a repository server or a database server (or both) as processing nodes, provided that MS Windows is installed.

Software Configuration

Verification Information


Verification is the process of analyzing the code loaded into CA. The verification process demands a lot of CPU power. COBOL Analyzer is designed to use a single CPU core for the main application, but each Queue Processor can run on a separate CPU core. The Queue Processors can even run on multiple computers to help with the verification database loading. There are two ways of initiating the verification process. The first is the online tool (COBOL Analyzer) and the second is through the Batch Refresh Process (BRP).

Parallel Verification

By default COBOL Analyzer uses serial verification both for the online tool and for BRP. If you have a multicore CPU, it is better to use parallel verification.

To enable it in COBOL Analyzer:


1. Click **Options > Workspace Options**.
2. Click **Verification > Parallel Verification** tab.
3. Select either
 - **Run Parallel Verification in the online tool**
 - **Run Parallel Verification in BRP**
4. Adjust the **Minimum Queue Processors** to improve performance when using an optimized number of CPU cores.

 **Note:** On a quad core CPU you get up to 4 times better performance provided that you use the optimal number of Queue Processors.

Choosing the Optimal Number of Queue Processors


The optimal number of Queue Processors is related to the number of physical CPU cores available.

If you have X physical CPU cores you should use at least X Queue Processors. Using more Queue Processors than physical CPU cores slightly increases performance but depending on your system setup it could also slow down the verification because of the large number of concurrent IO operations.

 **Note:** Micro Focus recommends using <number-of-physical-CPU-cores> multiplied by two.

Using More Computers

You can start a new Queue Processor on another computer to help the current verification.

 **Note:** CA must be installed on the computer that launches a Queue Processor.

1. Open Micro Focus COBOL Analyzer Administration and click **Administer > Build New Connection**. This creates a local `.rwp` file to connect to the workspace that needs help during the verification and database loading.
2. Click **Administer > Launch Queue Processor**. The Launch Queue Processor window opens.
3. Specify the local `.rwp` file for the workspace and enter the number of Queue Processors that you want to start.

Installing and Uninstalling COBOL Analyzer

Before running an installation program described in this section, make sure you have administrative permissions for the machine on which you are performing the installation. If you do not have administrative permissions, the installation program will not let you continue.



Note: You can run the Windows Installer (.msi) installation programs "silently." That is particularly useful when you are installing clients on a network. Check with support services for the appropriate sequence of commands.

Installing CA on the Server or Client

The CA Server hosts CA workspace files and related support files. CA Clients host the link files used to connect to workspaces on the server. Follow the instructions in this section to install CA on the client or server. The Micro Focus License Manager is packaged with the installation.



Note: Having the CA server and the database on separate machines could improve performance. The installation program is the same for the CA client and CA server.

1. Double-click `COBOLAnalyzer.exe`. Note that there might be a version number in the name of the .exe file.
2. Change the installation directory if not installing to the default location.
3. Select which optional installs you want to include:
 - Microsoft SQL Server - Express Edition
 - Web Client
4. Read the End User License Agreement and check **I agree to the End User License Agreement**.
5. Click **Install**.

Uninstalling COBOL Analyzer

Follow the instructions below to uninstall an COBOL Analyzer product. Make sure you close any open CA programs before running the uninstall program.

1. Click **Start > Control Panel > Programs > Programs and Features > Uninstall a program**
2. In the **Uninstall or change a program** window, right-click the product you want to uninstall, then click **Uninstall**. You are prompted to confirm that you want to uninstall the product. Click **Uninstall**.

Post-Installation Administrative Tasks

Before you can work in COBOL Analyzer, you must complete the basic administrative tasks described in this section. You are prompted to perform the first of these tasks, configuring the CA, when you complete the installation.

Configuring COBOL Analyzer

Use the Configuration Manager in the COBOL Analyzer Administration tool to configure CA options and displays for the programming languages, dialects, character sets, and products in use at your site.

If you configure your CA for COBOL, for example, you will see only the CA options and displays appropriate for COBOL modernization. You need to configure your CA for PL/I as well, you can come back to the Configuration Manager later and select PL/I in addition to COBOL.



Note: The COBOL Analyzer Administration tool will prompt to be run with elevated privileges. This is required because some features of the tool require write access to the Program Files directory.

1. Open the Configuration Manager window.

- If you are installing CA, the Configuration Manager window opens after you finish the installation.
- If you have previously installed CA and want to reconfigure it, choose **Start > Programs > Micro Focus > Administration**. The COBOL Analyzer Administration window opens. In the **Administer** menu, choose **Configure Micro Focus**. The Configuration Manager window opens.



Note: If CA is open, you are prompted to exit. Click **OK**, then close CA.

2. Select each programming language, dialect, and character set in use at your site. Select each CA product you want to use at your site. The core Application Analyzer product is always selected. Select **Additional Tools** to enable tools not used in typical CA configurations. When you are satisfied with your choices, click **OK**.



Note: If you modify your CA configuration, make sure you upgrade workspaces created under the previous configuration.



Note: The COBOL Analyzer Administration tool will prompt to be run with elevated privileges. This is required because some features of the tool require write access to the Program Files directory.

Licensing

Although COBOL Analyzer and Business Rule Manager) are both installed by the COBOL Analyzer installer, each product is licensed separately. Follow the instructions in the topics listed below to manage your licenses.

Upgrading Customers

A customer upgrading from Modernization Workbench version 3.1 or later may use the existing license key to enable COBOL Analyzer for the same number of users.

Concurrent Use

You can license COBOL Analyzer and Business Rule Manager for concurrent use. In this mode, one license key for the purchased number of end users will be registered to a license server and each user instance will contact it for authorization.

COBOL Analyzer Licensing

A Sentinel RMS license server is installed with COBOL Analyzer.

Depending on the licensing model in use at your site, you can manage your license:

- Locally, using the installed Sentinel RMS license server.
- Remotely, using a Sentinel RMS license server that has been configured to manage licenses centrally.



Note: The remote license server uses the same software as the local license server that is installed with COBOL Analyzer.

You will need to apply a license in the Micro Focus License Administration to use COBOL Analyzer. To start the Micro Focus License Administration, choose **Start > All Programs > Micro Focus License Manager > License Administration**.

- If you use a local license server, you can apply a license using an authorization code or a licensing file in the **Install** tab of the License Administration.
- **Authorization code:** Enter the 16-character license authorization code for COBOL Analyzer, then click **Authorize**.

- **Licensing file:** Enter your downloaded license file (.mflic) by browsing or dropping the file, then click **Install Licenses**.
- If you use a remote license server, click **Options > Advanced Configuration**, then click **Change** in the **License server** field. Enter the IP address or server name of the license server, then click **Save**.

Your License System Administrator can tell you which licensing model your site uses and provide you with the name of the remote license server if needed. For complete information, including how to configure a remote license server, refer to the help provided with the License Administration tool.

Business Rule Manager Licensing

Trial licenses are not provided for Business Rule Manager. Before you can work with it, you must enter the 16-character license authorization code:

- Choose **Start > Programs > Micro Focus License Manager > License Management System**. In the Licensing System Administration tool, click the Authorize tab, then enter the authorization code in the **Enter authorization code** field and click **Authorize**.

Micro Focus End User License Agreement

IMPORTANT: LICENSOR PROVIDES LICENSED SOFTWARE TO LICENSEE UNDER THIS END USER LICENSE AGREEMENT (THE "AGREEMENT"). THIS AGREEMENT GOVERNS LICENSEE'S INSTALLATION AND USE OF THE VERSION OF THE LICENSED SOFTWARE IDENTIFIED IN THE APPLICABLE PRODUCT ORDER, OR IF NOT ACQUIRED VIA A PRODUCT ORDER, LICENSEE'S INSTALLATION OR USE OF THE LICENSED SOFTWARE CONSTITUTES ACCEPTANCE OF THIS AGREEMENT. **THE TERMS AND CONDITIONS OF THIS AGREEMENT MAY BE DIFFERENT FROM THE AGREEMENT(S) THAT ACCOMPANIED EARLIER RELEASES OF THE LICENSED SOFTWARE. PLEASE READ THIS AGREEMENT CAREFULLY BEFORE PROCEEDING, AS IT MAY CONTAIN ADDITIONAL RESTRICTIONS ON YOUR USE OF THE SOFTWARE. THIS AGREEMENT SUPERSEDES AND CONTROLS OVER ANY OTHER TERMS PROVIDED TO LICENSEE REGARDING LICENSEE'S USE OF THE LICENSED SOFTWARE, WHETHER WRITTEN OR ORAL, AS PART OF A SIGNED AGREEMENT (INCLUDING, BUT NOT LIMITED TO, MASTER AGREEMENTS AND PORTFOLIO TERMS, UNLESS A DIFFERENT AGREEMENT IS EXPRESSLY REFERENCED IN A PRODUCT ORDER OR EXECUTED BY LICENSOR AND LICENSEE SPECIFYING THAT IT APPLIES TO THE VERSION OF THE LICENSED SOFTWARE TO WHICH THIS AGREEMENT RELATES), A CLICK-WRAP AGREEMENT PROVIDED WITH THE LICENSED SOFTWARE OR OTHERWISE (SUCH TERMS REFERRED TO AS THE "OTHER AGREEMENT"), EVEN IF SUCH OTHER AGREEMENT WAS EMBEDDED WITHIN PREVIOUSLY LICENSED SOFTWARE.** LICENSOR RESERVES THE RIGHT TO UPDATE, AMEND, AND/OR MODIFY THIS AGREEMENT FROM TIME TO TIME, AND MAY INCLUDE SUCH UPDATED AGREEMENT WITH OR EMBEDDED IN FUTURE VERSIONS OF THE LICENSED SOFTWARE. PLEASE DIRECT ANY QUESTIONS TO THE MICRO FOCUS LEGAL DEPARTMENT AT LEGALDEPT@MICROFOCUS.COM.

ENTERING INTO THIS AGREEMENT DOES NOT CONSTITUTE A SALES TRANSACTION. THE SALE OF A LICENSE TO SOFTWARE PRODUCTS TAKES PLACE UNDER PRODUCT ORDERS WHICH (UNLESS OTHERWISE STATED IN THE PRODUCT ORDER) INCORPORATE THE TERMS OF THIS AGREEMENT.

Capitalized terms in this Agreement are defined as follows:

"**Additional License Authorization**" or "**ALA**" means the additional specific software license terms that govern the authorized use of a given software product of Licensor, including requirements for any non-production use rights described in the Non-Production Licensing Guide, and License Options available for that software product, along with additional terms or conditions applicable to a given License Option, all of which are made a part of this Agreement. The applicable Additional License Authorization for the Licensed Software is either attached hereto or can be found here: <https://software.microfocus.com/en-us/about/software-licensing> by product name and version; all references in this Agreement to "Additional License Authorization" or "ALA" shall refer to the ALA that corresponds to the version of the Licensed Software.

"Documentation" means the user documentation that Licensor makes available for the Licensed Software in electronic form or paper form.

"Licensee" means the legal entity or individual that is identified in the applicable Product Order or who has rightfully received a license to the Licensed Product.

"License Option" means the type of license available for a given software product (such as a named user license, concurrent user license or server license). In addition to the ALA, a License Option may be set forth in a Product Order or an agreement executed in writing by Licensee and Licensor.

"Licensor" means the applicable Micro Focus entity and its affiliates who own the intellectual property rights in Licensed Product.

"Licensed Product" means the Licensed Software and Documentation.

"Licensed Software" means the executable version of Licensor's software listed in the Product Order or otherwise provided to or rightfully acquired by Licensee. This Agreement shall govern the use of any update to the Licensed Software that Licensee receives pursuant to a separate support and maintenance agreement as described in Section 4 below, unless such update contains, comes with, or is otherwise specifically governed by a different end user license agreement.

"Product Order" means an agreement between Licensor and Licensee that consists of a document that has been (i) submitted by Licensee describing the License Option(s) to be purchased for the Licensed Software, and (ii) accepted by Licensor (a) in writing or (b) by delivering the Licensed Software to Licensee, whichever occurs first. A Product Order may also consist of a written quote, or other written document issued by Licensor (the "Quote"), (i) describing the License Options for the Licensed Software to be purchased, and (ii) that is accepted by Licensee before the Quote expires either by (a) returning the Quote signed by an authorized representative of Licensee, (b) issuing a purchase order that references the Quote (if the Quote expressly allows acceptance in this manner), or (c) paying Licensor the fees listed in the Quote. Unless otherwise expressly set forth in the relevant Product Order, each Product Order incorporates the terms and conditions of this Agreement, and in no event will any different or additional terms of a purchase order or similar document issued by Licensee in connection with this Agreement or a Product Order apply, and any such additional or different terms are hereby rejected by the Licensor. For purposes of this paragraph, "Licensor" shall also include the relevant Micro Focus entity as defined above and its authorized distributors and resellers. Any conflicting or additional terms in a Product Order accepted by an authorized distributor or reseller of Licensor shall have no effect unless such terms have been agreed to by the applicable Micro Focus entity in writing.

"Third Party Component" means any run time or other elements owned or licensed to Licensor by a third party (other than open source code or elements) which may be embedded in the Licensed Software.

"Third Party Software" means additional or accompanying software owned or licensed by a third party (such as Adobe Acrobat or Microsoft Internet Explorer, but not any open source code or elements) that may be specified in the Documentation or in a file accompanying such Licensed Software.

"Warranty Period" means the ninety (90) day time frame beginning on date of delivery of the Licensed Software to Licensee (Licensed Software delivery is deemed to occur when the Licensed Software is physically delivered to Licensee EXWORKS or made available for download to Licensee).

1. GRANT OF LICENSE; LICENSE CONDITIONS.

- a. License Grant. Subject to Licensee's compliance with the terms and conditions of this Agreement (including but not limited to payment of applicable fees), Licensor grants to Licensee a personal, non-transferable, non-sublicensable and non-exclusive license to use the Licensed Software as authorized by the License Option(s) specified in the ALA solely for Licensee's internal business operations, functions and benefit, and not for commercialization of the Licensed Software or to provide services or benefit to any affiliates or subsidiaries of Licensee or any other third party. Throughout the license term, Licensee agrees to: (i) implement internal safeguards to prevent any unauthorized copying, distribution, installation, use of, or access to, the Licensed Products and associated support and maintenance, or any other breach of this Agreement; and (ii) take all necessary steps to destroy or erase all Licensed Software codes, programs, Documentation, and other proprietary information of Licensor before disposing of any media or hardware. Licensor will

provide any license key necessary for activation and use of the Licensed Software. Licensor is not liable or responsible for lost or broken license keys and is not obligated to replace license keys or issue new license keys unless (1) Licensee has purchased a support and maintenance plan for the applicable Licensed Software that specifically covers the issuance of new or replacement keys and (2) the applicable version of the Licensed Software is then generally available for distribution by Licensor. If Licensee has not paid for such a support and maintenance plan, replacement or new license keys may be available for purchase at Licensor's then-current list price for applicable new licenses.

- b. **Evaluation Licenses.** With respect to any Licensed Software provided to Licensee solely for evaluation purposes (an "Evaluation License"), in the event of conflict this Section 1.b shall prevail over any other provisions set forth in this Agreement. An Evaluation License may be used for a period of no more than thirty (30) days from the date the Licensed Software is provided to Licensee ("Evaluation Term"), unless a different period is specified in writing by Licensor. An Evaluation License may be used solely for Licensee's internal evaluation and testing purposes on a single computer system and not for development, commercial, or production purposes. For Licensed Software subject to an Evaluation License, (i) Licensee may not reproduce or distribute the Licensed Products; and (ii) Licensee's results of benchmark or other performance tests run on or using the Licensed Software may not be disclosed to any third party without Licensor's prior written consent. At any time during the Evaluation Term or upon completion thereof, Licensee may, upon written notification to Licensor and payment of the applicable license fee, replace the Evaluation License with a license to use the Licensed Software that is not restricted to evaluation purposes. In the absence of such notification by Licensee, the Evaluation License shall automatically terminate at the end of the Evaluation Term, and Licensee shall return, or, if Licensor so directs, delete and destroy all such Licensed Software and provide Licensor with written confirmation of its compliance with this provision. Upon written request from Licensee, Licensor may, in its sole discretion, grant Licensee an extension in writing prior to the expiration of the Evaluation Term. Other than updates to Licensed Software provided as part of support and maintenance, Licensed Software provided by Licensor free of license fee charge shall be deemed to be provided for evaluation purposes only. Licensed Software furnished under an Evaluation License is provided without any contractual obligation of maintenance and support by Licensor and is provided "as is" without warranties, implied or express, of any kind.
- c. **Bundles/Suites.** If the Licensed Software is licensed in a bundle or suite of multiple products, and the applicable Product Order specifies the License Option and license count for the bundle or suite (but not the individual product components of the bundle or suite), then each product in the bundle or suite shall share such license type and count. For example, individual products in the bundle or suite cannot be used by multiple users if only one user license is purchased (for user-based licenses) and cannot be installed on multiple devices or servers if only one device or server license is purchased (for device or server-based licenses).

2. USE RESTRICTIONS. Except as may be otherwise specifically permitted in the applicable ALA, Licensee agrees not to, directly or indirectly:

- a. Copy, distribute or use the Licensed Software, in whole or in part (such as any portion, feature, function, or user interface) without paying Licensor the applicable fees;
- b. Use the Licensed Software as a service, or for timesharing, facilities management, outsourcing, hosting, service bureau use, or for providing other application service (ASP) or data processing services to third parties or for like purposes, or permit the use of the Licensed Software by a third party or permit access by or use for the benefit of any third party without executing a separate distribution agreement for the Licensed Software and paying Licensor the applicable required additional fees;
- c. Modify or create derivative works of the Licensed Software, or decrypt, translate, disassemble, recompile, decompile or reverse engineer the Licensed Software or attempt to do so (except to the extent applicable law specifically permits such activity, in which case Licensee must provide Licensor with detailed information regarding such activities);
- d. Alter, destroy, or otherwise remove any proprietary notices or labels on or embedded within the Licensed Software;
- e. Use the Licensed Software in a manner other than as specifically permitted in this Agreement or an ALA;

- f. Assign, sell, resell, license, rent, lease, lend, sublicense, outsource or otherwise transfer the Licensed Software to any third party, without first paying Licensor the applicable required license fees and obtaining Licensor's prior written consent;
- g. Authorize, allow or appoint any third party to do any of the foregoing. For the avoidance of doubt, third parties include, without limitation, contractors and consultants (including contractors and consultants retained to provide services solely for the benefit of Licensee), outsourcers, Licensee's affiliates and subsidiaries, parent companies, customers, and the public; or
- h. Publish or disclose to third parties any evaluation or benchmarking of the Licensed Software without Licensor's prior written consent.

Notwithstanding the foregoing, Licensee may: (i) make a reasonable number of archival back-up copies of the Licensed Software and (ii) make a reasonable number of copies of the Documentation. Licensee shall reproduce all copyright and other proprietary rights notices appearing in or on the Licensed Products, including notices of all third party suppliers.

- 3. TERM OF LICENSE.** This Agreement and the license term for the Licensed Software granted herein is perpetual, unless a subscription/term license has been purchased by Licensee (in which case the license term shall be set forth in the Product Order or ALA), and is subject to earlier termination as provided in this Section 3. If Licensee has purchased a subscription/term license, such license shall automatically terminate upon expiry of such subscription/term, unless earlier terminated under this Section 3. Licensor may terminate this Agreement, along with any or all licenses then in effect with Licensee, immediately by giving Licensee written notice of termination in the event that (i) Licensee breaches any term or condition of this Agreement and fails to remedy such breach within ten (10) days of receipt of Licensor's notice detailing such breach; (ii) Licensee becomes insolvent, has a receiver appointed, or files for or has filed against it, liquidation, bankruptcy or analogous proceedings; or (iii) Licensee infringes or misappropriates the intellectual property rights of Licensor. Termination shall be without prejudice to any other rights or remedies Licensor may have. In the event of any termination, Licensee's license(s) to install, access or use the Licensed Software will immediately terminate, and Licensee shall destroy and erase all copies of such Licensed Software in its possession or control and provide written certification to Licensor that it has complied with this provision. Early termination of this Agreement shall not entitle Licensee to any refund or reimbursement of any previously paid fees. The rights and obligations of the parties contained in Sections 3 (Term of License), 6 (Disclaimer of Warranty), 7 (Limitation of Liability), 8 (High Risk Uses), 9 (Ownership), 10 (Third Party Software and Components), 11 (Notice to U.S. Government End Users), 12 (License Fees and Payment Terms), 13 (Audits), 15 (Privacy and Use of Licensee Information), 16 (Licensee Trademark and Feedback) and 17 (Miscellaneous) will survive the termination or expiration of this Agreement.
- 4. SUPPORT AND MAINTENANCE.** Licensee is not entitled to any updates to the Licensed Software, unless Licensee purchases maintenance and support services pursuant to Licensor's then-current applicable standard maintenance and support agreement, which can be found at <https://www.microfocus.com/support-and-services/maintenance-and-support-agreements/> or can be provided by Licensor at Licensee's request. Maintenance and support services (including, but not limited to, any new versions, bug fixes, and patches) provided by Licensor will be subject to such agreement. Where Licensee purchases maintenance and support for any Licensed Software, Licensee hereby agrees to purchase or keep current on such maintenance and support services for all of Licensee's licensed units of such Licensed Software product, regardless of License Option.
- 5. LIMITED WARRANTY.** Licensor warrants for the Warranty Period that: (i) if the Licensed Software is supplied via media, the media will be free from defects in materials or workmanship under normal use, and (ii) the copy of the Licensed Software delivered to Licensee substantially conforms in all material respects to the Documentation. Licensee's sole and exclusive remedy for any defective media supplied by Licensor shall be Licensor's repair or replacement of such defective media free of charge, provided that the defective media is returned to Licensor during the Warranty Period. During the Warranty Period, Licensee's sole and exclusive remedy for not meeting part (ii) of the above warranty shall be the repair or replacement of the Licensed Software by Licensor free of charge so that it substantially conforms to the Documentation or, if Licensor reasonably determines that such remedy is not economically or technically feasible, Licensee shall be entitled to a full refund of the license fee and any maintenance fee paid for such Licensed Software. Upon such refund, Licensee's license to use such Licensed Software will immediately terminate. The warranties set forth in this Section 5 shall not apply if the defects in the Licensed Software or media result from: (a) failure to use the Licensed Software in

accordance with the Documentation, this Agreement or ALAs; (b) the malfunctioning of Licensee's equipment or network; (c) accident, neglect, or abuse; (d) service by any unauthorized person; (e) other software used by Licensee and not provided by Licensor, or for which the Licensed Software is not designed or licensed for such use; (f) Third Party Software that is not a Third Party Component; (g) any other cause occurring after initial delivery of the Licensed Software or media to Licensee, unless caused directly by Licensor. Licensor has no responsibility for any claims made outside of the Warranty Period. The foregoing warranty shall not apply to any free-of-charge Licensed Software or updates provided under support and maintenance. THE FOREGOING WARRANTIES DO NOT APPLY, AND LICENSOR DISCLAIMS ALL WARRANTIES, WITH RESPECT TO ANY THIRD PARTY SOFTWARE THAT IS NOT A THIRD PARTY COMPONENT. The warranties set forth in this Section 5 will not apply and will become null and void if Licensee materially breaches any provision of this Agreement.

6. DISCLAIMER OF WARRANTY. EXCEPT FOR THE LIMITED WARRANTY SET FORTH IN SECTION 5, THE LICENSED PRODUCTS ARE PROVIDED TO LICENSEE "AS-IS" WITHOUT WARRANTY OF ANY KIND. LICENSOR DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE LICENSED SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS, THAT OPERATION WILL BE UNINTERRUPTED, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE, OR WORK IN COMBINATION WITH ANY OTHER SOFTWARE, APPLICATIONS, OR SYSTEMS, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS, OR BE ERROR FREE, OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED. EXCEPT AS SET FORTH HEREIN AND TO THE EXTENT PERMITTED BY LAW, ALL OTHER WARRANTIES WITH RESPECT TO THE LICENSED PRODUCTS, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, NON-INFRINGEMENT, AND WARRANTIES THAT MAY ARISE OUT OF COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE, OR TRADE PRACTICE ARE EXPRESSLY DISCLAIMED BY LICENSOR AND ITS THIRD-PARTY SUPPLIERS AND AFFILIATES. LICENSEE ACKNOWLEDGES THAT LICENSEE IS RESPONSIBLE FOR THE SELECTION OF THE LICENSED SOFTWARE TO ACHIEVE LICENSEE'S INTENDED RESULTS AND FOR THE INSTALLATION AND/OR USE OF, AND RESULTS OBTAINED FROM, THE LICENSED SOFTWARE.

7. LIMITATION OF LIABILITY.

- a. Aggregate Cap. IN NO EVENT SHALL ANY LIABILITY OF LICENSOR OR ITS AFFILIATES OR ANY OF ITS OR THEIR RESPECTIVE LICENSORS OR SERVICE PROVIDERS UNDER OR IN CONNECTION WITH THIS AGREEMENT EXCEED, IN THE AGGREGATE, THE AMOUNTS PAID BY LICENSEE FOR THE LICENSED SOFTWARE AND THE INITIAL PERIOD OF MAINTENANCE AND SUPPORT GIVING RISE TO THE APPLICABLE CLAIM.
- b. Waiver of Liability. IN NO EVENT SHALL LICENSOR OR ITS AFFILIATES OR ANY OF ITS OR THEIR RESPECTIVE LICENSORS OR SERVICE PROVIDERS BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR SIMILAR DAMAGES, LOSS OF PROFITS, BUSINESS, DATA, OR PROGRAMS (INCLUDING, BUT NOT LIMITED TO, THE COST OF RECOVERING OR REPLACING SUCH DATA OR PROGRAMS), LOSS, DAMAGE OR ANY COSTS DUE TO INTERRUPTION, DELAY, OR INABILITY TO USE THE LICENSED SOFTWARE, WHETHER ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT, EVEN IF INFORMED OF THE POSSIBILITY OF SUCH DAMAGES IN ADVANCE.
- c. Scope. THE LIMITATIONS AND EXCLUSIONS OF THIS SECTION 7 APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS. THESE LIMITATIONS AND EXCLUSIONS APPLY COLLECTIVELY TO LICENSOR, ITS PARENTS, AFFILIATES, AND SUBSIDIARIES AND EACH OF THEIR RESPECTIVE EMPLOYEES, CONTRACTORS, AND SUPPLIERS. NOTWITHSTANDING THE FOREGOING, NOTHING IN THIS SECTION 7 EXCLUDES LIABILITY FOR WILLFUL MISCONDUCT OR FRAUDULENT MISREPRESENTATION.
- d. Exclusive Remedy. LICENSEE'S REMEDIES IN THIS AGREEMENT ARE LICENSEE'S EXCLUSIVE REMEDIES. LICENSEE AGREES THAT, IN ENTERING INTO THIS AGREEMENT, IT DID NOT RELY ON ANY REPRESENTATIONS (WHETHER WRITTEN OR ORAL) OF ANY KIND OTHER THAN THOSE EXPRESSLY SET OUT IN THIS AGREEMENT.
- e. Essential Purpose. LICENSEE FURTHER ACKNOWLEDGES THAT THE LIMITATIONS AND EXCLUSIONS OF LIABILITY IN THIS SECTION 7 APPLY TO THE FULLEST EXTENT

PERMITTED BY LAW AND ARE AN ESSENTIAL ELEMENT OF THIS AGREEMENT AND THAT, IN THE ABSENCE OF SUCH LIMITATIONS AND EXCLUSIONS, THE PRICING AND OTHER TERMS AND CONDITIONS SET FORTH HEREIN WOULD BE SUBSTANTIALLY DIFFERENT. THE LIMITATIONS AND EXCLUSIONS SET FORTH IN THIS SECTION 7 SHALL APPLY EVEN IF THE LICENSEE'S REMEDIES UNDER THIS AGREEMENT FAIL OF THEIR ESSENTIAL PURPOSE.

- f. **Free Software.** IF LICENSOR PROVIDES LICENSEE WITH ANY LICENSED SOFTWARE FREE-OF-CHARGE OR UNDER AN EVALUATION LICENSE, TO THE EXTENT PERMITTED BY LAW, LICENSOR SHALL NOT BE RESPONSIBLE FOR ANY LOSS OR DAMAGE TO LICENSEE, ITS CUSTOMERS, OR ANY THIRD PARTIES CAUSED BY THE LICENSED SOFTWARE THAT IT MAKES AVAILABLE TO LICENSEE.
- 8. HIGH RISK USES.** The Licensed Software is not fault tolerant, nor designed, manufactured, or intended for use in hazardous environments requiring fail-safe performance (including, without limitation, the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems) in which failure of the Licensed Software could lead directly or indirectly to death, personal injury, or severe physical or environmental damage. Licensor and its suppliers shall have no liability for any use of the Licensed Software in any high-risk situations.
- 9. OWNERSHIP.** The Licensed Products are provided under license, and not sold, to Licensee. The only rights in the Licensed Products are the licenses expressly stated in this Agreement and no other rights are implied or granted by estoppel. Licensor (and its affiliates) and its and their licensors and third party suppliers retain ownership of, and reserve all rights in and to, the Licensed Products, including all copies thereof, and all intellectual property rights arising out of or relating to the Licensed Products. Licensee shall use reasonable efforts to safeguard the Licensed Products (including all copies thereof) from infringement, misappropriation, theft, misuse, or unauthorized access. Licensee shall promptly notify Licensor if it becomes aware of any infringement or misappropriation of the Licensed Products and shall fully cooperate with Licensor, at Licensor's expense, in any legal action taken by Licensor to enforce its intellectual property rights.
- 10. THIRD PARTY SOFTWARE AND COMPONENTS.** The Licensed Software may come with or require Third Party Software that Licensee shall license directly from the third party licensor pursuant to such third party's terms and conditions and not this Agreement. Additionally, some Licensed Software may include certain Third Party Components and open source software. Such open source software and Third Party Components may also be loaded on the Licensed Software media. Third Party Components are licensed to Licensee under this Agreement; open source software is licensed pursuant to the applicable open source license. To the extent applicable, information about the open source software may be found (i) in a file accompanying the applicable Licensed Software or (ii) in the Documentation or ALA. Licensee shall not directly access any Third Party Components other than with or as part of the Licensed Software. Licensee agrees that to the extent required by a third party licensor or supplier of a Third Party Component, that third party licensor or supplier is an intended third party beneficiary of this Agreement as necessary to protect intellectual property rights in the Licensed Software and limit certain uses thereof.
- 11. NOTICE TO U.S. GOVERNMENT END USERS.** The Licensed Products are deemed to be "Commercial Items," as defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with such sections, the Licensed Products are licensed to U.S. Government end users (i) only as Commercial Items, and (ii) with only those rights as are granted pursuant to this Agreement.
- 12. LICENSE FEES AND PAYMENT TERMS.** Licensee agrees to pay the applicable license fees for the Licensed Products within thirty (30) days of the date of invoice or such other date as agreed in writing by the parties. Software license fees are non-refundable, except as provided in Section 5 above, and shall be paid without any deduction or tax withholding. Software License fees are exclusive of any applicable transportation charges, sales, use, value added tax, and other applicable taxes and duties, and all such amounts shall be paid or reimbursed by Licensee. Licensee shall be liable for all outstanding past due amounts, which shall accrue interest at the rate of 1.5% per month compounded or, if lower, the maximum rate allowed by applicable law, and any collection costs associated with the collection of any past due amounts.
- 13. AUDITS.** Licensor or an Auditor (as defined below) has the right to verify Licensee's compliance with the licenses issued under Product Orders, the applicable ALAs and this Agreement (please see Micro

Focus License Compliance Charter at <http://supportline.microfocus.com/licensing/licVerification.aspx>, which can also be provided by Licensor at Licensee's request). Licensee agrees to:

- a. Recordkeeping. Keep, and upon Licensor's request, provide records, sufficient to certify Licensee's compliance with this Agreement based on the applicable License Option(s) (including applicable license metric and other terms and conditions) for the Licensed Software, which may include but are not limited to, serial numbers, license keys, logs, the location, model (including quantity and type of processor) and serial number of all machines on which the Licensed Software is installed or accessed or from which the Licensed Software can be accessed, the names (including corporate entity) and number of users accessing or otherwise able to access the Licensed Software, metrics, reports, copies of the Licensed Software (by product and version), and network architecture diagrams as they may relate to Licensee's licensing and deployment of the Licensed Products and associated support and maintenance;
- b. Questionnaire. Within seven (7) days of Licensor's request, Licensee shall furnish to Licensor or its designated independent auditor ("Auditor") a completed questionnaire provided by Licensor or Auditor, accompanied with a written statement signed by a director of Licensee certifying the accuracy of the information provided; and
- c. Access. Provide representatives of Licensor or Auditor any necessary assistance and access to records and computers to allow an inspection and audit of Licensee's computers and records, during Licensee's normal business hours, for compliance with licenses, the applicable ALAs, and this Agreement, and fully cooperate with such audit.
- d. Non-Compliance. In the event that Licensee has, or at any time has had, unlicensed installation, use of, or access to the Licensed Software or has otherwise breached this Agreement or an ALA (a "Non-Compliance"), without prejudice to any other rights or remedies Licensor may have, including, without limitation, injunctive relief, Licensee shall, within thirty (30) days' notice of such Non-Compliance to Licensee, purchase sufficient licenses and/or subscriptions and associated support and maintenance to cure the Non-Compliance, by paying Licensor's current (as of the date of such additional purchase) list license fees and 12-month support and maintenance fees to Licensor for such additional licenses, plus Licensor's current (as of the date of such additional purchase) list term license and support and maintenance fees and interest (compounded at 1.5% monthly or the maximum rate permitted by applicable law if lower) for such additional licenses for the time period from the commencement of the Non-Compliance until payment of the aforementioned fees, with interest payable even if an invoice was not issued at the time the Non-Compliance occurred. For purposes of the foregoing, "list" shall mean Licensor's full list price as set forth in Licensor's standard price list that is current as of the commencement of the audit without any volume or other discount. If Licensee's Non-Compliance results in an underpayment of license fees of 5% or greater, Licensee shall also reimburse Licensor for the reasonable cost of such audit in addition to other amounts due. In the event of a dispute related to a Non-Compliance, Licensor shall have the right to collect from Licensee its reasonable costs and attorneys' fees incurred in enforcing this Agreement.

14. RELATED SERVICES. Licensee shall be responsible for obtaining and installing all proper hardware and other third party support software (including operating systems) for the proper installation and implementation of the Licensed Software. In the event that Licensee retains Licensor to perform any services with respect to the Licensed Software (for example: installation, implementation, maintenance, consulting, or training services), Licensee and Licensor agree that such services shall be provided at Licensor's then- current standard terms, conditions, and rates for such services unless otherwise agreed in writing by Licensor.

15. PRIVACY AND USE OF LICENSEE INFORMATION.

- a. Responsibility and Compliance with Laws. Licensee is solely responsible for and assumes all liability with respect to its own collection, processing, storage, and transfer of any user data, including, but not limited to, personally identifiable information and personal health and financial information (collectively, "Personal Information"). Licensee shall be solely responsible for notifying its users of proper use of such data. Each party is responsible for complying with its respective obligations under all applicable laws, regulations, and industry standards regarding data collection and data privacy applicable for the use of the Licensed Software by the relevant party.

Licensee shall not provide any Personal Information to Licensor for processing by Licensor on behalf of Licensee, unless otherwise agreed by the parties in writing in an applicable transaction document

with applicable privacy terms. If the parties agree that processing Personal Information is necessary for the performance of this specific transaction, and when such Personal Information processing falls within the scope of the General Data Protection Regulation (EU) 2016/679 ("GDPR"), before any Personal Information is made available to Licensor, the parties agree that Licensee will be the data controller and Licensor will be the data processor, and when Licensor is processing Personal Information on behalf of Licensee, such processing shall be governed by terms that comply with Article 28 of the GDPR including standard contractual clauses to be included in such transaction document.

Licensor will not have access to protected health information unless the parties have an executed business associate agreement in place for this transaction. Licensee is solely responsible for assessing the Licensed Product or any related product or service for compliance with any industry requirements applicable to Licensee.

- b. Consent to Use of Licensee Information.** To the extent required or permitted by law, Licensee hereby expressly consents to (i) receiving information from Licensor from time to time advertising Licensor's products; (ii) the use of Licensee's name in Licensor customer lists, promotional materials, and press releases; and (iii) the collection and use of information about the computer system on which the Licensed Software is installed (e.g. product version, serial number) for internal security and licensing purposes. Further information about Licensor's processing of personally identifiable data is available at <https://www.microfocus.com/about/legal/#privacy> (click "Privacy Notice" tab) or can be provided by Licensor at Licensee's request.
- c. Other Use of Licensee Information.** To the extent required or permitted by law, and notwithstanding the terms in Section 15.a, Licensor may also process personally identifiable information of Licensee and Licensee's users (i) in order to comply with a legal obligation to which Licensor is subject; (ii) as is necessary for the performance of this Agreement; and (iii) where necessary for the purposes of Licensor's legitimate interests, except where such interests are overridden by the interests or fundamental rights and freedoms of the Licensee or Licensee's users which require protection of personally identifiable information.

16. LICENSEE TRADEMARK AND FEEDBACK. Licensor may use Licensee's name and logo for business development and marketing purposes, including, but not limited to, online and printed sales and marketing materials. Any other use of Licensee's name or logo, or a description of Licensee's use of the Licensed Software, shall be subject to Licensee's prior consent. Any suggestions, ideas for modifications, enhancements, and other feedback from Licensee regarding the Licensed Software provided at any time (collectively, the "Feedback"), including (but not limited to) all intellectual property rights in and to such Feedback, shall be owned exclusively by Licensor. Licensee hereby assigns all right, title and interest in and to such Feedback and all the intellectual property rights therein to Licensor, without the necessity of any further consideration. To the extent any Feedback cannot be assigned to Licensor, Licensee hereby grants to Licensor a perpetual, irrevocable, exclusive, worldwide, royalty-free, fully paid up license, with the right to sublicense through multiple tiers to use, make, sell, distribute, execute, adapt, translate, reproduce, display, perform, modify, create derivative works of and otherwise exploit the Feedback in any manner.

17. MISCELLANEOUS.

- a. Assignment.** Licensor may assign this Agreement, including any rights or obligations under the Agreement (in whole or in part) to a parent or an affiliate. Licensee may not assign or transfer this Agreement or any of its rights or duties hereunder, including (but not limited to) by merger, acquisition by any entity of all or substantially all of Licensee's stock or assets, change of control, operation of law, or otherwise, without the prior written consent of Licensor and payment by Licensee of the applicable assignment fee. Any attempted assignment not in accordance with this Section shall be null and void.
- b. Governing Law.** If Licensee is located in North America, the laws of the State of California govern this Agreement and the licenses granted hereunder, and the parties hereto consent to the exclusive jurisdiction of the State and Federal courts of the State of California in any action based on this Agreement or the Licensed Software hereunder or any License Option under an ALA. Each party waives any right it may have to object to such venue, including objections based on personal jurisdiction or forum non conveniens (inconvenient forum). The parties agree that the Uniform Computer Information Transaction Act or any version thereof, adopted by any state, in any form

("UCITA"), shall not apply to this Agreement. To the extent that UCITA is applicable, the parties hereby opt out of the applicability of UCITA pursuant to the opt-out provision(s) contained therein. If Licensee is located in France, Germany or Japan, this Agreement is governed by the laws of the country in which Licensee is located. In the rest of the world the laws of England govern this Agreement. In each case, the applicable law shall apply without regard to conflict of laws provisions thereof, and without regard to the United Nations Convention on the International Sale of Goods. Other than for North American transactions, this Agreement, the licenses granted hereunder, and the parties hereto, shall be subject to the exclusive jurisdiction of the courts of the country determining the applicable law as aforesaid.

- c. **Export Control.** This Agreement may be subject to export control laws, regulations, and other restrictions of the United States (including, but not limited to, the U.S. Export Administration Regulations (the "EAR")), United Kingdom, or the European Union regarding export or re-export of computer software and technology. Licensee agrees to comply with all applicable export control laws, regulations, and restrictions, including the EAR, where applicable.
- d. **Entire Agreement.** The applicable Product Order and this Agreement including the applicable ALA, constitutes the complete and exclusive statement of agreement between the parties relating to the license for the Licensed Products and supersedes all prior proposals, communications, purchase orders, and agreements (including, without limitation, Other Agreements), without need for a mutually executed amendment to any such Other Agreement. Any conflicting terms and conditions shall be resolved according to the following order of precedence: the applicable Product Order, the applicable ALA, and then this Agreement in all other respects.
- e. **Amendment.** No representation, supplement, modification, or amendment of this Agreement will be binding on either party unless executed in writing by duly authorized representatives of both parties (excluding any distributor or reseller of Micro Focus) to this Agreement.
- f. **Waiver.** No waiver of any right under this Agreement will be effective unless in writing and signed by authorized representatives of both parties (excluding any distributor or reseller of Licensor). No waiver of any past or present right arising from any breach or failure to perform will be deemed to be a waiver of any future right arising under this Agreement.
- g. **Severability.** If any provision in this Agreement is invalid or unenforceable, that provision will be construed, limited, modified or, if necessary, severed, to the extent necessary, to eliminate its invalidity or unenforceability, and the other provisions of this Agreement will remain unaffected.
- h. **No Reliance.** Each party acknowledges that in entering into this Agreement it has not relied on any representations, agreements, warranties or other assurances (other than those repeated in this Agreement) and waives all rights and remedies which but for this Section 17 would be available to it.

Micro Focus EULA (1 November 2019)

Additional License Authorizations for Visual COBOL & Enterprise software products

This Additional License Authorizations document ("ALA") set forth the applicable License Options and additional specific software license terms that govern the authorized use of the software products specified below, and are part of the applicable agreement (i.e., Micro Focus End User License Agreement; and/or any separate agreement that grants Licensee a license to such products (e.g., Customer Portfolio Terms or other Master Agreement); and/or Quotation) (the "Applicable Agreement"). Capitalized terms used but not defined herein shall have the meanings set forth in the Applicable Agreement.

Products and Suites covered

Products	E-LTU or E-Media available *	Non-production software class **	Term License Non-production software class (if available)
Business Rule Manager	Yes	Class 2	Class 2
COBOL Analyzer	Yes	Class 2	Class 2
COBOL Server	Yes	Class 2	Class 2

Products	E-LTU or E-Media available *	Non-production software class **	Term License Non-production software class (if available)
COBOL Server for Stored Procedures	Yes	Class 2	Class 2
COBOL 2010 Runtime	Yes	Class 2	Class 2
COBOL 2010 Runtime Test Server	Yes	Class 2	Class 2
Database Connectors (all variants)	Yes	Class 2	Class 2
Enterprise Analyzer (all variants)	Yes	Class 2	Class 2
Enterprise Analyzer Server (all variants)	Yes	Class 2	Class 2
Enterprise Developer (all variants)	Yes	Class 2	Class 2
Enterprise Server (all variants)	Yes	Class 2	Class 2
Enterprise Server for Stored Procedures (all variants)	Yes	Class 2	Class 2
Enterprise Test Server	Yes	Class 2	Class 2
Enterprise Test Server Premium	Yes	Class 2	Class 2
Enterprise View	Yes	Class 2	Class 2
Relativity Designer Suite for MF COBOL	Yes	Class 2	Class 2
Relativity for Windows Workstations for MF COBOL	Yes	Class 2	Class 2
Relativity Server for MF COBOL	Yes	Class 2	Class 2
Visual COBOL (all variants)	Yes	Class 2	Class 2
Visual COBOL Database File Handler	Yes	Class 2	Class 2

Suites	E-LTU or E-Media available *	Non-production software class **	Term License Non-production software class (if available)
Visual COBOL for Eclipse Distributed Edition	Yes	Class 2	Class 2
Visual COBOL for ISVs	Yes	Class 2	Class 2
Visual COBOL Studio Distributed Edition for ISVs	Yes	Class 2	Class 2

* Any product sold as E-LTU or E-Media shall be delivered electronically regardless of any contrary designation in a purchase order.

** Additional licenses solely for non-production use may be available as specified in the Non-Production Licensing Guide found at software.microfocus.com/legal/software-licensing depending on the non-production software class specified above. Any such non-production licenses will be subject to the Non-Production Licensing Guide and the applicable License Option terms and conditions set forth in this ALA.

Definitions

Term	Definition
Authorized User	Means a single user that has been authorized by Licensee to use the Licensed Software or any device or other software program that may access the Licensed Software, either directly or indirectly through any other software program, regardless of how such access occurs or if such individual uses any hardware or software that reduces the apparent number of users who are using the Licensed Software, such as by using a terminal service.
Batch Processing	Means the execution of one or more tasks, transactions or programs by a computer system without the continuous interaction of the end user with the Licensed Software for the entire period during which each such task, transaction or program is being executed by the system. Batch processing includes situations where a queue of jobs is processed as resources become available and is in contrast to transaction or interactive processing.
Concurrent Users	Means individuals who, at the same point in time, have accessed or used the Licensed Software via any device or other software program, directly or indirectly, regardless of how such access occurs or whether such user uses any hardware or software that reduces the apparent number of users who are using the Licensed Software, such as by using a terminal service.
Container	Means an isolated user-space instance created using operating-system-level virtualization, also known as containerization. A program running inside a Container whether standalone or using an orchestration technology, can only see the resources and devices assigned to the Container. Examples of such container technologies include Docker and podman and examples of orchestrators include Kubernetes and Open Shift.
Core	Means a physical (not virtual) subunit within a CPU on a single chip that handles the main computational activities of a computer. A CPU may have one or more Cores and therefore be a "Multicore CPU" if it has more than one Core.
CPU or Central Processing Unit	Means a data processing unit, normally identified with a single microchip mounted on a single circuit board and can comprise several Cores.

Term	Definition
E-LTU <i>and</i> E-Media	Means products which are electronically delivered only, and as such any reference to FOB Destination or delivery methods that are stated on your purchase order other than electronic shall be null and void with respect to these E-LTU or E-Media products.
Hard Partitioning	Means using hard physical partitioning to physically segment a single larger server or machine into separate and distinct smaller systems where each separate system acts as a physically independent, self-contained server or machine with its own CPUs, operating system, separate boot area, memory, input/output subsystem and network resources (each known as a "Hard Partition"). Examples of Hard Partitioning methods include: Dynamic System Domains (DSD) -- enabled by Dynamic Reconfiguration (DR), Solaris Zones (also known as Solaris Containers, capped Zones/Containers only), IBM LPAR (adds DLPAR with AIX 5.2), IBM Micro-Partitions (capped partitions only), vPar, nPar, Integrity Virtual Machine (capped partitions only), Secure Resource Partitions (capped partitions only), and Fujitsu's PPAR. All approved hard partitioning technologies must have a capped or a maximum number of Cores/processors for the given partition.
Named User	Means a single user that has been authorized by Licensee to use the Licensed Software.
Platform	Means a hardware chipset (e.g., PA-RISC, Itanium, x86, or SPARC) and operating system (e.g., Windows, Linux, Solaris, AIX, or HP-UX) combination. Each unique Linux distribution is considered a different operating system from other Linux distributions and a different Platform even with the same hardware chipset.
Soft Partitioning	Means using an operating system resource manager to segment and limit the number of Cores, CPUs, or other processing devices utilized by the Licensed Software by creating areas where processor resources are allocated and limited within the same operating system (each such area known as a "Soft Partition"). Examples of such Soft Partitioning include: Solaris 9 Resource Containers, AIX Workload Manager, HP Process Resource Manager, Affinity Management, Oracle VM, and VMware.
vCPU	Means a virtual central processing unit that represents a portion or share of the underlying, physical CPU that is assigned to a particular Virtual Machine.
Virtual Machine	Means a software implementation that can run its own operating system and execute programs like a physical machine where the software running inside the virtual machine is limited to the resources and abstractions provided by the virtual machine.

License Options

The following License Options are the types of licenses available for a given software product as further specified in this ALA. The applicable License Option for a license shall be as set forth in the Applicable Agreement or Product Order. Those products with only one License Option available shall be governed by such License Option whether or not stated in the Applicable Agreement or Product Order unless otherwise agreed in writing between Licensee and Licensor.

Named User License

Licensed Software provided under this License Option gives Licensee the right to install the Licensed Software on a single physical or Virtual Machine or server, or one or more Containers on such single physical or Virtual Machine or server, for use solely by the number of Named Users expressly authorized by Licensor in the Product Order for such license for whom Licensee has paid the applicable license fee. Each such Named User has the right to unlimited access to the Licensed Software on such single machine or server. A Named User License is required for each Named User that uses an installation of the Licensed Software. For example, if Named User A and Named User B both use the same two separate installations of the Licensed Software, a total of four Named User Licenses would be required for such use. Licensor reserves the right at any time to require Licensee to provide a list of the Named Users. Licensee may change a Named User provided that the change is made either (i) permanently, or (ii) temporarily to accommodate the use of the Licensed Software by a temporary worker while the Named User is absent, but in each case no more than once every 30 days. Licensee may not use the Licensed Software for user acceptance testing, system/load testing, production or deployment. For purposes of this ALA, a Named User License is considered to be a Development License.

Concurrent User License

Licensed Software provided under this License Option gives Licensee the right to install the Licensed Software on multiple physical or Virtual Machine host machines or servers, or one or more Containers on such physical or Virtual Machine host machines or servers, for use solely by the maximum number of Concurrent Users expressly authorized by Licensor in the Product Order for such license for whom Licensee has paid the applicable license fee. Licensee may not use the Licensed Software for user acceptance testing, system/load testing, production or deployment. For purposes of this ALA, a Concurrent User License is considered to be a Development License.

User License (Authorized Users)

Licensed Software provided under this License Option gives Licensee the right to install the Licensed Software on an unlimited number of physical or Virtual Machine servers or Containers for use by up to the total number of Authorized Users expressly authorized by Licensor in the Product Order for such license for whom Licensee has paid the applicable license fee. The Licensed Software may not be used or accessed by (i) individuals who are not Authorized Users; or (ii) any other software or hardware device that does not require an individual to use or access it, including, without limitation, Batch Processing. Licensor reserves the right at any time to require Licensee to provide a list of the Authorized Users. Licensee may change an Authorized User provided that the change is made either (i) permanently, or (ii) temporarily to accommodate the use of the Licensed Software by a temporary worker while the Authorized User is absent, but in no event more than once every 30 days. This License Option may be further subject to the Additional License Options set forth below in this ALA. For purposes of this ALA, a User License (Authorized Users) is considered to be a Deployment License.

Server License (Concurrent Users)

Licensed Software provided under this License Option gives Licensee the right to install the Licensed Software on a single physical machine or server or Virtual Machine, or one or more Containers on such physical single machine or server or Virtual Machine, for use by the maximum number of Concurrent Users expressly authorized by Licensor in the Product Order for such license for whom Licensee has paid the applicable license fee. **The Licensed Software may not be used or accessed by any other software or hardware device that does not require an individual to use or access it including, without limitation, Batch Processing.** A Server License (Concurrent Users) cannot be split into multiple licenses of lesser quantity (e.g., a license for 10 Concurrent Users cannot be split into two licenses of five Concurrent Users each) without Licensor's prior written consent and possible provision of different license keys. This License

Option may be further subject to the Additional License Options set forth below in this ALA. For purposes of this ALA, a Server License (Concurrent Users) is considered to be a Deployment License.

Server License (per core) or Server License (per IFL)

Licensed Software provided under this License Option gives Licensee the right to install the Licensed Software on a single physical machine or server ("Host Server"), or one or more Containers on the Host Server (provided such Containers are not running in a Soft Partition or Virtual Machine unless Licensee has purchased the Virtualization License Extension or otherwise been granted such rights separately from Micro Focus), and have the Licensed Software executed by up to the total number of (i) Cores in the case of a Server License (per core), or (iii) Integrated Facility for Linux processors ("IFLs") in the case of a Server License (per IFL), specified for the license in the applicable Product Order ("License Specification"). If the number of Cores is not specified for a CPU in the event a CPU is specified in the License Specification, such CPU shall be considered to be single-Core. A license covering all physical Cores or IFLs, as applicable, that are contained in and/or can be accessed by the Host Server ("Total Processors") is required with all applicable license fees paid, even if one or more of such Cores or IFLs are not accessing or running the Licensed Software for any reason, including, for example, installation of the Licensed Software in a Container that has fewer than the Total Processors allocated to it. For example, if 32 Cores are the Total Processors on the Host Server, but only 16 Cores are utilized to execute the Licensed Software, a 32-Core license is required notwithstanding the fact that 16 of the 32 Cores may not actually be accessing the Licensed Software. Each Core on a multi-core CPU requires a license covering each such Core. For example a Host Server with Total Processors consisting of a single quad-core CPU will require a 4-Core license.

Licensed Software provided under this License Option is for use by an unlimited number of Licensee's internal users, other software devices and hardware devices. **Server License (per core) and Server License (per IFL) are the only license types authorized for any Batch Processing.** Licensor reserves the right at any time to require Licensee to provide a specification of the Host Server. Licensee is required to purchase a license for the Total Processors on each Host Server that accesses or uses the Licensed Software either directly or indirectly through any other software program, regardless of how such access occurs, or if the Host Server, CPU or operator uses any hardware or software that reduces the apparent number of Cores, IFLs, or other processing devices that are using the Licensed Software, such as by using a terminal service. This License Option may be further subject to the Additional License Options set forth below in this ALA. For purposes of this ALA, a Server License (per core) and Server License (per IFL) are considered to be Deployment Licenses.

Server License (per vCPU)

Licensed Software provided under this License Option gives Licensee the right to install and use the Licensed Software in a virtualized environment that uses vCPUs as the metric for assigned processing resources, on any number of instances, provided that the collective amount of vCPUs accessible by those instances does not exceed the applicable license quantity. An instance may be within a single Virtual Machine or Container that is not necessarily tied to just one physical machine. Licensee may make one (1) failover or disaster recovery copy of the Virtual Machine or Container in which the Licensed Software is installed and instantiate such Virtual Machine or Container in the event and for as long as the primary installation is unavailable. For purposes of this ALA, a Server License (per vCPU) is considered to be Deployment License.

Additional License Options

The following additional License Options may also apply to one of the License Options above if set forth in the applicable Product Order notwithstanding anything else to the contrary in this ALA:

Test License

Licensed Software provided under this License Option gives Licensee a limited license to use the Licensed Software solely for Licensee's internal testing purposes on a single computer subject to this section and the restrictions applicable to the type of license specified in the applicable Product Order (e.g., Server License (per core)) as set forth elsewhere in this ALA. In the event of a conflict, the terms and conditions in this section shall prevail. At no time may Licensee use the Licensed Software for development, commercial, or production purposes, nor may Licensee reproduce or distribute the Licensed Software or any software

application programs created with it. Licensee's results of benchmark or other performance tests run on or using the Licensed Software may not be disclosed to any third party without Licensor's prior written consent.

Disaster Recovery License

Licensed Software provided under this License Option gives Licensee a limited license to use the Licensed Software solely on a Disaster Recovery System subject to this section and the restrictions applicable to the type of license specified in the applicable Product Order (e.g., Server License (per core)) as set forth elsewhere in this ALA. In the event of a conflict, the terms and conditions in this section shall prevail. A Disaster Recovery System is a single machine on which the Licensed Software is installed but is not instantiated, running, or otherwise in use except: (i) if a disaster arises and the single machine for which the Disaster Recovery System is configured to recover or replace (known as the "Primary Machine" in this clause) is unavailable, (ii) for the purposes of periodic disaster recovery testing, and/or (iii) for periodic system diagnostics or maintenance of the Disaster Recovery System itself. A Disaster Recovery License cannot be used: (a) concurrently with any other license for the Licensed Software or in any production, test, or development environments except while the Primary Machine is being recovered or replaced in a disaster recovery situation, or (b) for load-balancing, failover, testing (other than disaster recovery testing), clustering, or training purposes. Additional copies for use in production, test, and/or development environments, or for load-balancing, failover, clustering, or training purposes, must be purchased separately. The license quantity and metric of any Disaster Recovery License installed may not be less than the license quantity and metric of the standard license for the Licensed Software on the Primary Machine.

Failover License

Licensed Software provided under this License Option gives Licensee a limited license to use the Licensed Software solely on a Failover System subject to this section and the restrictions applicable to the type of license specified in the applicable Product Order (e.g., Server License (per core)) as set forth elsewhere in this ALA. A Failover System is a single machine on which the Licensed Software is installed and running, but only used to prepare for the recovery or replacement of the single machine for which the Failover System is configured to recover or replace (known as the "Primary Machine" in this clause), up to and including real-time data backups, mirroring, or any other activity to allow a synchronized switch-over from the Primary Machine. Licensee may not use the Failover License for any other purpose such as production, testing (other than failover testing), development, or training except during a disaster where the Primary Machine is unavailable. Licensee may not use the Licensed Software for load-balancing or clustering; a separate license for each of those use cases is required to be purchased by Licensee. The license quantity and metric of any Failover License installed may not be less than the license quantity and metric of the standard license for the Licensed Software on the Primary Machine.

Batch License

Where any Licensed Software is designated as a Batch License or "Batch-Only" in the applicable Product Order using such description or similar terminology, such Licensed Software may only be used to perform batch processing on a single computer subject to the restrictions applicable to the type of license specified in the applicable Product Order (e.g., Server License (per core)) as set forth elsewhere in this ALA. In the event of a conflict, the terms and conditions in this section shall prevail.

Virtualization License Extension

For a license for which a Virtualization License Extension has been purchased, the restrictions applicable to the type of license specified in the applicable Product Order (e.g., Server License (per core)) as set forth elsewhere in this ALA for such license shall apply subject to the terms in this clause. For as long as such license is current on support and maintenance with Licensor, such license can be installed, used, or accessed on, in, or from a Virtual Machine. *Such license may not be installed or used in a public cloud service.* With respect to licenses based on processing devices such as Cores, the Licensed Software may be installed and used on any number of instances provided that the collective amount of licensed processing devices accessible by those instances does not exceed the applicable license quantity. For example, for Core-based licenses, the license can be used on any number of instances provided that the collective amount of Cores accessible by those instances does not exceed the applicable license quantity.

An instance may be within a single Virtual Machine or Container that is not necessarily tied to just one physical machine. Licensee may make one (1) failover or disaster recovery copy of the Virtual Machine or Container in which the Licensed Software is installed and instantiate such Virtual Machine or Container in the event and for as long as the primary installation is unavailable.

Third Party Usage License Extension

For a license for which a Third Party Usage License Extension has been purchased, such license can be used by third party contractors of Licensee, notwithstanding any restriction to the contrary in the Applicable Agreement, provided that (1) each such third party is under a written obligation of confidentiality and restrictions of use which provide for protections and limitations of use of the license no less restrictive than the Applicable Agreement, (2) the license will only be used on behalf of Licensee, (3) such use is subject to the terms and conditions of the Applicable Agreement, and (4) Licensee remains liable for any breach by any such third party of the terms of the Applicable Agreement.

Off Continent License Extension

For a license for which an Off Continent License Extension has been purchased, such license can be used worldwide, except where prohibited by applicable law, notwithstanding any restriction to the contrary in the Applicable Agreement.

Software Specific Terms

Business Rule Manager

The following License Options apply: Named User License, Concurrent User License.

COBOL Server, COBOL 2010 Runtime

The following License Options apply: User License (Authorized Users), Server License (Concurrent Users), Server License (per core), Server License (per IFL), Server License (per vCPU) (COBOL Server only).

With any valid and properly granted license to this Licensed Software, Licensee may reproduce and distribute internally, in whole or in part, subject to the applicable License Option as set forth in this ALA and the applicable Product Order, any software application program created by Licensee using Visual COBOL (if separately licensed to Licensee by Licensor) ("Licensee Application Software"). In such event Licensee shall (a) include Licensor's copyright notice for the Licensed Software on any product label and as a part of any sign-on message for such Licensee Application Software product; and (b) indemnify, hold harmless and defend Licensor and its third-party suppliers from and against any claims or lawsuits, including attorneys' fees, legal fees and court costs that arise out of, or result from, the use or distribution of such Licensee Application Software.

Licensee can only reproduce and distribute Licensee Application Software to third parties after entering into a separate distribution agreement with Licensor. Usage and distribution of the Licensee Application Software arises both by the explicit distribution of any Licensee Application Software and by the implicit distribution and usage of any of the Licensee Application Software's functionality when linked into another software application program. Access and/or use of data, results and/or output obtained and/or generated, either directly or indirectly, and in any format whatsoever, through the use of any Licensed Software (such as when a user accesses an application that calls or uses output from another application running the Licensed Software, whether or not on the same machine or server) are deemed to be access and/or use of such Licensed Software. Licensor offers specific production licensing options for distribution to third parties which vary depending upon the license fees paid by Licensee and Licensee should contact its Licensor sales representative for more details.

COBOL Server for Stored Procedures

The following License Options apply: Server License (per core), Server License (per IFL).

The terms applicable to COBOL Server above apply to COBOL Server for Stored Procedures, except that COBOL Server for Stored Procedures may only be used to enable deployment of COBOL stored procedures on a single database server per license.

Database Connectors (all variants)

The following License Options apply: Server License (Concurrent Users), Server License (per core).

Enterprise Analyzer (all variants), Enterprise Analyzer Server (all variants), COBOL Analyzer, Enterprise View

The following License Options apply: Named User License, Concurrent User License, Server License (Enterprise View only).

Named User Licenses for this Licensed Software may be installed on multiple machines for use solely by the licensed Named Users. In the event the applicable Product Order specifies an amount of lines of code for a license, then the application portfolio size (aggregate number of lines of code in all repositories created) for such license is limited to the maximum lines of code (LOC) set forth for such license in the applicable Product Order. LOC includes all lines of code from all programs (including each version of the same program), all Java files, and all job control and database definition files in the repository. Copybook and other included code is counted separately for each program in which it appears. Each Server License of Enterprise View may only be installed on one server and may only access one repository at a time.

With respect to Enterprise Analyzer, the Licensed Software may require certain third party database software beyond the limited database software that may be included with such Licensed Software, and in such event Licensee must separately purchase or otherwise obtain such database software from the applicable third party.

Enterprise Developer (all variants)

The following License Options apply: Named User License, Concurrent User License.

Licensee shall not use the Licensed Software, nor reproduce or distribute any files supplied as part of the Licensed Software, in order to create a compiler, interpreter, runtime support product or any other product that is generally competitive with or a substitute for the Licensed Software or any other Licensor product.

This Licensed Software may include or generate limited licenses for Enterprise Server or other application deployment product. If such licenses are included or generated, these limited licenses are provided solely for use by the licensed user(s) of the Licensed Software in order to unit test an application developed with the Licensed Software on the same machine or server used to develop the application and may not be used for system testing, production or deployment. Any use of this limited license outside of the development and unit testing of such applications is not permitted.

Enterprise Developer for Stored Procedure Debug. This Licensed Software may only be used to debug stored procedures on a single database server per license.

Enterprise Developer Connect. Licensee may install certain Enterprise Developer components and models in another Eclipse or IBM Developer for z environment in accordance with the applicable Documentation solely for use with a valid license for Enterprise Developer Connect.

Enterprise Developer Models. This Licensed Software may only be used with a valid license for Enterprise Developer and shall be governed by the same terms and conditions as such license. No support or maintenance is provided or available for this Licensed Software.

Enterprise Server (all variants)

The following License Options apply: User License (Authorized Users), Server License (Concurrent Users), Server License (per core), Server License (per IFL), Server License (per vCPU).

With any valid and properly granted license to this Licensed Software, Licensee may reproduce and distribute internally, in whole or in part, subject to the applicable License Option as set forth in this ALA and the applicable Product Order, any software application program created by Licensee using Micro Focus Enterprise Developer (if separately licensed to Licensee by Licensor) ("Licensee Application Software"). In such event Licensee shall (a) include Licensor's copyright notice for the Licensed Software on any product label and as a part of any sign-on message for such Licensee Application Software product; and (b) indemnify, hold harmless and defend Licensor and its third-party suppliers from and against any claims or lawsuits, including attorneys' fees, legal fees and court costs that arise out of, or result from, the use or distribution of such Licensee Application Software.

Licensee can only reproduce and distribute Licensee Application Software to third parties after entering into a separate distribution agreement with Licensor. Usage and distribution of the Licensee Application Software arises both by the explicit distribution of any Licensee Application Software and by the implicit distribution and usage of any of the Licensee Application Software's functionality when linked into another software application program. Access and/or use of data, results and/or output obtained and/or generated, either directly or indirectly, and in any format whatsoever, through the use of any Licensed Software (such as when a user accesses an application that calls or uses output from another application running the Licensed Software, whether or not on the same machine or server) are deemed to be access and/or use of such Licensed Software. Licensor offers specific production licensing options for distribution to third parties which vary depending upon the license fees paid by Licensee and Licensee should contact its Licensor sales representative for more details.

Enterprise Server for Stored Procedures

The following License Options apply: Server License (per core), Server License (per IFL), Server License (per vCPU).

The terms applicable to Enterprise Server above apply to Enterprise Server for Stored Procedures, except that Enterprise Server for Stored Procedures may only be used to enable deployment of stored procedures (either COBOL, PL/I, or both depending on what is specified for such license in the applicable Product Order) on a single database server per license.

Enterprise Test Server, Enterprise Test Server Premium, COBOL 2010 Runtime Test Server

The following License Option applies: Server License (per core), Server License (per vCPU) (Enterprise Test Server only).

This Licensed Software may be used only for Licensee's internal testing purposes. At no time may Licensee use the Licensed Software for any other purpose including production.

Relativity Designer Suite for MF COBOL

The following License Options apply: User License (Authorized Users).

Relativity for Windows Workstations for MF COBOL

The following License Options apply: User License (Authorized Users), Server License (Concurrent Users).

Relativity Server for MF COBOL

The following License Options apply: Server License (per core), Server License (Concurrent Users).

Visual COBOL (all variants)

The following License Option applies: Named User License.

Licensee shall not use the Licensed Software, nor reproduce or distribute any files supplied as part of the Licensed Software, in order to create a compiler, interpreter, runtime support product or any other product that is generally competitive with or a substitute for the Licensed Software or any other Licensor product.

This Licensed Software may include or generate limited licenses for COBOL Server or other application deployment product. If such licenses are included or generated, these limited licenses are provided solely for use by the licensed user(s) of the Licensed Software in order to unit test an application developed with the Licensed Software on the same machine or server used to develop the application and may not be used for system testing, production or deployment, and may only be used if the Licensed Software is current on support and maintenance. Any use of this limited license outside of the development and unit testing of such applications is not permitted.

Visual COBOL Database File Handler

The following License Options apply: Server License (Concurrent Users), Server License (per core).

SUITES

Suite	Offering includes
Visual COBOL for Eclipse Distributed Edition	<ul style="list-style-type: none"> • Visual COBOL Development Hub • Visual COBOL for Eclipse
Visual COBOL for ISVs	<ul style="list-style-type: none"> • Visual COBOL Development Hub • Visual COBOL for Eclipse • Visual COBOL for Visual Studio
Visual COBOL Studio Distributed Edition for ISVs	<ul style="list-style-type: none"> • Visual COBOL Development Hub • Visual COBOL for Eclipse • Visual COBOL for Visual Studio

Additional License Terms

The following additional license terms shall apply to any software governed by this ALA:

Additional License Restrictions

Licensee shall not:

1. Permit use or access by any third party to any output directly or indirectly created by the Licensed Software without first paying Licensor any applicable additional fees required by Licensor and entering into a separate distribution license agreement with Licensor. For the avoidance of doubt, third party(ies) include without limitation contractors, outsourcers, Licensee's customers and the public.
2. Transfer, ship or use the Licensed Software outside the continent in which it was originally licensed to Licensee without first purchasing an Off-Continent License Extension or otherwise paying Licensor any applicable additional fees required by Licensor. For purposes of the foregoing, "continent" shall mean North America, South America, Europe, Africa, Asia, Australia, or Antarctica.
3. Install, use, or access the Licensed Software on, in, or from a Hard Partition, Soft Partition, or Virtual Machine, except with respect to Server License (per vCPU) and the Virtualization License Extension, and except where Licensee is otherwise expressly authorized in each instance by Licensor in the applicable Product Order or otherwise in writing separate from this ALA (the "Authorization") which in each event will be subject to Licensee's payment to Licensor of additional applicable fees required by Licensor and, if requested by Licensor, Licensee's written confirmation to Licensor of Licensee's planned installation environment signed by a director or officer of Licensee (accompanied by all information as Licensor reasonably requires to verify). Except as otherwise specified in the Authorization, where authorized in accordance with the foregoing and to the extent applicable to the subject license: (i) such Hard Partition, Soft Partition, or Virtual Machine, may only be tied to one physical machine and shall be considered the entire machine for licensing purposes; and (ii) Licensee shall provide to Licensor, at such times as Licensor may reasonably request, confirmation that Licensee's computer systems comply with the applicable Authorization and the requirements of this ALA. Such confirmation shall be signed by a director or officer of Licensee and shall be accompanied by all information as Licensor reasonably requires to verify that Licensee is utilizing the Licensed Software in compliance with the terms and conditions of this ALA and the Applicable Agreement, including without limitation any relevant hypervisor logs. The foregoing is in addition to Licensor's audit rights under the Applicable Agreement.
4. Utilize any software application and/or program created, in whole or in part, with any Development License without first purchasing a valid and properly granted Deployment License.
5. Install, access or use the Licensed Software on a Platform other than the Platform for which the Licensed Software was originally licensed to Licensee. Additional licenses for use of the Licensed Software on a Platform other than the one set forth in the applicable Product Order may be available upon payment of an additional license fee.

All-or-None Support and Maintenance

Where Licensee purchases support and/or maintenance for the Licensed Software, Licensee hereby agrees that it shall purchase such support and/or maintenance services for all of Licensee's licensed units of such Licensed Software including all related Development Licenses and Deployment Licenses.

Academic Editions/Users

Where Licensed Software is licensed under the Micro Focus COBOL Academic Program:

- (i) Licensee shall not use the Licensed Software for any purpose other than for non-commercial education or academic research activities;
- (ii) Licensee shall not copy the Licensed Software, nor shall Licensee distribute, transfer or assign the Licensed Software without specific permission from Licensor; and
- (iii) If Licensee is the academic institution using the Licensed Software for teaching purposes, Licensee will manage all student licenses through a license server provided by Licensor in accordance with guidelines provided by Licensor. Licensee agrees to provide reports describing the number of licenses Licensee has issued in each period upon Licensor's request.

Personal Editions

With respect to any Personal Edition versions of the Licensed Software:

- (i) Licensee shall not use the Licensed Software for any purpose other than for personal educational and non-commercial activities. The Licensed Software may not be used for training or teaching purposes.
- (ii) Licensee shall not: (a) use the Licensed Software to compile source code for applications of more than two thousand, two hundred (2200) lines of procedural code (exclusive of lines containing comments and/or blanks) in any one (1) application or deploy or transfer such application on or to any other machine or third party; or (b) with respect to Enterprise Developer, use the Licensed Software for any purpose other than to input, edit and syntax check code for personal educational use.
- (iii) Licensee shall not copy the Licensed Software, nor shall Licensee distribute, transfer or assign the Licensed Software without specific permission from Licensor.

Subscription Licenses

Where any Licensed Software is designated as a Subscription or Conversion Subscription license (rather than a "Sub" license) or is an existing Upfront Sub or No Upfront Sub license, then the following additional terms apply:

1. Term. Each license shall automatically terminate at the end of term for such license as stated in the applicable Product Order. The license must be de-installed upon termination unless Licensee purchases an additional subscription term for such license from Licensor.
2. Payment for Existing No Upfront Sub Licenses.. Fees for No Upfront Sub licenses will be billed monthly in advance or per the frequency noted in the applicable Product Order if different. Licensee irrevocably commits to pay such fees to Licensor for the entirety of the subscription term purchased.
3. Applicable License Extensions. The Off Continent License Extension and Third Party Usage License Extension are automatically included. The Licensed Software may be installed or used in a public cloud service. Server License (per vCPU) is the only authorized Deployment License type for Subscription and Conversion Subscription licenses. If Licensee has an Applicable Agreement that permits use of the Licensed Software to provide services to end user customers, Licensee may only use a license for the Licensed Software subject to this ALA to provide services to one end user customer.
4. Support and Maintenance. Standard support and maintenance are included during the term of the license.
5. Platform Transfer. Licensee may freely transfer a license during the term of such license from one x86 platform to another for which a version of the Licensed Software is generally available as of the commencement of such license.
6. Reporting and True-Up. Licensee shall provide to Licensor on a quarterly basis, or at such other times as Licensor may reasonably request, with a written report detailing the amount of usage of the Licensed Software in a format agreed in advance by the parties. At a minimum this report shall include, as applicable, the number of (i) instances of the Licensed Software, (ii) users of products licensed by

named user, (iii) cores, vCPUs or other processing devices accessed by instances of products licensed by server or processing device, and (iv) hours of usage supported by hardware/infrastructure consumption reporting. Such report shall be signed by a director or officer of Licensee and shall be accompanied by all information as Licensor reasonably requires to verify that Licensee is utilizing the Licensed Software in compliance with the terms and conditions of the Applicable Agreement, including without limitation any relevant hypervisor logs. The foregoing is in addition to Micro Focus' audit rights under the Applicable Agreement. If the above report indicates that Licensee has exceeded the amount of instances, users, cores, vCPUs, or IFLs, then Licensee agrees to immediately pay Licensor for such additional usage at Licensor's On-Demand Subscription per-day price specified in the applicable Product Order, or if no such pricing is specified, then Licensor's then-current On-Demand Subscription per-day list pricing.

Host Connectivity Products

The Licensed Software may come with one or more software products from Licensor's Host Connectivity portfolio, and in such event, Licensee may only use such Host Connectivity software with the Licensed Software on the same machine as the Licensed Software subject to the terms and conditions applicable to the Licensed Software. For example, if Rumba or Host Access for Cloud were included with Enterprise Developer, the licensed Named User for Enterprise Developer could only use such products to access applications being developed using Enterprise Developer on the same machine as the licensed installation of Enterprise Developer and not applications on any other machine. Notwithstanding the foregoing, in the event Host Access for Cloud is included with Enterprise Server or any other product licensed by a Deployment License, such software may only be used by up to 25 Concurrent Users per Enterprise Server licensed deployment solely to perform administrative functions to support and maintain the applications utilizing Enterprise Server to run.

software.microfocus.com/legal/software-licensing - Latest version of software licensing documents

© Copyright 2022 Micro Focus.

June 3, 2022

Creating a Shared Folder for Workspaces

The folder in which you plan to create workspaces must be shared with team members. It is typically more convenient to share the folder before you create workspaces, but you can do it afterward if necessary, then refresh the workspace path as described in *Getting Started* in the CA documentation set.

Follow the instructions below to create a shared folder for workspaces. You must be an administrator to create a shared folder.

1. On the CA server, create a folder for CA workspaces.
2. Select the folder and choose **Sharing and Security** from the right-click menu. The Properties dialog for the folder opens. In the Sharing tab for the Properties dialog, select **Share this folder on the network** (or, on a machine in a Windows domain, **Share this folder**). Enter the share name in the **Share name** field and click **Apply**.



Note: Do not embed spaces in the share name. Doing so may cause other users to be unable to access the folder.

3. Click **Permissions**. The Permissions for Folder dialog opens. Specify the appropriate permissions for users sharing the folder and click **OK**.

Upgrading Workspaces

When you modify your CA configuration, you must upgrade every workspace created with the previous configuration. Only the master user can upgrade a workspace.

1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**. The CA Administration window opens.

2. In the Administration window, choose **Administer > Upgrade Workspace**. The Upgrade workspace dialog opens.
3. Choose the workspace you want to upgrade, then click **Open**. A Workspace Upgrade window opens.
4. Click **Start** to begin the upgrade. In the Workspace Upgrade window you see the upgrade process (The parts that are being upgraded are highlighted and the checkbox next to each upgraded part is checked when the upgrade is complete.) You can Pause the upgrade and Resume it later.



Note: Upgrades that have stopped due to an error can be reset from the **File** menu.

5. (optional) Click **Workspace Upgrade > File > Export DDL Script** to export the DDL script to perform the upgrade and give it to a DBA.

Troubleshooting the Installation

Follow the instructions in this section to troubleshoot an CA installation.

Troubleshooting Workspace Access

The folder in which you plan to create workspaces must be shared with team members. If users are unable to access workspaces, it may be because:

- You have not shared the folder for workspaces. Share the folder, then refresh the workspace path as described in *Getting Started* in the CA documentation set.
- You shared the folder for workspaces after creating a workspace. Refresh the workspace path as described in *Getting Started* in the CA documentation set.
- You embedded spaces in the value of the **Share name** field for the folder. Remove the spaces, then refresh the workspace path as described in *Getting Started* in the CA documentation set.

Getting Started

Introducing COBOL Analyzer

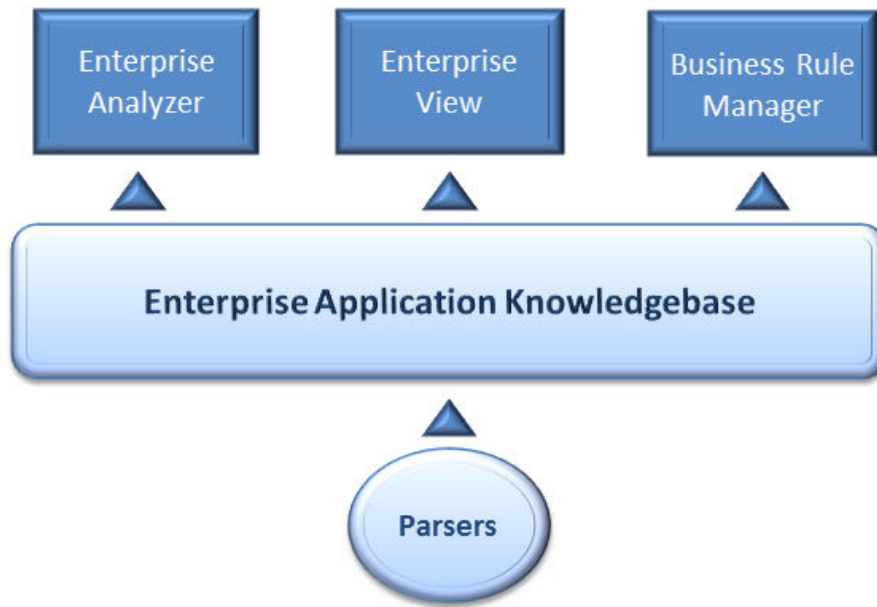
Micro Focus COBOL Analyzer (CA) offers centralized storage and analysis of enterprise application portfolios:

- Language-specific parsers generate a common repository of application objects, in the industrial-strength DBMS of your choice.
- Sophisticated analysis tools give you convenient interactive access to the repository, in a multiuser environment that facilitates exchange of information and insight across the enterprise.

The result is a single easy-to-use interface in which you analyze assets written in modern languages like Java and C# alongside assets written in legacy languages like COBOL and PL/I. And in which you perform the analysis with industry-leading tools that offer deep, detailed, and comprehensive insight into complex modern distributed applications.

COBOL Analyzer Products

COBOL Analyzer is a suite of PC-based software products for storing, analyzing, and re-architecting enterprise 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. Each product is separately licensed.



COBOL Analyzer

COBOL Analyzer is a robust set of non-invasive interactive tools that work with source code, system files, DDL, screen maps, and more. Use it for analyzing and documenting legacy systems at both the application and program levels.

Use COBOL Analyzer at the application level to:

- Create diagrams of applications.
- Perform change analysis across applications.
- Estimate application complexity and effort.

Use COBOL Analyzer at the program level to:

- Perform program analysis in stages.
- Perform impact analysis on a program.
- Model and search the repository.

Enterprise View

Enterprise View (EV) is a Web server-based tool that offers IT executives "at-a-glance" application portfolio management (APM). EV charts and tables provide powerful graphical insight into APM trends and current status, on dashboards that make it easy to drill down from the "big picture" captured at the highest level to the lower-level details that support it.

With EV you can:

- Create, distribute, and gather responses to surveys about your organization's application portfolio and business processes.
- Mine application complexity metrics from the CA repository.
- Track survey and repository metrics in EV's rich variety of interactive charts, displayed on dashboards structured to show the rollup of data from lower to higher levels.

For users who need to drill down even deeper into an application portfolio, EV lets you search, filter, and analyze CA repository data. You can:

- Query the repository for information about application objects.
- Search, browse, and edit business rules and triggers.

Business Rule Manager

Business rule mining encapsulates your application’s business logic, making the application easier to understand, document, and test. It lowers the risk and cost of maintenance and enhancement, and ensures that all business logic is implemented in modernization efforts.

A *business rule* is a named container for program code that performs a discrete task in a business process. It identifies and documents the code segment that performs this task. A business rule named Calculate Date Difference, for example, might consist of this segment:

```
COMPUTE WS-DATE-VARIANCE = WS-C-CARD-DATE-CCYYMM - WS-TODAYS-DATE-CCYYMM.
```

Business Rule Manager (BRM) offers a powerful set of tools for aut documenting, business rules. You can batch edit rule attributes; create custom attributes suited to your particular needs; match input/output data elements with the business names you’ve assigned to them in your project glossary; and much more.

How COBOL Analyzer Models Applications

COBOL Analyzer is a model-based technology that generates information about legacy applications at two levels:

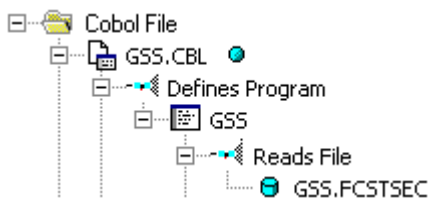
- Application-level information describes the relationships between the *objects* that comprise an application: its programs, copybooks, JCLs, BMS files, and so forth.
- Object-level information describes the relationships between the abstract syntactical *constructs* that comprise an object: its sections, paragraphs, statements, conditions, variables, and so forth.

At both levels, the models are conceived as *Entity Relationship Diagrams (ERDs)*. Each object or construct the parser generates is modeled as an entity in one of the diagrams. The models are represented in tree form to users.

Understanding Application-Level Information

The *object model* for an application defines the relationships between the objects that comprise the application. These can be physical objects, like program source files or JCLs, or logical objects that identify abstract program elements: entry points, data stores, screens, jobs, and the like.

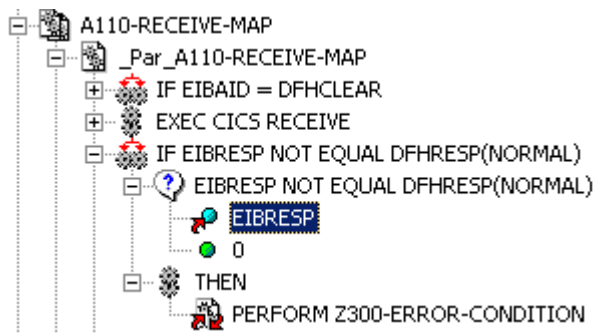
The *relationships* between objects describe the ways in which they interact. In the figure, the source file GSS.CBL *defines* the GSS program. The program, in turn, *reads* the data file GSS.FCSTSEC.



Understanding Object-Level Information

The *construct model* for an object defines its syntax. It shows in abstract form how the syntactical constructs that comprise the object (its sections, paragraphs, statements, conditions, variables, and so forth) are related. A variable, for example, can be related in the construct model to its declaration, a dataport (if it is used in an I/O statement), or a condition (if the condition uses an expression of which the variable forms a part).

The figure shows a portion of the construct model for the GSS program. The model shows that the program executes a PERFORM statement if the value of the variable EIBRESP satisfies the condition EIBRESP NOT EQUAL DFHRESP(NORMAL).



If you are interested in investigating other uses of EIBRESP in the program, you can navigate to the declaration of the variable in the construct model, and from the declaration to each instance of the variable's use in the program.

Workspaces and Projects

COBOL Analyzer uses the familiar notions of workspace and project to help users organize application source files conveniently. A *workspace* is a named container for one or more applications. Every workspace has a corresponding database *repository* of model objects.

You can divide a workspace up into *projects* that represent different applications or different portions of an application. You might have a project for the batch portion of an application and another project for the online portion, for example. You can also use a project to collect items for discrete tasks: all the source files affected by a change request, for example.

When you set up a workspace in COBOL Analyzer, the system creates a default project with the same name as the workspace. You can create new projects and move or copy entities between projects as needed.

A workspace can contain objects coded in different programming languages. You can create multiple workspaces on the same machine.

Single-User versus Multiuser Environments

COBOL Analyzer can be deployed in a single-user or multiuser environment:

- In a single-user environment, the workspace repository resides on the local machine, and can be accessed by the owner of the machine only. Limited facilities exist for sharing work with other users.
- In a multiuser environment, the workspace repository resides on a database server, and can be accessed by any user with appropriate database privileges.

Most installations deploy COBOL Analyzer in a multiuser environment. Single-user mode typically is reserved for special needs.

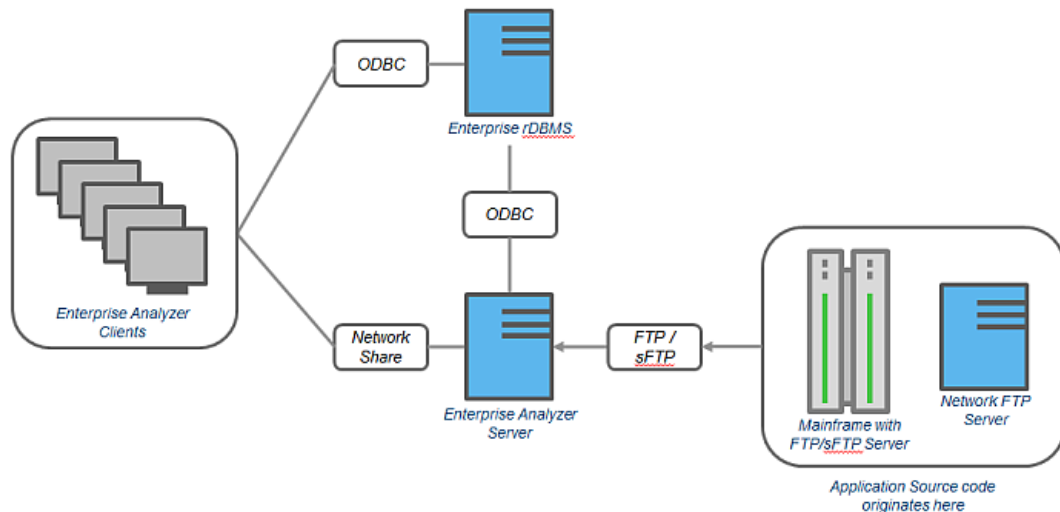
The guidelines that follow for multiuser environments apply equally to single-user environments. For installation instructions, see the installation guide for your product.

Multiuser Environment Basics

A multiuser COBOL Analyzer environment gives members of a development team common access to a workspace repository on a database server. Workspace setup is the responsibility of a master user, leaving team members free to focus on their tasks. Common access makes it easy to share insights across the team, and ensures that team members view the same source and work with the same option settings.

Deployment Scenario

The figure below shows a standard deployment scenario for the COBOL Analyzer multiuser environment. The role each machine performs and its key relationships are described below. Keep in mind that the view is logical. Machine roles can be combined or separated as required.



Mainframe and Network Server

The mainframe hosts the application to be modeled in COBOL Analyzer. Application source files are downloaded from the mainframe (and, if necessary, a network server) to the COBOL Analyzer server via FTP or SFTP.

COBOL Analyzer Server

The COBOL Analyzer server hosts workspaces, workspace support files (including the copies of application source files the product creates at workspace loading), and workspace output. This server leverages multiple processes to enhance parsing performance in online or batch mode.

Parsed data is sent via ODBC to the database server. Some analysis output is stored on the CA server as well.

Database Server

The database server hosts the database for one or more multiuser repositories. This server provides centralized, network-accessible storage for parsed data and analysis output.

The CA server, Enterprise View Web Server, and CA clients access the database server via ODBC. For guidance on database setup, see the installation guide for your product.

COBOL Analyzer Client

CA clients host the *link files* that let team members connect to workspaces on the CA server. These clients access repository data stored in the database server via ODBC.

Enterprise View Web Server

The Enterprise View web server hosts services used by intranet and internet clients to display charts and to access web-based view of repositories.

Enterprise View Web Client

Enterprise View web clients provide multiuser access to charts and workspace repository data via the Enterprise View web server.

CA Privileges

Privileges in a multiuser environment work on two assumptions:

- Team members typically do not need to modify files until they have completed analysis and are ready to test a proposed change.
- The team lead is best situated to make the change and reverify the modified file.

For these reasons, the CA multiuser environment has no need for file checkout and locking of the kind seen in conventional multiuser environments.

Understanding Master Users

The user who creates a workspace on a COBOL Analyzer server machine is referred to as its *owner*. Initially, only the owner has *master user* privileges for the workspace.

The master user can delete the workspace, upgrade the workspace configuration, register and verify source files, modify source, autoresolve decisions, perform restricted Code Search searches, create, assign, and delete tags, and so forth. The master user can also designate new master users for the workspace.

Ordinary users *connect* to the workspace from COBOL Analyzer client machines. These users can perform any task not restricted to the master user: analyze programs, mine business rules, extract components, and so forth.

If you do not have master user privileges for a workspace, you will not see menu choices for privileged methods. The same team member can be a master user for one workspace and an ordinary user for another.

Understanding Workspace Security Policies

When you create a workspace, you can choose from two workspace security policies:

- The *Simple Security Policy* recognizes two classes of users, master users and ordinary users.
- The *Three-Group Security Policy* recognizes a third class of users, *subject matter experts (SMEs)*, with special privileges to create, assign, and delete tags.

In each policy, the master user can create new master users.


Using the Optional File Server

When you create a workspace, you can restrict Windows access to workspace source files to master users, while using the optional File Server to make the source files available in CA only to ordinary users. As long as you set up workspace folder privileges correctly, ordinary users can view and analyze sources in the product, but cannot access them otherwise. For file server installation instructions, see the installation guide for your product.

Public and Private Visibility


In a multiuser environment, the user who creates a workspace resource (a project or a Code Search list, for example) is referred to as its *owner*. Only the owner can *share* the resource with other users.

A shared, or *public*, resource is visible to other members of your team. A *private* resource is not. If a public resource is *protected*, team members can view but not edit or delete the resource. A Code Search list, for example, is always protected.


You can turn sharing on and off in the tool you use to create the resource. Look for a symbol like this one for a shared project  to indicate that the resource is shared.

Protecting Projects

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

 **Note:** Only a master user or the owner of a shared unprotected project can delete it.

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 **Project > Toggle Protection**. Choose **Project > Toggle Protection** again to turn it off. Look for a symbol like this one  to indicate that a project is protected.

Dropping Indexes

If you are a master user with responsibility for verifying source files in a multiuser environment, you will almost always want to *drop database indexes* to improve verification performance. You will be prompted to drop indexes when you verify the application, or you can drop indexes at a time of your own choosing by selecting **Prepare > Drop Repository Indexes**.


When you drop indexes, you are prompted to restore the indexes when you analyze the files. If you want to restore indexes at a time of your own choosing, choose **Prepare > Restore Repository Indexes**.

COBOL Analyzer Basics

This section describes elementary CA tasks: how to create a workspace and project, register and verify source files, use the browser and other CA windows, set options, and export system diagrams and reports.

Creating a Workspace in SQL Server


You create a workspace on the COBOL Analyzer server machine. Other users *connect* to the workspace from COBOL Analyzer client machines. The workspace repository resides on a database server, and can be accessed by any user with database privileges to the repository.

 **Note:** Do not modify the name of the workspace after you create it. Doing so will cause unpredictable results.

The folder in which you plan to create workspaces must be shared with team members. It's usually more convenient to share the folder before you create workspaces, but you can do it afterward if necessary, then refresh the workspace path.


It is the responsibility of the workspace creator to designate users for security policies. Perform this task as soon as you have finished creating the workspace, then periodically edit the list of users as required.

You can use Microsoft SQL Server Express to create a workspace on your local machine. COBOL Analyzer creates a database for the workspace "on the fly," with no intervention on your part. If you do use SQL Server Express, keep in mind that the Windows user who creates the workspace must have been configured with appropriate permissions in SQL Server Express. The user who installed SQL Server Express will always have the appropriate permissions. See the SQL Server Express documentation for configuration instructions.

 **Note:** For Windows XP installations using the Italian locale, you must set the Time format in the Control Panel Regional Language Options to "HH:mm:ss" before attempting to verify an CA workspace with a SQL Server repository. Click **Settings > Control Panel > Regional and Language Options > Customize > Time** and choose "HH:mm:ss" from the **Time format** drop-down.


1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**. The Administration window opens.
2. Choose **Administer > New Workspace**. The Create new workspace dialog opens.
3. In the Create new workspace dialog, double-click the **New Workspace** icon for the RDBMS in use at your site or select it and click **OK**. The New workspace dialog opens.

4. In the New workspace dialog, choose the location for the workspace in the **Save in** drop-down. In the **File name** field, enter the name of the workspace. Choose a name that describes the legacy application as closely as possible. Click **Save**. The Define Connection Parameters dialog opens.


 **Note:** This dialog does not open for Microsoft SQL Server Express. The product starts creating the workspace immediately.

5. In the Define Connection Parameters dialog, click the Connection tab. Define the connection:

- In the **Server** field, enter the server name. The server name must be of the form `<machine>\<SQL server>`.

 **Note:** DNS issues can result in an SQL Server does not exist or access denied error. If you get this error, try replacing the machine name with the IP address.

- In the **Database Name** field, enter the database name your DBA created for the workspace repository.
- In the **Integrated Security** drop-down, select the integrated security provider interface in use at your site. Choose:
 - **Windows Authentication** if users connect to the workspace repository through a Windows user account.
 - **Server Authentication** if users connect to the workspace repository through an SQL Server login account.
- If you chose **Server Authentication**, enter the database login name in the **User Name** field and the login password in the **Password** field.


 **Note:** The database user name and login name are typically, but not always, the same. If you have trouble connecting, it may be because you are specifying the database user name rather than the login name.

6. Click the Security tab. In the Policies list box, select:


- **Simple Security Policy** if you want to enforce a workspace security policy that recognizes two classes of users, master users and ordinary users.
- **Three-Group Security Policy** if you want to enforce a workspace security policy that recognizes a third class of users, *subject matter experts (SMEs)*, with special privileges to create, assign, and delete tags.

7. In the Files access group box on the Security tab, select:

- **Direct sources access via Windows share** if your site does not restrict Windows access to workspace source files to the master user.
- **Secure sources access via CA file server** if your site restricts Windows access to workspace source files to the master user, while using the optional file server to make the source files available in the product to ordinary users. Make sure the values for the server name and port number fields are correct.

 **Note:** For File Server installation and setup instructions, see the installation guide for your product.


8. Click **OK**.

 **Note:** If you created the workspace in a non-shared folder, you are warned to that effect and prompted to share the folder. After you share the folder, make sure to refresh the workspace path.

You are notified that the workspace was created successfully. COBOL Analyzer creates a workspace file (.rwp) and a folder with the same name in the specified location. The workspace folder contains support files and subfolders for CA output. A project with the same name as the workspace is displayed in the Repository Browser.

Creating a Workspace Using the Workspace Build Wizard

The Workspace Build Wizard helps you create an COBOL Analyzer workspace. Stepping through it, you add your source files, associate files with source types and analyze the files so that they are ready to be used in COBOL Analyzer.

 **Note:** To use the wizard, you must have SQL Server Express installed on the machine.

Designating Users for Security Policies



When you create a workspace, you choose from two workspace security policies:


- The *Simple Security Policy* recognizes two classes of users, master users and ordinary users.
- The *Three-Group Security Policy* recognizes a third class of users, *subject matter experts (SMEs)*, with special privileges to create, assign, and delete tags.

In each policy, the master user can create new master users.

Designating Users in a Simple Security Policy

A Simple Security Policy recognizes two classes of users, master users and ordinary users. Initially, only the workspace creator has master user privileges. The master user can designate new master users as described in this section.


1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**. The Administration window opens.
2. Choose **Administer > Edit Users**. The Edit Users dialog opens.
3. In the Edit Users dialog, select the workspace (.rwp file) for which you want to edit users and click **Open**. The Security dialog opens.
4. The Security dialog displays a list of users who have connected to the workspace in the Known Users list box and a list of master users in the Master Users list box. Select a known user and click the  button to move the known user into the list of master users. Select a master user and click the  button to move the master user into the list of known users.


 **Note:** Click **Refresh** at any time to update the lists.


5. To create a new master user, click **(New User...)** in the Master User list box. A dialog opens prompting you to enter the name of the new master user. Enter the name and click **OK**.
6. When you are satisfied with your entries, click **Close**.

Designating Users in a Three-Group Security Policy

A Three-Group Security Policy recognizes a third class of workspace users in addition to master users and ordinary users: *subject matter experts (SMEs)*, with special privileges to create, assign, and delete tags. Initially, only the workspace creator has master user privileges. The master user can designate SMEs and new master users as described in this section.

1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**. The Administration window opens.
2. Choose **Administer > Edit Users**. The Edit Users dialog opens.
3. In the Edit Users dialog, select the workspace (.rwp file) for which you want to edit users and click **Open**. The Security dialog opens.
4. The Subject Matter Experts and Master Users tabs of the Security dialog display a list of users who have connected to the workspace in the Known Users list box and a list of subject matter experts and master users, respectively, in the opposite list box. Select a known user and click the  button to move

the known user into the list of subject matter experts or master users. Select a master user and click the  button to move the subject matter expert or master user into the list of known users.

 **Note:** Click **Refresh** at any time to update the lists.


5. To create a new subject matter expert or master user, click **(New User...)** in the Subject Matter Experts or Master Users list box. A dialog opens prompting you to enter the name of the new subject matter expert or master user. Enter the name and click **OK**.
6. When you are satisfied with your entries, click **Close**.

Refreshing the Workspace Path

The folder in which you plan to create workspaces must be shared with team members. It's usually more convenient to share the folder before you create workspaces, but you can do it afterward if necessary, then refresh the workspace path as described below. You can also use the refresh path feature when you move a workspace.

 **Note:** For instructions on how to share a workspace folder, see the installation guide for your product.

1. Click **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**. The Administration window opens.
2. Click **Administer > Refresh Workspace Path**. The Select workspace dialog opens.
3. Select the workspace (.rwp file) you want to refresh and click **Open**. The Workspace directory network path dialog opens. The Workspace directory network path dialog opens.
4. In the Workspace directory network path dialog, enter the path of the workspace in the text field. The path must be of the form `\machine_name\share_name\workspace_folder_name`, where `machine_name` is the name of the machine where the shared folder resides, `share_name` is the value of the Share name field on the Sharing tab of the Properties dialog for the shared folder, and `workspace_folder_name` is the name of the folder for the workspace.

 **Note:** The shared folder is the folder that contains the .rwp file for the workspace, not the workspace folder. The workspace folder (containing support files and subfolders for CA output) resides on the same level as the .rwp file.

5. Click **OK**.

Connecting to a Workspace in SQL Server

Once a workspace has been created on the COBOL Analyzer server machine, other users can *connect* to the workspace from CA clients.



If you use the local MS SQL connection, you need db_owner permissions, because this type of connection creates a database. But if you use the left option, MS SQL via OLE DB, you need a database already created by an administrator. For this database you may have less permissions and the roles can be: db_writer and db_reader.

To grant permissions, run these queries:

```
USE mydatabase;  
GRANT ALTER TO myuser;  
GRANT REFERENCES TO myuser;  
GO
```



ALTER permissions are required when upgrading, or using Configuration Manager in the Administration tool. ALTER is also needed by the Batch Refresh Process and Code Search.

When you connect to a workspace, you create a *link file* with connection information that points to the workspace. Because the connection information is already stored in the file, you don't have to enter it again when you reopen the workspace connection. Like a workspace file, a link file has a .rwp extension.

1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer**. The Open Existing workspace dialog opens above the COBOL Analyzer main window. Click **Cancel** to dismiss the dialog.
2. Choose **File > Build New Connection**. The Define Server Workspace Connection dialog opens.
3. In the Define Server Workspace Connection dialog, click the Connection tab. Define the connection:
 - In the **Server** field, enter the server name. The server name must be of the form `<machine>\<SQL server>`.
 -  **Note:** DNS issues can result in an SQL Server does not exist or access denied error. If you get this error, try replacing the machine name with the IP address.
 - In the **Database Name** field, enter the database name your DBA created for the workspace repository.
 - In the **Integrated Security** drop-down, select the integrated security provider interface in use at your site. Choose:
 - **Windows Authentication** if you connect to the workspace repository through a Windows user account.
 - **Server Authentication** if you connect to the workspace repository through an SQL Server login account.
 - If you chose **Server Authentication**, enter the database login name in the **User Name** field and the login password in the **Password** field.
 -  **Note:** The database user name and login name are typically, but not always, the same. If you have trouble connecting, it may be because you are specifying the database user name rather than the login name.
4. When you are satisfied with your entries in the Define Server Workspace Connection dialog, click the **Browse** button for the **Save Link As (.rwp)** field. The Save Server Workspace Connectivity dialog opens.
5. In the Save Server Workspace Connectivity dialog, choose the location for the workspace link in the **Save in** drop-down. In the **File name** field, enter the name of the link. Click **Save**. The linked workspace opens in the COBOL Analyzer main window. CA creates a link file (.rwp) in the specified location.

Opening a Workspace


The procedure for opening a workspace is the same whether you created the workspace or simply connected to it. You can also open a workspace by double-clicking it in the file system. You don't have to close a workspace before opening another workspace.

1. Choose **Start > All Programs > Micro Focus > COBOL Analyzer**. The Open Existing workspace dialog opens above the COBOL Analyzer main window.
 -  **Note:** If COBOL Analyzer is already open, choose **File > Open Workspace**.
2. Click the tab you want to work in:
 - In the Existing tab, choose the drive and folder for the workspace you want to open. A list of workspaces in the selected folder is displayed in the righthand pane. Double-click the workspace you want to open, or select it and click **OK**.
 - In the Recent tab, double-click the recently opened workspace you want to open, or select it and click **OK**.
 -  **Note:** If you created the workspace in a previous release of COBOL Analyzer, the system prompts you to upgrade the workspace to the new release. Click **Yes**.


The workspace is displayed in the COBOL Analyzer window. If a workspace was already open in the window, the selected workspace replaces it.

Registering Source Files

Before you can analyze application source files in COBOL Analyzer, you need to load, or *register*, the source files in a workspace.

 **Note:** In a multiuser environment, only a master user can register source files.


CA registers sources files in-place from the location they are selected rather than creating copies of the files in the workspace folder. These are the files you view and edit in the CA tools.

 **Note:** Workspaces that were created without in-place registration and upgraded from previous versions will continue to create copies of the files in the Sources folder for the workspace when registering new files.

Source files must have recognized DOS file extensions before they can be registered. You can view and add to the recognized extensions in the Workspace Registration options window. Files without extensions are checked for content, and if the content is recognized, the files are registered with appropriate extensions appended.


CA 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 COBOL Analyzer accepts these files as input, some CA 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. COBOL Analyzer automatically unpacks the compressed file and registers its contents.


 **Note:** CA 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.

1. In the Repository Browser, create a project for the source files you want to register, or use the default project. To create a project, choose **Project > New Project**. The Create Project dialog opens. Enter the name of the new project and click **OK**. The new project is displayed in the Repository Browser.
2. Select the project in the Repository Browser, then copy-and-paste (**Ctrl+C** and **Ctrl+V**) or drag-and-drop (if UAC is disabled) the file or folder for the source files you want to register onto the Repository Browser. You are notified that you have registered the files successfully and are prompted to verify the files. Click **Close**. The Repository Browser displays the contents of the new workspace either by folder, or by type, depending on the tab you have selected - **Folders** or **Types**. The former displays the contents of the repository in the same folder structure as the one they have on the host environment, the latter displays them by object type.

 **Note:** For non-inplace registration workspaces, when updating or adding new sources through the CA main window, the files or folders being registered will be included into the selected folder in the Folders view. Registration into the Types view will copy the selected files and folders directly to the root Sources folder, i.e. they will not be copied into any other existing folders. If you are registering files or folders in Folders view, make sure you've selected in the tree the exact folder in which you want them to be registered. If you haven't selected any folder, they will be copied into the root Sources folder.

 **Important:** Before registering your source files, you must create a folder structure that replicates the original. This helps to avoid errors when you work with homonyms. Micro Focus recommends putting each application in a separate folder, putting system files that are common for more than one application in one directory. After you do that, register your sources by selecting the folders and including them in the project as described in step 2.

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

Host Environment Source Compatibility

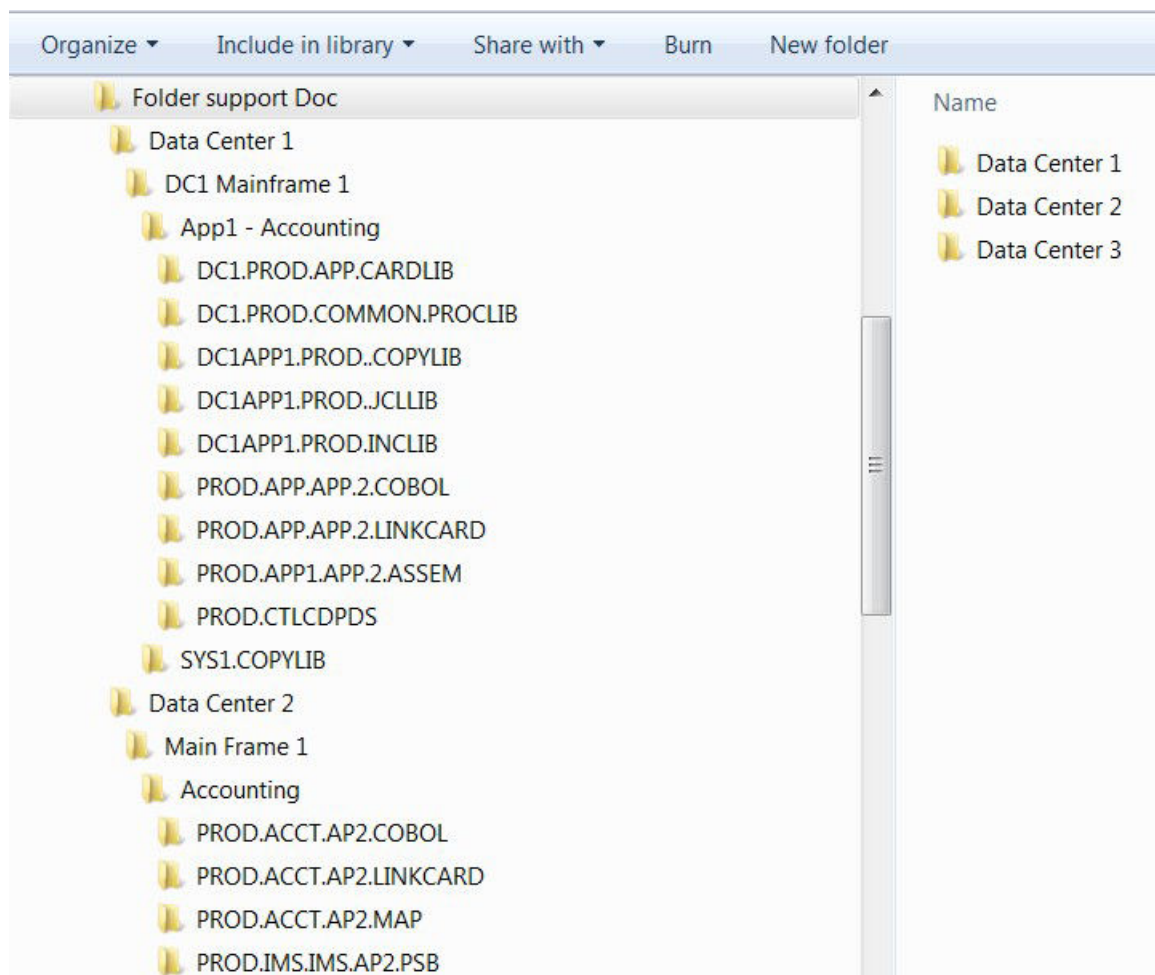
When you register your source files, it is essential that they are organized in the same folder structure as the one on your host environment. This allows CA to create the correct relationships when connecting programs with their copybooks, JCLs with the proper version of the procedures that are used, etc. Runtime connections are also established based on this structure. Furthermore, this structure allows for having several members with the same name, i.e. homonyms.

The sources to be loaded into CA should be in folders named after the PDS libraries on the Mainframe and those folders should be the ones loaded.

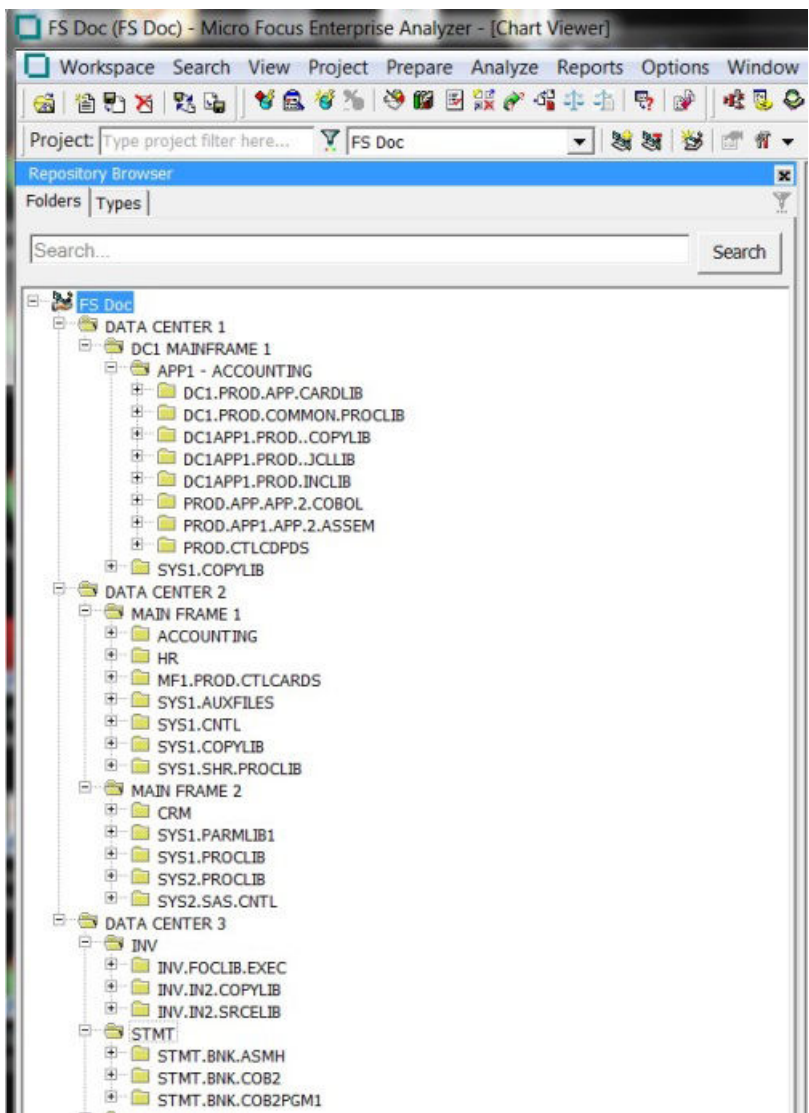
Example

There is an enterprise organization with 3 data centers: Data Center 1 with one Mainframe, Data Center 2 with two Mainframes, and Data Center 3 with one Mainframe.

The following image shows the folders in a Windows environment:



And this is what the three data center folders look like after they have been copied to the workspace:



Resolving Include Files

When trying to resolve include files (copybooks, PL/I includes, JCL procedures, and so on), the CA parser uses different search methods that you can set in the **Homonyms** tab in **Project Options**. The three methods are described below:

Search all, use the provided paths as preference

If you choose this option, the parser will look at all folders trying to find names that match the included file. If only one match is found, it will use that file. If more than one match is found, the parser will take the following steps to resolve which of the matching include files is the one to use:

1. It will check if one of the folders is listed under **Search Path** in the **Homonyms** tab in **Project Options**. The folder should be listed as the full path to the registered sources. It could be local or shared, for example:
 - local:
C:\Sources\Data Center 1\DC1 Mainframe 1\SYS1.COPYLIB
 - shared:
\\server\Sources\Data Center 1\DC1 Mainframe 1\SYS1.COPYLIB

2. It will look for proximity of the included file to the including file in the source folder structure. First, it will check the files that are in the same folder as the "including" source, then the files in the dependent folder, and then up the folder hierarchy.
3. Finally, folder names will be matched and the "include" folder that has a name with a better match to the source folder name will be selected.

Search only in the provided paths When this option is selected, only the paths specified in the Search Path list will be used as locations for looking for include files. As this option works on project level, you can configure each subsystem or application in a separate project, and then specify the copybook locations for each project. This option is useful when the number of applications/subsystem is not too big.

Search in common location and only the same subsystem This option might help when there is a large number of applications/subsystems. It lets you use the folder structure in the file system to define the boundaries between subsystems and to restrict retrieval of include files based on the physical folder structure. If you choose it, you can put all the sources in a single project and avoiding the need to add a list of folders to search in. If you choose this option, you need to specify the following two paths:

Common folder All common libraries need to be located inside the same root folder, for example, C:\Sources\Common. Files in folders located here will be "visible" to all programs being analyzed.

Subsystems root level All subsystems need to have a common root folder, for example, C:\Sources\Subsystems. If you have two subsystems in this folder, one of them will not get copybooks from the other and vice versa, but they can both get copybooks from the common libraries folder specified in the **Common folder** field.



Note: The subsystems root level selection is checked first and then the common folder.



Note: For COBOL only, if there is a registered homonym (copybook) in the program's folder it will be used in preference to the homonym search configuration.

Resolving Program-to-Program Calls

When resolving call relationships between programs, it is possible to reach a point where there is more than one candidate for a called program. In this case CA works the same way as with includes (see *Resolving Include Files*), but since the program objects themselves don't have folders, the folders of the source files that generated them are used.



Note: This same algorithm applies to any relations between non-legacy entities (e.g. PROGRAM reads TABLE, TRANSACTION starts PROGRAM, etc.).

Resolving Control Cards

Control cards can be any members explicitly mentioned in the JCL. When parsing JCLs, CA will look for a file with the member name in the JCL in a folder with a name matching the PDS name from the mainframe.

Resolving JCL Procedures

When looking for JCL procedures, the parser uses the JCLLIB statement, if present. If there is only one copy of the searched procedure, that copy is used. Otherwise, if more than one copy is available, the libraries specified with the JCLLIB statement are used to pick the right one.

Example:

```
Path to the JCL file: <workspace_path>\Sources\APPL\MYJCL\MYJOB.JCL
//MYJOB1 JOB ...
//MYLIBS1 JCLLIB ORDER=CAMPBEL.PROCS.JCL
//S1 EXEC PROC=MYPROCL
```

In this case the procedure <workspace_path>\Sources\APPL\CAMPBEL.PROCS.JCL\MYPROCL, if present, will be used.



Refreshing Registered Source Paths



The Refresh Registered Source Paths feature enables you to change the location of registered source files. It is useful in cases when the sources have been moved to another location, or a folder that contains sources has been moved or renamed, or sources were registered from a non-shared location, and the location has since been shared.

To use this feature:


1. Open COBOL Analyzer Administration.
2. From the **Administer** menu, click **Refresh Registered Source Paths**.
3. Choose a workspace from the list in the **Select Workspace** window.
4. Click **OK**. The **Change Registered Source Paths** window opens. It displays the registered source paths and lets you select the new source paths.

You can choose whether to hide or display system paths in the list of source paths by toggling the **Hide System Paths** check box.

5. Click the yellow folder icon  under **New Source Path**.
6. Choose the new directory of the source files and click **Select Folder**. The selected path and a red cross icon  appear under **New Source Path**.

 **Note:** Clicking  removes the selected path.

7. Click **Change**. A message asking you if you are sure you want to change the registered source paths comes up.
8. Click **OK**. You are notified that the paths have been refreshed.
9. Click **OK**. The Change Registered Source Paths window closes.

 **Note:** Alternatively, you can change the registered source paths from Source Synchronization. See *Source Synchronization* for more information.

After changing the registered source paths, a file must be verified again for a Logic Analyzer extraction to work correctly.

Queue Processor

There are various ways to set parameters for the Queue Processor.

To launch the Queue Processor

1. Start **CA Administration**.
2. Open **Administer > Launch Queue Processor** or press **F7**.

The Launch Queue Processor window opens.

3. In the **Serve workspace** field type the name of the workspace you want to use or press the ... button to browse for a workspace.
4. Check the **Processing Mode - Conversion, Verification** or both.
5. Choose the number of processors to start. The number depends on your system capabilities.

6. Check **Produce Log File** if you want to produce a log file and then click **OK** to start the verification or **Cancel** to quit.

Check Workspace Queue

Check workspace Queue displays the queued processes that are waiting in queue for the current workspace. To view them:

1. Open **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration**.
2. Select **Tools > Check Workspace Queue**. The Select workspace window opens.
3. Select a workspace file (.rwp) and click **Open**. A message with the number of queued records is displayed.

Manage Queue

1. Click **Start > All Programs > Micro Focus > COBOL Analyzer > COBOL Analyzer Administration** to start the COBOL Analyzer Administration. Manage Queue shows the processes that are waiting to be processed. After the Queue processor has finished the list is automatically cleared.
2. Click **Tools > Manage Queue**. The Manage Queue window opens.

It contains a list of all processes waiting to be processed by the Queue Processor. The available actions are:

- **File > Close** - closes the window.
- **Record > Delete** - if you delete an item from the list it remains unverified and you must re-verify the sources again.
- **Record > Unlock** - unlocks records that are locked due to some verification error.
- **Record > Select All** - selects all records from the queue list.
- **Record > Unselect All** - deselects all records from the queue.
- **Queue > Refresh** - refreshes the queue list.
- **Queue > Show All** - shows all records for the queued processes.
- **Queue > Auto Refresh** - automatically refreshes the list.

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.

You can verify a single file, a group of files, all the files in a folder, or all the files in a project. It's usually best to verify an entire project. COBOL Analyzer parses the files in appropriate order, taking account of likely dependencies between file types.



Note: You need not verify copybooks. Copybooks are parsed when the including source file is verified.

Workspace and Project Verification options determine verification behavior. The default values for these options are preset based on your configuration and should be appropriate for most installations.




1. In the Repository Browser, select the project, folder, or files you want to verify and choose **Prepare > Verify**.
2. 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.


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.

How Verification Results Are Indicated

If you have not verified a workspace file, the file is displayed in the Repository pane in **bold** type. Since the system 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 that the file has been successfully verified under the relaxed parsing option.

 **Note:** When you edit a file, the modified file and any dependent files are *invalidated*, and need to be verified again. You can learn more about how objects are invalidated and how the system refreshes the repository in *Preparing Projects* in the product documentation set.

Using the COBOL Analyzer Main Window

The COBOL Analyzer main window lets you view and edit projects, set product options, and open CA tools. The window's state is saved across sessions.

Open a pane in the main window by selecting the appropriate choice in the **View** menu. Resize a pane by using the mouse to grab its border with another pane, called a *splitter*, and dragging it to a new location. Close a pane by clicking the  button in the upper right corner.


When you start COBOL Analyzer for the first time, the main window displays the following panes: Repository Browser, Getting Started, Chart Viewer and Activity Log.

The Getting Started pane gives you basic information about some of the main features and provides quick access to them.

The other panes are described in the topics below.

Menus

COBOL Analyzer menus let you access COBOL Analyzer tools and other product functionality. Menus are displayed only for the products you have enabled in the Configuration Manager. The table below shows the relationship between products and menus.

 **Note:** The screens shown in the product documentation set display the menu bar with all products installed.

Most COBOL Analyzer menu choices have equivalent choices in the right-click menu. Menu selections for panes in the main window are available only if the pane has the focus.

Product	Menu
COBOL Analyzer	Analyze
Enterprise View	Profile
Business Rule Manager	Extract

Toolbars

You can access a COBOL Analyzer menu function by clicking its tool bar icon. The tool tip for an icon tells you what it does. Place your cursor over the icon for a moment. The tool tip will display.

A tool bar is displayed only if its corresponding menu is displayed. You can change the position of a tool bar by grabbing the delimiter at the left of the tool bar and dragging it to a new location.

Project Filtering

Filter the projects in the repository browser to see only the one(s) that you want to work with. The Project filter field is on the toolbar. Enter the name of the project you want to see in the Project filter and all other project will be hidden. The filter string is saved when closing and reopening the workspace. To clear the filter, delete the text in the input box and press Enter.

Use wildcards and regular expressions to refine the project filtering.



Note: An asterisk wildcard is assumed at the end of the filter name.

Example

If you have 5 projects named Project 1 through Project 5 and enter Project [1-3] in the project filter, only Project 1, Project 2, and Project 3 will be shown in the browser. If no projects match the filter, the Browser will be empty.

Using the Repository Browser Pane

The Repository Browser pane shows the contents of the repository for the current workspace. It has two tabs:

Folders Displays the contents of the repository in the same folder structure as the one they have in the host environment. This allows for elements with the same name (homonyms) to be loaded and displayed.

Types Displays the contents of the repository by project, object type, and object relationship.

Each of the two tabs consists of the following elements:

Browser Shows the objects in the repository of the current workspace in a tree structure. Depending on the tab you have selected in the Repository Browser pane, the contents are displayed either by type or by folder.

Search Lets you execute various searches in the repository of the current workspace.

In the Repository Browser window, hover your cursor over an object to display a tooltip that describes the object in more detail.



Note: The Repository Browser pane displays automatically when you launch COBOL Analyzer. If you close it, you can reopen it by clicking **View > Repository**.

Using the Search Assistant


The Search Assistant is integrated in the Repository pane. It searches all sources in the current workspace. The results are displayed in a tree in the browser window, either by type or by folder, depending on the tab you have selected in the Repository pane - **Folders** or **Types**. The information in the tree is grouped in three levels:


- Project level - shows the projects that contain the objects.
- Folder level - shows the relative folder structure. If there is more than one folder level, they are grouped and displayed with slashes - Folder1/Folder2/Folder3(*n*). The number in brackets indicates the number of search result objects in the folder.
- Object level - shows the objects that are the result of the search.




Note: For information about customizing the search options, see *Configuring the Search Assistant*.


To use the Search Assistant:

1. Type in a search string in the search box and click . The search returns all workspace objects whose names match the search criteria.
2. Drill down the tree to see the relationship types and available methods for the objects.
3. Double-click an object in the search result tree to see its source code.


 **Note:** To see the source code of a declaration, double-click it or right-click and select **Go To Source**.

4. Right-click the search result objects in the Browser to see the list of available context-sensitive actions for the selected objects.

 **Note:** Not all of the context menu options available in the Repository Browser are available in the search results tree.

 **Note:** Multiselect is also available for the items in the search result tree.

To reset a search or change the search criteria in the search field:

1. Click **Close**  in the left corner of the search field.

The search is reset and the last Repository Browser view you used is displayed.

COBOL Analyzer keeps a history of the searches you have done so when you start typing a search string, it lets you select from a list of search strings you have previously used that begin with the same characters as the ones you have typed.

Customizing the Search Assistant

You can customize the Search Assistant to suit your needs by setting some options in the Workspace Options and User Preferences that you can access from the Options menu.

Setting Search Assistant options in the Workspace Options

There are two options that you can set in **Options > Workspace Options > Search Assistant**:

- Maximum number of results to display. The default is 10,000.
- Timeout for displaying a complete set of results. It is configured in seconds and the default is 60 seconds.

Setting Search Assistant options in the User Preferences

From the Search Assistant tab in **Options > User Preferences** you can set the following options:

- Select the type of the objects that will be shown in the search results. The available types are: all legacy, system program, program entry point, data store, table, transaction and ADABAS file.
- Include hypercode objects. Depending on the source type, you can select different types of hypercode entities. The table below displays the available source code types and their corresponding hypercode objects.

Source type	Hypercode entity
COBOL file	<ul style="list-style-type: none"> • Declaration • Paragraph • Variable
JCL file	Data set
Natural file	<ul style="list-style-type: none"> • Declaration • Variable


Source type	Hypercode entity
PL/I file	<ul style="list-style-type: none"> • Declaration • Variable
RPG file	<ul style="list-style-type: none"> • Declaration • Variable

By default, declaration is selected for COBOL, Natural and PL/I files.

Using the Browser

The Browser displays the sources in the repository of the current workspace, organized in a tree. Depending on the tab you have selected in the Repository pane, the contents are displayed either by type or by folder. Switch between them to browse the repository in different ways.

- Type view - shows the object model for the workspace, organized in a tree form by project, object type, and object relationship.
- Folder view - shows the sources in the tree in the same folder structure as the one they have in the host environment.


 **Note:** When you work with homonyms, you will see a number in parentheses after the name of the object if there is more than one instance of it.

Selecting Objects

Select an object in the Browser by clicking it with the left mouse button. You can select all, select multiple, or type-ahead select.

Use the **Select** choices in the **Edit** menu to select objects based on their verification status. The choices operate on the entire list in the Search tab and on the objects under the selected workspace, project, or folder in the Browser.

The **Project** drop-down displays the current project. You can navigate quickly to another project in the Browser by clicking the project name in the drop-down.

 **Note:** In the Folders view, you can use multiselect for objects in different folders on the same level. You cannot use multiselect for child and parent nodes. You cannot use multiselect across projects.

Selecting All


To select all the objects in the tree or all the objects in the selected node, choose **Edit > Select All**. To deselect the objects, choose **Edit > Unselect All**.




Selecting Multiple

To select a range of objects, hold down the Shift key, click the first object in the range, then click the last object in the range. To select objects that are not in a range, hold down the Control key, then click each object you want to select.



Using the Type-Ahead Select Functions



Use the *simple type-ahead select function* to select displayed objects as you type. To use the simple function, make sure the Browser tab is current, then type the characters you want to match.

 **Note:** You cannot use the simple type-ahead select function in the Search tab.


Use the *advanced type ahead select function* to select objects in the Browser or Search tab whether or not they are currently displayed. To use the advanced function, click an object that has the same type as the object you want to select, then click the  button next to the  button at the bottom of the Repository pane (or if the filter subtree function is enabled, next to the  button). From the drop-down menu, choose:

- **Select** to match text explicitly.
- **Wildcard Select** to match text using wildcard patterns.



Enter the text you want to match in the field next to the  button. The tool selects matched objects as you type, as long as they are in the same subtree. Click the  button to repeat the select with the same criterion.

To select multiple objects, enter the text you want to match in the field next to the  button, then click the adjacent  button. From the drop-down menu, choose:

- **Select All** to select multiple objects that match text explicitly.
- **Wildcard Select All** to select multiple objects that match text using wildcard patterns.

The tool selects matched objects as long as they are in the same subtree. Click the  button to repeat the select with the same criterion.

Expanding Subtrees




Click the  symbol next to an item to expand its subtree one level. Click the  symbol to collapse the subtree one level.


In the Repository Browser, select an item and choose **Expand Subtree** in the right-click menu to expand all its subtree levels in a single step. Select an item and choose **Save Subtree** in the right-click menu to export the label text for expanded subtree levels to a text file.



Note: For subtrees with a great many objects, you can improve browser display performance by filtering the subtree and/or by lowering the fetch buffer size in the General tab of the Workspace Options window.

Filtering Subtrees

For a subtree with a great many objects, you can improve browser display performance by showing only objects of interest in the subtree. To filter out objects from a subtree, select the subtree, then enter the text you want to match in the field next to the  button at the bottom of the Repository pane (or if the type-ahead select function is enabled, next to the  button). Click the adjacent  button and choose **Filter** from the drop-down menu.

The tool filters out unmatched objects from the subtree, and places a  symbol next to the subtree in the browser display. The filter and filter text are persistent for the subtree across sessions. To remove the filter and its text, enter an empty string in the text field and choose **Filter** again.

Refreshing the Browser

In a multiuser environment, you need to *refresh the browser* after another user adds, deletes, or modifies a workspace file. Choose **View > Refresh Browser** to refresh the browser.

Viewing the Entire Workspace

By default, the Browser tab displays repository trees for projects only. These trees show only objects you can act on in the browser. To view all the objects in the workspace, choose **View > Entire Workspace**, or

select (none) in the **Project** drop-down. Choose **View > Entire Workspace** again, or select a project in the **Project** drop-down, to display the project only.

Performing a Text Search in the Repository

The Find functionality lets you do a plain text search in the repository files. The results are displayed in the Found Results tab in the Repository Browser. You can select to view the results by folder structure or object types. Drilling down the results tree, you can see the object name as well as the specific line of code where the searched text is.

To use this functionality, go to **Search > Find**, type a text and click **Find All**.

Before you click **Find All**, you can select various options such as skipping comments from the search or matching the whole words and/or the case of the searched text. You can also specify the search mode, i.e. whether to search for plain text, regular expressions and wildcards.

If you want to further specify your search, click **Advanced Find**. Here you can select the type of objects in which to search depending on their parse status: unverified, verified with errors, verified with relaxed parser and verified successfully. You can also specify the scope of the search (workspace or selection), the project where to search and an object name.

You can additionally customize the text search functionality from **Options > Workspace Options > Find Options**. In this tab you can set the following options:

- Maximum number of results to display. The default is 10,000.
- Font style of the results: regular (default), bold, italic, strikeout or underline.
- Edge Color - lets you select the color in which the searched text appears in the results. The default color is red.

Using Wildcard Patterns in Searches

In most searches you perform in the CA tools, you can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). The table below shows the wildcard characters and what they match.

Character	Matches
?	Any single character.
*	Zero or more characters.
#	Any single digit (0–9).
[charlist]	Any single character in charlist.
[!charlist]	Any single character not in charlist.

To match the special characters left bracket ([), question mark (?), number sign (#), and asterisk (*), enclose them in brackets. The right bracket (]) cannot be used within a group to match itself, but it can be used outside a group as an individual character.

Specify a range of characters in a charlist by using a hyphen (–) to separate the upper and lower bounds of the range. [A-Z], for example, results in a match if the corresponding character position in a string contains any uppercase letters in the range A–Z. You can specify multiple ranges in brackets without delimiters.

Finding and Replacing Text in Source Files

To find and replace text in the Editor:


1. In the Repository Browser, double-click a source file. The file opens in the Editor.
2. Right-click and select **Find** or click **Edit > Find**. The Find and Replace dialog opens.
3. Click the tab for the operation you want to perform.

The find functionality in the Find tab is bidirectional. You can search up or down, and specify if you want the search to wrap as well as match the whole word or the case. The find functionality in the Replace

tab is unidirectional. From both tabs you can specify the search mode, i.e. whether to search for plain text, regular expressions and wildcards.



Note: The remaining steps assume that you want to both search and replace.

4. In the **Find What** box, type the search criterion. Check the necessary check boxes.
5. In the **Replace With** box, select the replacement text. You can select text strings from previous replace operations or type a new replacement text. If you have selected to search for regular expressions, click the  button next to the box to display a list of elements you can insert in the replacement text field.
6. Click **Find Next**. When the search finds a match, click **Replace**. Click **Find Next** again to find the next match, or click **Replace All** to replace all matching text.

Using the Editor and Viewer

The Editor lets you view and edit the source code for a file. The Viewer lets you view renderings of logical objects like screens and transformation target models. Double-click a file in the Repository pane to view it in the Editor or Viewer. Choose **File > Recent Sources** to open a file from a list of recently opened files.

Editing Source

Edit file source as you would text in a word processor (cut-and-paste, drag-and-drop, undo, and so forth). Click **File > Save** to save your changes. Edits are saved in the copy of the file maintained in the `\Workspace\Source` subfolder.



Note: When you edit a file, the modified file and any dependent files are invalidated, and need to be verified again. Only a master user can edit and verify source in a multiuser environment.

Finding and Replacing Source in a Selection

To find and replace source in a selection of text, select the text, then follow the instructions for the source file find and replace function.

Viewing a List of Errors

For a source files with errors or warnings, right-click in the Editor and choose:

- **Errors > View Errors** from the popup menu to list the errors and warnings in a separate pane below the Editor. Choose **Edit > Errors > View Errors** again to hide the pane.
- **Errors > Next Error** from the popup menu to navigate to the offending source code for the next error in the list.
- **Errors > Previous Error** to navigate to the offending source code for the previous error in the list.
- **Errors > Severity Filter** from the popup menu to filter the list by severity.



Note: You can set a severity filter for all your workspaces in the Editor User Preferences.

- **Errors > Copy Errors** in the pop-up menu to copy the list of errors and warnings to the clipboard.

To hide the Editor and show only the list of errors and warnings for the selected file, right-click in the Repository pane and choose **Show Errors** in the pop-up menu. Double-click the source file in the Repository pane to show the Editor again.

Commenting Out Source

To comment out a block of source, select the block and choose **Edit > Comment Block**. To uncomment the block, select it and choose **Edit > Uncomment Block**.

Printing Source

To print the source for a file, choose **File > Print**.

Setting Editor User Preferences

Use the Editor tab of the User Preferences window to specify the display characteristics of the Editor.

1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Editor tab.
2. In the **Font** drop-down, select the display font for source code.
3. Select the **Colorize Text** check box if you want to use a color scheme to identify source elements, then specify the color of each source element.
4. In the Show Errors pane, select the check box for each type of error you want to show in the error list.

Specifying the Color of Items


You use a standard Windows color control to specify a color scheme for source elements and other items. Use the Palette tab to select a color from the Windows palette. Use the System tab to match the color of a standard Windows element. Depending on the element, you can specify the foreground color, background color, and/or font color.


Using the Chart Viewer

The Chart Viewer in the main window lets you view information about the objects in the workspace in the form of various charts. You can choose between two types of views: Interactive Chart View, which is the default one, and Report Chart View.


To switch between the two views, use the drop-down list at the top of the Chart Viewer.

To change the default chart view, go to **Options > User Preferences > Environment > Chart Viewer** and select a new default view from the drop-down list. Note that there is also an option letting you choose whether to have the Chart Viewer displayed when you start CA or not.

 **Note:** To open Chart Viewer when it's not displayed in the main window, go to **View > Chart Viewer**.

To refresh the charts, click .

Interactive Chart View

The Interactive Chart View provides information about the objects in the workspace in the form of interactive charts. Clicking the charts lets you drill down and access more detailed information in the form of charts and tables. To drill back up, click .

Right-clicking in the tables lets you select from a number of options: edit, copy or delete objects, include in or exclude from the project, assign tags, view properties. You can also choose to have the files shown in the Repository Browser pane or to open Interactive Analysis. The last option in the context menu, **Query Repository**, lets you view additional information about the object such as: dependent or used sources, direct references and what the legacy source is.

There are four main pie charts:

- **Object Types** - a pie chart representation of the types of objects and their number. Clicking on any segment opens the **Average Lines of Code** chart which is a representation of all objects of the selected type grouped by the number of lines of code. Clicking on a segment in this chart displays a table with a list of all objects that fit into this segment as well as information about them such as: type, source name, source path and lines of code.
- **Cyclomatic Complexity** - presents the cyclomatic complexity index for programs of all types. Clicking on a segment in this chart displays a list of programs that fit the selected cyclomatic complexity index boundary. The list is presented in a table containing information such as: type, program name, cyclomatic complexity, maintainability index and legacy name.
- **Average Modifications** - represents the objects that have been modified grouped by the number of modifications. Clicking on a segment drills down to the **Object Types** chart. Clicking further opens a table with information about object type, source name and source path.

- **Average Modification Date** - represents the number of files of all types that have been modified within specific time periods. Clicking on a segment drills down to the **Object Types** chart. Clicking further opens a table with information about object type, source name and source path.

Report Chart View

The Report Chart View displays information about the objects in the workspace in three charts: Object Types, Inventory Report and Complexity Report.

The Object Types is a pie chart representation of the types of objects and their number.


The Inventory Report Chart shows the number of objects that are verified, unverified, erroneous and relaxed, grouped by type.

The Complexity Report chart provides additional information about the various types of objects such as number of source lines, lines with comments, blank lines and include statements.

Using the Object Information Pane


The Object Information pane displays comments you have entered for an object. Select an object in the Repository pane and choose **View > Object Information** to show the Object Information pane.

Enter comments for an object by selecting the object in the Repository pane and choosing **Properties** in the right-click menu. In the Description tab, enter your comment in the text area and click **OK**.

 **Note:** You cannot use this method to enter comments for a workspace (that is, when an entire workspace is displayed in the Repository Browser). Use the General tab of the Workspace Options window to enter comments for a workspace.

Viewing Object Properties

The *properties* of an object describe its characteristics: its language, complexity metrics, relationships, and the like. Select an object in the Repository pane and choose **View > Properties** to display a set of tabs with object properties. The table below describes the tabs in the Properties window.


 **Note:** In other CA tools, use the right-click menu or the **View** menu to display the properties of the selected object.


Tab	Description
Description	Displays comments you have entered for the object.
General	Displays the name and other general information for the object.
Internal	Displays verification option settings and other internal information for the object.
System	Displays system information for the object.
Statistic	Displays the complexity metrics for the object and their values.
Relationships	Displays the relationships of the object. Only the first relationship level is shown but if you want to explore a particular branch you can do it by double-clicking the object. Select Restricted to the current project to restrict relationships to those in the current project. Select Workspace wide to show all the relationships of the object in the workspace.
Object Tags	Displays the tags assigned to the object.

Viewing Inventory Reports

Inventory Reports give high-level statistics for source file types in the current workspace: number of files of each type, whether verified, number of lines of source code (verified files only), and the like. To generate an

Inventory Report, choose **Reports > Inventory Report** . The table below describes the columns in the Inventory Report.

Column	Description
Type	The source file type.
Source Lines	The number of physical source code lines in the source text files (including blank lines and comments).
Quantity	The number of source files of the specified type.
Unverified	The number of source files of the specified type for which verification was not attempted or for which verification failed completely .  Note: Copybooks do not need to be verified. Copybooks are parsed when the including source file is verified.
Verified	The number of successfully verified source files of the specified type.
Erroneous	The number of unsuccessfully verified source files of the specified type.
Relaxed	The number of source files of the specified type that were verified despite errors, under the relaxed parsing option.
Missing	The number of missing support files (copybooks, JCL procedures, and the like).

 **Note:** The workspace might have more copybooks or include files than the projects because some default system copybooks may be included automatically to support specific environments or dialects.

Viewing Key Object Relationships with Query Repository

The Query Repository feature lets you home 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.

1. In the Repository pane, select the workspace, project, folder, files, or objects you want to view relationships for and choose **Query Repository > <Relationship>** in the right-click menu. The Query Repository results pane opens over the Editor/Viewer.
2. To launch a query from the results, select one or more objects in the right-hand column and choose **Query Repository > <Relationship>** in the right-click menu.

Using the Activity Log

The Activity Log is a chronological record of your activities in the current session. Choose **View > Activity Log** to show the Activity Log pane.

Viewing Errors

If the parser generated errors or warnings for a file, double-click the Activity Log entry for the file or select the entry and choose **Expand** in the right-click menu to show a list of the errors and/or warnings. Double-click a message to navigate to the offending source in the Editor. Choose **Collapse** in the right-click menu to collapse the list.

Viewing History

To view Activity Log entries across sessions, choose **History** in the right-click menu. Click a date to view the session log. To clear the session log, right-click in the log and choose **Clear Log** in the pop-up menu. To clear a log entry, select the entry and choose **Delete** in the right-click menu.

Copying Records

To copy Activity Log records to the clipboard, select the records and choose **Copy** in the right-click menu.

Generating Reports

To generate a report from the Activity Log, right-click in the log and choose **Report** in the pop-up menu.

Clearing the Log

To clear the Activity Log, right-click in the log and choose **Clear Log** in the pop-up menu. To clear a log entry, select the entry and choose **Delete** in the right-click menu.

Hiding a Column

To hide a column in the Activity Log, right-click in the log and deselect the column name in the **Columns** choice in the pop-up menu. Select the column name to show the column again.

Creating a To Do List


A *to do list* is a record of tasks you need to perform before completing your project. You can create a to do list for a workspace, project, folder, source file, or extracted object.

1. In the Repository Browser, select a workspace, project, folder, source file, or generated object and choose **View > ToDo List**. The ToDo List window opens.
2. If you selected:
 - A workspace, project, or folder, choose **Edit > Add** to enter a task for the item and all the items it contains. Otherwise, choose **Add for Item** to enter a task for the workspace, project, or folder only.
 - A source file or generated object, choose **Edit > Add** to enter a task for the item.
3. The New Properties dialog opens. In the **Subject** field, enter the name of the task. In the **Content** field, enter a description of the task. Click **OK**. The task is listed in the ToDo List window.

Viewing a To Do List

To view the to do list for an item, select the item in the Repository Browser and choose **View > ToDo List**. Use the options in the ToDo List window **Edit** menu to mark a task as done, delete a task, or view its properties. You can generate a report from a to do list by choosing **File > Report**.

Using Tool Windows

When you open a COBOL Analyzer tool, the window for the tool appears above the product main window. Simply move the tool window if you want to work in the main window again. A tool window's state is saved across sessions. Close a tool window by clicking the  button in the upper righthand corner.

Tool windows are *modeless*: in principle, you can open as many tool windows as you want. In practice, COBOL Analyzer will prompt you to close an "unprotected" tool window if the activity you want to perform could result in a repository change. An unprotected tool window cannot be updated to reflect a concurrent repository change.

In tool windows with lists, you can select all, select multiple, size columns, and sort by column.

Resizing Panes

Resize a pane in a tool window by using the mouse to grab its border with another pane, called a *splitter*, and dragging the border to a new location. To control how panes are resized when you resize the window, select the splitter and choose one of the following in the right-click menu:

- For a vertical splitter, **Keep First** preserves the size of the panes to the left of the splitter. For a horizontal splitter, **Keep First** preserves the size of the panes above the splitter.

- For a vertical splitter, **Keep Second** preserves the size of the panes to the right of the splitter. For a horizontal splitter, **Keep Second** preserves the size of the panes below the splitter.
- **Keep Proportion** preserves the length and width ratio of the panes on either side of the splitter.

Moving Panes

Move a pane in the window by grabbing its title bar to show the “virtual rectangle” for the pane. The virtual rectangle is a dotted color frame that you can drag to the position in the window at which you want the pane to appear. When you are satisfied with the new position, release the mouse button. The other panes in the window are adjusted as necessary.



Note: You may have to drag the virtual rectangle beyond the edge of the tool window to achieve the effect you want.

To swap the positions of the panes on either side of a splitter, select the splitter and choose **Swap** in the right-click menu. To change the color of the splitter when it is selected (and of its virtual rectangle), select the splitter and choose **Color** in the right-click menu. Use the standard Windows color control to specify the color.

Setting Options

COBOL Analyzer options control the look and feel of the product tools and how they perform their tasks. The **Tools** menu gives you access to three sets of options:

- **Workspace Options** control the appearance and behavior of CA tools for the current workspace.
- **Project Options** control the appearance and behavior of CA tools for the current project.
- **User Preferences** control the appearance and behavior of the Editor, Interactive Analysis, and Change Analyzer across workspaces, and miscellaneous other tasks.

Each tool has an **Options** choice in the **View** menu that lets you set the options for that tool.



Note: Many options are preset based on your choices in the COBOL Analyzer Configuration Manager. For more information, see the installation guide for your product.

Tool options are described in detail in the relevant sections of the documentation set. This section shows you how to manage option sets and how to set general CA and environment options


Managing Option Sets


An *option set* is a particular configuration of settings for one of the option groupings. In a given Project Options set, for example, you might choose to show GO TO statements in the Interactive Analysis Program Control Flow tool, while in the other Project Options sets you would not.

Option sets let you switch back and forth between configurations in the same session, without having to change settings manually. You can also use them to ensure uniform results across your team by sharing customized settings with team members.

Understanding the Options Manager Window

Use the Options Manager to define custom options sets. Choose **Options > Options Manager** to open the Options Manager window.

The figure below shows the Options Manager window at startup, with the tab for the Workspace Options grouping selected. For each grouping, there is an initial read-only *default option set*, denoted by a  symbol, that reflects your current CA configuration. The *active* option set is displayed in **bold** type in the Options Manager tab for the grouping. You make an option set active by *assigning* it. Assignments persist across COBOL Analyzer sessions.

The user who defines a custom option set is referred to as its *owner*. Only the owner can *share* a custom option set with other users, as indicated by a  symbol. A shared option set cannot be deleted or unshared if it is being used. The **Used By** drop-down at the bottom of the owner’s Options Manager

Copying an Option Set

To copy an option set, select it and click **Copy**. Options Manager prepends “Copy of” to the name of the copied option set.

Deleting an Option Set


To delete an option set, select it and click **Delete**. You cannot delete a default option set, or a shared option set that has been assigned by another user.

Sharing an Option Set

To share an option set, select it and click **Sharing**. The button is a toggle. Select it again to revoke sharing. You cannot revoke sharing for an option set that has been assigned by another user.

Making an Option Set the Default


To make a custom option set the default, share it first, then select it and click **Default**.

 **Note:** In the Project Options tab, a dialog opens that prompts you to select or create a new project. Select the project and click **OK**.

The new default option set is automatically assigned to every user who has not explicitly defined a custom option set. A default option set cannot be modified or deleted.

Assigning an Option Set

To make an option set active, select it and click **Assign**.

 **Note:** In the Project Options tab, a dialog opens that prompts you to select or create a new project. Select the project and click **OK**.

Importing and Exporting Option Sets


To import an option set, click **Import**. A Load dialog opens, where you can specify the option set you want to import.

To export an option set, select it and click **Export**. A Save dialog opens, where you can specify the name and location of the option set.

Setting General Options

Use the General tab of the Workspace Options window to specify the description of the workspace displayed in the Object Information pane, the attributes displayed in the Repository pane Search tab, and the browser fetch buffer size.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the General tab.
2. In the **Workspace description** field, enter a description of your workspace. This description is shown in the Object Information pane.
3. In Repository Browser (Search), check the boxes for the object attributes that should be displayed in the Search tab of the Repository pane. Click **Select All** to check all the attributes.

 **Note:** Only attributes in the repository model for an object are displayed. That is, if you choose Cyclomatic Complexity, and only JCL files appear in the Search tab results list, the attribute will not be displayed in the list, because Cyclomatic Complexity is not an attribute of a JCL file.

4. In the **Browser Fetch Buffer Size** field, enter the size of the buffer COBOL Analyzer uses to populate the repository tree. When you expand a node in the Repository Browser, COBOL Analyzer populates the display from a buffer that contains the objects last fetched from the repository. For very large repositories, you may want to lower the buffer size so that you can work with objects in the Repository Browser while the product performs subsequent fetches.

Setting Environment User Preferences

Use the Environment tab of the User Preferences window to specify whether you want to be prompted to verify source files after registering them and the scope of relationships shown in the Object Properties window.

1. Choose **Options > User Preferences > Environment**.
2. In After new files are registered, choose:
 - **Verify registered files** if you want COBOL Analyzer to verify files after they have been registered.
 - **Ask user about verification** if you want to be prompted to verify files after they have been registered. Available whether or not **Verify registered files** is selected.
3. From the **Code Search Report Options** you can select whether to be prompted that queries will be run on all programs when you try to run a Code Search report for a project.
4. The Chart Viewer options let you select:
 - **Show Chart Viewer on startup** - if you want to have the Chart Viewer displayed when you start CA.
 - **Default Chart View** - if you want to change the default chart view.
5. In Show relationships in the object properties view window, choose whether you want to restrict the relationships displayed in the Relationships tab of the Object Properties window:
 - **Restricted to the current project** if you want to restrict the relationships of the object to those in the current project.
 - **Workspace wide** if you want to display all the relationships of the object in the current workspace.
6. Select a main menu scheme: simplified or classic.
7. For Java support, specify the path of the folder for the Java Runtime Environment (JRE) on your machine in the **Path to JRE Folder** field. If you do not enter a path, the current version specified in the Windows registry is used. You must use JRE version 1.5 or later.



Note: If your application was compiled with a previous version of the JRE, specify the path of the Java SE runtime library (rt.jar) used to compile the application in the **Enter classpath to JAR Files and/or path to external Java file root directories** field in Workspace Options > Verification > Settings.

Setting Archivers User Preferences

The CA registration process extracts compressed source files using the command line syntax for archiver versions most widely in use. Use the Archivers tab of the User Preferences window to specify the command line syntax for newer archiver versions.

1. Choose **Options > User Preferences**. The User Preferences window opens. Click the Archivers tab.
2. In the **Command Line for Archiver** drop-down, choose the archiver whose command line syntax you want to specify.
3. In the **Unpack All** field, enter the command line syntax for the unpack all operation.
4. In the **Extract File** field, enter the command line syntax for the extract file operation.

Setting Export Options

Use the Export Options tab of the Project Options window to specify where COBOL Analyzer stores generated files. The default location is the `\<Workspace Home>\Output` folder.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Export Options tab.
2. In the **Target directory** field, enter the folder you want COBOL Analyzer to store generated files in.
3. Deselect the **Move Files** check box if you want to copy generated files to both the default and the specified directories.

Adding a Selectable Item

Many options windows allow you to add an item to a list of available items. In the Extensions tab of the Workspace Registration options, for example, you can add an extension to the list of recognized extensions.

Add an item by right-clicking in the selection pane and choosing **Add** in the pop-up menu. The CA displays an empty text field next to a selected check box. Enter the name of the item in the text field and click outside the field.



Note: If you are entering a file extension, make sure to enter the dot (.).

Edit an item 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.

Working with Reports and Diagrams

When you generate a diagram or report in the COBOL Analyzer, you typically have the option of saving it to a file or printing it. You can save to a wide variety of standard formats.



Note: Word must be installed to save reports to RTF and Word. Excel must be installed to save reports to Excel, CSV, and TXT. Visio 2002 must be installed to save a diagram to Visio. Visio 2002 is not required to save to Visio XML.

Click **Page Setup** in a report window to set up the page for a printed report. Click **Print** to print the report. The Print dialog opens, where you can set options for the print job. Click **OK**.

Click **Save** in a report window to export a report. A Save As dialog opens, where you can specify the name, location, and file type of the report.

Using the Guide for a Tool

Many COBOL Analyzer 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 **View > Guide**. Use the table of contents in the **Page** drop-down to navigate quickly to a topic.

Purge Activity Log


Activity Log grows in size. To remove unwanted information, use Purge Activity Log to permanently delete contents from the database.

1. Open Micro Focus COBOL Analyzer Administration.
2. Click **Tools > Purge Activity Log**. The **Purge Activity Log** window opens.
3. Select a date from which to delete Activity Log data.
 - Click **Delete** to proceed with log deletion. This will actually delete items from the Activity Log history table in the database.
 - Click **Close** to exit without any change.


Preparing Projects

Registering Source Files

Before you can analyze application source files in COBOL Analyzer, you need to load, or *register*, the source files in a workspace.

 **Note:** In a multiuser environment, only a master user can register source files.


CA registers source files in-place from the location they are selected rather than creating copies of the files in the workspace folder. These are the files you view and edit in the CA tools.

 **Note:** Workspaces that were created without in-place registration and upgraded from previous versions will continue to create copies of the files in the Sources folder for the workspace when registering new files.

Source files must have recognized DOS file extensions before they can be registered. You can view and add to the recognized extensions in the Workspace Registration options window. Files without extensions are checked for content, and if the content is recognized, the files are registered with appropriate extensions appended.


CA 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 COBOL Analyzer accepts these files as input, some CA 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. COBOL Analyzer automatically unpacks the compressed file and registers its contents.


 **Note:** CA 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.

1. In the Repository Browser, create a project for the source files you want to register, or use the default project. To create a project, choose **Project > New Project**. The Create Project dialog opens. Enter the name of the new project and click **OK**. The new project is displayed in the Repository Browser.
2. Select the project in the Repository Browser, then copy-and-paste (**Ctrl+C** and **Ctrl+V**) or drag-and-drop (if UAC is disabled) the file or folder for the source files you want to register onto the Repository Browser. You are notified that you have registered the files successfully and are prompted to verify the files. Click **Close**. The Repository Browser displays the contents of the new workspace either by folder, or by type, depending on the tab you have selected - **Folders** or **Types**. The former displays the contents of the repository in the same folder structure as the one they have on the host environment, the latter displays them by object type.

 **Note:** For non-inplace registration workspaces, when updating or adding new sources through the CA main window, the files or folders being registered will be included into the selected folder in the Folders view. Registration into the Types view will copy the selected files and folders directly to the root Sources folder, i.e. they will not be copied into any other existing folders. If you are registering files or folders in Folders view, make sure you've selected in the tree the exact folder in which you want them to be registered. If you haven't selected any folder, they will be copied into the root Sources folder.

 **Important:** Before registering your source files, you must create a folder structure that replicates the original. This helps to avoid errors when you work with homonyms. Micro Focus recommends putting each application in a separate folder, putting system files that are common for more than one application in one directory. After you do that, register your sources by selecting the folders and including them in the project as described in step 2.

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

Host Environment Source Compatibility

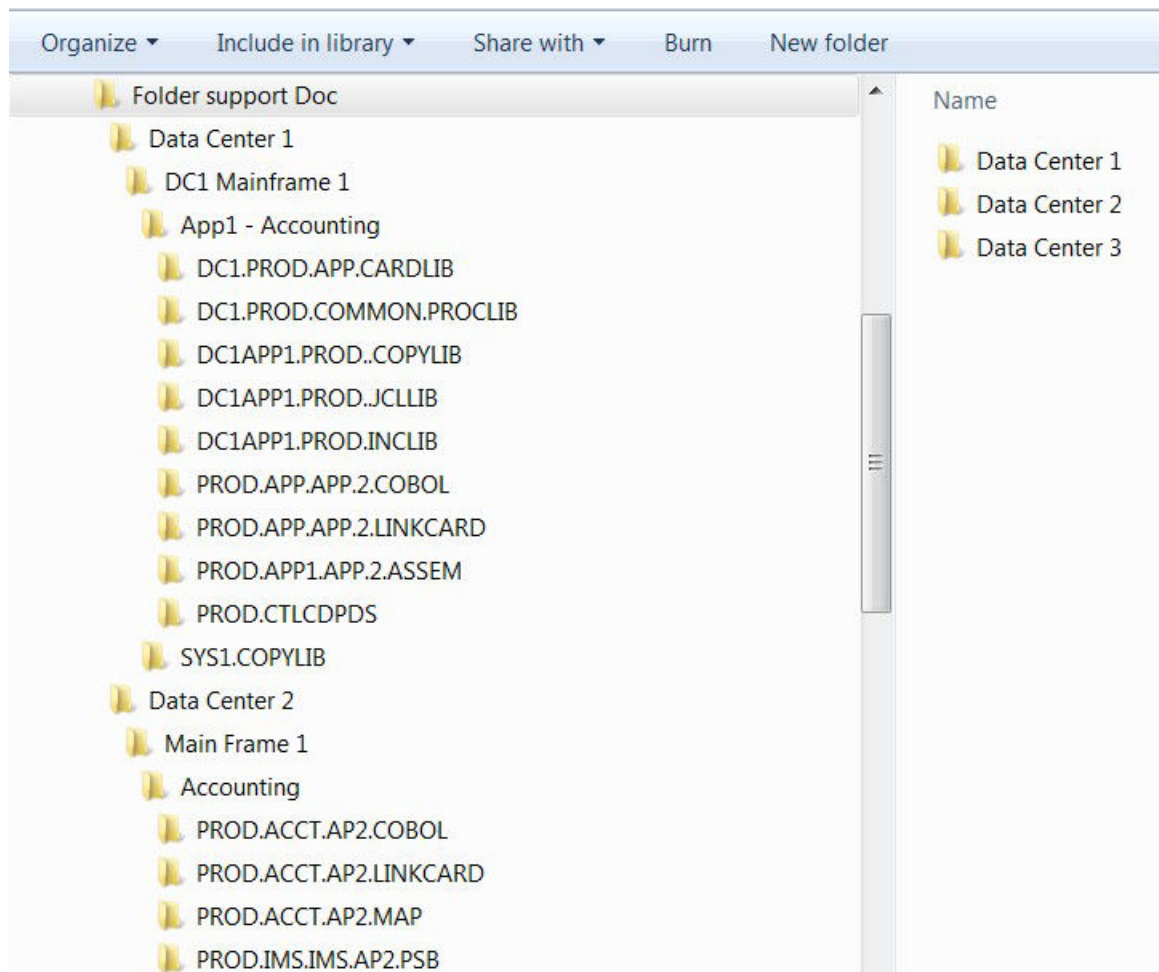
When you register your source files, it is essential that they are organized in the same folder structure as the one on your host environment. This allows CA to create the correct relationships when connecting programs with their copybooks, JCLs with the proper version of the procedures that are used, etc. Runtime connections are also established based on this structure. Furthermore, this structure allows for having several members with the same name, i.e. homonyms.

The sources to be loaded into CA should be in folders named after the PDS libraries on the Mainframe and those folders should be the ones loaded.

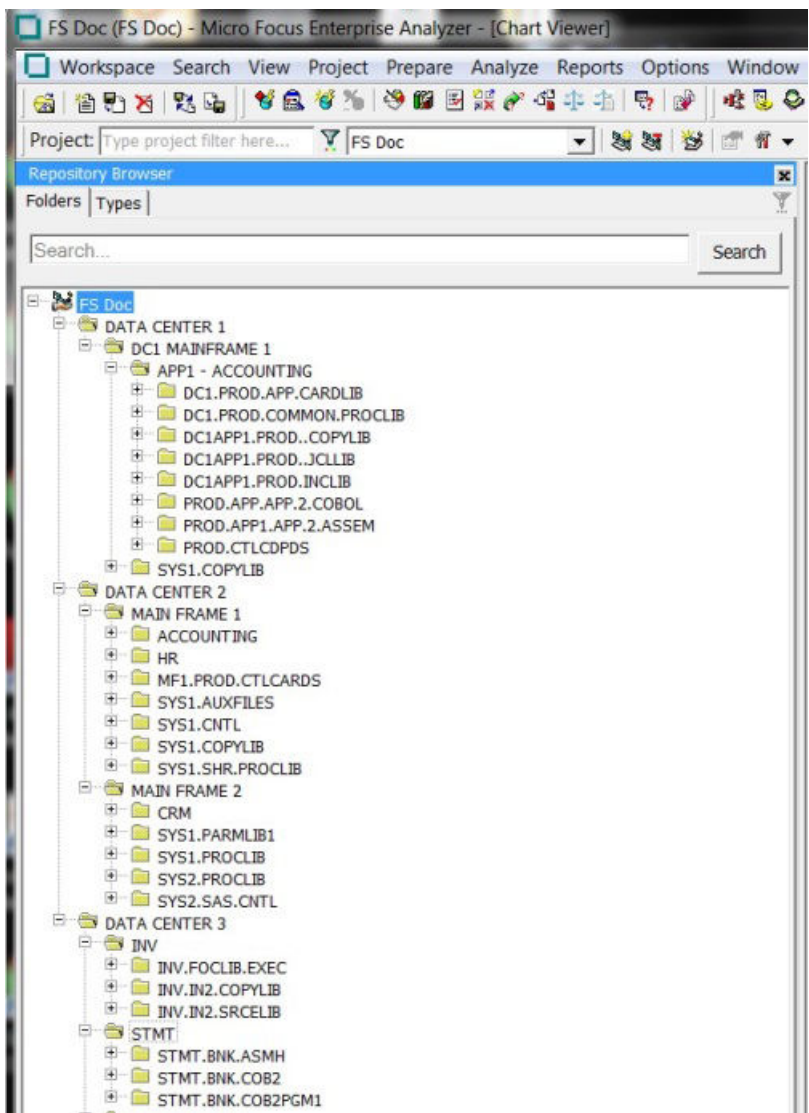
Example

There is an enterprise organization with 3 data centers: Data Center 1 with one Mainframe, Data Center 2 with two Mainframes, and Data Center 3 with one Mainframe.

The following image shows the folders in a Windows environment:



And this is what the three data center folders look like after they have been copied to the workspace:



Resolving Include Files

When trying to resolve include files (copybooks, PL/I includes, JCL procedures, and so on), the CA parser uses different search methods that you can set in the **Homonyms** tab in **Project Options**. The three methods are described below:

Search all, use the provided paths as preference

If you choose this option, the parser will look at all folders trying to find names that match the included file. If only one match is found, it will use that file. If more than one match is found, the parser will take the following steps to resolve which of the matching include files is the one to use:

1. It will check if one of the folders is listed under **Search Path** in the **Homonyms** tab in **Project Options**. The folder should be listed as the full path to the registered sources. It could be local or shared, for example:
 - local:


```
C:\Sources\Data Center 1\DC1 Mainframe 1\SYS1.COPYLIB
```
 - shared:


```
\\server\Sources\Data Center 1\DC1 Mainframe 1\SYS1.COPYLIB
```


2. It will look for proximity of the included file to the including file in the source folder structure. First, it will check the files that are in the same folder as the "including" source, then the files in the dependent folder, and then up the folder hierarchy.
3. Finally, folder names will be matched and the "include" folder that has a name with a better match to the source folder name will be selected.


Search only in the provided paths When this option is selected, only the paths specified in the Search Path list will be used as locations for looking for include files. As this option works on project level, you can configure each subsystem or application in a separate project, and then specify the copybook locations for each project. This option is useful when the number of applications/subsystem is not too big.

Search in common location and only the same subsystem This option might help when there is a large number of applications/subsystems. It lets you use the folder structure in the file system to define the boundaries between subsystems and to restrict retrieval of include files based on the physical folder structure. If you choose it, you can put all the sources in a single project and avoiding the need to add a list of folders to search in. If you choose this option, you need to specify the following two paths:

Common folder All common libraries need to be located inside the same root folder, for example, C:\Sources\Common. Files in folders located here will be "visible" to all programs being analyzed.

Subsystems root level All subsystems need to have a common root folder, for example, C:\Sources\Subsystems. If you have two subsystems in this folder, one of them will not get copybooks from the other and vice versa, but they can both get copybooks from the common libraries folder specified in the **Common folder** field.

 **Note:** The subsystems root level selection is checked first and then the common folder.

 **Note:** For COBOL only, if there is a registered homonym (copybook) in the program's folder it will be used in preference to the homonym search configuration.

Resolving Program-to-Program Calls

When resolving call relationships between programs, it is possible to reach a point where there is more than one candidate for a called program. In this case CA works the same way as with includes (see *Resolving Include Files*), but since the program objects themselves don't have folders, the folders of the source files that generated them are used.

 **Note:** This same algorithm applies to any relations between non-legacy entities (e.g. PROGRAM reads TABLE, TRANSACTION starts PROGRAM, etc.).

Resolving Control Cards

Control cards can be any members explicitly mentioned in the JCL. When parsing JCLs, CA will look for a file with the member name in the JCL in a folder with a name matching the PDS name from the mainframe.

Resolving JCL Procedures

When looking for JCL procedures, the parser uses the JCLLIB statement, if present. If there is only one copy of the searched procedure, that copy is used. Otherwise, if more than one copy is available, the libraries specified with the JCLLIB statement are used to pick the right one.

Example:

```
Path to the JCL file: <workspace_path>\Sources\APPL\MYJCL\MYJOB.JCL
//MYJOB1 JOB ...
//MYLIBS1 JCLLIB ORDER=CAMPBEL.PROCS.JCL
//S1 EXEC PROC=MYPROCL
```

In this case the procedure <workspace_path>\Sources\APPL\CAMPBEL.PROCS.JCL\MYPROCL, if present, will be used.



Refreshing Registered Source Paths

The Refresh Registered Source Paths feature enables you to change the location of registered source files. It is useful in cases when the sources have been moved to another location, or a folder that contains sources has been moved or renamed, or sources were registered from a non-shared location, and the location has since been shared.


To use this feature:

1. Open COBOL Analyzer Administration.
2. From the **Administer** menu, click **Refresh Registered Source Paths**.
3. Choose a workspace from the list in the **Select Workspace** window.
4. Click **OK**. The **Change Registered Source Paths** window opens. It displays the registered source paths and lets you select the new source paths.

You can choose whether to hide or display system paths in the list of source paths by toggling the **Hide System Paths** check box.

5. Click the yellow folder icon  under **New Source Path**.
6. Choose the new directory of the source files and click **Select Folder**. The selected path and a red cross icon  appear under **New Source Path**.



Note: Clicking  removes the selected path.

7. Click **Change**. A message asking you if you are sure you want to change the registered source paths comes up.
8. Click **OK**. You are notified that the paths have been refreshed.
9. Click **OK**. The Change Registered Source Paths window closes.



Note: Alternatively, you can change the registered source paths from Source Synchronization. See *Source Synchronization* for more information.

After changing the registered source paths, a file must be verified again for a Logic Analyzer extraction to work correctly.

Setting Registration Options: Extensions Tab

Source files must have recognized DOS file extensions before they can be registered. Files without extensions are checked for content, and if the content is recognized, the files are registered with appropriate extensions appended.

Files with unknown extensions are flagged, provided that you uncheck **Ignore Unknown and Overloaded Extensions** on the Extensions tab of the Workspace Registration options window. If a file fails to register because its extension is unknown, simply add the extension to the list of recognized extensions on the Extensions tab and register the file again.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the Registration tab, then the Extensions tab.

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 Extensions pane. Select each extension you want the system to recognize. Add extensions as necessary.



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. Check **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, check **Remove Sequence Numbers** if you want the system to replace preceding enumeration characters, or *sequence numbers*, with blanks. Sequence numbers are removed only from the source file versions maintained by the product.

Setting Registration Options: Source Files Tab

If your 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 COBOL Analyzer convert it for you, using the options on the Registration > Source Files tab of the Workspace Options window.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the Registration tab, then the Source Files tab.
2. In the Legacy Source Encoding group box, choose:
 - **Workstation** if the source is workstation-encoded. For DBCS configurations, if Japanese-language source files were downloaded in workstation (text) mode, specify how DBCS escape control characters were handled.
 - **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 COBOL Analyzer are converted.
3. In the Object System Encoding group box, choose:
 - **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 (DBCS configurations only).
 - **Japanese (EBCDIC-CCSID-939, 5035)** if the original source was Japanese EBCDIC-encoded, CCSID-939, 5035 (DBCS configurations only).

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



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

4. Check **Strip trailing numeration** 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 CA.
5. Check **Expand tabulation symbols** 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 CA. You must select this option if you want to view Interactive Analysis information for C or C++ programs.
6. Check **Do not invalidate dependent files** if you do not want the system to invalidate files that are dependent on a file you are refreshing. Normally, the system invalidates all dependent files. Selecting this option shifts the burden of determining which files need to be invalidated (and reverified) onto the

user. If you select this option and know that a file is affected by a change in the file being refreshed, you must reverify the affected file.


Creating New Source Files

To create a new source file, select the project for the source file in the Repository Browser and choose **File > New**. A dialog box opens, where you can specify the file name (with extension) and source file type. To create a new source file with the same content as an existing file, select the file and choose **File > Save As**. The system automatically registers the created files and stores them in the appropriate folder.

Updating Source Files with Source Synchronization

Source Synchronization checks if the source files registered in COBOL Analyzer have been modified or deleted from their original location. It also detects if new files or folders have been added to a folder registered in CA.

To use Source Synchronization, do one of the following:

- Select **Workspace > Source Synchronization**.
- Click .

If there are any new, modified or deleted files or folders, Source Synchronization informs you about that and asks you if you want to synchronize the workspace. To view the updates, click **Show Changes**. Click **Close** to go back to the Source Synchronization window.

If you click **Yes**, the new files will be added to the workspace and verified, the modified files be updated and re-verified, and the deleted files will be removed.

Sources that were registered in-place but are not found in the expected locations are not automatically deleted from the repository during Source Synchronization. If files have been deleted from the original location, you can view them if you click **Missing files** button on the Source Synchronization confirmation dialog. You can then decide if you need to restore the missing files to the expected locations, or you can manually select which of the missing files to be removed from the repository.

For in-place registration workspaces, source files removed from registered source directories can be deleted from the workspace during BRP. If you check the **Automatically unregister missing sources when running update** option in the BRP configuration **Advanced** tab, the missing sources will be deleted from the workspace when running BRP. The deleted files are logged in the `Update.log`. If the option is unchecked, the removed sources will not be deleted. This option is unchecked by default.


Source Synchronization Settings

To set Source Synchronization options, go to **Options > Workspace Options > Registration > Source Synchronization**.

The first option allows you to enable or disable Source Synchronization. The default is enabled. Disabling Source Synchronization stops any related checks and automated executions (if any were set). On a non-in-place workspace, Source Synchronization is automatically disabled if BRP registration is run. BRP on in-place registration workspace still uses Source Synchronization for updating, ignoring whether Source Synchronization is enabled or not.


In this tab you can select to check for updated sources automatically. If you do so, you must select whether to have the checks performed periodically or at a specific time. If you want to have periodic checks, you must select an interval. The default is 20 hours. When this option is selected, the check is added as a task to the Windows Task Scheduler.

Another option in this tab is to select to execute a batch file on the files in the original source location before the workspace is synchronized.

If you select **Synchronize the workspace after checking for outdated sources without asking me**, every time you click  or select **Workspace > Source Synchronization** the workspace will be automatically synchronized.

Source Synchronization and Interactive Analysis

If you select a file that has been modified in the original location and try to run Impact Analysis, you will be notified that it has been changed and will be asked to synchronize the workspace.

 **Note:** If you have selected the **Synchronize the workspace after checking for outdated sources without asking me** option in **Options > Workspace Options > Registration > Source Synchronization**, the workspace will be synchronized when you try to run Impact Analysis on a modified file.

The Source Synchronization window also pops up when you try to select a file that has been modified in the Objects pane in Interactive Analysis.

Using Source Synchronization to Change the Registered Source Paths


When the folder containing the registered source files has been changed or renamed, or the files have been moved to a different location, or sources were registered from a non-shared location, and the location has since been shared, you can refresh the registered source paths from the Source Synchronization window.

In these cases, a **Change Registered Source Paths** appears in the **Source Synchronization** window.

To refresh the registered source paths from the **Source Synchronization** window, perform the following steps:

1. Click **Workspace > Source Synchronization**.


2. Select the Removed folders / subfolders tab to see the list of folders that could not be found.



 **Note:** Only top-level folders can be refreshed. This means that if the location of a top-level folder has been changed, you cannot choose to update only one or more of its subfolders.

3. Click **Change Registered Source Paths**. The **Change Registered Source Paths** window opens. This displays the registered source paths and lets you select the new source paths.


You can choose whether to hide or display system paths in the list of source paths by toggling the **Hide System Paths** check box.

4. Click the yellow folder icon  under **New Source Path**.

5. Choose the new directory of the source files and click **Select Folder**. The selected path and a red cross icon  appear under **New Source Path**.

 **Note:** Clicking  removes the selected path.


6. Click **Change**. A message asking you if you are sure you want to change the registered source paths comes up.

 **Note:** If you click **Change** without selecting a new source path before that, a message that there is nothing to change will pop up.

7. Click **OK**. You are notified that the paths have been refreshed.

8. Click **OK**. You are notified that all files are up-to-date.

9. Click **OK**. Source Synchronization closes.

 **Note:** Alternatively, you can change the registered source paths from the COBOL Analyzer Administration tool. See *Refreshing Registered Source Paths* for more information.

After changing the registered source paths, a file must be verified again for a Logic Analyzer extraction to work correctly.

Exporting Source Files from a Workspace


To export the workspace source for a project or file to a new location, select the project or file in the Repository Browser and click **File > Export Sources**. A dialog box opens, where you can specify the location.

Deleting Objects from a Workspace

To delete an object from a workspace, select it and choose **File > Delete from Workspace**. To delete a folder and all its contents from a workspace, select it and choose **File > Delete Contents from Workspace**.

Deleting a Workspace

To delete a workspace, choose **Administer > Delete Workspace...** in the COBOL Analyzer Administration tool. A Delete workspace dialog opens, where you can select the workspace you want to delete.

 **Note:** Only a master user can delete a workspace in a multiuser environment.

Japanese Language Support

COBOL Analyzer provides full support for mainframe-based Cobol or PL/I Japanese-language applications. Make sure you set Windows system and user locales to Japanese before registering source files.

You can register Japanese source files downloaded in text or binary mode:

- Source files downloaded in text (workstation) mode must be in Shift-JIS encoding. If Shift-Out and Shift-In delimiters were replaced with spaces or removed during downloading, COBOL Analyzer restores them at registration.
- Source files downloaded in binary (mainframe) mode are recoded by COBOL Analyzer from EBCDIC to Shift-JIS encoding at registration.

Use the options on the Registration > Source Files tab of the Workspace Options window to specify how DBCS escape control characters were handled in source file downloaded in text mode:

- **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.

We recommend you preserve delimiters during download. Replacing delimiters with spaces during download generally yields better restoration results than removing them.

In all COBOL Analyzer 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.

Setting Up Projects

Workspace *projects* typically represent different portions of the application modeled in the workspace. You might have a project for the batch portion of the application and another project for the online portion. You can also use a project to collect items for discrete tasks: all the source files affected by a change request, for example.

Creating Projects


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

1. Choose **Project > New** project. The Create Project dialog opens.
2. Enter the name of the new project and click **OK**. The new project is displayed in the Repository Browser. 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. A *private* project is not. 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 **Project > Toggle Sharing**. Choose **Project > 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 add source files, include or exclude objects.



Note: Only a master user or the owner of a shared unprotected project can delete it.

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 **Project > Toggle Protection**. Choose **Project > 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 into Projects

Copy the contents of a project, folder, or file to a different project by selecting it and dragging and dropping the selection onto the project, or by using the **Edit** menu choices to copy and paste the selection. Use the **Project** menu choices described below to move selections, or to include referenced or referencing objects in a move or copy.



Note: In other CA tools, use the right-click menu or the **File** menu to include files in projects.

1. In the Repository Browser, select the project, folder, or file you want to move or copy, then choose **Project > Copy Project Contents** (if you selected a project) or **Project > Include into Project** (if you selected a folder or file). The Select Project window opens.
2. In the Select Project window, select the target project. Click **New** to create a new project.
3. Select either:
 - **Include All Referenced Objects** if you want to include objects referenced by the selected object (the Cobol copybooks included in a Cobol program file, for example).
 - **Include All Referencing Objects** if you want to include objects that reference the selected object.



Note: This feature is available only for verified files.

4. Select:
 - **Copy** to copy the selection to the target project.
 - **Move From Current Project** to move the selection to the target project.
 - **Move From All Projects** to move the selection from all projects to the target project.
5. Click **OK** to move or copy the selection.

Including Referenced and Referencing Objects in a Project

After verifying a project, you can ensure a closed system by including referenced or referencing objects in the project. The objects to be included must also have been verified, so it's best to verify every project in the workspace before including objects.

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.

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 **Project > Include All Referenced Objects**.
- Every object that references the objects in the project (including indirectly referencing objects), select the project in the Repository Browser and choose **Project > Include All Referencing Objects**.
- Every object that directly references the objects in the project, select the project in the Repository Browser and choose **Project > Include Directly Referencing Objects**.

Removing Unused Support Objects from a Project

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 **Project > Compact Project**.

Emptying a Project

To empty a project (without deleting the project or its contents from the workspace), select the project and choose **Project > Empty Project Contents**.

Deleting a Project

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



Note: Only the owner of a project can delete it.

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.

You can verify a single file, a group of files, all the files in a folder, or all the files in a project. It's usually best to verify an entire project. COBOL Analyzer parses the files in appropriate order, taking account of likely dependencies between file types.



Note: You need not verify copybooks. Copybooks are parsed when the including source file is verified.


Workspace and Project Verification options determine verification behavior. The default values for these options are preset based on your configuration and should be appropriate for most installations.

1. In the Repository Browser, select the project, folder, or files you want to verify and choose **Prepare > Verify**.
2. 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.

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.


Enabling Parallel Verification

Parallel verification typically improves verification performance for very large workspaces by using multiple execution agents, called *Queue Processor*, to process source files concurrently. You can start any number of Queue Processors on the local machine, remote machines, or some combination of local and remote machines. You can run parallel verification online in the COBOL Analyzer or in batch mode with the Batch Refresh Process (BRP).


 **Important:** When you run parallel verification on more than one machine, you need to make sure that workspace and project verification options are set identically on each machine. The easiest way to do this is to log in as the same Windows user on each machine. Alternatively, you can define a default option set that is automatically assigned to every user in the environment who has not explicitly defined a custom option set. See the related topics for more information on option sets.


You enable parallel verification in three steps:

- Select the parallel verification method and the minimum number of concurrent Queue Processors on the **Verification > Parallel Verification** tab of the Workspace Options.
- Start the Queue Processors on the local and/or remote machines. If you start fewer than the minimum number of Queue Processors specified on the Parallel Verification tab, the verification process starts the needed Queue Processors automatically on the local machine.
- Verify the workspace online in the COBOL Analyzer or in batch mode using the Batch Refresh Process (BRP).

 **Note:** Verification results are reported in the Activity Log History window. They are not reported in the Activity Log itself (for online verification) or BRP log files (for batch verification). You can also use a Verification Report to view the results.

Follow the instructions below to launch Queue Processors and to specify the type of work they perform. You can launch multiple Queue Processors on the same machine. Once the minimum number of Queue Processors has been started, you can launch them at any point in the verification process.

1. In the COBOL Analyzer Administration window, choose **Administer > Launch Queue Processor**. The Launch Queue Processor window opens.
2. In the **Serve workspace** combo box, specify the workspace to be processed.
3. In the Processing Mode pane, select any combination of:
 - **Conversion** to perform operations used to generate a Interactive Analysis construct model.
 - **Verification** to perform verification operations.
 - **Code Search Query** to perform Code Search searches in offline mode.
4. Select **Produce Log File** to generate a log file for parallel verification. The log file has a name of the form `<workspace_name>HCC.<random_number>.log` and is stored at the same level as the workspace (.rwp) file.
5. Click **OK**. The product launches the Queue Processor. Click the  button on the Windows toolbar to view the Queue Processor window.

 **Note:** Once verification has started, you can change the processing mode for a Queue Processors by selecting the appropriate choice in the **Processing** menu in the Queue Processor window.

Invalidating Files Before Reverification


Invalidating a source file restores it to its state before it was verified, deleting the object model the parser created for the file. COBOL Analyzer automatically invalidates previously verified files when you reverify them.


When reverifying sources in the repository, all selected objects are invalidated prior to being verified. When the reverification is cancelled, all selected objects that were not verified, are invalidated.


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:

- The entire workspace in the COBOL Analyzer Administration tool. Choose **Administer > Invalidate Workspace**. The Select workspace dialog opens, where you can select the workspace you want to invalidate.
- Selected projects, folders, or files in the COBOL Analyzer Repository pane. Select the items you want to invalidate and choose **File > Invalidate Selected Objects**.

Invalidated files are displayed in **bold** type in the Repository pane.

 **Note:** To improve performance when reverifying objects in the repository, all selected objects will be invalidated prior to being verified. When the reverification is cancelled, all selected objects that were not verified will be invalidated.

 **Note:** To prevent the database transaction log from filling up during the invalidation of a large number of files, commits to the database are made regularly rather than waiting until the completion of the invalidation process.



 **Note:** Invalidating an entire workspace is significantly faster than invalidating projects, folders, or files. You must be a master user to invalidate source files.

The invalidate workspace process opens several console windows displaying the progress of component tasks. Close these windows manually when invalidation is complete.

Setting Workspace Verification Options: Legacy Dialects Tab

Use the **Verification > Legacy Dialects** tab of the Workspace Options window to identify the dialect of the source files in your application.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the Verification tab, then the Legacy Dialects tab.
2. In the Source Type pane, select the source file type whose dialect you want to specify, then select the dialect in the dialect pane.
3. Set verification options for the dialect. The table below shows the available options.

Option	Dialect	Description
Binary Storage Mode	ACUCOBOL-GT®, Micro Focus COBOL	Specifies the binary storage mode: Word (2, 4, or 8 bytes) or Byte (1 to 8 bytes).
Comment out unsupported statements	Micro Focus COBOL, .NET/JVM COBOL	Excludes directives and unsupported statements which are set implicitly in the source files and/or the directive files used in source files.  Note: XMLGEN and any other directive affecting the <code>kenlygen.xml</code> generation set explicitly in the CA Options will not be excluded and will lead to failures in verification.
Compiler Directives	All COBOL	Specifies directives to be used during compilation. To add a directive, right-click in the Compiler Directives field and select Add from the context menu. The format is: <code><directive>=<value></code> . Using the context menu you can also delete, edit, select, unselect, move up/down the directives.  Note: Project-level compiler directives override workspace-level directives.

Option	Dialect	Description
Copybook Directories	Micro Focus COBOL, .NET/JVM COBOL	Specifies copybook directories that contain system copybooks, frameworks and other copybooks that are not part of the source code but are used by it.
COPY REPLACING Substitutes Partial Words	All COBOL	Specifies that the application was compiled with partial-word substitution enabled for COPY REPLACE operations.
COPY statements as in COBOL-68	All COBOL	Specifies that the application was compiled with the OLDCOPY option set.
Currency Sign	All COBOL	Specifies the national language currency symbol.
Data File Assignment	Micro Focus COBOL	Specifies the setting of the compiler ASSIGN option: Dynamic or External.
Enable MF comments	Micro Focus COBOL	Specifies that the application contains comments in the first position.
Environment	Micro Focus COBOL, .NET/JVM COBOL	Specifies environment variables.
Graphical System	ACUCOBOL-GT®	Specifies that the application was executed on a graphical rather than character-based system.
In Margins	All	Specifies the current margins for source files.
Micro Focus compiler compatibility level	Micro Focus COBOL, .NET/JVM COBOL	Enables forward compatibility with Micro Focus COBOL systems by selectively enabling Micro Focus-specific reserved words and changing the behavior of certain features to be compatible with particular versions.
PERFORM behavior	ACUCOBOL-GT®, Micro Focus COBOL	Specifies the setting of the compiler PERFORM-type option: Stack, to allow recursive PERFORMS, or All exits active, to disallow them.
Preprocessors	Micro Focus COBOL, .NET/JVM COBOL	Specifies the preprocessors' order of execution. To change the order of execution, right-click in the Compiler Directives field and select Move Up or Move Down from the context menu. Using the context menu you can also add, delete, edit, select or unselect the preprocessors.
Preserve dialect for verified objects	All COBOL	Specifies that the parser reverify COBOL files with the same dialect it used when the files were verified first.
RM/COBOL compatibility	ACUCOBOL-GT®	Specifies that the parser ensure proper memory allocation for applications written for Liant RM/COBOL. Emulates behavior of -Ds compatibility option.
Source Format	ACUCOBOL-GT®	Specifies the ACUCOBOL-GT® source format: ANSI, Long or Terminal.
SQL Dialect	All SQL	Specifies the SQL dialect. The options are: Standard SQL , DB2 for AS/400 , Unisys RDMS . For ANSI SQL-92 and DB2 for OS/390 use Standard SQL .
Support Hogan Framework	All COBOL	Specifies that the parser create relationships for Hogan COBOL and Hogan batch applications. Specify the full path of the folder for Hogan

Option	Dialect	Description
		Framework configuration files in the Hogan Files Location field. For more information, see "Hogan Framework Support."
Treat COMP-1/COMP-2 as FLOAT/DOUBLE	ACUCOBOL-GT®	Specifies that the parser treat picture data types with COMP-1 or COMP-2 attributes as FLOAT or DOUBLE, respectively.

Setting Workspace Verification Options: Settings Tab


Use the **Verification > Settings** tab of the Workspace Options window to specify verification behavior for the workspace. Among other tasks, you can:



- Enable *staged parsing*, which may improve verification performance by letting you control which verification stages the parser performs.
 - Enable *relaxed parsing*, which lets you verify source despite errors.
1. Click **Options > Workspace Options**. The Workspace Options window opens. Click the Verification tab, then the Settings tab.
 2. In the Source Type pane, select the source file type whose verification options you want to specify.
 3. Set verification options for the source file type. The table below shows the available options.



Note: Click the **More** or **Details** button if an option listed below does not appear on the Settings tab.

Option	Source File	Description
Allow Implicit Instream Data	JCL	Specifies that a DD * statement be inserted before implicit instream data if the statement was omitted from JCL.
Allow Keywords to Be Used as Identifiers	COBOL, Copybook	Enables the parser to recognize COBOL keywords used as identifiers.
At Sign	System Definition File	Specifies the national language character for the at symbol.
C/C++ Parser Parameters	C, C++	Specifies the parameters used to compile the application. You can also specify these parameters in the Project Verification options, in which case the project parameters are used for verification.
Create Alternative Entry Point	COBOL	Specifies that an additional entry point be created with a name based on the conversion pattern you enter in the Conversion Pattern field. Supports systems in which load module names differ from program IDs. For assistance, contact support services.
Cross-reference Report	TWS Schedule	Specifies the full path of the TWS cross-reference report (.xrf).
Currency Sign	System Definition File	Specifies the national language character for the currency symbol.
Debugging Lines	COBOL	Specifies parsing of debugging lines: Off, to parse lines as comments; On, to parse lines as normal statements; Auto, to parse lines based on the program debugging mode.

Option	Source File	Description
Detect Potential Code Anomalies	COBOL	Enables generation of Interactive Analysis information on potential code anomalies.
Enable Interactive Analysis	COBOL, Natural, PL/I, RPG, JCL	Enables generation of Interactive Analysis information.
Enable Quoted SQL Identifiers	Assembler File, COBOL, DDL, PL/I	Enables the parser to recognize quoted SQL identifiers. Strings delimited by double quotation marks are interpreted as object identifiers.
Enable Reference Reports	COBOL, Control Language, ECL, JCL, Natural, PL/I, RPG, W.L.	Enables generation of complete repository information for logical objects.
Enter classpath to JAR Files and/or path to external Java file root directories	Java	<p>Specifies any combination of:</p> <ul style="list-style-type: none"> JAR files containing class libraries referenced in the Java application, or Zip files containing external Java files referenced in the application. Alternatively, specify the path of the folder for the JAR or Zip files, then select Include Jar/Zip Files From Directories. Paths of the root folders for external Java files referenced in the application. <p> Note: This option is checked after the identical option in the project verification options. Setting the project verification option effectively overrides the setting here.</p>
Extralingual Characters	System Definition File	Adds the specified lower-case national language characters to the supported character set. Do not separate characters with a space.
Extralingual Upper Characters	System Definition File	Adds the specified upper-case national language characters to the supported character set. Do not separate characters with a space.
Generate program entry points for functions with same name as file	C	Specifies that a program entry point be created for the function that has the same name as the file. Typically used to trace calls to C programs from COBOL, PL/I, Natural, RPG, or Assembler programs.
Generate program entry points for main functions	C	Specifies that a program entry point be created for functions named "main". Typically used to trace calls to C programs from COBOL, PL/I, Natural, RPG, or Assembler programs.
Ignore Duplicate Entry Points	All	Enables the parser to recognize 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. To use this option, you must select Enable Reference Reports. You cannot use this option to verify multiple programs with the same program ID.
Ignore Text After Column 72	DDL	Enables the parser to ignore trailing enumeration characters (columns 73 through 80) in source lines.

Option	Source File	Description
Imply CMPAT=Yes	PSB	Specifies that <code>CMPAT=Yes</code> is assumed even if it hasn't been set in <code>PSBGEN</code> . When <code>CMPAT=Yes</code> is specified, it instructs IMS to generate an I/O PCB for all uses, even in environments where it is not used, such as batch or CICS. If you use this option, you won't need to recompile the program between batch and online executions.
Import instream data from DSN	JCL	Imports data sets in HyperCode: <ul style="list-style-type: none"> • Instream data sets. • Control card patterns.  Note: This option can increase the verification time of JCL and also increase the size of the database.
JCL Variable Definitions	JCL	Use this to set global symbol definitions used during JCL parsing. This works in the same way as the <code>SystemSymbols</code> element in the JCL section of <code>Legacy.xml</code> file. You can specify a <code>Name=Value</code> pair for each line. To add a new entry right-click in the JCL Variable Definitions field, and then click Add . Only lines with a checked box will be used in verification.  Note: Do not specify the "&" character used in a symbol reference. For example, use <code>PRG</code> not <code>&PRG</code> as the name. Delimiters are not required around the values; all text after the "=", including any spaces, are treated as the value. In JCL files, a reference to <code>&Name</code> will be substituted with the available <code>Value</code> , unless there is a more local definition of that symbol name in the context of that reference. For example, a PROC statement might have a symbol definition that redefines the value of that symbol within the scope of that procedure. There is no special processing done based on the type of <code>Value</code> - values with spaces or digits are substituted in for the reference as they are specified in the option.
Libraries support	Natural	Enables Natural library support. For more information, see "Natural Support" in the online help.
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. Add folders as necessary. You can also specify these folders in the Project Verification options, in which case the parser looks only for the folders for the project.
Number Sign	System Definition File	Specifies the national language character for the number symbol.
Perform Dead Code Analysis	COBOL, PL/I, RPG	Enables collection of dead code statistics.

Option	Source File	Description
Perform DSN Calling Chains Analysis	Control Language, ECL, JCL, WFL	Enables analysis of dataset calling chains.
Perform System Calls Analysis	JCL	Enables analysis of system program input data to determine the application program started in a job step.
Relaxed Parsing	AS400 Screen, BMS, COBOL, Copybook, CSD, DDL, Device Description, DPS, ECL, MFS, Natural, Netron Specification Frame, PL/I	Enables relaxed parsing.
Relaxed Parsing for Embedded Statements	Assembler File, COBOL, PL/I	Enables relaxed parsing for embedded SQL, CICS, or DLI statements.
Resolve Decisions Automatically	Control Language, WFL	Enables automatic decision resolution.
Show Macro Generation	C, C++	Specifies whether to display statements that derive from macro processing in Interactive Analysis.
Sort Program Aliases	JCL	Enables batch sort card analysis. Choose a recognized sort utility in the Sort Program Aliases pane. Add sort utilities as necessary.
SQL Statements Processor	COBOL	Specifies whether the SQL Preprocessor or Coprocessor was used to process embedded SQL statements.
System Procedures	JCL	Specifies the system procedures referenced by JCL files. Add system procedures as necessary.
Timeout in seconds to stop verification execution	All	The number of seconds to wait before stopping a stalled verification process.
Treat every file with main procedure as a program	C, C++	Specifies whether to treat only files with main functions as programs.
Trim Date from Active Schedule Names	TWS Schedule	Specifies whether to append the effective date range to a TWS jobstream object.
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.
Use Database Schema	Assembler File, 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.
Workstation Report	TWS Schedule	Specifies the full path of the TWS workstation report (.wdr).

Enabling Staged Parsing

File verification generates repository information in four stages, as described in this section. You can control which stage the CA parser performs by setting the staged parsing options on the Settings tab for Workspace Verification options. 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 at each point. 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.

Basic Repository Information

To generate basic repository information only, deselect **Enable Interactive Analysis**, **Enable Reference Reports**, and **Perform Dead Code Analysis** on the Workspace Verification options Settings tab. 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** on the Workspace Verification options Settings tab. Set this option to generate all relationships between logical objects, and to enable non-Interactive Analysis tools, including Reference Reports and Orphan Analysis.



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

Interactive Analysis Information

To generate an Interactive Analysis construct model, select **Enable Interactive Analysis** on the Workspace Verification options Settings tab. An Interactive Analysis 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 Interactive Analysis information on potential code anomalies, select **Detect Potential Code Anomalies** on the Workspace Verification options Settings tab.



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

Dead Code Statistics

To generate dead code statistics, and to set the Dead attribute to True for dead constructs in Interactive Analysis, select **Perform Dead Code Analysis** on the Workspace Verification options > Settings tab. 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.

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). Select **Relaxed Parsing** or **Relaxed Parsing for Embedded Statements** as appropriate on the Workspace Verification options Settings tab.



Note: Relaxed parsing may affect the behavior of other tools. You cannot generate component code, for example, from source files verified with the relaxed parsing option.

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. Select **Truncate Names of Absolute Elements** on the Workspace Verification options Settings tab for the ECL file.

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 Workspace Verification Options: Parallel Verification Tab

Use the **Verification > Parallel Verification** tab of the Workspace Options window to enable online or batch parallel verification and to specify the minimum number of Queue Processors CA should expect.



Important: When you run parallel verification on more than one machine, you need to make sure that workspace and project verification options are set identically on each machine. The easiest way to do this is to log in as the same Windows user on each machine. Alternatively, you can define a default option set that is automatically assigned to every user in the environment who has not explicitly defined a custom option set. See the related topics for more information on option sets.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the Verification tab, then the Parallel Verification tab.
2. Select:
 - **Run Parallel Verification in the Online Tool** to enable parallel verification online. In the **Minimum Queue Processors** combo box, specify the minimum number of concurrent Queue Processors the product should expect.
 - **Run Parallel Verification in BRP** to enable parallel verification in the Batch Refresh Process (BRP) tool. In the **Minimum Queue Processors** combo box, specify the minimum number of concurrent Queue Processors CA should expect.



Note: Before running verification, start the necessary Queue Processors on the local and/or remote machines. If you start fewer than the minimum number of Queue Processors, the verification process starts the required Queue Processors automatically on the local machine.

Setting Project Verification Options

Use the Verification tab of the Project Options window to specify verification behavior for the selected project. Among other tasks, you can:


- Specify how the parser treats environment-related code.
- Specify the conditional constants the parser uses to compile programs in the project.
- Specify schedule IDs for CA-7 jobs triggered by datasets.
- Optimize verification for advanced program analysis.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Verification** tab.

2. In the Source Type pane, select the source file type whose verification options you want to specify.
3. Set verification options for the source file type. The table below shows the available options.



Note: Click the **Environments**, **CopyLibs**, **Advanced**, or **Cobol Dialect** button if an option listed below does not appear on the Verification tab.

Option	Source File	Description
AIM/DB Environment	COBOL	Specifies how the parser treats AIM/DB-environment-related code or its absence.
CICS Environment	COBOL	Specifies how the parser treats CICS-environment-related code or its absence.
Compiler Directives	COBOL	<p>Specifies directives to be used during compilation. To add a directive, select Use project-specific compiler directives, then right-click in the Compiler Directives field and select Add from the context menu. The format is: <code><directive>=<value></code>.</p> <p>Using the context menu you can also delete, edit, select, unselect, move up/down the directives.</p> <p> Note: Project-level compiler directives override workspace-level directives.</p>
Context-Sensitive Value Analysis	COBOL	Specifies that the parser perform context-sensitive automatic decision resolution for Unisys MCP COMS analysis. Choosing this option may degrade verification performance.
Dialect Specific Options	COBOL	Specifies dialect-specific options, including the conditional constants the parser uses to compile programs in the project. Select the Cobol dialect, then choose the constant in the Macro Settings pane. Add constants as necessary.
DMS Environment	COBOL	Specifies how the parser treats DMS-environment-related code or its absence.
DMSII Environment	COBOL	Specifies how the parser treats DMSII-environment-related code or its absence.
DPS routines may end with error	COBOL	Specifies that the parser perform call analysis of Unisys 2200 DPS routines that end in an error. Error-handling code for these routines is either analyzed or treated as dead code.
Enable Data Element Flow	COBOL	Enables the Global Data Flow, Change Analyzer, and impact trace tools.
Enable Execution Flow	COBOL	Enables the Execution Path tool.
Enable Impact Report	COBOL	Enables the impact trace tools. You must also set Enable Data Element Flow to perform impact analysis.
IDMS Environment	COBOL	Specifies how the parser treats IDMS-environment-related code or its absence.
IMS Environment	COBOL	Specifies how the parser treats IMS-environment-related code or its absence.
Maximum Number of Variable's Values	COBOL	Specifies the maximum number of values to be calculated for each variable during verification for advanced program analysis. Limit is 200.

Option	Source File	Description
Maximum Size of Variable to Be Calculated	COBOL	Specifies the maximum size in bytes for each variable value to be calculated during verification for advanced program analysis.
Override CICS Program Terminations	COBOL	Specifies that the parser interpret CICS RETURN, XCTL, and ABEND commands as not terminating program execution. Error-handling code after these statements is either analyzed or treated as dead code.
Perform COMS Analysis	COBOL	Specifies that the parser define relationships for Unisys MCP COMS SEND statements.
Perform Generic API Analysis	COBOL	Specifies that the parser define relationships with objects passed as parameters in calls to unsupported program interfaces, in addition to relationships with the called programs themselves.
Perform Program Analysis	COBOL	Enables program analysis and component extraction features.
Perform Unisys Common-Storage Analysis	COBOL	Specifies that the parser include in the analysis for Unisys COBOL files variables that are not explicitly declared in CALL statements, but that participate in interprogram communications. You must set this option to include Unisys COBOL common storage variables in impact traces and global data flow diagrams.
Perform Unisys TIP and DPS Calls Analysis	COBOL	Specifies that the parser perform Unisys 2200 TIP and DPS call analysis.
Report Writer Environment	COBOL	Specifies how the parser treats Report Writer-environment-related code or its absence.
Resolve Decisions Automatically	COBOL	Specifies that the parser autoresolve decisions after successfully verifying files.
SQL Environment	COBOL	Specifies how the parser treats SQL-environment-related code or its absence.
Support CICS HANDLE statements	COBOL	Specifies that the parser detect dependencies between CICS statements and related error-handling statements.
Use overwritten VALUEs	COBOL	Specifies that the parser use constants from VALUE clauses as known values even if they are overwritten in the program by unknown values.
Use VALUEs from Linkage Section	COBOL	Specifies that advanced analysis tools not ignore parameter values in the Linkage Section.

Specifying the Processing Environment

The COBOL Analyzer parser autodetects the environment in which a file is intended to execute, 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.
- Adds the phrase Procedure Division using DFHEIBLK, DFHCOMMAREA to the internal parse tree.

Autodetection is not always appropriate. 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 example. For each autodetected environment on the Project Verification options tab, 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.

Specifying Conditional Compiler Constants

Compiler constant directives let you compile programs conditionally. Specify the conditional constants the parser uses to compile programs in the project in the **Dialect Specific Options** for your dialect on the Project Verification options tab. For Micro Focus COBOL, two formats are supported:

- *constant_name=value* (where no space is allowed around the equals sign). In the following example, if you specify WHERE=PC on the Project Verification options tab, the source that follows the \$if clause is compiled:

```
$if WHERE="PC"
    evaluate test-field
        when 5 perform test-a
    end-evaluate
```

- *constant_name*. In the following example, if you specify NOMF on the Project Verification options tab, the source that follows the \$if clause is compiled:

```
$if NOMF set
    $display Not MF dialect
    go to test-a test-b depending on test-field
$end
```

Optimizing Verification for Advanced Program Analysis

When you enable advanced program analysis options for COBOL projects, the parser calculates constant values for variables at every node in the Interactive Analysis parse tree. For this reason 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 the **Maximum Number of Variable's Values** and **Maximum Size of Variable to Be Calculated** options in the **Project Verification Options** tab. 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 re-verify the application.

Identifying System Programs

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

The most convenient way to identify the system programs your 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 in the System Programs tab of the Workspace Options window and reverify any one of their referencing source files.



Note: The reference report tool lets you bring up the System Programs tab of the Workspace Options window while you are in the tool itself. Choose **View > System Programs** in the reference report window to display the tab.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the 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 patterns as necessary.

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*)
```

You can resolve the decision objects to the appropriate resources in the Decision Resolution tool.

1. Choose **Options > 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, 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, the method type signature would be `int f1f(char*)`. Add signatures as necessary. 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 (VBA).



Note: Keep in mind that the signatures of C or C++ functions can contain an asterisk (*) character, as in the example. 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.

4. Select the project, folder, or source files for the application and choose:
 - **Prepare > Verify** if the source files have not been verified.
 - **Prepare > Apply Boundary Decisions** if the source files have been verified, but boundary decisions have not been specified for the application, or the specification has changed.

A decision object is added to the tree for the source files in the Repository Browser.

Performing Post-Verification Program Analysis

Much of the performance cost of program verification for COBOL projects is incurred by the advanced program analysis options in the Project Verification Options window. 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. As long as you have verified source files with the **Enable Reference Reports** and **Enable Interactive Analysis** workspace verification options, you can use the post-verification program analysis

feature to collect the remaining program analysis information without having to re-verify your entire legacy program.

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

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 on Cobol Post-Verification Program Analysis

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 Cobol program analysis the *first* time you collect analysis information, whether during verification or subsequent post-verification analysis. This is because, with the exception of **Enable Impact Report** and **Enable Execution Flow**, selecting *any* of the options dependent on the **Perform Program Analysis** project verification option, whether during a previous verification or a previous program analysis, results in *none* of the information for those options being collected in a subsequent post-verification program analysis.

Therefore 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.

Restrictions on PL/I Post-Verification Program Analysis

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.

Using Post-Verification Reports

Use COBOL Analyzer post-verification reports to check verification results, perform detailed executive assessments of the application, and view key data operations:

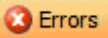
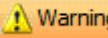
- Verification Reports offer a convenient way to analyze project verification results.
- Executive Reports offer HTML views of application inventories that a manager can use to assess the risks and costs of supporting the application.
- CRUD Reports show the data operations each program in the project performs, and the data objects on which the programs operate.

Viewing Verification Reports

Use Verification Reports to display the verification errors and warnings for each source file in the selected project and to navigate to the offending code in source. To open the Verification Report window, select a project in the Repository Browser and choose **Reports > Verification Report**. The verification report is displayed in the Verification Report window.

To show the verification report for a different project, select the project in the **Projects** drop-down on the toolbar. To refresh the report after reverifying a file, click **File > Refresh Errors**.

Errors Pane



The Errors pane of the Verification Report window displays the errors and warning for the project, sorted by the Files Affected column. Click  to display the errors for the project. Click  to display the warnings for the project. The buttons are toggles. Click the buttons again to hide errors or warnings.

Click an error or warning to display the affected files in the Files Affected pane and to highlight each instance of offending code in the Source pane. Mark an error or warning to mark the affected files in the Files Affected pane. The table below describes the columns in the Errors pane.

Column	Description
Severity	The severity of the error or warning.
Number	The error or warning number.
Count	The number of occurrences of errors or warnings of this type in the project.
Files Affected	The number of files affected by the error or warning.
Sample Text	The text of the error or warning message. If there are multiple occurrences of the error or warning, and the text of the message differs among occurrences, then the text of a sample occurrence.

Files Affected Pane

The top portion of the Files Affected pane displays the files affected by the error or warning selected in the Errors pane, the verification status of the file, and the numbers of occurrences of the error and warning in the file. Click a file to display the occurrences in the Details portion of the Files Affected pane. Select **Show All Errors** to display every error and warning for the selected file.

Errors are indicated with a  symbol. Warnings are indicated with a  symbol. Click an occurrence of an error or warning to navigate to the offending code in the Source pane.

Source Pane

The Source pane displays view-only source for the file selected in the Files Affected pane. Offending code is highlighted in red.

Usage is similar to that for the COBOL Analyzer Interactive Analysis Source pane. For more information, see *Analyzing Programs* in the product documentation set.

Marking Items

To mark an item, place a check mark next to it. To mark all the items in the selected tab, choose **Edit > Mark All**. To unmark all the items in the selected tab, choose **Edit > Unmark All**.

Including Files into Projects

In very large workspaces, you may find it useful to move or copy files into different projects based on the errors they contain. Follow the instructions below to move or copy files listed in a Verification Report into a project.

1. In the Files Affected pane, mark the items you want to move and choose **File > Include Into Project**. The Select Project window opens.
2. In the Select Project window, select the target project. Click **New** to create a new project.

3. Select either:

- **Include All Referenced Objects** if you want to include objects referenced by the selected object (the Cobol copybooks included in a Cobol program file, for example).
- **Include All Referencing Objects** if you want to include objects that reference the selected object.



Note: This feature is available only for verified files.

4. Select:

- **Copy** to copy the selection to the target project.
- **Move From Current Project** to move the selection to the target project.
- **Move From All Projects** to move the selection from all projects to the target project.

5. Click **OK** to move or copy the selection.

Generating HTML Reports

To generate HTML reports of Verification Report results, choose **File > Report > <Report Type>**. Before generating the Details for Checked Files report, mark each file on which you want to report.



Tip: The Missing Files reports are a convenient alternative to an Unresolved Report when you are interested only in missing source files, and not in unresolved objects like system programs. You can generate a missing files report for a project or for the entire workspace.

Viewing CRUD Reports

The CRUD Report for a project shows the data operations each program in the project performs, and the data objects on which the programs operate. To generate a CRUD Report, select a project in the Repository Browser and choose **Reports > CRUD Report**. The figure below shows a CRUD Report.

Project options on the **Options > Project Options > Reports > CRUD Report** tab determine the data operations and program-to-data object relationships displayed in CRUD reports. To refresh the report after modifying display options, choose **File > Refresh** in the CRUD Report window. To generate the report in HTML, choose **File > Report**.



Note: The IMS data column of the CRUD report behaves differently from the columns for other data types. What appears in the IMS data column cells depends on what can be determined. If the segment can be determined, the cell is populated with the PSB name and segment name. Otherwise, the segment name is left blank. The format is `xxxxxxx.yyyyyyy`, where `xxxxxxx` is the PSB name and `yyyyyyy` is the segment name or blank if the segment cannot be determined.

Setting CRUD Report Options

Use the **Report > CRUD Report** tab of the Project Options window to specify the data operations and program-to-data object relationships displayed in CRUD Reports.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Report tab, then the CRUD Report tab.
2. Place a check mark next to each type of program-to-data object relationship you want to display.
3. Place a check mark next to each type of data operation you want to display.

Exporting for Fortify Security Analysis (MBS)

You can create and save a Mobile Build Session (MBS) using verified COBOL files. The MBS includes COBOL files and their respective copybooks. When created the MBS is saved to a single file with a `.mbs` file extension. The `.mbs` file can then be imported into the Fortify Static Code Analyzer (SCA) for security analysis.

To create and export an MBS file, perform the follow steps:

1. In the Repository browser, select a project that contain verified COBOL files or select individual verified COBOL files.
2. Click **Reports > Export for Fortify Security Analysis (MBS)**.
This opens the **Export for Fortify Security Analysis (MBS)** dialog box.
3. Browse to the location you want to save the file to.
4. In the **File name** field, type the name of the file to save.
5. Click **Save**.

Distinguishing between Homonyms in the Reports

Sometimes there are several items with the same name in the repository. To distinguish between them in the reports, you can set COBOL Analyzer to display their location.

To do so:

1. Go to **Options > Workspace Options > Homonyms**.
2. Check **Show Source Path in Reports**.



Note: This option is unchecked by default.

When **Show Source Path in Reports** is enabled, the Complexity Metrics report, Unresolved Report, Cross Reference Report, and CRUD Report will contain a column showing the path with the file name to distinguish between objects with the same name. In the Complexity Metrics report the column is called **Source Path**.

Inventorying Applications

Users often ask why the COBOL Analyzer 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.

Use COBOL Analyzer *inventory analysis* tools to ensure that all parts of your application are available to the parser:

- Reference Reports let you track referential dependencies in verified source.
- Orphan Analysis 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.
- Decision Resolution 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.

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 copybook 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:

- An Unresolved Report identifies missing application elements.
- An Unreferred Report identifies unreferenced application elements.
- A Cross-reference Report identifies all application references.
- An External Reference Report identifies references in object-oriented applications to external files that are not registered in the workspace, such as .java, Java Archive (JAR), or C++ include files (assuming you have identified the locations of these files in the Workspace Verification options window for the source files). These references are not reported as unresolved in the Unresolved Report.



Tip: The Missing Files report in the Verification Report tool is a convenient alternative to an Unresolved Report when you are interested only in missing source files, and not in unresolved objects like system programs.

Understanding the Reference Reports Window

Use Reference Reports to track referential dependencies in verified project source. To open the Reference Reports window, select a project in the Repository Browser and choose **Reports > Reference Reports**.

When the Reference Reports window opens, choose the Reference Report type in the **Report type** drop-down. To limit the report to references in the current project, choose **View > Restrict References to Project**. To generate the report in HTML, choose **File > Report**.

The figure below shows an Unreferred Report window. The windows for the other reports are similar. By default, all Reference Report panes are displayed. Select the appropriate choice in the **View** menu to hide a pane. Select the choice again to show the pane.

Main Pane

The Main pane displays the objects in the Reference Report and their relationships. The table below describes the columns in the Main pane.

Column	Report Type	Description
Object Name	All	The name of the unresolved, unreferenced, cross-referenced, or externally referenced object.
Object Type	All	The entity type of the unresolved, unreferenced, cross-referenced, or externally 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, External Reference Report	The name of the referring object.
Referring Object Type	Unresolved Report, Cross-reference Report, External Reference Report	The entity type of the referring object.
Relationship	Unresolved Report, Cross-reference Report, External Reference Report	The relationship between the unresolved, cross-referenced, or externally referenced object and the referring object.
Object Description	All	The description of the unresolved, unreferenced, cross-referenced, or externally referenced object entered by the user on the Description tab of the Object Properties window.

Preview Pane

The Preview pane lets you browse Interactive Analysis information for the object selected in 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 Interactive Analysis information for a program. Choose the information you want to view for the object from the **Source** drop-down.

Setting Reference Reports Options

Use the **Report > Reference Reports** tab of the Project Options window to specify the entity types for which reference report information is collected.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Report** tab, then the **Reference Reports** tab.
2. Place a check next to each type of entity you want to be included in reference reports.

Using Orphan Analysis Reports

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. Use the Orphan Analysis tool to find orphans.

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

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

Suppose 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.

Understanding the Orphan Analysis Window

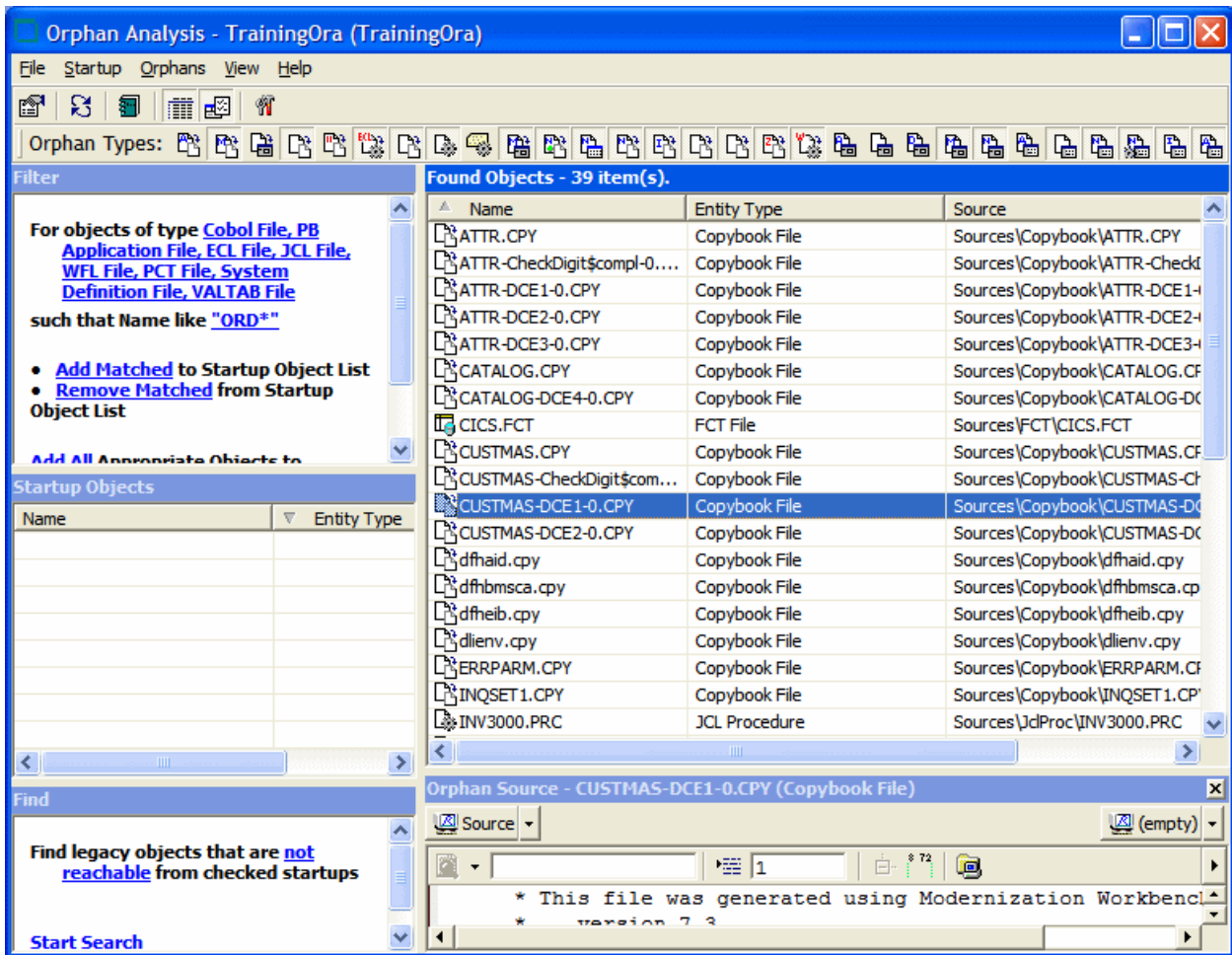
Use the Orphan Analysis tool to determine whether an object exists in the reference tree for a top-level program object. To open the Orphan Analysis tool window, select a project in the Repository Browser and choose **Prepare > Orphan Analysis**.

Project options on the Report > Orphan Analysis tab specify the search filter for the report. To refresh the report after modifying the options, choose **Orphans > Refresh** in the Orphan Analysis window. To generate the report in HTML, choose **File > Save Report As**.



Note: The Filter, Startup, and Find panes let you use hyperlinks to set up and apply the Orphan Analysis search filter. Use these panes instead of the options window if you prefer.

The figure below shows the Orphan Analysis window. By default, all Orphan Analysis panes are displayed. Select the appropriate choice in the **View** menu to hide a pane. Select the choice again to show the pane.



Found Objects Pane

The Found Objects pane shows the name, type, and source location of orphans. To show the list of orphans only, deselect **View > Report View**.

Orphan Source Pane

The Orphan Source pane lets you browse Interactive Analysis information for the object selected in the Found Objects pane. The information available depends on the type of object selected. You see only source code for a copybook, for example, but full Interactive Analysis information for a program. Choose the information you want to view for the object from the **Source** drop-down.

Setting Orphan Analysis Options

Use the **Report > Orphan Analysis** tab of the Project Options window to specify the search filter for an Orphan Analysis report.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Report tab, then the Orphan Analysis tab.
2. In the Startup pane, click **Select Startup Types**. The Startup dialog opens.
3. In the Startup dialog, set up a search filter for the startup objects in the orphan analysis. You can filter on entity type, entity name, or both:
 - To filter on entity type, place a check next to the entity type you want to search for in the Roots pane.

- To filter on entity name, place a check next to a recognized matching pattern in the Like pane, the Unlike pane, or both. Add patterns as necessary. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).
4. When you are satisfied with your choices in the Startup dialog, click **OK**.
 5. In the Find pane, define the terms of the search by selecting the appropriate choice in the **Relationships to Checked Startups** drop-down, the **Relationships to Unchecked Startups** drop-down, or both.
 6. In the Entities pane, click **Displayed Types**. The Entities dialog opens. In the Entities dialog, place a check next to each type of entity to include in the report. When you are satisfied with your choices in the Entities dialog, click **OK**.

Deleting Orphans from a Project

To delete an orphan from a project (but not the workspace), select the orphan in the Found Objects pane and choose **Orphans > Exclude from Project**.

Deleting Orphans from a Workspace

To delete an orphan from the workspace, select the orphan in the Found Objects pane and choose **Orphans > Delete from Workspace**.

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 COBOL Analyzer 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.

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.

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

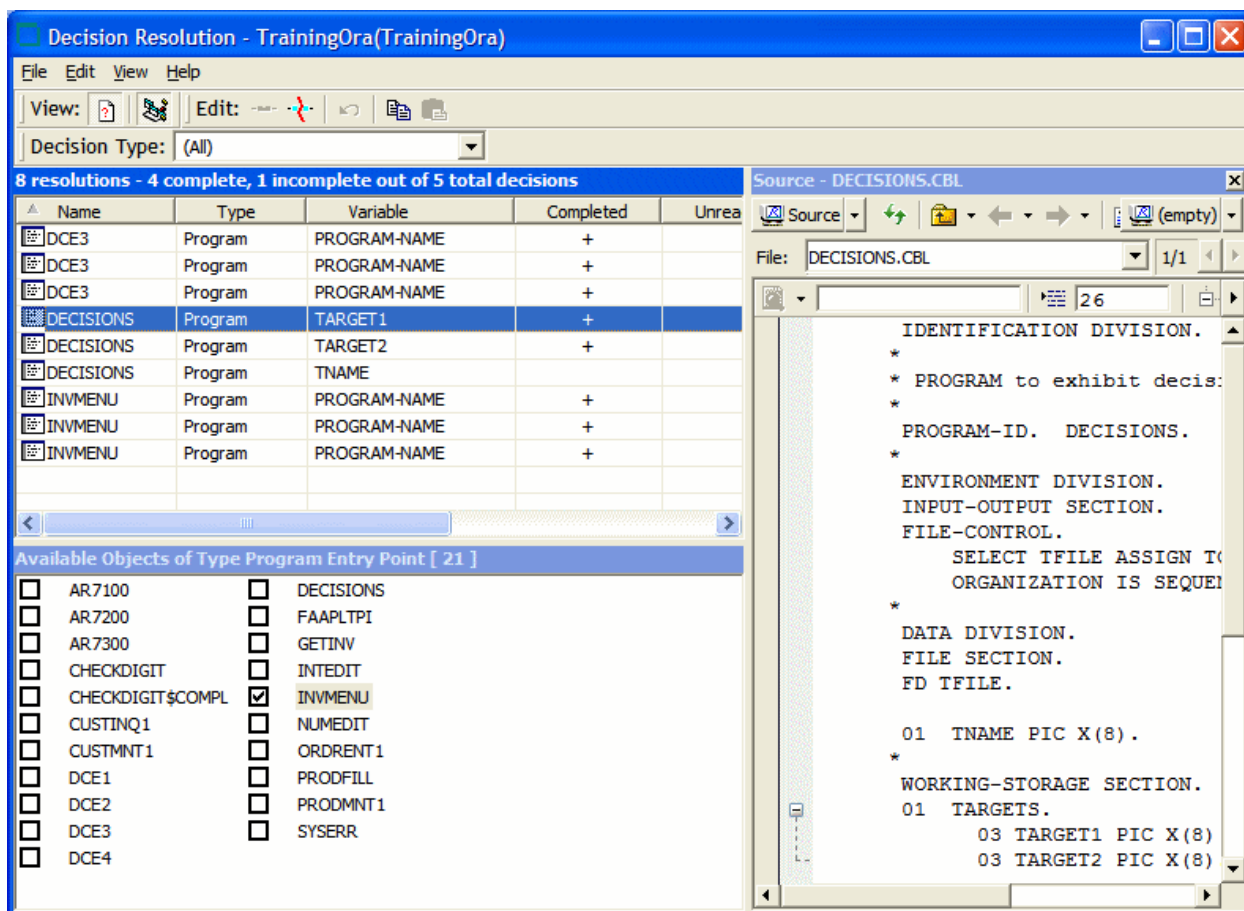
to autoresolve every decision, or even every decision completely, but it should get you to a point where it is practical to complete decision resolution manually.

Understanding the Decision Resolution Tool Window

Use the Decision Resolution tool to view and manually resolve decisions. To open the Decision Resolution tool window, select a project in the Repository Browser and choose **Prepare > Resolve Decisions**.

To save decision resolutions to the repository, choose **File > Save**. To generate the Decision Resolution report in HTML, choose **File > Report**.

The figure below shows the Decision Resolution window. By default, all Decision Resolution panes are displayed. Select the appropriate choice in the **View** menu to hide a pane. Select the choice again to show the pane.



Decision List Pane

The Decision List pane displays the decisions in the project. To filter the list, choose the type of decision you want to display in the **Decision Type** drop-down. The table below describes the columns in the Decision List pane.

Column	Description
Name	The name of the object that contains the decision.
Type	The type of the object that contains the decision.
Variable	The program variable that requires the decision.

Column	Description
Completed	Whether the decision has been resolved.
Unreachable	Whether the decision is in dead code.
Manual	Whether the decision was resolved manually.
Resolved to	The target object the variable resolves to (an entry point, for example). An unresolved decision contains the grayed-out text <i>Some Object</i> .

Available Targets Pane

For the selected decision type, the Available Targets pane lists the objects in the workspace to which the decision can be resolved. To resolve a decision to an available target, select the decision in the Decision List pane and place a check next to the target.

To limit the targets to objects in the current project, choose **View > Restrict to Current Project**. To delete a decision resolution, uncheck the target. To undo changes, choose **Edit > Undo all changes**.

Source Pane

The Source pane lets you browse Interactive Analysis information for the object selected in the Decision List pane. The information available depends on the type of object selected. You see only source code for a copybook, for example, but full Interactive Analysis information for a program. Select the information you want to view for the object from the drop-down in the upper lefthand corner of the pane.

Resolving Decisions Manually

Follow the instructions below to resolve decisions manually to targets in or out of the workspace.

1. To resolve decisions to available targets, select one or more entries in the Decision List pane and check one or more target objects in the Available Targets pane. If you link an entry to multiple targets, the Decision Resolution tool creates as many entries in the Decision List pane as there are targets.




Note: If you are linking an entry to multiple targets, you can save time by selecting the targets and choosing **Edit > Link Selected Targets**. You can also choose **Edit > Copy** to copy selected targets to the clipboard, then **Edit > Paste** to link the targets to an entry.

2. To resolve decisions to targets not in the workspace, select one or more entries in the Decision List pane and choose **Edit > Link New Target**. The Link New Target window opens.
3. 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. 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).
4. Check **Completed** if you want the resolution to be marked as completed. When you are satisfied with your entry, click **OK**.

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. Click **Yes** if you want to restore the decisions. Click **No** otherwise.

 **Note:** Check **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, check **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 on the Verification tab of the Project Options window for a source file type.

For programs only, you can autoresolve decisions after verification by selecting the project, folder, or files for which you want to autoresolve decisions and choosing **Prepare > AutoResolve Decisions**. Only a master user can autoresolve decisions in a multiuser environment.

 **Note:** Decision Resolution cannot autoresolve every decision. The target name may be read from a data file, for example.

Setting Decision Resolution Tool User Preferences

Use the Decision Resolution Tool tab of the User Preferences window to specify whether you want to be prompted to restore invalidated manually resolved decisions when you reopen the Decision Resolution tool.

1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Decision Resolution Tool tab.
2. Select **Ask before restoring previous manual changes** if you want to be prompted to restore manually resolved decisions when you reopen the Decision Resolution tool.

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 refers 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 CA repository.

You enable the Generic API Analysis feature by identifying unsupported APIs and their parameters in the file `\<CA Home>\Data\Legacy.xml`. Before you verify your application, set **Perform Generic API Analysis** on the **Verification** tab of the **Project Options** window. That option tells the parser to define relationships with the objects passed as parameters in the calls, in addition to relationships with the unsupported APIs themselves.

This section shows you how to identify the programs and parameters to the parser before verifying your application. You can specify both object and construct model information, and create different relationships or entities for the same parameter in a call.



Attention:

The specification requires a thorough understanding of the COBOL Analyzer repository models. For background information on the repository models, see the following documentation sections:

- *How COBOL Analyzer Models Applications in Getting Started.*
- *Understanding the Application-Level Metamodel in Using Architecture Modeler*

- *Using the Model Reference Pane in Interactive Analysis*



Note: Only the predefined definitions described in this section are guaranteed to provide consistent data to CA databases.



Note: For altered `Legacy.xml` files: The `Legacy.xml` file that resides in the `\<CA Home>\Data` directory gets overwritten when a new HotFix is installed. To preserve the changes made to the file:

1. Create an empty file `Legacy.xml` in the workspace directory. You can use the `\<CA Home>\Data\Legacy.xml.user` file as a template.
2. Add only the tags that need to be added or altered
3. Save the file and verify the sources.

We do not recommend copying the `\<CA Home>\Data\Legacy.xml` file to the workspace directory and then altering it as needed, since the altered file is merged with the original file every time the parsers are required to read them. An example of a user-defined `Legacy.xml` is:

```
<Legacy>
  <Cobol>
    <OverloadedDiagnostics>
      <msg num="2054" sev="2" />
    </OverloadedDiagnostics>
  </Cobol>
</Legacy>
```

Identifying Unsupported API Calls to the Parser

Follow the instructions in the steps below and the detailed specifications in the following sections to identify unsupported API calls to the parser. For each call, you need to define an entry in `\<COBOL Analyzer 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 `\<COBOL Analyzer 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.

The syntax for specifying predefined patterns in a `<rep>` or `<hc>` definition is described in the section for the tag. Consult `Legacy.xml.api` for supported patterns and for required parameters and values.

1. Open the file `\<COBOL Analyzer 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, following the specifications in the sections below and the samples of Generic API usage in `Legacy.xml`.
4. Click **Options > Project Options > Verification** tab. Click **Advanced** in the Advanced Program Analysis section, then check **Perform Generic API Analysis** and then click **OK**.
5. Verify the project.

Using the API Entry Tag

The *name* attribute of the `<API Entry>` tag is the name of the entry, used for error diagnostics only.

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.

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: Do not use *auto* when the content of a numeric field is binary. Use *int*. The parser extracts the binary content and converts it to the corresponding numeric string.

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.

Natural-Specific Usage

For Natural, use the *auto* data type in the <vars> tag. *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.

Using the rep and hc Tags

Use the <rep> tag to represent the API call in the object model repository. Use the <hc> tag and the <attr> subtag to represent the construct model (HyperCode) attributes of entities defined in the call.

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.

Use the case statement to produce relationships between programs. When a `Legacy.xml` value resolves to `"__fail__"` in a case statement, the relationship will not be produced. For example:

```
<progrname switch-var='%fname'>
  <case eq='LINKDATA' value='%pname' type='Calls' params='%3' />
  <case eq='LINK' value='%pname' type='Calls' params='%3' />
  <case eq='READQTS' value='__fail__' type='__fail__' />
</progrname>
```

will not produce a relationship for 'READQTS'

Using Predefined Patterns

Most repository entities can be represented with the predefined patterns in `\<CA 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 CA databases.

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

Look at 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 CA databases.

Using the entity Subtag

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

- *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.

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.

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.



Note: 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

Assume that we know that the second parameter in the API call described earlier for the <vars> tag 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 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. The table below describes the 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.

Name	Description
_yes	A non-empty string for use as a true value.
_no	An empty string for use as a false value.
_pgmname	Name of the calling program.
_hcid	HyperCode ID of the call node.
_varname nn	If parameter number nn 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.

Set the value and type of the variable to `__fail__` (two underscores) if you do not want a relationship to be produced for the variable:

```
<progrname switch-var="%fname">
  <case eq="LINKDATA" value="%pname" type="Calls" params="%3" />
  <case eq="LINK" value="%pname" type="Calls" params="%3" />
  <case eq="READQTS" value="__fail__" type="__fail__" />
  <default value="" />
</progrname>
```

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.

In cases where more than one switch variable is used, than the result will not be context dependent, this means that all possible combinations between possible values will be produced. The following example legacy.xml file demonstrates the behavior:

```
<Legacy>
  <Cobol>
    <MicroFocus>
      <GenericAPI>
        <APIEntry name='CALL XBASE'>
          <match stmt='CALL'>
            <name value='XBASE' />
          </match>
          <flow halts='no'>
            <param index='1' usage='r' />
            <param index='2' usage='r' />
            <param index='3' usage='r' />
            <param index='4' usage='r' />
            <param index='5' usage='r' />
            <param index='6' usage='r' />
          </flow>
          <vars>
            <arg var='code' param='1' type='data' offset='0' size='1' />
            <arg var='filecode' param='2' type='data' offset='0' size='1' />
          </vars>
          <rep>
            <sql-database>
              <table switch-var = '%filecode'>
                <case eq='1' value='FILE1' />
                <case eq='2' value='FILE2' />
              </table>
              <column switch-var = '%filecode'>
                <case eq='1' value='FILE1' />
                <case eq='2' value='FILE2' />
              </column>
              <origin value='XBASE' />
              <action switch-var = '%code'>
                <case eq='1' value='Inserts' />
                <case eq='2' value='Deletes' />
              </action>
            </sql-database>
          </rep>
        </APIEntry>
      </GenericAPI>
    </MicroFocus>
  </Cobol>
</Legacy>
```

Using the following code:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    test1.
AUTHOR.
ENVIRONMENT DIVISION.

DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01 CODE1 PIC 9.
01 FILECODE PIC 9.
LINKAGE SECTION.

PROCEDURE DIVISION.
```

```

FIRST-SECTION SECTION.
FIRST-PARAGRAPH.
    MOVE 1 TO CODE1.
    MOVE 1 TO FILECODE.
    PERFORM SECOND-PARAGRAPH.

    MOVE 2 TO CODE1.
    MOVE 2 TO FILECODE.
    PERFORM SECOND-PARAGRAPH.
    GOBACK.

SECOND-PARAGRAPH.
    CALL 'XBASE' USING CODE1 FILECODE.
    EXIT.

```

This results in the following relations being generated:

- Insert into FILE1.
- Delete from FILE1.
- Insert into FILE2.
- Delete from FILE2.

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. The table below 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.

Function	Description
substr,<start>,<size>,<variable>	Extracts a substring from the value of the variable. The substring begins at position <start> (counted from 0), and is <size> characters long. If <size> is an empty string, the substring extends up to the end of the value string.
int,<variable>	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.
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}.
lookup, <file>, <keyCol>, <valueCol>, <variable>	Looks up a value from a comma-separated value (CSV) file. The CSV file is located on disk at <file>. For each line in that file, <keyCol> specifies the column (counted from 0) containing the key to match against <variable> and <valueCol> specifies the column containing the value to return. If <keyCol> is an empty string, column 0 is assumed. If <valueCol> is an empty string, column 1 is assumed. If no matching row is found in the file, an empty string is returned.
upper, <variable>	Returns the value of the variable converted to all upper-case characters.
lower, <variable>	Returns the value of the variable converted to all lower-case characters.

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 section on decisions 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 section on decisions, 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 for true and false, respectively. 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.

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

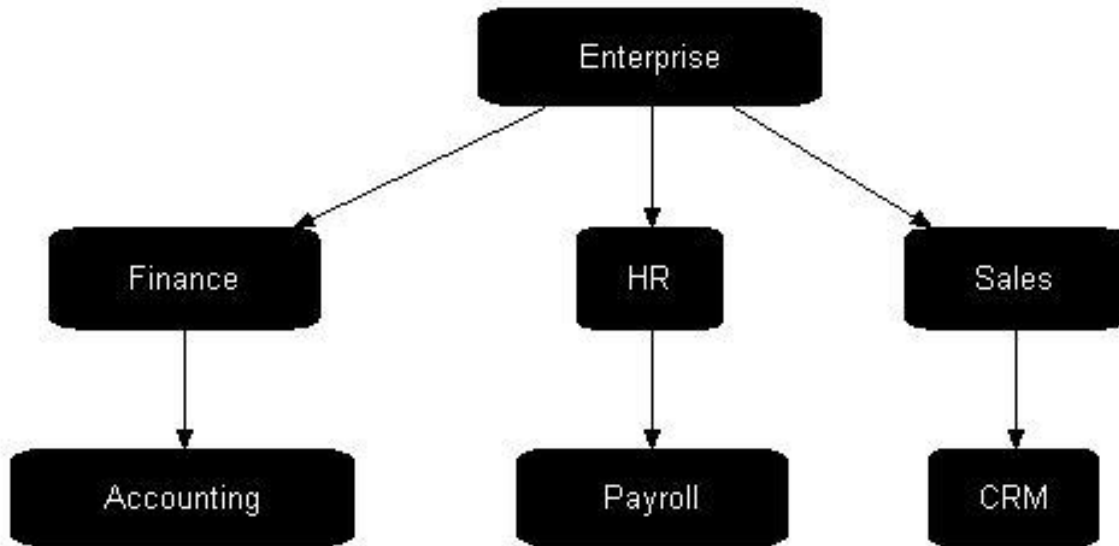
```

Analyzing Projects

Managing Tags

You use a simple tagging language to identify workspace objects as members of functional, structural, or other types of groupings. After you set up these groupings, you can chart them in EV or “black-box” them in the Diagrammer.

Each member of a grouping is identified by a tag, "Payroll," for example. Each grouping can, in turn, reference a more inclusive grouping, "HR," for instance. The figure below shows the tag hierarchy for a typical enterprise:



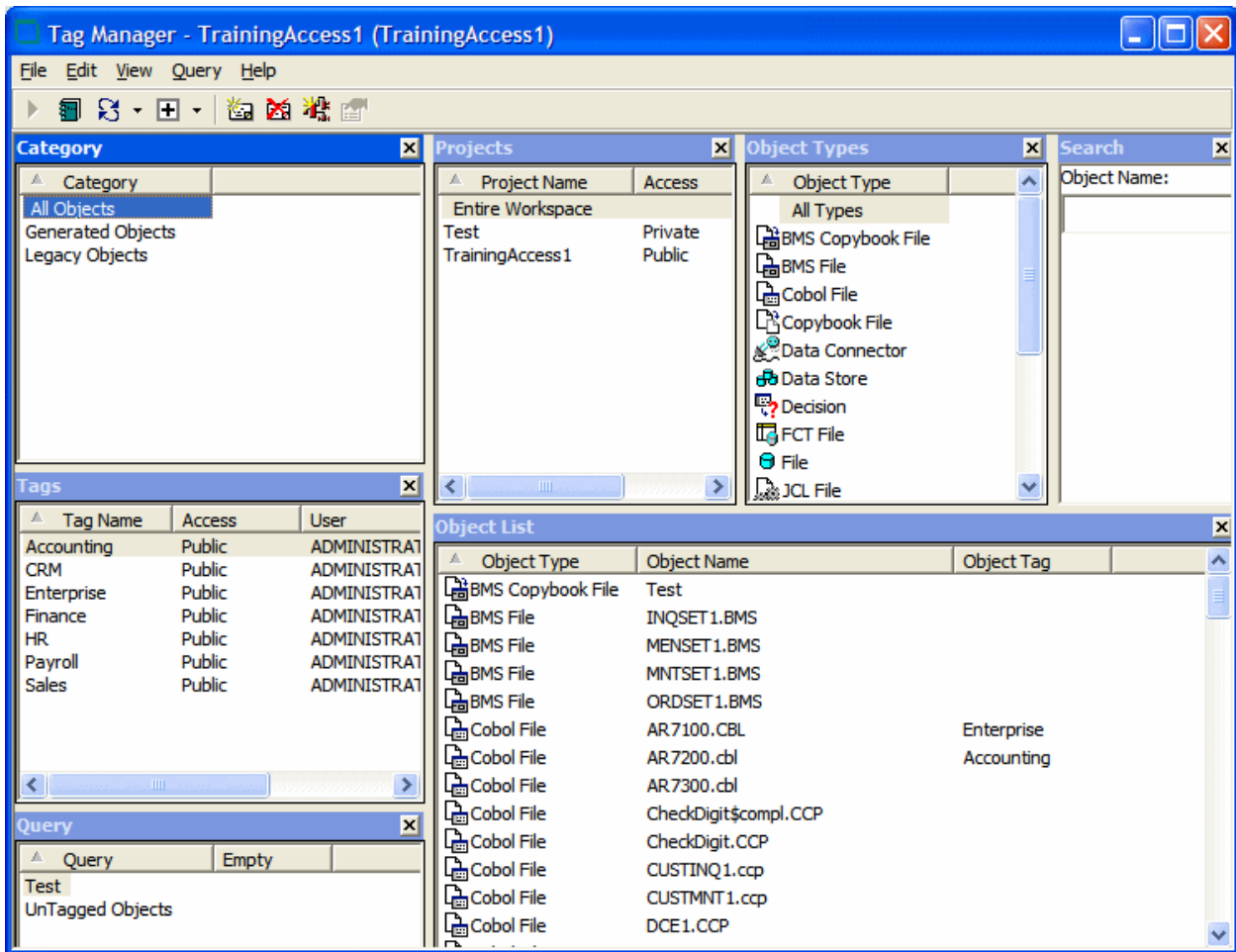
In fact, this is the default tag structure that ships with CA. The "Enterprise" tag is the root of the structure. The "Finance," "HR," and "Sales" tags represent departments in the enterprise. The "Accounting," "Payroll," and "CRM" tags are the applications owned by the departments.


Understanding the Tag Manager Window

Use the Tag Manager to create and assign tags to objects, and create relationships between tags. To open the Tag Manager, click **Analyze > Tag Manager**.

The figure below shows the Tag Manager window. The Objects List pane shows repository objects and their tags. Your selections in the other Tag Manager panes filter the objects displayed in the Objects List pane.



By default, all Tag Manager panes are displayed. Select the appropriate choice in the **View** menu to hide a pane. Select the choice again to show the pane.



 **Note:** When you launch a query based on All Types selection in the Object Types pane, a dialog opens prompting that you are about to run a long running query and asking if you want to proceed. This message appears only when the query is likely to run for more than a few seconds. If you choose not to run it, the Object List's header turns red to indicate that the displayed information does not match the query.

Object List Pane

The Object List pane displays workspace objects filtered by your selections in the Category, Project, Object Types, Search, and/or Query panes; or by your selections in the Tags pane.

By default, the Object List pane displays up to 100 repository objects and their tags. Click  next to the  icon on the toolbar and choose one of:

- **More Objects** in the drop-down menu to display the next 100 objects that match the filter criteria.
- **All Objects** in the drop-down menu to display all the objects that match the filter criteria.

Category Pane

The Category pane displays the categories you can use to filter objects in the Object List pane:

- All Objects comprises source and generated objects in the workspace.
- Generated Objects comprises generated objects in the workspace.
- Legacy Objects comprises source objects in the workspace.

Projects Pane

The Projects pane displays the workspace projects you can use to filter objects in the Object List pane. Entire Workspace comprises all the projects in the workspace.

Object Types Pane

The Object Types pane displays the object types you can use to filter objects in the Object List pane.

Search Pane

The Search pane lets you filter the objects in the Object List pane using wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). Enter the pattern in the **Object Name** field. The Object List pane is refreshed as you type.



Note: In large repositories, the Object List pane may not immediately refresh as you type. Press the Enter key to instantly display matching objects for the current string.

Query Pane

The Query pane lets you filter the objects in the Object List pane using complex queries. Select a query to filter the Object List by its terms. Select another query and choose one of:

- **Add Tags > with <And>** in the right-click menu to add its results to the results of previous queries.
- **Add Tags > with <Or>** in the right-click menu to compare its results with the results of previous queries.

Tags Pane

The Tags pane lists the tags in the workspace. To view the objects that have tags assigned, select one or more tags and the result will be displayed in the Object List pane.

Creating Tags

To create a tag, choose **Edit > Create** in the Tag Manager window. A dialog box prompts you to enter the name of the new tag. Enter the name and click **OK**. The tag is displayed in the Tags pane.

Specifying Access to Tags

By default, the tags you create are public. To make a tag private, that is, visible only to you, select it and choose **Set Access > Private** in the right-click menu. To make the tag public again, choose **Set Access > Public** in the right-click menu.

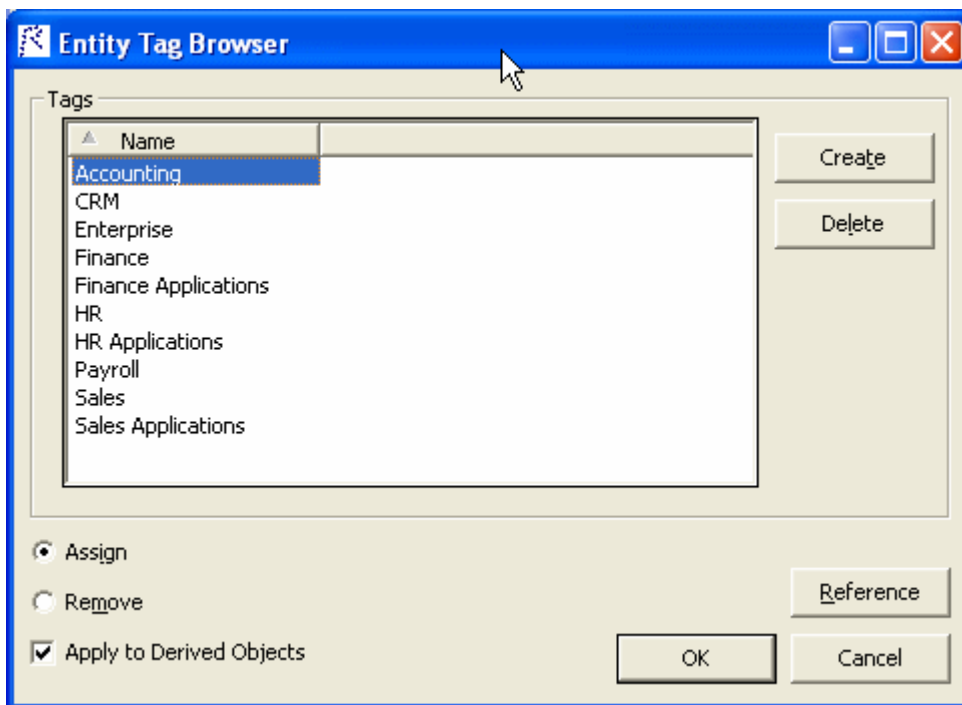


Note: Tags shipped with the product are always public. You cannot change their access.

Assigning Tags

You can assign tags to objects individually, or to all the objects generated from a source file. You can also assign tags to objects in the Repository pane, by selecting the objects and choosing **Assign/Remove Tags** in the right-click menu. Select a project or folder to assign tags to every first-level object in the project or folder.

1. In the Object List pane, select the object(s) you want to assign a tag and choose **Edit > Assign/Remove Tags**. The Entity Tag Browser window opens.



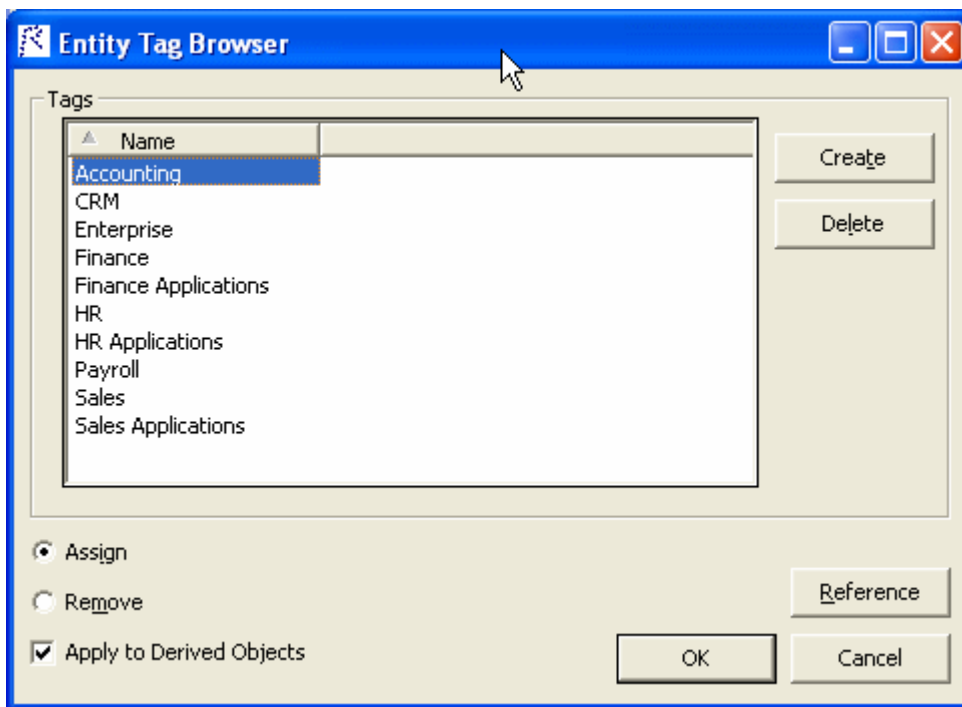
2. In the Entity Tag Browser window, select the tag(s) you want to assign and click **Assign**.
3. If you are assigning a tag to a legacy source file, check **Apply to Derived Objects** to assign the selected tag(s) to all objects generated from the source file.
4. Click **OK** to assign the selected tags.

The Object Tag column for the selected object(s) is updated to show the assigned tag(s) in the Objects List pane.

Removing Tags

Removing a tag simply “unassigns” the tag from the selected object. The tag is not deleted from the repository. You can also remove tags from objects in the Repository pane, by selecting the objects and choosing **Assign/Remove Tags** in the right-click menu. Select a project or folder to remove tags from every first-level object in the project or folder.

1. In the Object List pane, select the object(s) for which you want to remove tags and choose **Edit > Assign/Remove Tags**. The Entity Tag Browser window opens.



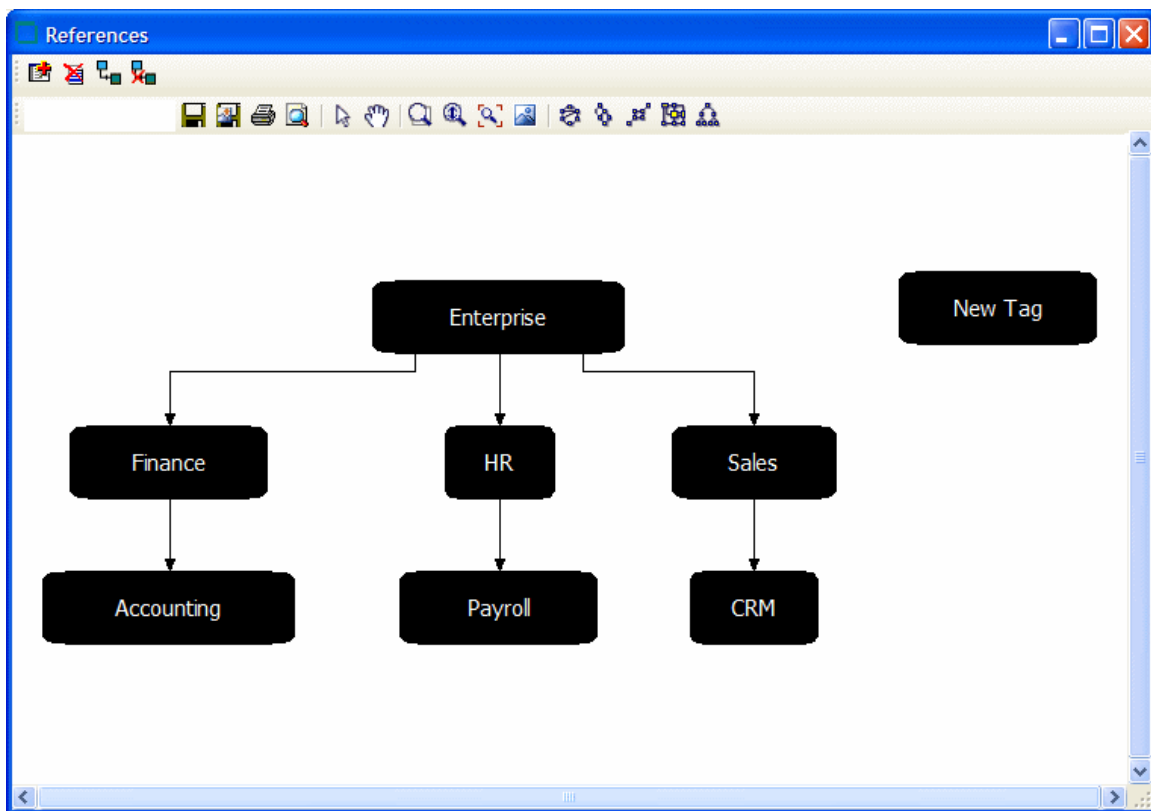
2. In the Entity Tag Browser window, select the tag(s) you want to remove and click **Remove**.
3. If you are removing a tag from a legacy source file, check **Apply to Derived Objects** to remove the selected tag(s) from all objects generated from the source file.
4. Click **OK** to remove the selected tags.


The Object Tag column for the selected object(s) is updated in the Objects List pane.



Creating Relationships Between Tags


The relationships between tags define the tag hierarchy. Create relationships between tags as described below.

1. Choose **Edit > Reference** in the Tag Manager window. The References window opens.




2. In the References window, click the  button on the toolbar.
3. The Select dialog opens. Select the tags you want to create relationships between and click **OK**.

 **Note:** You can also draw relationships between tags by clicking the  button on the toolbar. Select the first tag in the relationship and drag the mouse to the second tag in the relationship, then release the mouse button. See the Diagrammer help for instructions on how to use the other diagramming functions in the References window.

The tool draws a line between the referenced tags. To delete a relationship line, select it and click the  button on the toolbar.

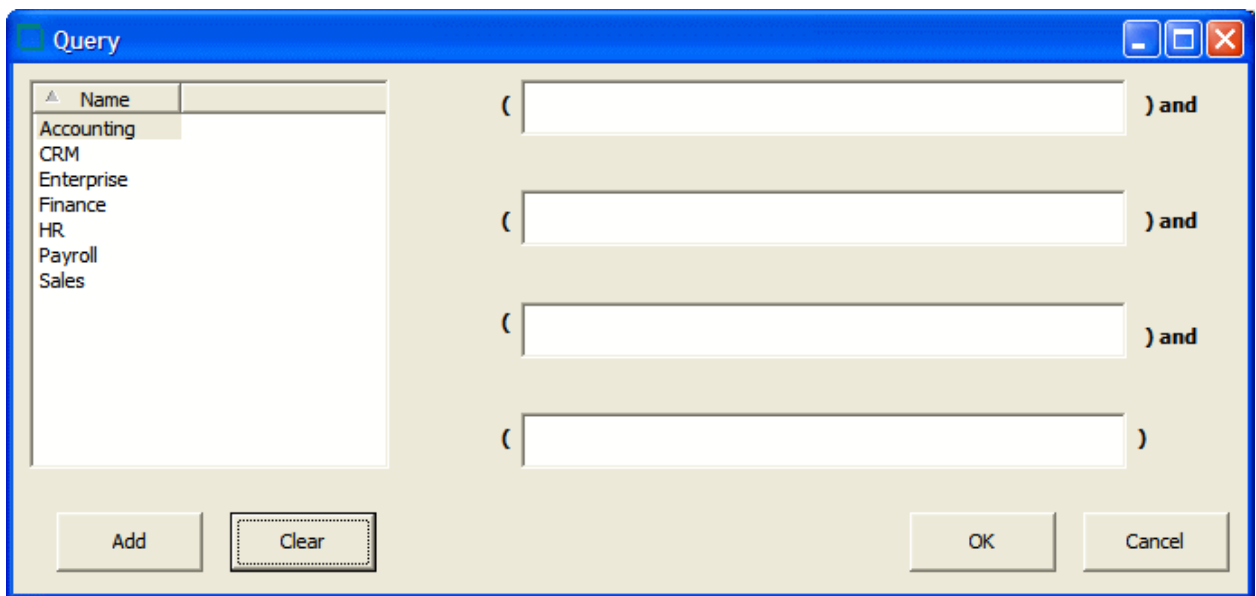
Deleting Tags

To delete tags from the repository, select them in the Tags pane and choose **Edit > Delete Tags**. You can also delete tags in the References window, by selecting them and clicking the  button on the toolbar.

Creating Tag Queries

Use tag queries to filter the objects in the Object List pane in more complex ways than the other panes allow. Create a tag query as described below.

1. Choose **Query > New Query** in the Tag Manager window. A dialog box prompts you to enter the name of the new query. Enter the name and click **OK**. The query is displayed in the Query pane.
2. Select the query in the Query pane and choose **Query > Edit Query**. The Query window opens.





3. Define the terms of the query. Click in the term field you want to define, then select the tag you want to add to the term definition and click **Add**. Tags added to the same term are ORed. Tags added to a different term are ANDed. Click **Clear** to clear the term definition.
4. When you are satisfied with the query definition, click **OK**.



Deleting Tag Queries

To delete a tag query, select it and choose **Delete** in the right-click menu.

Refreshing the Tag Manager Window

To refresh the Tag Manager window with the latest information from the repository, including changes made by other users, click the  button next to the  button on the toolbar and click **Refresh** in the drop-down menu. If objects have been deleted from the repository, they will remain visible in the Object List pane until you clean up the Tag Manager window.

Cleaning Up the Tag Manager Window

When objects are deleted from the repository, Tag Manager continues to display them in the Object List pane until you *clean up* the Tag Manager window. To clean up the window, click  next to the  button on the toolbar and click **Clean Up** in the drop-down menu.



Note: This feature is only available to master users. Running a clean up may take several minutes, depending on the size of your repository.

Generating Reports

To generate an HTML report for all objects currently displayed in the Object List pane, choose **File > Save Report**.

Analyzing Relationship Flows

Use the Diagrammer to view the relationships between objects in a scope, a data flow or call map, for example. These relationships describe the ways in which objects in the scope interact. In the data flow fragment below, for example, the GSS2 program reads the file GSS2.STATEMS. The file, in turn, is assigned to the CXXCP.OPK00.SNFILE data store.



The Diagrammer’s high-level analysis of relationships lets you quickly trace the flow of information in an application. You can use it to view program to program calls; program, transaction, and screen flows; job, program, and data store flows; and much more.

Understanding Relationship Flow Diagrams

Relationship flow diagrams use boxes and lines to show the relationships between objects in a scope. Depending on the Diagrammer mode, you can display the relationships for:

- All the objects in the selected project.
- Only the objects you copy and paste onto the canvas.

In each mode, all related workspace objects are included in the diagram. In copy-and-paste mode, you can add objects to an existing diagram and expand the diagram one level at a time.


Objects are color-coded, based on a color scheme of your choosing. If you have assigned business names to objects, their business names appear in the diagram as well as their technical names. You can “black-box” items in higher-level groupings that make it easy to visualize their roles in your application, cluster highly related objects, filter drivers and utilities, and much more.





Understanding Diagram Scopes


The *scope* of a diagram determines the relationships it displays. To view a diagram of the relationships, select the scope in the **Scope** drop-down and choose **Scope > View Scope**.


The Diagrammer provides default scopes that you can use for most analysis tasks, but you can create your own scopes if you like. You might want to exclude data stores from a data flow, or include source files in a call map. Use the Scope Editor to create your own scopes.

Understanding Diagram Layouts

Diagrammer layout styles are designed to produce aesthetically pleasing, easy-to-understand diagrams for a wide variety of needs. To choose a layout style, click  on the button for the active layout style, then select the layout style in the drop-down menu. The following styles are available:

- The  **Circular Layout** style organizes nodes in clusters of related objects.
- The  **Hierarchical Layout** style organizes nodes in precedence relationships, with levels for parent and child object types.
- The  **Orthogonal Layout** style organizes nodes in relationships drawn with horizontal and vertical lines only.
- The  **Symmetric Layout** style (the default) organizes nodes based on symmetries detected in their relationships.


 **Note:** The symmetric style typically offers the best performance for very large diagrams.

- The  **Tree Layout** style organizes nodes in parent-child relationships, with child nodes arranged on levels farther from the root node than their parents.

Click a layout style button in the drop-down to change the style after a diagram is drawn.

Generating a Diagram for the Selected Project

Follow the instructions below to generate a relationship flow diagram for the selected project. You can exclude related workspace objects that are not in the project, colorize project boundaries, and show incomplete relationship chains. You can use tags to “black-box” objects in the diagram.


 **Note:** The Browser pane and project drop-down list in Diagrammer will use the same project filter from the main window when Diagrammer is opened. Any changes made to the project selection or project filter in the main window after Diagrammer is opened will not be reflected in Diagrammer.

1. Choose the type of the diagram - **Project Based** or **Entity Based**.
2. Select a **Project** from the drop-down list.
3. In the **Scope** drop-down, choose the scope of the diagram.
4. Select a **Group By** criterion from the drop-down list.
5. On the Diagram pane tool bar, choose the layout style for the diagram.
6. In the **Project** drop-down or Browser pane, select the project.
7. Click **Build Diagram** on the Diagrammer tool bar. The diagram is displayed.

Generating a Diagram for Objects Copied and Pasted onto the Canvas

Follow the instructions below to generate a relationship flow diagram for objects copied and pasted onto the canvas. In this mode, you can expand the diagram one level at a time and add objects to an existing diagram. You can exclude related workspace objects that are not in the project, colorize project boundaries, and show incomplete relationship chains. You can use tags to “black-box” objects in the diagram.

1. In the **Scope** drop-down, choose the scope of the diagram.
2. On the Diagram pane tool bar, choose the layout style for the diagram.
3. In the Browser pane, select the startup objects for the diagram, then press CTRL-C to copy them onto the clipboard.

 **Note:** Make sure to select startup objects in the diagram's intended scope. Most scopes include logical objects rather than source files.



4. To show the entire relationship flow for the selected objects, choose **View > Auto Expand**. Turn off **Auto Expand** if you want to expand the diagram one level at a time after it is drawn. Only immediate relationships of the selected objects are drawn initially.
5. Click in the Diagram pane, then choose **Diagram > Paste** or press **CTRL-V** to paste the objects onto the Diagrammer canvas.
6. Repeat the preceding step for each object whose relationships you want to add to the diagram. The diagram is automatically redrawn.



Understanding the Diagrammer Window

Use the Diagrammer to view the relationships between application objects interactively. To open the Diagrammer, select a project in the Repository Browser and choose **Analyze > Diagrammer**.

By default all panes in the Diagrammer are empty. Select the attributes for your diagram and click **Build Diagram** to create the diagram. Select the appropriate choice in the **View** menu to hide a pane. Select the choice again to show the pane.

Diagram Pane

The Diagram pane displays the diagram for the selected scope. To save a diagram in one of the supported diagram formats, click the  on the Diagram pane tool bar. To save a diagram in one of the supported image formats, click the  on the Diagram pane tool bar.

To print a diagram, click the  button on the Diagram pane tool bar. To preview the printed diagram, click the  button on the Diagram pane tool bar.

Selecting an Object or Relationship

To enable select mode, click the  button on the Diagram pane tool bar.

- To select an object in a diagram, click it on the diagram canvas. Use Ctrl-click to select multiple objects, or click a blank area of the canvas, then drag the mouse over multiple objects to “lasso” the objects. Selected objects are displayed with handles that you can use to size the object.
- To select a relationship in a diagram, click it on the diagram canvas. The selected relationship is displayed in blue.

Searching for an Object

To search for an object in a diagram, enter the text you want to match in the **Search** field on the Diagrammer tool bar and press Enter. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Navigating to an Object from a Relationship

To navigate to the diagram object on the left side of a relationship, select the relationship and choose **Locate Left** in the right-click menu. To navigate to the diagram object on the right side of a relationship, select the relationship and choose **Locate Right** in the right-click menu.



Note: Use the Overview pane to navigate to general locations in a diagram.

Expanding and Collapsing Relationships

To expand the incoming relationships for an object in a diagram generated in copy-and-paste mode, select the object on the diagram canvas and choose **Expand > Expand Incoming** in the right-click menu. To expand the outgoing relationships, select the object on the diagram canvas and choose **Expand > Expand Outgoing** in the right-click menu.

To collapse the incoming relationships for an object, select the object on the diagram canvas and choose **Collapse > Collapse Incoming** in the right-click menu. To collapse the outgoing relationships, click the object on the diagram canvas and choose **Collapse > Collapse Outgoing** in the right-click menu.

Moving an Object or Relationship

To move an object or relationship in a diagram, select it and drag it to the desired location on the diagram canvas.

Resizing an Object

To resize an object in a diagram, select it on the diagram canvas and drag one of its handles to the desired location.

Hiding an Object

To hide objects on the canvas, select an object and choose:

- **Hide > Selected** in the right-click menu to hide the selected object.
- **Hide > Incoming** in the right-click menu to hide incoming nodes of the selected object.
- **Hide > Outgoing** in the right-click menu to hide outgoing nodes of the selected object.

To display the object again, double-click the  button for a related object.

Deleting an Object or Relationship

To delete an object or a relationship in a diagram, select it on the diagram canvas and choose **Delete Selected** in the right-click menu.


Assigning Tags

You can assign tags manually in Diagrammer or automatically, to all the objects in a cluster. Follow the instructions below to assign tags manually.

1. Select the objects you want to assign tags on the diagram canvas and click **Assign Tags** in the right-click menu. The Entity Tag Browser window opens.
2. In the Entity Tag Browser window, select the tag(s) you want to assign and click **Assign**.
3. If you are assigning a tag to a legacy source file, check **Apply to Derived Objects** to assign the selected tag(s) to all objects generated from the source file.
4. Click **OK** to assign the selected tags.

Assigning Business Names




To assign a business name and business description to an object, select the object and choose **Set Business Attributes** in the right-click menu. A dialog box opens, where you can enter the business name and business description. To unassign a business name or business description, simply delete the value in the dialog box.

 **Note:** If multiple objects are selected, Diagrammer displays the dialog box as many times as there are objects.

Displaying Labels for Multiple Relationships

The Diagrammer denotes that an object has multiple relationships with another object by displaying a number in parentheses (n) along the relationship line, where n represents the number of relationships. Double-click the parenthesis to display the relationship labels. Double-click the relationship labels to hide them again.


Hiding Relationship Labels


To hide relationship labels on the canvas, click the  button on the tool bar and choose **Hide Relationship Labels** in the drop-down menu. To show the relationship labels again, click the  button on the tool bar and choose **Show Relationship Labels** in the drop-down menu. To show hidden nodes as well as hidden relationship labels, click the  button on the tool bar and choose **Unhide All** in the drop-down menu.


Displaying the Legend for a Diagram

To display the legend for a diagram, click the  button on the tool bar and choose **Show Legend** in the drop-down menu.

Zooming

To zoom in interactive mode, click  button on the Diagram pane tool bar, then drag the mouse over the diagram to zoom in on it.

To zoom in marquee mode, click  button on the Diagram pane tool bar, then drag the mouse over the diagram to draw a marquee. The Diagrammer displays the portion of the diagram inside the marquee.

To fit the diagram in the Diagram pane, click the  button on the Diagram pane tool bar.


Moving a Diagram


To move a diagram in the Diagram pane, click the  button on the Diagram pane tool bar to choose panning mode, then hold down the left mouse button and drag the mouse to move the diagram.

Clearing a Diagram



To clear the Diagram pane, choose **Diagram > Clear**.

Saving a Diagram

Click the  button on the Diagram pane tool bar to save a diagram in one of the supported diagram types. A Save dialog opens, where you can specify the name, type, and location of the file.

Click the  button on the Diagram pane tool bar to save a diagram in one of the supported image types. A Save dialog opens, where you can specify the image type and characteristics. You may also change the name and location of the file.

Printing a Diagram


Click the  button on the Diagram pane tool bar to print a diagram. A Print dialog opens, where you can specify the properties of the print job. Click the  button to preview the printed diagram.

Generating Diagram Reports

Choose **File > Save Type Report** to display a printable report based on a diagram. Choose **File > Save Blackbox Interface Report** to display a printable report showing the relationships between objects in different black boxes. In the printable report, click **Print** to print the report. Click **Save** to export the report to HTML, Excel, RTF, Word, or formatted text.

Browser Pane

The Browser pane displays the workspace repository in tree form. The current project is expanded one level when you first open the Diagrammer.

 **Note:** To improve Diagrammer startup performance in your next session, hide the Browser pane before you end your current session. Click the close box in the upper righthand corner to hide the pane. To show the pane again, choose **View > Browser**.

Relationships Pane

The Relationships pane displays the relationships in the selected scope, as they appear from left to right in the diagram. Select the relationship for an object in the Relationships pane to navigate to the object in the Diagram pane.

The list includes relationship chains, or *composite relationships*, such as Program[IsDefinedInCobol]Cobol[Includes]Copybook[GeneratesTargetXml]TargetXML. If you chose **View > Potential Incomplete Composite Relationships** when you drew the diagram, the list shows incomplete relationship chains, in which the final object in the chain is unresolved or otherwise unavailable.

Quick View Pane

The Quick View pane lets you browse Interactive Analysis information for the object selected in the Diagram pane. The information available depends on the type of object selected. You see only source code for a copybook, for example, but full Interactive Analysis information for a program. Select the information you want to view for the object from the drop-down in the upper lefthand corner of the pane.

Overview Pane

The Overview pane lets you navigate to a general location in a diagram. It displays the entire diagram, with a frame surrounding the portion of the diagram visible in the Diagram pane. Drag the frame to the area of the diagram you want to view in the Diagram pane. Diagrammer displays the selected area in the Diagram pane.

Excluding Objects Outside the Project

To restrict the diagram to related objects in the current project, choose **View > Exclude Objects Outside Project**. Otherwise, all related workspace objects are included in the diagram.

Showing Incomplete Composite Relationships

A *composite relationship* defines the indirect interaction of two objects. If a job runs a program entry point that belongs to a program, for example, the relationship between the job and program is said to be composite: defined by the chain of relationships between the job and program.

To show incomplete relationship chains in the diagram and list of relationships, choose **View > Potential Incomplete Composite Relationships**. Intermediate objects in the chain are displayed even if the final object in the chain is unresolved or otherwise unavailable. Relationships are displayed in red in the diagram.

Colorizing Project Boundaries

Boundary objects are objects with relationships to objects outside the project. To colorize boundary objects in the diagram, choose **View > Project Boundary Entities**. Boundary objects are displayed in the diagram with a red border. Related external objects are displayed in the diagram with a blue border.

Showing and Hiding Object Labels

The **View > Show in Label** menu choice controls the labels displayed for objects in a diagram. Choose one of:


- **View > Show in Label > Show Type** to display the type of an object in its label.
- **View > Show in Label > Show Name** to display the name of an object in its label.
- **View > Show in Label > Show Business Name** to display the business name of an object in its label.
- **View > Show in Label > Show Business Description** to display the business description of an object in its label.


A check mark next to a choice means that the label will be displayed. The choices are toggles. Click the choice again to hide the information in the label.

Working with Groups

Grouping objects reduces diagram clutter and makes it easy to visualize the roles of the grouped objects in your application. Groups are automatically created when you black-box tagged objects and when you filter or cluster objects.



Grouping and Ungrouping Objects

To group objects manually, select the objects on the diagram canvas, then click the  button on the tool bar and choose **Group Selected** in the drop-down menu. Inclusive groups are allowed.


To ungroup objects, select the group on the diagram canvas, then click the  button on the tool bar and choose **Ungroup Selected** in the drop-down menu.

Expanding and Collapsing Groups


Expand a group to view its contents. Collapse a group to hide its contents.

- If a group is collapsed (including a group in a more inclusive group), double-click it to expand its contents. To expand all the groups in an inclusive group, select the group on the diagram canvas, then click the  button on the tool bar and choose **Expand Groups** in the drop-down menu.
- If a group is expanded (including a group in a more inclusive group), double-click it to collapse it. To collapse all the groups in an inclusive group, select the group on the diagram canvas, then click the  button on the tool bar and choose **Collapse Groups** in the drop-down menu.

To view a diagram consisting only of objects in a group, select the group and choose **Navigate to Child**


Diagram in the right-click menu. To restore the full diagram, click the  button on the tool bar and choose **Return to Parent** in the drop-down menu.

Moving Objects Between Groups

To move objects between groups, expand the groups, then click the  button on the toolbar. Select the object you want to move and drag it to the new group.

Naming Groups

To rename a group, select it and choose **Rename Group** in the right-click menu.

 **Note:** The name does not persist after ungrouping. You cannot rename a black box.

Deleting Groups

To delete a group, select it and choose **Delete Group** in the right-click menu. You are prompted to confirm the deletion. Click **Yes**.


Black-Boxing Tagged Objects

The deeper your understanding of your application, the more confidently you can abstract from its lower-level details to a “bigger picture” view, one that organizes related programs in functional, structural, or other types of groupings: a Customer Maintenance subsystem, for example, in an Order Acceptance application. This is the kind of view a subject matter expert uses to evaluate whether an application does everything it is supposed to do, in the appropriate order.

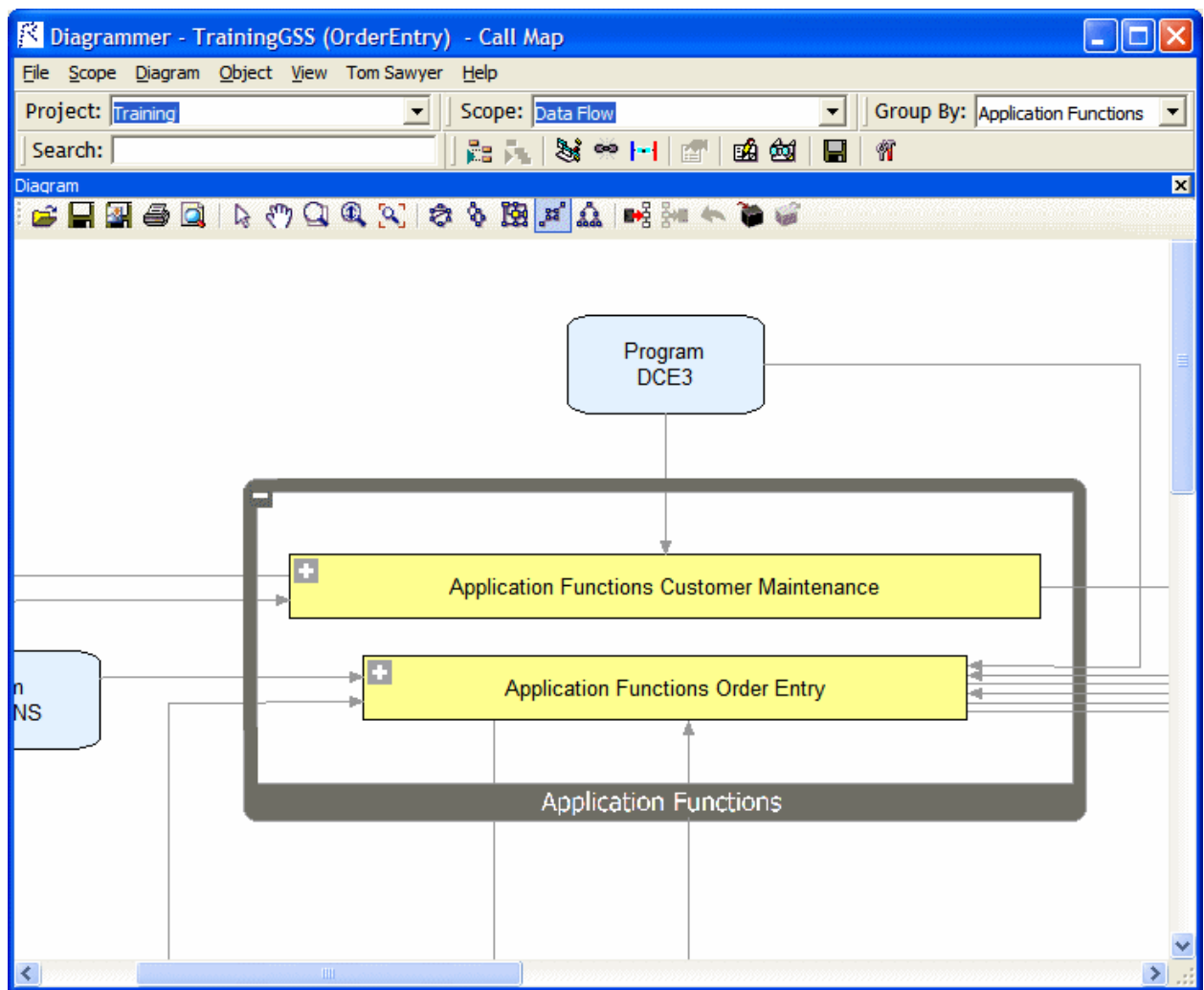
The Diagrammer *black-box* feature lets you assign lower-level objects to higher-level groupings that make it easy to visualize their roles in your application. Your diagram might have one black box for the Customer Maintenance subsystem, another for the Order Entry subsystem, and so forth. Because the details of these relationships are hidden in the black box until you need to view them, the subject matter expert can home in quickly on the higher-order functions you have abstracted from them.

You use the Tag Manager to identify the items in each higher-level grouping. Each grouping can, in turn, reference a more inclusive grouping. If you assign the Customer Maintenance tag to one set of programs, for example, and the Order Entry tag to another, and both tags reference the Application Functions tag, then when you choose Application Functions in the Diagrammer **Group By** drop-down, the Diagrammer puts the programs in black boxes named Application Functions Customer Maintenance and Application Functions Order Entry.

1. To black-box objects in a diagram, choose the tag for the objects in the **Group By** drop-down.



 **Note:** To show only top-level tags in the drop-down, choose **View > Root Tags Only**.


2. Generate the diagram in project or copy-and-paste mode. The figure below shows expected results.




Filtering Objects

A *driver* is an object with many outputs, a startup program, for example. A *utility* is an object with many inputs, DATEFMT, for example. Filtering drivers and utilities from large diagrams reduces diagram clutter and typically provides a better picture of what your application does.

You set thresholds for filtering in the Diagrams User Preferences. If you set the driver threshold for outgoing relationships to 40, for example, then any object with 40 or more outgoing relationship will be considered a driver and displayed outside the main relationship flow. The driver and its related nodes in the relationship flow appear with a  symbol in the diagram. Click the  symbol to display the relationships for the driver and its related nodes again.

You generate a filtered diagram the same way you generate any diagram. The only difference is that when the filtering thresholds are met, Diagrammer displays the Large Dataset Returned dialog. Choose **Enable Node Filtering** in the dialog to draw the diagram with the filter applied. Choose **Disable Node Filtering and Clustering** in the dialog to draw the diagram without the filter applied. A  symbol on the tool bar indicates that a diagram is filtered.




Tip: Use the **Filter Override** options on the  drop-down menu to override the thresholds in the Diagrams User Preferences. If the diagram has already been generated, choosing an override option redraws the diagram with the new filter applied.

Setting Filter Overrides

Use the Filter Override Options window to override the filtering thresholds in the Diagrams User Preferences. If the diagram has already been generated, choosing an override option redraws the diagram with the new filter applied.



Note: These settings do not persist between Diagrammer sessions.


1. Click  on the tool bar and choose **Filter Override Options** in the drop-down. The Filter Override Options window opens.
2. Select **Filters Enabled** to enable filtering.
3. Select **Hide Utilities** to hide objects with many inputs, DATEFMT, for example. Specify the filtering thresholds for inputs and outputs:
 - In the **Incoming Relationships** field, enter the filtering threshold for incoming relationships.
 - In the **Outgoing Relationships** field, enter the filtering threshold for outgoing relationships.
4. Select **Hide Drivers** to hide objects with many outputs, a startup program, for example. Specify the filtering thresholds for inputs and outputs:
 - In the **Incoming Relationships** field, enter the filtering threshold for incoming relationships.
 - In the **Outgoing Relationships** field, enter the filtering threshold for outgoing relationships.
5. Select **Hide High Input Nodes** to hide objects with many inputs that do not meet the incoming relationship threshold for utilities. Specify the thresholds for inputs in the **With Incoming Relationships** field.
6. Select **Hide High Output Nodes** to hide objects with many outputs that do not meet the outgoing relationships threshold for drivers. Specify the thresholds for outputs in the **With Outgoing Relationships** field.



Tip: Select **Reset Values** to restore the filtering thresholds to their settings in the Diagrams User Preferences.

Clustering Objects

Clustering objects on a diagram groups them by their degree of relatedness. Once clusters are generated, you can move objects in and out of the groups, name the groups, and create new clusters as needed. When you are satisfied that a cluster meets your needs, you can assign an existing tag to its members or generate a tag from the cluster itself, each member of which receives the tag.

To enable clustering click  and then click **Clustering Options**. Check if the appropriate filtering thresholds are met, drivers and utilities are grouped in special clusters named "Drivers" and "Utilities." A third special cluster, "Miscellaneous Components," contains relationship flows with too few objects to cluster.


You generate a clustered diagram the same way you generate any diagram. The only difference is that Diagrammer displays the Large Dataset Returned dialog. Choose **Enable Node Clustering** in the dialog to draw the diagram with clustering applied. Choose **Disable Node Filtering and Clustering** in the dialog to draw the diagram without clustering applied.

Creating Clusters Manually

To create a cluster manually, right-click on white space on the Diagrammer canvas. The Clustering dialog opens. Select **Create New Cluster**, then enter the cluster name in the **Cluster Name** field. Click **OK**.

Generating Tags from Clusters


To generate tags based on cluster names, right-click on white space on the Diagrammer canvas. The Clustering dialog opens. Select **Create Tags from Clusters**, then enter the name of the parent tag in the **Top Level tag** field.


 **Note:** The parent tag is required.

Click **OK**. Each member of a cluster is assigned a tag based on the cluster name.

Setting Clustering Factors


Use the Clustering Options window to specify values to use when factoring how closely related cluster objects are.


 **Note:** These settings do not persist between Diagrammer sessions.

1. Click  on the tool bar and choose **Clustering Options**. The Clustering Options window opens.
2. In the **Quality** drop-down, select one of:
 - **Draft** for the lowest clustering quality and best performance.
 - **Proof** for the highest clustering quality and worst performance.
 - **Medium** for the best balance of clustering quality and performance.
3. In the **Balance** combo box, specify the clustering balance factor. The higher the balance factor, the lower the number of relationships between objects in different clusters.
4. In the **Cluster Size Factor** combo box, specify the clustering size factor. The higher the clustering size factor, the larger and fewer the clusters.
5. In the **Max Number of Clusters** combo box, specify the maximum number of clusters. A value of 0 means any number of clusters.
6. In the **Min Number of Clusters** combo box, specify the minimum number of clusters. A value of 0 means any number of clusters.


Setting Diagrams User Preferences

Use the Diagrams tab of the User Preferences window to specify the color-coding used in relationship flow diagrams, the thresholds for clustering and filtering, and performance limits for large diagrams.


1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Diagrams tab.
2. Select **Prompt When Diagram Mode Changes** to be prompted when you switch between a diagram generated for a project and a diagram generated for copied-and-pasted objects.
3. In the Color Scheme pane, click the object type whose color you want to edit. The current background color of the type (if any) is displayed in the Color drop-down.
4. Click the adjacent  button and choose **ForeColor** from the pop-up menu if you want to specify the color of the caption for the object type, or **BackColor** if you want to specify the color of the background for the object type.
5. In the **Shape** drop-down, select the shape of the object type in the diagram. Select **Lock Shape Aspect** to specify that the shape not be adjusted to fit the caption.
6. Click **Performance** button. The Performance Option Limits dialog opens.
7. Deselect **Relayout Subgraphs** if you do not want child groups to be redrawn when you choose a new layout style.

 **Note:** Selecting this option may result in poor performance for very large diagrams.

8. In the **Maximum relationships to return from repository** field, enter the maximum number of relationships to display in the diagram. Relationships are truncated when the maximum is exceeded.

 **Note:** Setting this option too high may result in poor performance for very large diagrams.

9. In the **Force Symmetric Layout (all relationships)** field, enter the maximum number of relationships to display in the diagram before forcing symmetric layout style.

 **Note:** Symmetric layout style typically offers the best performance for very large diagrams. Setting this option too high may result in poor performance for very large diagrams.

10. In the **Offer Auto Clustering Layout (all relationships)** field, enter the maximum number of relationships to display in the diagram before offering to perform clustering.

11. Select **Filter Options Enabled** to enable filtering.

12. In the **Hide Drivers (all relationships)** field, enter the maximum number of relationships to display in the diagram before filtering drivers. (A *driver* is an object with many outputs, a startup program, for example.) Specify the filtering thresholds for inputs and outputs:

- In the **Incoming Relationships** field, enter the filtering threshold for incoming relationships.
- In the **Outgoing Relationships** field, enter the filtering threshold for outgoing relationships.

13. In the **Hide Utilities (all relationships)** field, enter the maximum number of relationships to display in the diagram before filtering utilities. (A *utility* is an object with many inputs, DATEFMT, for example.) Specify the filtering thresholds for inputs and outputs:

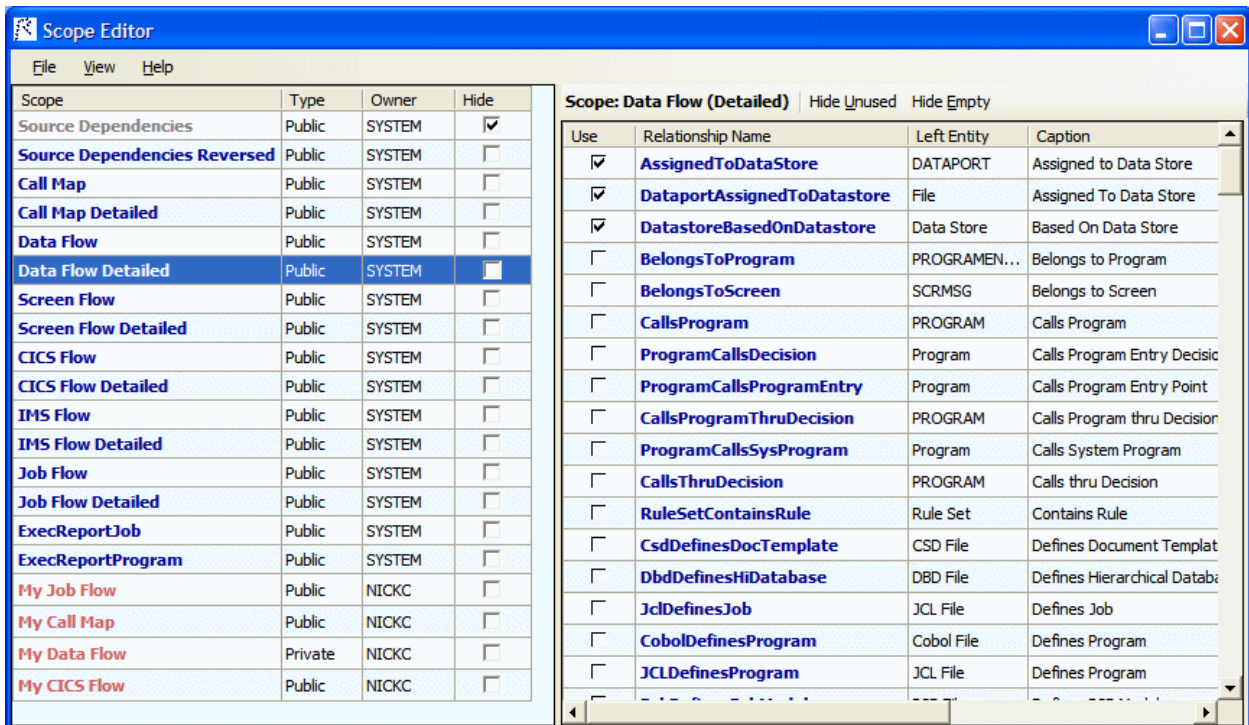
- In the **Incoming Relationships** field, enter the filtering threshold for incoming relationships.
- In the **Outgoing Relationships** field, enter the filtering threshold for outgoing relationships.

14. In the **Hide High Output Nodes (all relationships)** field, enter the maximum number of relationships to display in the diagram before filtering objects with many outputs that do not meet the outgoing relationships threshold for drivers. Specify the filtering thresholds for outputs in the **Outgoing Relationships** field.

15. In the **Hide High Input Nodes (all relationships)** field, enter the maximum number of relationships to display in the diagram before filtering objects that do not meet the incoming relationship threshold for utilities. Specify the filtering threshold for inputs in the **Incoming Relationships** field.

Using the Scope Editor

Use the Scope Editor to manage the scopes in your repository, create custom composite relationships, and define relationship filters. To open the Scope Editor, choose **Scope > Scope Editor**. The Scope Editor opens, with the workspace scopes listed in the lefthand pane and their relationships in the righthand pane. The figure below shows the Scope Editor.



Understanding the Lefthand Pane of the Scope Editor

The lefthand pane of the Scope Editor window lists every scope in the repository. The table below describes the columns in the lefthand pane.

Column	Description
Scope	The name of the scope. Scopes are color-coded as follows: <ul style="list-style-type: none"> Blue means that the scope is available in the Diagrammer Scope drop-down. Gray means that the scope is not available in the Scope drop-down, because it is hidden or because it is a private scope owned by another user. You can copy a private scope, then edit it as you would your own scope Red means that the scope has no relationships associated with it.
Type	The type of the scope, public or private. The type determines availability of the scope in the Diagrammer Scope drop-down: <ul style="list-style-type: none"> A public scope is available to every user of the workspace. A private scope is available only to its owner. You can copy a private scope, then edit it as you would your own scope.
Owner	The owner of the scope. SYSTEM denotes a default scope.
Hide	Whether the scope is hidden in the Diagrammer Scope drop-down. A check mark means that the scope is hidden.

Understanding the Righthand Pane of the Scope Editor

The righthand pane of the Scope Editor window lists every relationship in the repository. The table below describes the columns in the righthand pane.

Column	Description
Use	Whether the relationship is used in the selected scope. A check mark means that the relationship is used.
Relationship Name	The name of the relationship. Relationships for which a filter has been defined are color-coded magenta.
Left Entity	The entity on the left end of the relationship. Entities for which a relationship filter has been defined are color-coded magenta. A tool tip displays the text of the condition.
Caption	The caption of the relationship.
Right Entity	The entity on the right end of the relationship. Entities for which a relationship filter has been defined are color-coded magenta. A tool tip displays the text of the condition.
Class	The class of the relationship, basic or composite: <ul style="list-style-type: none"> • A basic relationship defines the direct interaction of two objects: a program and a program entry point, for example. • A composite relationship defines the indirect interaction of two objects. If a job runs a program entry point that belongs to a program, for example, the relationship between the job and program is said to be composite: defined by the chain of relationships between the job and program.
Occurs	The number of times a basic relationship occurs in the repository. For composite relationships, N/A, indicated with an asterisk (*).

Managing Scopes

Use the Scope Editor to edit the Diagrammer **Scope** drop-down, view scope diagrams, create, edit, copy, and delete scopes, import and export scopes, and more.

Hiding a Scope in the Scope Drop-down

To hide a scope in the Diagrammer **Scope** drop-down (but not remove it from the repository), select the scope in the Scope Editor and put a check mark next to it in the **Hide** column.

Displaying Only the Relationships Used in a Scope

To display only the relationships used in a scope, select the scope in the Scope Editor and click **Hide Unused** button in the righthand pane.

Hiding Empty Relationships

To hide relationships that have no instances in the repository, click the **Hide Empty** button in the righthand pane.

Viewing a Diagram of a Scope

To view the diagram of a scope, select the scope in the Scope Editor and choose **File > Diagram**. You can also view a scope diagram by selecting the scope in the Diagrammer **Scope** drop-down and choosing **Scope > View Scope**.

Viewing a Diagram of a Composite Relationship


To view the diagram of a composite relationship, select the relationship in the Scope Editor and choose **Diagram** in the right-click menu.

Creating a Scope

To create a scope, choose **File > New Scope**. A dialog box prompts you to enter the name, optional caption, class, and owner of the scope. Enter the requested information and click **Save**.


Specifying the Relationships in a Scope

To specify the relationships in a scope, select the scope in the Scope Editor and put a check mark in the **Use** column next to each relationship you want to include in the scope. To save the scope, choose **File > Save**.

 **Note:** You can specify relationships only for a scope you own.

Editing Scope Details

To edit the name, caption, class, and owner of a scope, select the scope in the Scope Editor and choose **File > Edit**. A dialog box displays the current details for the scope. Enter the new information and click **Save**.


 **Note:** You can edit details only for a scope you own.

Copying a Scope

To copy a scope, select the scope in the Scope Editor and choose **File > Copy**. A dialog box prompts you to enter the name, optional caption, class, and owner of the new scope. Enter the requested information and click **Save**. You can copy default scopes and public or private scopes owned by another user, then edit them as you would your own scope.


Deleting a Scope

To delete a scope, select the scope in the Scope Editor and choose **File > Delete**.

 **Note:** You can delete only a scope you own.


Deleting a Composite Relationship

To delete a composite relationship, select the relationship in the Scope Editor and choose **Delete** in the right-click menu.

 **Note:** You can delete only a composite relationship you own.

Importing and Exporting Scopes

To import a scope from an XML file, choose **File > Import**. A dialog opens, where you can specify the scope you want to import. The type of an imported scope defaults to public. The owner defaults to the current user.

 **Note:** If a scope or relationship being imported conflicts with an existing scope or relationship, you are notified and the conflicting item is ignored.

To export a scope to an XML file, select the scope in the Scope Editor and choose **File > Export Selected**. A Save dialog opens, where you can specify the name and location of the scope. To export all the scopes in the repository to an XML file, choose **File > Export All**.

Hiding Columns in the Scope Editor

To hide a column in the Scope Editor, deselect the column name in the **View** menu.

Creating Custom Composite Relationships

A *composite relationship* defines the indirect interaction of two objects. If a job runs a program entry point that belongs to a program, for example, the relationship between the job and program is said to be composite: defined by the chain of relationships between the job and program.

You can use the default composite relationships provided with the system, or use the Scope Editor to create custom composite relationships. Custom composite relationships are available to all users of the workspace.



Note: Custom composite relationships are not copied or exported when a scope is copied or exported.

1. In the Scope Editor, choose **File > New Relationship**.
2. The New Relationship Wizard opens. In the **Enter Relationship Name** field, enter the name of the composite relationship, then select the object you want to appear on the left side of the relationship chain in the Select First Entity pane. Click **Next**.
3. In the Select relationship pane, select the relationship you want to appear next in the relationship chain. Click **Next**.
4. Repeat the preceding step for each relationship you want to include in the chain. When you are satisfied with your choices, click **Finish**. The New Relationship Wizard closes and the new composite relationship is added to the Scope Editor window.

Defining a Relationship Filter


You can define a relationship filter by setting conditions for either side of the relationships defined in a scope you own. You might want a scope to be restricted to programs that have a cyclomatic complexity greater than 300 and that are defined in COBOL source files, for example. The example below shows you how to do that step by step.

Condition definitions are based on the Repository Exchange Protocol (RXP), an XML-based API that you can use to interact with application-level information in the workspace repository.

In the Scope Editor, relationships and entities for which a filter has been defined are color-coded magenta. A tool tip over the filtered entity displays the text of the condition.

1. In the Scope Editor, select the relationship you want to set conditions for and choose:
 - **Left Cond** in the right-click menu to qualify the entity on the left side of the relationship.
 - **Right Cond** in the right-click menu to qualify the entity on the right side of the relationship.
2. The Condition window opens. In the Condition window, click:
 - **attribute** to qualify the entity according to the value of a given attribute. Programs that have a cyclomatic complexity greater than 300, for example.
 - **has (not) related object** to qualify the entity according to a given relationship type. Programs defined in COBOL source files, for example.
 - **add AND condition** to qualify the entity according to inclusive criteria using an AND operator. Programs that have a cyclomatic complexity greater than 300 and that are defined in COBOL source files, for example.
 - **add OR condition** to qualify the entity according to exclusive criteria using an OR operator. Programs that have a cyclomatic complexity greater than 300 or that are defined in COBOL source files, for example.
3. The Condition window displays the shell definition for the selected condition. The following steps describe how to qualify the program entity using an AND operator. The procedure for other entities and conditions is similar.
4. In the definition for the AND condition, click **attribute**. The definition for the attribute condition is displayed:

```
<attr name="..." op="..." arg="..." negate="..." />
```

 **Note:** Click the X in a definition to delete the condition.

5. Click the ellipsis (...) in `name="..."`. The User Input dialog opens. In the User Input dialog, select the Program entity in the **Choose entity** drop-down and the Cyclomatic Complexity attribute in the **Attributes for Program** drop-down, then click **OK**. The attribute is added to the condition definition.

 **Note:** Click **Delete** in the User Input dialog to delete the criterion defined in the dialog.

6. Click the ellipsis (...) in `op="..."`. The User Input dialog opens. In the **Choose new value** drop-down, choose the greater than (>) symbol, then click **OK**. The greater than symbol is added to the condition definition.
7. Click the ellipsis (...) in `arg="..."`. The User Input dialog opens. In the **Enter new value** drop-down, enter 300, then click **OK**. The new value is added to the condition definition:

```
<attr name="Cyclomatic Complexity" op=">" arg="300"
negate="..." />
```

 **Note:**

In a condition definition, `negate` means “not the specified criterion.” Programs that do not have a cyclomatic complexity greater than 300, for example. Click the ellipsis (...) in `negate="..."` to set its value to true. Ignore the field otherwise.

8. In the definition for the AND condition, click **has (not) related object**. The definition for the relationship type condition is displayed:

```
<hasrelated negate="...">
```

9. In the choices for the relationship type condition, click **define relationship type**.

10. The choices for the relationship type are displayed. Click **define relationship**. The definition for the relationship type is displayed:

```
<rel name="..." negate="..." />
```

11. Click the ellipsis (...) in `name="..."`. The User Input dialog opens. In the **Choose entity** drop-down, select the Program entity. In the **Relations for Program** drop-down, select the `IsDefinedInCobol` relationship, then click **OK**. The relationship is added to the condition definition:

```
- <hasrelated negate="...">
- <reltype>
<rel name="IsDefinedInCobol" negate="..." />
</reltype>
</hasrelated>
```

12. The AND condition is now complete. The diagram scope will be restricted to programs that have a cyclomatic complexity greater than 300 and that are defined in COBOL source files. The full condition looks like this:

```
- <cond>
<and negate="...">
<attr name="Cyclomatic Complexity" op=">"
arg="300" negate="..." />
- <hasrelated negate="...">
- <reltype>
<rel name="IsDefinedInCobol"
negate="..." />
</reltype>
</hasrelated>
</and>
</cond>
```

Pruning a Scope

You can prune relationships from a scope you own directly in the Diagrammer window. When you prune a scope, keep in mind that:

- You are deleting relationships from the current scope exactly as if you were deleting them in the Scope Editor window. For that reason, you might want to save the original scope with a different name and use the renamed scope as the basis for the pruned diagram.
 - All the relationships of the selected type are deleted for the selected object, not just the single relationship you selected in the diagram.
1. To prune a scope, select the relationship you want to prune in the diagram and choose:
 - **Prune type for right object** in the right-click menu to delete from the current scope all relationships of the selected type for the right object in the relationship.
 - **Prune type for left object** in the right-click menu to delete from the current scope all relationships of the selected type for the left object in the relationship.
 2. Generate the diagram in project or copy-and-paste mode. The Diagrammer deletes the relationships from the redrawn diagram.

Mirroring a Scope

By default, Diagrammer shows the flow of relationships from a diagrammed object rather than to a diagrammed object. Choose **Scope > Mirror Scope** to show the flow of relationships to the object. Choose **Scope > Mirror Scope** again to return to the original view.

Analyzing Global Data Flow

The Global Data Flow traces incoming and outgoing data flows for a program variable up to a *dataport*, an I/O statement or call to or from another program. You can view the memory allocation and offset for the variable to determine how changes to the variable may affect other variables, and trace assignments to and from the variable across programs.



Note: Projects must have been verified with the **Enable Data Element Flow** option set in the Project Verification options.

The Global Data Flow tool is available in Interactive Analysis by clicking **View > Data Flow**.

Understanding the Global Data Flow Panes

Use the Global Data Flow panes to trace incoming and outgoing data flows for a program variable up to a *dataport*, an I/O statement or call to or from another program. To open the Global Data Flow panes, select a program in the Repository Browser and choose **View > Data Flow**.


By default, all Global Data Flow panes are displayed. If you want to show/hide one of the panes click **View** and then click the name of the pane you want to hide/show.

Data View Pane

For the selected program, the Data View pane shows variable structures, substructures, and fields in hierarchical order from left to right. Double-click a variable name to generate a data flow diagram for the variable in the Data Flow pane, and a list of offsets and memory allocations for the variable and any related variables in the Origin pane.

Data Flow Pane

For the variable selected in the Data View pane, the Data Flow pane displays a diagram that traces its incoming and outgoing data flows up to a *dataport*, an I/O statement or call to or from another program. The selected variable is displayed in red, constants in gray, and dataports in blue.

 **Note:** Make sure you have enabled incoming and/or outgoing data flows appropriately in the Global Data Flow project options.

Two diagram views are available:

- The Expanded View displays as many nodes for the selected variable as it has assignments.
- The Compact View displays a single node, regardless of the number of assignments.

Click the tab for the view you want to display.

Select a dataport and choose **Data Flow > Reconstruct** to display the data flow diagram for the variable identified at the dataport. Select a relationship line to display all the statements that determine the data flow between variables in the Origin pane.






Place your cursor over a variable for a moment to display a tool tip that identifies the memory offset and allocation for the variable. The diagram uses COBOL Analyzer common diagramming features.



Origin Pane

For the variable selected in the Data View or Data Flow panes, the Origin pane displays a list of offsets and memory allocations for the variable and any related variables. For the dataport selected in the Data Flow pane, it displays related variables. For the relationship selected in the Data Flow pane, it displays the intraprogram or interprogram relationship detected by data flow analysis.

Data Flow Relationships

The table below describes the intraprogram and interprogram relationships detected by the data flow analysis tools.

Relationship	Definition	Type	Icon	Description
calls	N/A	interprogram	N/A	A parameter passed in a call to another program.
cast	MOVE A TO B with data conversion	intraprogram		A data item moved to a data item of a different type.
common area transitions	N/A	interprogram	N/A	For Unisys 2200 Cobol, a common-storage data area item passed in a call to another program. Perform Unisys Common-Storage Area Analysis must be set in the project verification options.
comp	STRING A ... INTO B	intraprogram		An arbitrary computation. The result is produced by applying complex rules to the argument, such as STRING.
comp+	ADD A TO B	intraprogram		An addition-like operation: ADD, SUBTRACT, or corresponding parts of COMPUTE.
comp*	MULTIPLY A BY B	intraprogram		A multiplication-like operation: MULTIPLY, DIVIDE, or corresponding parts of COMPUTE.
comp@	MOVE ARRAY (IDX) TO A	intraprogram		An operation with array elements.

Relationship	Definition	Type	Icon	Description
cond	IF A = B ...	intraprogram		Comparison of data items with a symmetric relationship.
cond*	IF A * X = B ...	intraprogram		Comparison of a multiple of a data item with another data item.
const cond	IF A = 1 ...	intraprogram		Comparison of a data item with a constant.
const.move	MOVE 1 TO B	intraprogram		A constant moved into a data item.
const.comp	ADD 1 TO B	intraprogram		An arithmetic operation with constants.
const.init	03 A ... VALUE 1	intraprogram		A data item initialized by a constant.
DMS records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via Unisys DMS database records.
files	N/A	interprogram	N/A	Data communication via files. Traced only when corresponding JCL, ECL, FCT, or CSD files are verified.
files in jobs	N/A	interprogram	N/A	Data flow in JCL datasets when <i>files</i> is selected.
input port	N/A	intraprogram		A data item in which data is received.
move	MOVE A TO B	intraprogram		A data item moved to a data item of the same type.
network records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via network records.
output port	N/A	intraprogram		A data item from which data is sent.
screens	N/A	interprogram	N/A	Data sent to a screen by one program and received in a screen by another.
screen definitions	N/A	interprogram	N/A	Data flow in screen fields when <i>screens</i> is selected.
start	N/A	intraprogram		The startup item in an Impact pane consolidated analysis.
used	MOVE ... TO A ... MOVE A TO ...	intraprogram		A value assigned in a statement used as an argument in another statement.

Assigning Business Names Manually

To assign a business name and business description to an object, select the object and choose **Set Business Attributes** in the right-click menu. A dialog box opens, where you can enter the business name

and business description. To unassign a business name or business description, simply delete the value in the dialog box.

Setting Global Data Flow User Preferences

Use the **Interactive Analysis > Data View** tab of the User Preferences window to specify the color-coding used in the Data View pane and the order of display of variables with the same offset.

1. Choose **View > User Preferences**. The User Preferences window opens. Click the **Interactive Analysis > Data View** tab.
2. In the Same offset order group box, specify the order you want variables with the same offset to be displayed in the Data View pane. Choose:
 - **Data item size** if you want variables with the same offset to be displayed in size order, largest to smallest.
 - **Appearance in the source code** if you want variables with the same offset to be displayed in the order they appear in the source code.
3. Click the arrow beside the drop-downs for **Free Space Color**, **Used Space Color**, and **FILLER Color** to specify the color of each item in the Data View pane.
4. Select **Display Business Names** to display business names for objects in the Data View pane.

Setting Global Data Flow Project Options

Use the Global Data Flow tab of the Project Options window to specify whether data flow diagrams include literals, variable values set at initialization, and Unisys Cobol common storage variables; whether they show incoming, outgoing, and recursive data flows; and the number of nodes they display.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Global Data Flow tab.
2. In the Relationships pane, choose any combination of:
 - **Literal Flow** to include literals in the data flow diagram for the selected variable
 - **Initial Values** to include variable values set at initialization in the data flow diagram for the selected variable.
 - **Common Area Transition** to include Unisys Cobol common storage variables in the data flow diagram for the selected variable. Common storage variables are not explicitly declared in CALL statements.
3. In the Directions pane, choose any combination of:
 - **Causes** to generate data flows into the selected variable.
 - **Consequences** to generate data flows from the selected variable.
 - **Self-Dependencies** to show recursive data flows for the selected variable.
4. In the **Node Limit** combo box, enter the maximum number of diagram nodes you want to display. You might restrict the number of nodes to improve performance or make the diagram easier to read. You can also use the slider on the Node Limit tool bar to adjust the number of displayed nodes.



Note: To include Unisys Cobol common storage variables, you must have verified the project with the **Perform Unisys Common-Storage Analysis** option set in the project verification options.



Note: All children of a parent node are displayed even if some of them exceed the specified maximum.

Estimating Complexity and Effort

COBOL Analyzer Legacy Estimation tools let you compare programs based on weighted values for selected complexity metrics. The metrics used in the calculation are a combination of industry standard and COBOL Analyzer-generated statistics. Based on the comparison, you can develop a credible estimate of the time required to satisfy a change request or perform other maintenance tasks.



Note: For definitions of the supported complexity metrics, see the "Complexity Metrics" section of this help.

Viewing Complexity Metrics

Use the Complexity Metrics report to compare raw complexity values for the objects in your project. To open the Complexity Metrics report, select a project in the Repository Browser and choose **Analyze > Complexity**.

When the Complexity Metrics window opens, choose the type of object you want to compare in the **Entity Type** drop-down. To generate the report in HTML, choose **File > Report**. The figure below shows the Complexity Metrics window.

The screenshot shows a window titled "Complexity Metrics - TrainingAccess2 (TrainingAccess2)". The "Entity Type" is set to "Program". The table below lists the metrics for various programs.

Name	Source Name	Lines Of Code	Executable Statements	Operators	Op
AR7100	AR7100.CBL	277	25	25	
AR7200	AR7200.cbl	265	30	30	
AR7300	AR7300.cbl	371	30	30	
CHECKDIGIT	CheckDigit.CCP	130	29	103	
CHECKDIGIT\$COMPL	CheckDigit\$compl.CCP	741	290	434	
CUSTINQ1	CUSTINQ1.ccp	304	55	71	
CUSTOMNT1	CUSTOMNT1.ccp	727	231	286	
DCE1	DCE1.CCP	788	316	534	
DCE2	DCE2.CCP	788	316	534	
DCE3	DCE3.CCP	251	45	62	
DCE4	DCE4.CBL	197	25	25	
DECISIONS	DECISIONS.CBL	64	8	8	
FAAPLTPI	FAAPLTPI.ccp	124	14	14	
GETINV	GETINV.ccp	99	9	9	
INVMENU	INVMENU.ccp	300	45	62	
NUMEDIT	NUMEDIT.cbl	118	30	47	
ORDRENT1	ORDRENT1.ccp	878	316	534	
PRODFILL	PRODFILL.ccp	257	24	24	
PRODMNT1	PRODMNT1.ccp	727	231	286	
SYSERR	SYSERR.ccp	123	9	9	

Setting Complexity Metrics User Preferences

Use the Complexity Metrics tab of the User Preferences window to specify the metrics displayed in the Complexity Metrics report for each type of object in your application.

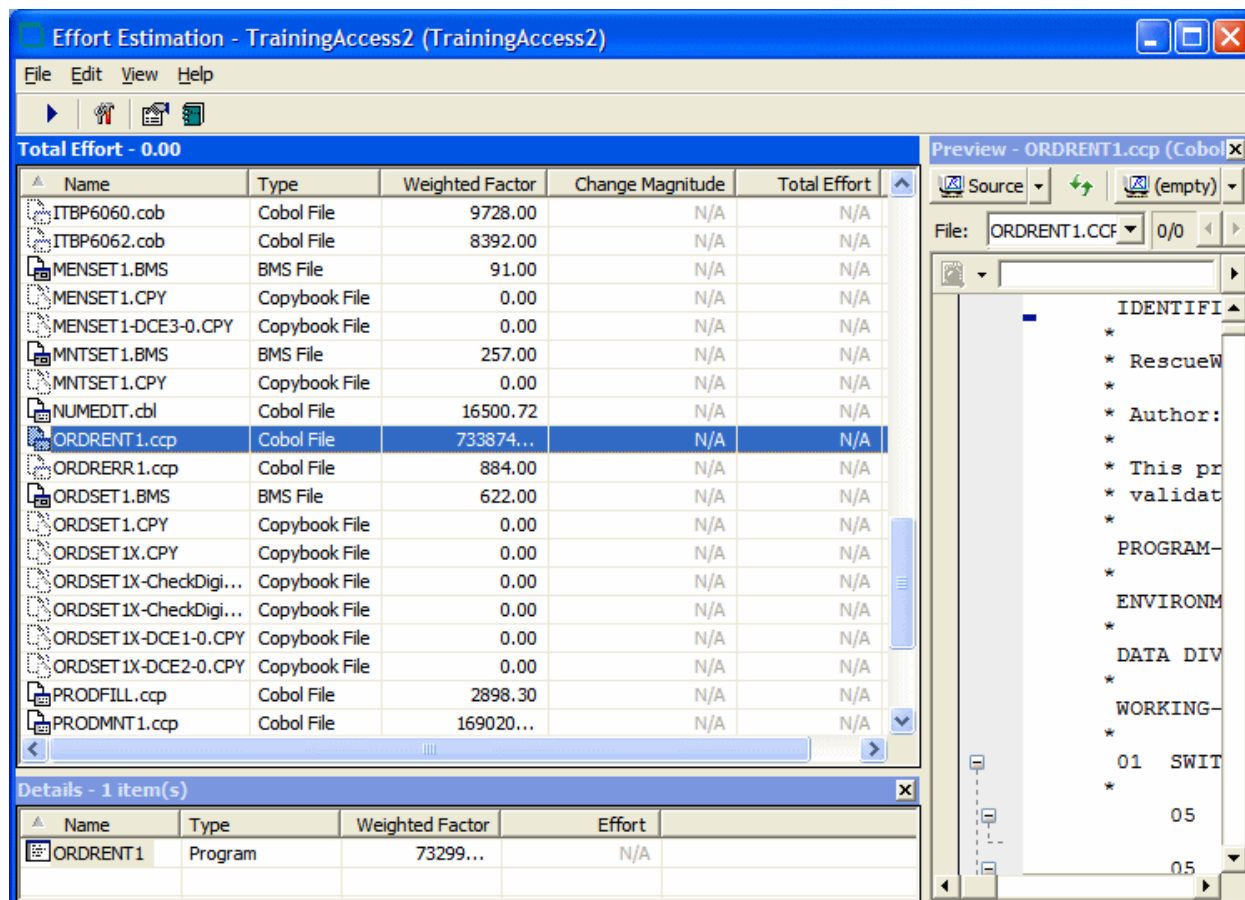
1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Complexity Metrics tab.
2. In the **Source Type** drop-down, select the type of object you want to display complexity metrics for.
3. In the Attributes pane, select each complexity metric you want to be displayed for the selected object type.

Estimating Effort

Use the Effort Estimation tool to compare source files based on weighted values for selected complexity metrics. To open the Effort Estimation tool, select a project in the Repository Browser and choose **Analyze > Effort**.

Select the source files types to be included in the calculation and the weighted values for the metrics in the Effort Estimation options. When you are satisfied with your selections, choose **File > Compute Effort**.

When the effort estimation report is displayed, select a source file in the Effort pane to show the effort statistics for its generated objects in the Details pane and the Interactive Analysis information for the file in the Preview pane. To generate the report in HTML, choose **File > Report**. The figure below shows the Effort Estimation window.



Setting Effort Estimation Options

Use the Effort Estimation tab of the Project Options window to specify the file types included in the effort estimation calculation and the complexity metrics for each type. These options also control the percentage factor for the change magnitudes used in the calculation.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Effort Estimation tab.
2. In the Factors pane, select each source file type you want to include in the calculation.
3. For each source file type selected in the Factors pane, click **Attributes** to edit the complexity metrics used in the calculation.
4. In the Attributes window, select each complexity metric you want to use in the calculation, then enter its weight in the **Weighting Factor** field. For example, if you want Cyclomatic Complexity to have twice the weight of Conditional Complexity, set the weighting factor for Cyclomatic Complexity to 2 and the weighting factor for Conditional Complexity to 1.
5. To set the percentage factor for the change magnitudes used in the calculation, enter the percentage you want to be applied in the combo box for each change magnitude.

Specifying the Change Magnitude for a Source File

Suppose you are planning to implement a change request and want to know how long it will take to complete the change. In that situation, you typically run an effort estimation report for your project based on weighted values for selected complexity metrics.

But what if your own analysis of the project shows that a given program will actually take much less time to change than the weighted calculation would suggest. The program might have thousands of source lines, for example, increasing its calculated complexity, while actually being very easy to modify.

A change magnitude is a way of overriding the calculated value for a source file. Your "subjective" estimate of the effort involved, Small, Medium, Large, Extra Large, becomes an input to the effort calculation, along with the weighted values.

You can specify a change magnitude in the Interactive Analysis Source or Context pane, or in the Effort Estimation tool itself. Select the source file and choose **Set Change Magnitude** in the right-click menu. Set the change magnitude to S for Small, M for Medium, L for Large, or XL for Extra Large. The effort estimation calculation for the selected file is automatically updated.

Identifying Classes of Data Items with Change Analyzer

Change Analyzer identifies the *class* of data items used to perform a business function in a legacy application. Amongst other uses, it lets you answer the kind of "What if?" questions posed in the past by industry-wide changes for Y2K, Zip+4, and the Euro dollar: "What if I change the type of this variable, or the length of this field? What other fields in the class will I also have to change?"

For each field in the class, you can use Change Analyzer to perform an impact trace that shows the flow of data to the field, and to detect and resolve program ports. You can generate reports showing source entities that might require modification, lines of affected code, and the like.

 **Note:** Projects must have been verified with the **Enable Data Element Flow** option set (see *Setting Project Verification Options*).

The project options on the **Impact > Relationships** tab of the Project Options window control the relationships detected during synonym and impact trace processing.

Understanding Data Item Classification

Suppose your organization is considering adding support for a new currency and that you are going to have to expand the existing exchange rate data field from 9(5)V9(3) to 9(5)V9(6) to accommodate the currency. You will need to know the data fields that are affected in the database, intermediate fields that might contain or use the exchange rate field in calculations, and so forth.

Seed Fields

Use Change Analyzer to search for the exchange rate field. The object of the search is called a *seed field*:

- If the application uses field names like EX-RATE, EXCH-RATE, EXCHANGE-RATE, and RATE-OF-EXCHG, you would search for data names that contain *EXCH* or *RATE*.
- If you know that some fields already contain the required number of decimal positions, and are interested only in those that don't, you might further limit the search by filtering on the PICTURE clause format of the variable, selecting only data items that have a format of 9(5)V9(3).
- You might limit the search even further by choosing fields that have a given initial value.
- If you know there are data fields that will meet the search criteria for name, but are not what you are looking for, you might set up a list of names to exclude, such as INTEREST-RATE and *PRORATE*.

Synonyms

Once you have found the exchange rate field, you can use Change Analyzer to detect its *synonyms*. A synonym is a data field whose value is related to the value of a seed field: a field whose value is assigned by a MOVE or REDEFINE statement, for example. In these cases, if you increase the size of the seed field, you will probably need to increase the size of its synonyms as well.

Analyzing Seed Fields and Synonyms

Not every seed field or synonym will be affected by a proposed change. You need to examine the source code for each instance of the field carefully in the Change Analyzer Source pane and in the Interactive Analysis tools accessible from the Source pane. You may want to perform an impact trace showing the flow of data to the field as documentation, or to ensure that you have accounted for each instance.

Seed Lists

Once you determine which fields are affected by a proposed change, you can move the fields between seed lists: Working and Affected, for example. You can generate reports based on your lists, as well as reports showing the affected source code in context.

Getting Started in Change Analyzer

Sample Usage of Change Analyzer

This section describes a sample use of Change Analyzer that should help you get oriented in the tool. Assume that you want to evaluate the impact of changing the YEAR field in your application.


1. In the Repository Browser, select the project you want to analyze and then click **Analyze > Change Analyzer**. This opens the **Change Analyzer** dialog box.
2. Click **Tools > Analysis Options**. This opens the **Options - Change Analyzer** dialog box.
3. Check **Use Cross Program Analysis** and then click **OK**.

4.

Step 1

Click **Step 1: Find Change Candidates** (). This opens the **Search** dialog box.

5.

Click **New Criterion** () on the tool bar. This opens the **Search** dialog box.

6. Type YEAR and then click **OK**. The new criteria, **YEAR** is added to the **Change Analyzer** tab.


7. Click **YEAR** and then type *DATE* in the **Name Like** tab.

8. Click **Find All Constructs**. The number of constructs found are displayed in a dialog box.

9. Click **OK**. The **Change Candidates** pane now displays a list matching objects.

10.

Step 2

Click **Step 2: Find Fields Affected by the Change Candidates** () and then **Find Affected Fields for All Programs**. If a messages pops up informing you that all existing analysis results will be cleared, click **OK**. The **Status** dialog box appears and displays progress.

11. In the **Change Candidates** pane, a list of change candidates and nested affected fields are displayed. Clicking on an affected field will display more information in the **Fields affected by** pane. Double-clicking on an affected field in either the **Change Candidates** or **Fields affected by** panes will display the relevant code in the **Source** tab.

12. In the **Fields affected by** pane, right-click any of the affected fields, and then click **Trace**. More information on the affected field is displayed in the **Impact Tracing** pane as well as displaying the relevant lines of code in the **Source** pane.

13. You can use this to examine the declarations for each affected field to determine whether the field will be affected by the proposed change.

14. Click **Tools > Reports**. This opens the **Report Selection** dialog box.

15. In the **Report Type** list, select **Report On The Whole List** and click **OK** and then click **Yes** if a message is displayed asking you if you want to continue. Change Analyzer displays a report for the affected fields.

16. You can now print or save the report.

Understanding the Change Analyzer Window

Use the Change Analyzer to identify classes of data items. To open the Change Analyzer, select a project in the Repository Browser and choose **Analyze > Change Analyzer**.

By default you see four panes which you can adjust - Programs, Lists, Synonyms and Ports, and Source.

Change Candidates Pane

The **Change Candidates** pane lists each program that contain matches to your search criterion. The number in parentheses after the program name indicates the number of affected fields in the program. Expanding the program node displays all the parent field names that matched your search criterion.


Field Pane

The Field pane displays a context dependent list of affected fields. The fields displayed is dependent on whether you click on a program displayed in the **Change Candidate** pane or one of the fields listed under a program. If you click on a program in the **Change Candidate** pane, only parent fields are displayed in the Field pane. Clicking on a program's field in the **Change Candidate** pane will displays all of its affected fields in the Field pane.

Clicking a field will display its declaration in the **Source** pane.

The table below describes the columns in the Field pane:

Column	Description
Name	The name of the field.
Length	The length of the field.
Value	The value of the field.
Picture	The format of the field.
Normalized Picture	The normalized picture of the field.
Comment	The comment for the field.
Program File	The program in which the field is used.
File	The source file that contains the declaration for the field.
Usage	The value of the USAGE clause.
Normalized Usage	The format of the data item in computer storage.

You can filter the field list by clicking the  icon in a column and selecting an item you want to filter by.



Tip: You can specify your own comment by right-clicking a field and then click **Comment**. This opens the **Edit Comment Field** dialog box. In the **Comment** field, type a comment and then click **OK**. This can be used to provide information on working process and can also be used for filtering.

Selecting Fields

Click a field in the list to select it. Use **Ctrl +** click to select multiple fields. Use **Shift +** click to select a range of fields. To select all the fields on a list, click the tab for the list and click **Lists > Select All**.

Editing the Comment for a Field

To edit the comment for a field, select the field and click **Lists > Comment....** A dialog opens where you can edit the comment.

Clearing Fields

To clear a field from a list:

1. Click a field from the list.
2. Click **Lists > Clear** and then choose between:
 - **Clear This Term Only** to clear the selected field.
 - **Clear All List Contents** to clear the whole list.

Impact Tracing Pane

The **Impact Tracing** pane displays a tree view of an affected field and its relationship to other synonyms, parent, child, or moved fields.

To display the synonyms and/or the impact trace, click the required field in the Field pane, this highlights the field row. Right-click the field row and then click **Trace**.

To delete an impact trace, select it in the tree and click **Delete**.

To clear the Impact Traces tree, click **Delete All**.

Source Pane

The Source pane lets you browse information for the object selected in the Lists or the Impact Tracing pane. You can select the information you want to view for the object from the drop-down in the upper lefthand corner of the pane. The available options are:

- Source
- Context
- Code Search
- Bird's Eye
- Program Control Flow
- Flowchart
- Data View
- Data Flow
- Impact
- Diagrammer
- Properties.

Usage is similar to that for the Source pane in Interactive Analysis. For Interactive Analysis usage information, see *Analyzing Programs* in the product documentation set.

Searching for Change Candidates in Change Analyzer

The Change Analyzer search facility contains two tabs:

- The General tab opens the Interactive Analysis advanced search facility.
- The Change Analyzer tab opens a scoped version of the advanced search facility for Change Analyzer.


Ordinarily, the scoped version of the tool should be sufficient for most searches. If you are already familiar with the advanced search facility, however, you may want to use it instead of the scoped tool.



Note: Change Analyzer returns only constants and literals found in the Procedure section.

1.



Select a file from the list of programs and click  (Step 1: Find Change Candidates). A message informing you that all existing search and analysis results will be cleared is displayed in a dialog box. Click **OK**.

2. The Change Analyzer **Search** window opens. Click the **Change Analyzer** tab. The Change Analyzer tab displays a list of recognized search criteria.



Note: Define a new criterion as described in the help for the advanced search facility.

3. Click a criteria to edit its definition in the **Criterion** tabs. Each tab specifies a condition in the definition. The definition can consist of any combination of conditions.
4. For each condition, enter a list of patterns you want to match, one pattern per line. You can use wildcard patterns used in LIKE statements by Visual Basic for Applications (VBA). Select:
 - In the **Name Like** tab, specify patterns that are like the name of the field you are searching for.
 - In the **Name Not Like** tab, specify patterns that are unlike the name of the field you are searching for.
 - In the **Picture Like** tab, specify patterns that are like the format of the field you are searching for.
 - In the **Value Like** tab, specify patterns that are like the initial value of the field you are searching for.
5. Check **Use OR Connection Between Lists** if you want the conditions specified in the tabs to have the OR operator applied. If this field is unchecked, the conditions have the AND operator applied.
6. Check **Used Data Items Only** if you want the search results to consist only of fields that are used in the selected programs. If you do not select this option, search results include fields that are declared but not used.
7. In the **Search In** list on the tool bar, you can choose from the following:
 - The selected file only.
 - The selected file with all included files.
 - The selected construct - Declaration.
 - All objects in the project.
 - A Code Search report.
8. Click **Find All Constructs** to execute the search.

Change Analyzer returns the change candidates for the criterion in the starting point list. Any previous results are deleted.

Creating Projects in Change Analyzer

You can create a project directly in Change Analyzer from the results of your analysis. The project contains only source files with fields in the selected list.

1. Click **Tools > Add to Project**. This opens the **Select List** window.
2. Click the list you want to add to the new project and click **OK**. This opens the new project name dialog box.
3. Type the name of the new project and click **OK**. This opens the **Change Magnitude** window.
4. Check **Automatically calculate Change Magnitude** if you want Change Analyzer to set change magnitudes for the listed source files based on the ranges specified for the fields in the **Change Magnitude ranges** group. The Change Analyzer settings will override any existing change magnitudes for the files.
5. Click **OK**. The new project is created and displayed in the **Repository** pane.

Setting Change Analyzer Options

To specify the default depth of synonym and impact trace processing and the amount of information to display:

1. Click **Options > Project Options**. The Project Options window opens. Click the **Change Analyzer** tab.



Note: You can also set the Change Analyzer from the Change Analyzer window. To do so, select **Tools > Analysis Options**.

2. In the Synonyms pane, specify the depth of synonym and impact trace processing in the **Default Depth** combo box. In this pane you can also select to use file descriptors in the analysis, use cross-program analysis and/or if you want to have a cross-program analysis warning before starting the analysis. When the **Check for variables that write to synonym** option is checked, the analysis traces both variables that use the synonym and variables that affect the synonym. If it is unselected, the analysis traces only the variables that use the synonym. Note that unselecting the **Use Cross-Program Analysis**, **Use File Descriptors in Analysis** and **Check for variables that write to synonym** options increases the analysis performance.
3. In the Affected Code Report pane, specify in the **Neighborhood Size** combo box the number of lines of unaffected code you want the Affected Code report to display above and below the line of affected code. (The line of affected code is displayed in bold in the report.) Select **Show unused datanames** if you want the report to include unused data fields.

Generating Change Analyzer Reports

To generate HTML reports in Change Analyzer, click **Tools > Reports**. The Report Selection dialog opens. In the Select Lists pane, select the list you want to report on, then select the report type in the **Report Type** drop-down. Choose:

- **Report on the Whole List** to report the contents of the list.
- **Report on Selected Program** to report the contents of the list filtered by the program selected in the Programs pane.
- **Metrics Report** to report, for each program with a list item, the percentage of declarations in the list relative to the total number of declarations in the program.
- **Affected Code Report** to report, for each program with a list item, code that would be impacted by changing the definition or usage of items in the list. The line of affected code is displayed in bold in the report.



Note: Use Change Analyzer project options to specify the amount of information to display in the Affected Code Report.

Click **Generate Report**. Change Analyzer displays the report.

Repository Exchange Protocol Syntax

The Repository Exchange Protocol (RXP) is an XML-based API that you can use to interact with application-level information in the workspace repository. This part of the documentation describes the RXP query syntax and provides examples of its use.

Query Syntax

An RXP query consists of the tags and attributes described in this section.

Query

```
<query [name='QueryName']> { Object } </query>
```

Objects

```
<object [global='false']>
[ <objecttype> ObjectTypeCondition </objecttype> ]
[ <cond> Condition </cond> ]
[ <fetchtype as='FieldName' /> ]
[ <fetchid as='FieldName' /> ]
```

```
[ <fetchdisplay as='FieldName' /> ]
[ <fetchorigin as='FieldName' /> ]
[ <fetchsource as='FieldName' /> ]
[ <fetchsize as='FieldName' /> ]
{ <fetchconst type='FieldType' value='Constant'
as='FieldName' /> }
{ <fetchattr attr='AttrName' as='FieldName' /> }
[ <related [countas='FieldName'] [optional='true']>
RelatedSpec
</related>]
</object>
```

ObjectTypeCondition

```
<typeset flag='FlagName' [negate='true'] />
| <type name='EntityName' [negate='true'] />
| <and [negate='true']> { ObjectTypeCondition }
</and>
| <or [negate='true']> { ObjectTypeCondition }
</or>
```

RelatedSpec

```
[ <reltype> RelTypeCondition </reltype> ]
[ <cond> Condition </cond> ]
[ <fetchtype as='FieldName' /> ]
{ <fetchattr attr='AttrName' as='FieldName' /> }
Object
```

RelTypeCondition

```
<relset flag='RelFlagName' [negate='true']
[incoming='true'] />
| <rel name='RelationName' [negate='true'] />
| <and [negate='true']> { RelTypeCondition } </and>
| <or [negate='true']> { RelTypeCondition } </or>
```

Condition

```
<attr name='AttrName' op='Operation'
arg='Argument' [negate='true'] />
| <hasrelated [negate='true']> RelatedSpec
</hasrelated>
| <id equals='Integer' [negate='true'] />
| <id in='Integer{,Integer}' [negate='true'] />
| <source equals='String' [negate='true'] />
| <source in='String{,String}' [negate='true'] />
| <origin equals='SID' [negate='true'] />
| <origin in='SID{,SID}' [negate='true'] />
| <and [negate='true']> { Condition } </and>
| <or [negate='true']> { Condition } </or>
```

EntityName

The name of a repository entity.

AttrName

The name of an entity attribute.

FieldName

The field name of the returned record set.

Operation

= | <> | > | >= | < | <= | like | in | between



Note: Since the RXP queries are XML constructs, make sure you use `<` and `>` instead of the less-than and greater-than signs.

FlagName

LEGACY | PROGRAMCODE | SYSTEM | KNOWLEDGE | GENERATED | EXTRACT | COMPOSITE

RelFlagName

REFER | USE | GENERATE | PRODUCE

RelationName

The name of a relationship.

Argument

The argument of the operation. Depends on both argument type and operation.

QueryName

A string.

Example 1

This example queries the repository for the object ID and parse status of the GSS.cbl source file:

```
<query name="Select a COBOL object by name">
  <object>
    <objecttype>
      <type name="COBOL" />
    </objecttype>
    <fetchid as="ID" />
    <fetchattr name="ParseStatus" as="Parsed" />
    <cond>
      <attr name="Name" op="=" arg="GSS.cbl" />
    </cond>
  </object>
</query>
```

Example 2

This example queries the repository for copybooks used in three Cobol programs:

```
<query name="Find COPYBOOKs used in given programs">
  <object>
    <objecttype>
      <type name="COBOL" />
    </objecttype>
    <cond>
      <attr name="Name" op="in"
        arg=" 'GSS1.CBL' , 'GSS2.CBL' , 'GSS3.CBL' " />
    </cond>
    <related>
      <reltype>
```

```
<reset flag="USE" />
</reltype>
<object>
  <fetchid as="ID2" />
  <fetchtype as="ObjectType2" />
  <fetchdisplay as="ObjectName2" />
</object>
</related>
</object>
</query>
```

Portability Assessment

Portability Assessment lets you quickly generate HTML reports to identify points of interest for migrations. All reports are presented under an easy to navigate main page. To run the report:

- Start COBOL Analyzer.
- Click **Reports > Portability Assessment**.

The Portability Assessment window opens.

- Select the types of HTML reports you want to generate. Use the checkbox next to each report or check **Enable/Disable All**.
- Enter the **Folder Name** and **Folder Location** where the report will be generated.
- Click **Start** to generate the reports or **Close** to quit.

When the report is ready a message is displayed asking if you want to see the report.

- Click **Yes** to view the report.

The results are displayed in your browser. Navigate through the reports. Check **Hide empty reports** to hide/display empty reports. The reports are organized in tables. Click on the heading of the columns to sort the data.



Note: Some of the reports may be available as advanced search queries in Interactive Analysis or in the Repository Browser > Query Repository feature. Running these queries interactively or editing them will not affect the results generated in the Portability Assessment report.

Quality Assessment

Quality Assessment lets you quickly generate HTML reports to monitor applications for conformity to quality rules or to identify candidates for quality improvements. All reports are presented under an easy to navigate main page. To run the report:

- Start COBOL Analyzer.
- Click **Reports > Quality Assessment**.

The Quality Assessment window opens.

- Select the types of HTML reports you want to generate. Use the checkbox next to each report or check **Enable/Disable All**.
- Enter the **Folder Name** and **Folder Location** where the report will be generated.
- Click **Start** to generate the reports or **Close** to quit.

When the report is ready a message is displayed asking if you want to see the report.

- Click **Yes** to view the report.

The results are displayed in your browser. Navigate through the reports. Check **Hide empty reports** to hide/display empty reports. The reports are organized in tables. Click on the heading of the columns to sort the data.



Note: Some of the reports may be available as advanced search queries in Interactive Analysis. Running these queries interactively or editing them will not affect the results generated in the Quality Assessment report.

Analyzing Programs

Introducing Interactive Analysis

Much of the power of COBOL Analyzer resides in a set of program analysis tools collectively called Interactive Analysis. Interactive Analysis lets you analyze legacy programs interactively, by examining synchronized, complementary views of the same information: source, context, impacts, and so forth. You can use Interactive Analysis to analyze procedure and data flows, stage program analyses, create a project glossary, extract business rules, and much more.

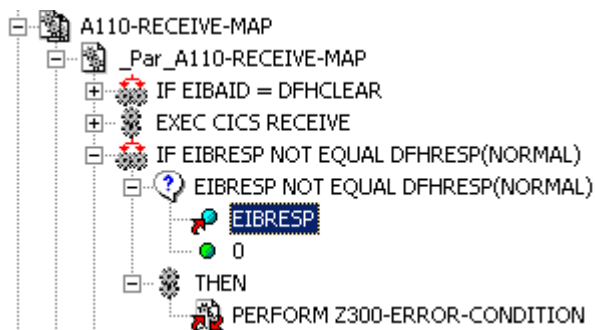
Interactive Analysis is designed primarily for programs, but you can also use it to analyze map files, JCL or ECL files, IDMS schemas, and the like.

Understanding Interactive Analysis Models

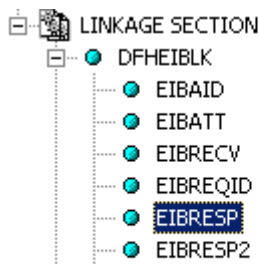
For each object that comprises an application, Interactive Analysis generates a *construct model* that defines its syntax. The construct model shows in abstract form how the syntactical constructs that comprise the object (its sections, paragraphs, statements, conditions, variables, and so forth) are related.

A variable, for example, can be related in the construct model to its declaration, a dataport (if it is used in an I/O statement), or a condition (if the condition uses an expression of which the variable forms a part). You view the construct model for a source file in the Interactive Analysis Context pane.

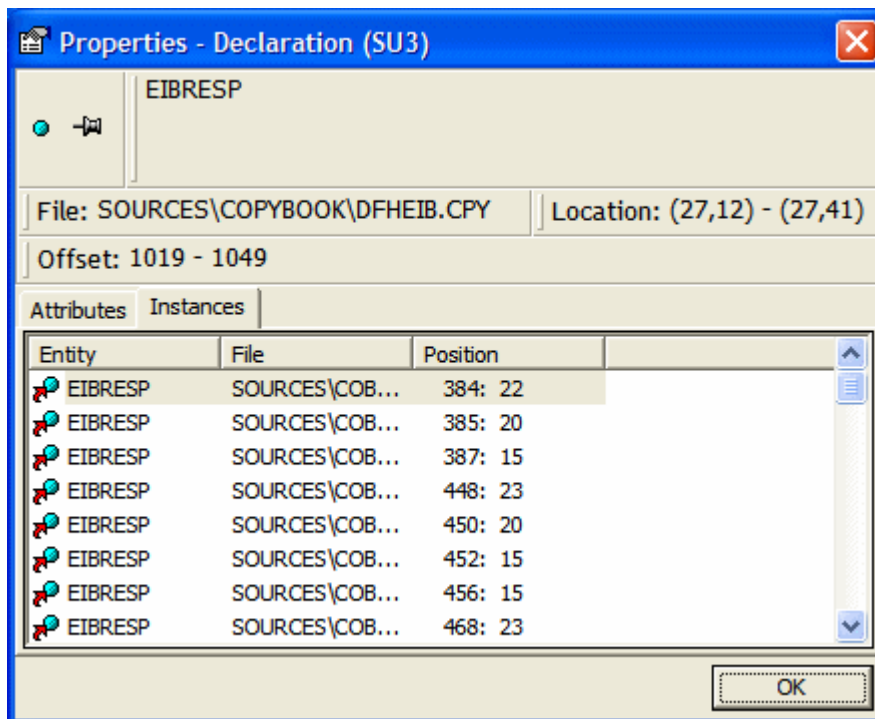
The figure below shows a portion of the construct model for the `GSS5.CBL` program. The model shows that the program executes a `PERFORM` statement if the value of the variable `EIBRESP` satisfies the condition `EIBRESP NOT EQUAL DFHRESP(NORMAL)`.



If you are interested in investigating other uses of `EIBRESP` in the program, you can navigate to the declaration of the variable in the construct model. Select `EIBRESP` in the Context pane and choose **Edit > Declaration**.



From the declaration, you can generate a list of instances in which the variable is used. Select the declaration in the Context pane and choose **Edit > Instances**.



Select an instance in the list to navigate to that instance in the construct model.

Using the Interactive Analysis Main Window


The Interactive Analysis main window consists of a series of panes that offer complementary views of code constructs in the selected source file. The display in each pane is synchronized with the others: selecting a construct in one pane automatically moves the cursor to the construct in the other panes.

You can invoke Interactive Analysis directly by selecting a source file in the Repository Browser and choosing **Analyze > Interactive Analysis**. Interactive Analysis is also available within the project analysis tools (Diagrammer, Change Analyzer, and so forth) but "silently", that is, without you ever actually invoking it as such. Tool usage is identical in either case.


The first time you open Interactive Analysis it displays the Source and Context panes. Select the appropriate choice in the **View** menu to show the other panes. A pane is available only if the view it offers is relevant for the source file type selected in the Objects pane. You will not see the Program Control Flow pane, for example, if a JCL file is selected in Objects pane. Choose the type of file to display from the

Select Type list under the title bar of the Objects pane, and then click **Search**. Select the file you want to open from the resulting list in the Objects tab.

Use the choices at the top of the View menu to configure the panes in logical groupings. Choose **Control Flow**, for example, to view the Source, Flowchart, Program Control Flow, and Animator panes. You can hide a pane by clicking the close box in the upper right corner. **Edit** menu choices and pane-specific menu choices have equivalent choices in the right-click menu.

 **Tip:** Double-click the title bar of a pane to maximize the pane in Interactive Analysis, hiding any other open panes. Double-click the title bar again to restore the previous view.







The Interactive Analysis window's state (size, location, configuration, and option settings) is saved across sessions. Interactive Analysis provides access to the product Activity Log and relevant options windows. To view these items, select the appropriate choice in the **View** menu.

 **Note:** Usage for the Data View and Data Flow panes is identical to that for the corresponding panes in the Global Data Flow tool. For more information, see *Analyzing Projects* in the product documentation set.

For Component pane usage, see *Creating Components* in the product documentation set.

Using Basic Navigation Controls

Interactive Analysis offers Explorer-style navigation controls on the tool bar, with corresponding choices in the **Edit** menu:

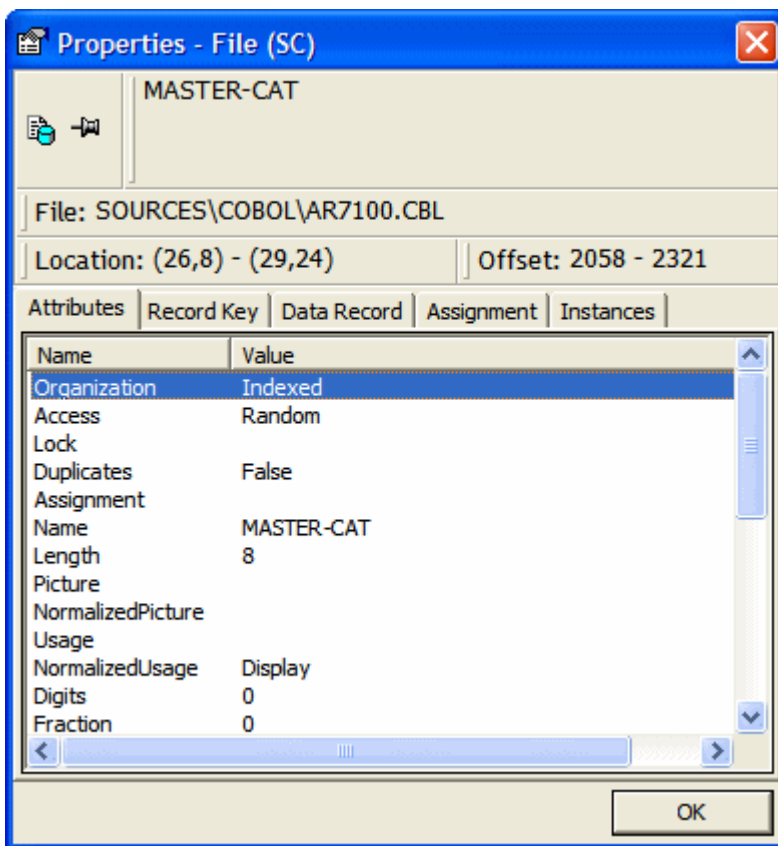
- Click the  button on the tool bar to navigate backward in the history of your selections in the Interactive Analysis window (regardless of the pane in which they were made). Click the adjacent  button to display the selection history in a drop-down menu. Choose a selection by clicking it in the menu.
- Click the  button on the tool bar to navigate forward in the history of your selections in the Interactive Analysis window (regardless of the pane in which they were made). Click the adjacent  button to display the selection history in a drop-down menu. Choose a selection by clicking it in the menu.
- Click the  button on the tool bar to navigate to the parent of the selected construct in the parse tree. Click the adjacent  button to display all of the ancestors of the selected construct in a drop-down menu. Choose a selection by clicking it in the menu.

Using the Properties Window

The Interactive Analysis Properties window for a construct identifies its attributes and related constructs. Attributes are the construct's type, name, location, and characteristics. Related constructs are the constructs with which the construct interacts. The Properties window is not modal, so you can leave it up throughout your Interactive Analysis session.

Opening the Properties Window

You can open the Properties window from any Interactive Analysis pane except the Model Reference pane. Select the construct whose properties you want to view and choose **Properties** in the right-click menu. The Properties window opens.



Viewing and Navigating to Related Constructs

To view related constructs in the Properties window, click the tab for the type of related constructs you want to view. Click a related construct to navigate to it in the Source or Context panes.




Note: Click the  button in the Properties window to navigate back to the original construct.

Opening a Properties Window for a Related Construct

To open a Properties window for a related construct, select the related construct in the Properties window and choose **Properties** in the right-click menu.



Tip: Click the  button to "pin" a Properties window, so that it remains open when you open another Properties window. Click the button again to unpin the window.

Assigning Business Names Manually

To assign a business name and business description to an object, select the object and choose **Set Business Attributes** in the right-click menu. A dialog box opens, where you can enter the business name and business description. To unassign a business name or business description, simply delete the value in the dialog box.

Understanding Program Context


The Interactive Analysis panes described in this chapter offer a set of complementary views of legacy source code. Together these views provide all the information you need to understand the context of a program item:

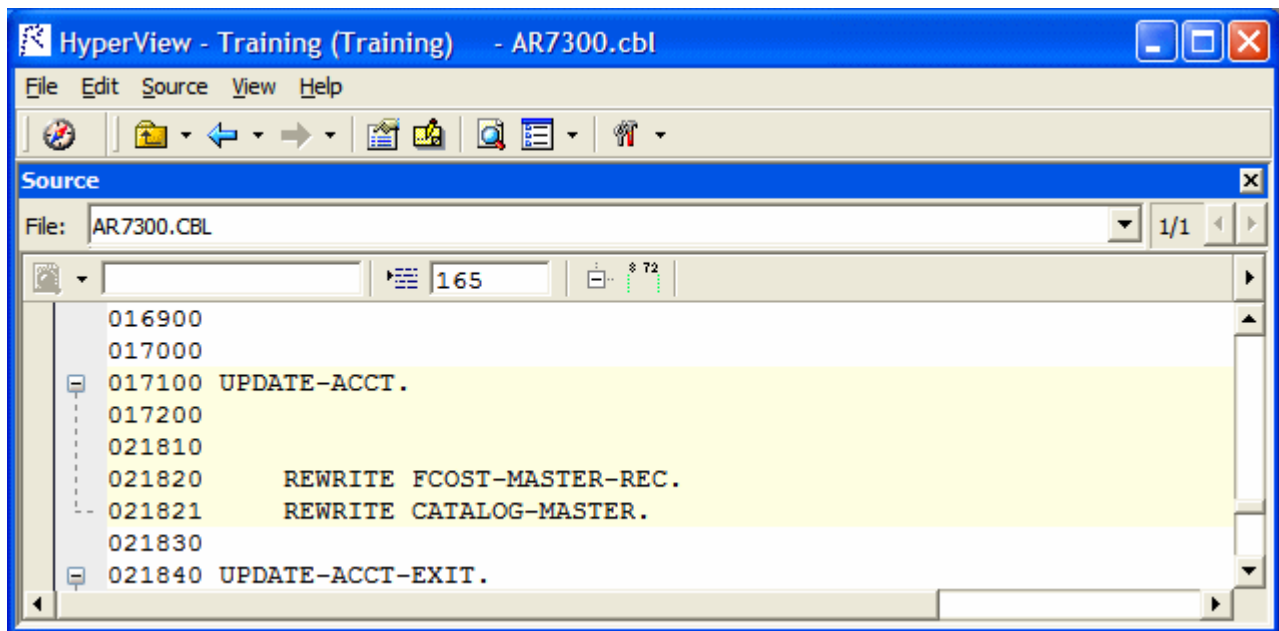
- The Source pane displays view-only source for the selected file.
- The Context pane displays the same code in hierarchical form, in a parse tree that defines the relationships between the code constructs that comprise the source.
- The Objects pane lists the files in the project. Select a file to open it in the Source or Context panes.
- The Watch pane displays program context in summary fashion, showing, in a single view, object model context for the selected source file and construct model context for the selected construct.
- The Screen pane displays the screen object defined in the selected map file, complementing the "bottom-up" analysis of application business processes in the Source and Context panes with a "top-down" view that lets you navigate quickly to an item of interest.

Using the Source Pane


The Source pane displays view-only source for the selected file. The name of the file appears in the **File** drop-down.

You can display the source code for an included file by choosing the file in the **File** drop-down. Use the Objects pane to select a different file.



 **Note:** Use the Editor in the COBOL Analyzer main window to modify program source code.



Navigating and Searching for Source

Click in the Source pane to navigate to a construct. Enter a line number in the field next to the  button and click the button or press **Enter** to navigate to the line. Tool tips show line numbers in the source file when you scroll the Source pane vertically.

Using the Simple Search Facility

Interactive Analysis provides simple and advanced search facilities for source code. To use the simple search facility, enter the text for the search in the field next to the  button on the tool bar. Interactive Analysis locates text matches as you type. Click the  button or press **Enter** to navigate to the next matching construct.

To view matching constructs in a list, click the adjacent  button. From the drop-down menu, choose:

- **Find All** to display pattern matches in a list.
- **Wildcard Find All** to display wildcard pattern matches in a list. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).
- **Recent Search List** to display the results of the last simple search.

Double-click an item in a list to navigate to it in the Source pane.

Navigating to Related Constructs

The **Edit** menu lists choices corresponding to each type of relationship the selected construct has with other constructs in the parse tree. If you select a variable in the Source pane, for example, the **Edit** menu shows **Conditions**, **Port**, and **Declaration** choices. The choices are grayed-out if no relationships of the given type exist for the selected construct.

To view all the constructs in the source file that have a given relationship with a construct, select the construct in the Source pane and choose the appropriate relationship in the **Edit** menu:

- If only one construct has the specified relationship, Interactive Analysis simply moves the cursor to that construct in every open Interactive Analysis pane.
- If more than one construct has the specified relationship, Interactive Analysis opens a list of related constructs in the Properties window and moves the cursor to the first item in the list in every open Interactive Analysis pane. To navigate to another item in the list, choose it in the Properties window.

Navigating to Multiple Occurrences of an Included Construct

If an included file is referenced multiple times in a program (in different structures, for example), you can use the arrows in the upper right corner of the Source pane to navigate between each occurrence of an included construct in the Context pane.

In the Source pane, click on the construct in the included file. The numbers in the upper right corner indicate the sequence number of the current construct versus the total number of constructs. The notation "2/3," for example, identifies the second occurrence of a construct that occurs three times. Use the arrows to navigate between each occurrence.

Selecting and Copying Code

There are two ways to select and copy code in the Source pane, as *construct* or as *text*. The one you choose depends on the task you want to perform.

Selecting and Copying Constructs

Click inside a construct in the Source pane to select it. The selected construct is highlighted. The number of the first line of the selection is displayed in the **Ln** field above the source code. To copy the construct to the clipboard, choose **Copy Selected Construct** in the **Source** menu.

Selecting and Copying Text





Copy code as text when you are assigning code segments to business rules manually and want to select either more or less code than a construct contains. To select a code segment as text, click-and-drag from the first line of the segment to the last line of the segment. The selected text is highlighted in blue. To copy the selected segment to the clipboard, choose **Source > Copy Selected Text**.



Tip: Click the minus sign (-) next to a paragraph, procedure, or data definition to collapse its contents. Click the plus sign (+) to expand its contents.

Setting Source Pane User Preferences

Use the Interactive Analysis > Source tab of the User Preferences window to specify the color-coding used in the Source pane and other display features.

1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Interactive Analysis > Source tab.
2. Select **Iconic cursor** to specify that the icon for a selected construct ( for a data port, for example) replace the standard cursor.
3. In the **Selected Construct Color** field, click the adjacent  button to edit the color used to highlight the selected construct.
4. In the **List Highlighting Color** field, click the adjacent  button to edit the color used to highlight lists.
5. In the Color Scheme pane, click the construct types whose text you want to colorize, or click **Select All** to select all the construct types. The current color of the selected types (if any) is displayed in the **Fore Color** drop-down. Click the adjacent  button to edit the color of the construct types.
6. Select **Display Business Names** to display business names rather than original identifiers in the Context pane.

Collapsing and Expanding Paragraphs or Subprograms

In very long programs, it may be helpful to display the names of paragraphs or subprograms only and hide their source code. To collapse paragraphs or subprograms, choose **Edit > Collapse All**. Click the plus sign (+) next to a collapsed paragraph or subprogram to show its source code again. To show the source code for all collapsed paragraphs or subprograms, choose **Edit > Expand All**.

Showing Source Code Boundaries

To demarcate source code from leading and trailing enumeration characters in source lines, choose **Edit > Show Boundaries**.

Using the Context Pane

The Context pane displays the *parse tree* for the selected source file. The parse tree displays source code constructs (sections, paragraphs, statements, conditions, variables, and so forth) in hierarchical form, making it easy to locate code constructs quickly. Constructs are displayed in the order they appear in your code.



Note: Interactive Analysis adds artificial "owning" sections or paragraphs to the parse tree as necessary. An added, or faked, section or paragraph is denoted in the parse tree with a leading underscore: `_S1`, for example.

The scope of the parse tree determines the constructs and relationships it displays: all constructs, only control statements, only declarations, and so forth. Scopes include the lists Interactive Analysis generates when you perform advanced searches, navigate to a related construct, or identify candidates in Code Search. Choose the scope of the parse tree in the **Scopes** drop-down on the Context pane tool bar.

Click the plus sign (+) next to a construct to expand its hierarchy. Click the minus sign (-) to collapse the hierarchy. The **Edit** menu lists every relationship type for the construct selected in the Context pane.

Perform other tasks as you do in the Source pane. Select **Display Business Names** in the User Preferences for the Source pane to display business names rather than original identifiers in the Context pane.

Using the Objects Pane

The Objects Pane lists the files in the selected project. To view the files you want:

1. Select a type from the drop-down list.



Note: The drop-down menu shows only types that have objects in the project.


2. (optional) Enter a search criteria.

 **Note:** You can use wildcard characters in the search.

3. Click **Search**.

The first 100 rows matching the search criteria will be displayed.

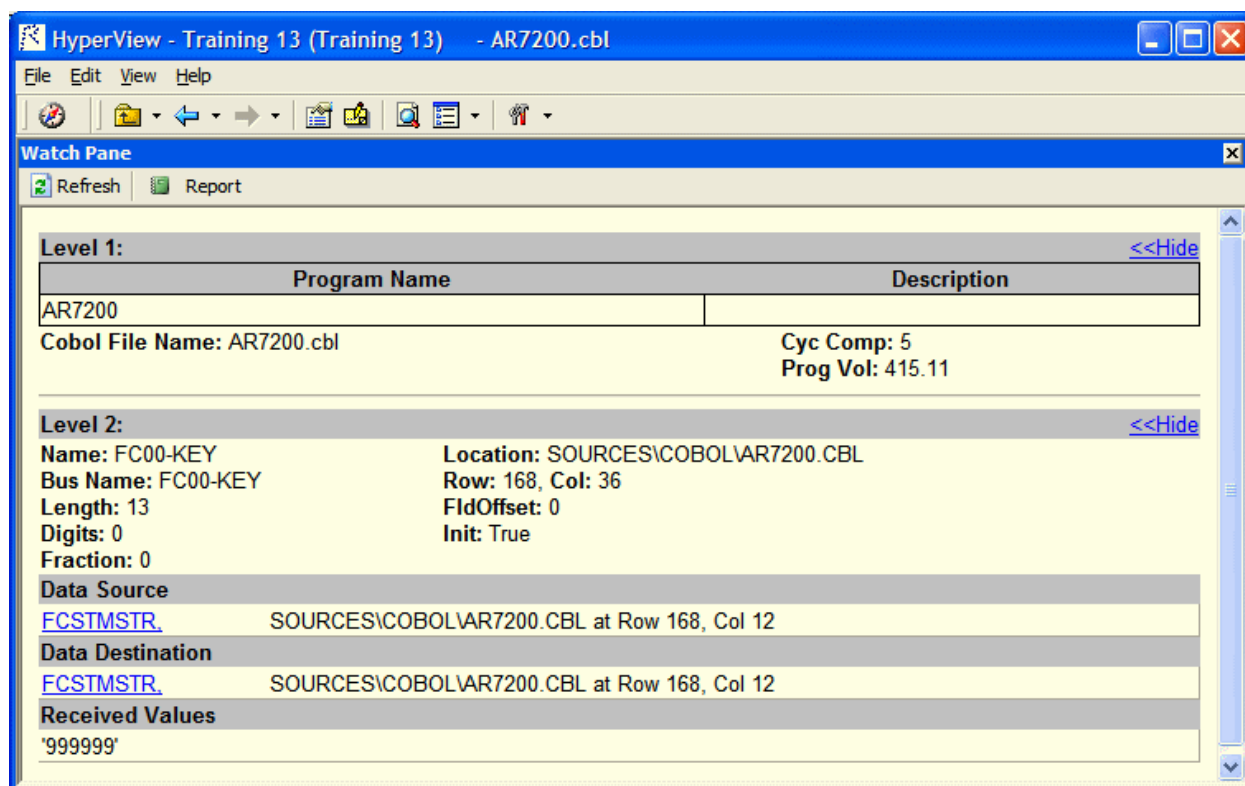
4. (optional) Click **View More** to load another 100 objects to the list.


 **Note:** Once all objects have been loaded the button becomes inactive.


Using the Watch Pane

The Watch pane displays current program context in summary form, showing, in a single view, object model context (Level 1) for the selected source file and construct model context (Level 2) for the selected construct. Both the content of the Watch pane and its layout are customizable with the externalized XML queries and CSS stylesheets provided with the product. For more information, contact support services.

Follow the instructions below to view current context summaries in the Watch pane.



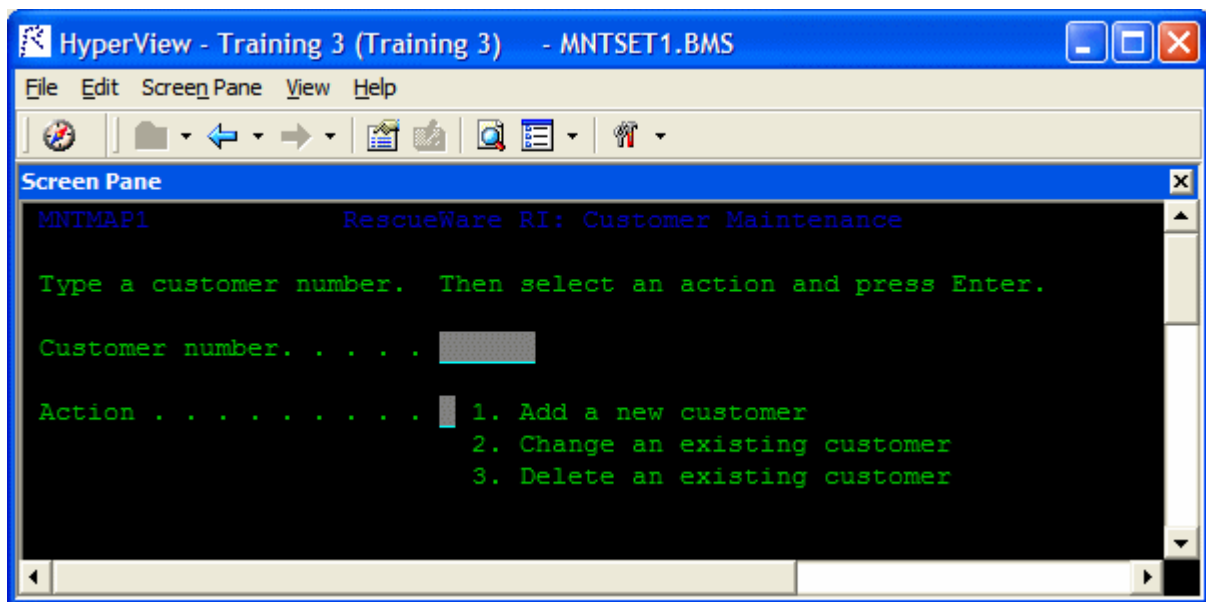
1. Select the source file you want to view in the Objects pane.
2. Select the construct you want to view in the Source or Context pane.
3. Click the  button in the Watch pane to refresh the display:
 - The object model context (Level 1) displays the logical object extracted from the selected source file and its key metrics and relationships.
 - The construct model context (Level 2) displays the selected construct, its business name, its location in source, and other characteristics. For a variable, click the **Build** link to view a summary of impact trace results:
 - Data Source displays the flow of data through each port in the trace to the originating port.
 - Data Destination displays the flow of data through each port in the trace to the destination port.

- Received Values displays the value of the watched variable at each location in the impact trace.
4. Click the **Hide** link to hide a summary. Click the **Show** link to restore a summary.
 5. Click the  button to save the contents of the Watch pane to an HTML file. The Save Watch View dialog opens, where you can specify the name and folder of the HTML file. The report opens when the save operation is complete.

Using the Screen Pane

The Screen pane displays the screen object defined in the selected map file. It complements the "bottom-up" analysis of application business processes in the Source and Context panes with a "top-down" view that lets you navigate quickly to data items of interest in your application.

To view a screen object in the Screen pane, select the map file in the Objects pane, then select the screen in the Source or Context panes. Choose **Screen > Best Fit** to fit the screen to the Screen pane. Choose **Screen > Actual Size** to restore the screen to its actual size.



Gray areas on the Screen pane represent screen fields. Click on a field or screen label to select it. Select a field and choose **Data Items** in the right-click menu to view the declarations for the field in the Properties window.

Manually Extracting Business Names for Program Variables

One of the most powerful features of COBOL Analyzer lets you *auto-extract screen labels* as business names for program variables. For each screen field with a label immediately to its left in the Screen pane, the Glossary tool generates a business name for the program variable whose value is displayed in the field. If the screen field has the label **Customer Number**, for example, Glossary assigns the business name "Customer Number" to the variable whose value is displayed in the field.

Occasionally, however, you may need a finer-grained approach than the auto-extract feature offers. The field of interest may not be immediately to the right of the label, or the field may have no label at all. In these situations, you can extract business names for program variables manually in the Screen pane.

1. Double-click a screen label and select the portion of the label you want to use as the business name, then click Ctrl+C.

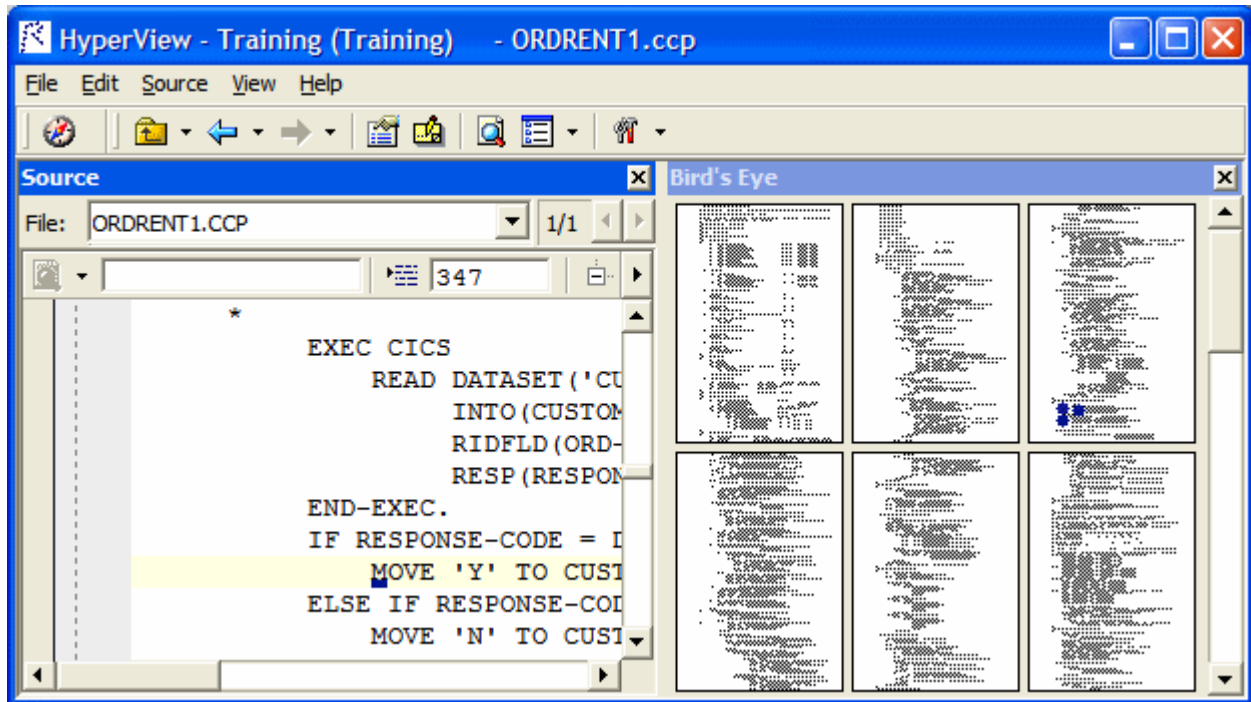


Note: You don't have to use the screen label. You can put any text on the clipboard and assign it as the business name.

2. Select the field of interest and click Ctrl+V. Interactive Analysis paints a yellow box in the lower lefthand corner of the field to indicate that the program variable for the field has been assigned a business name. Double-click the field to navigate to the variable in the Glossary.

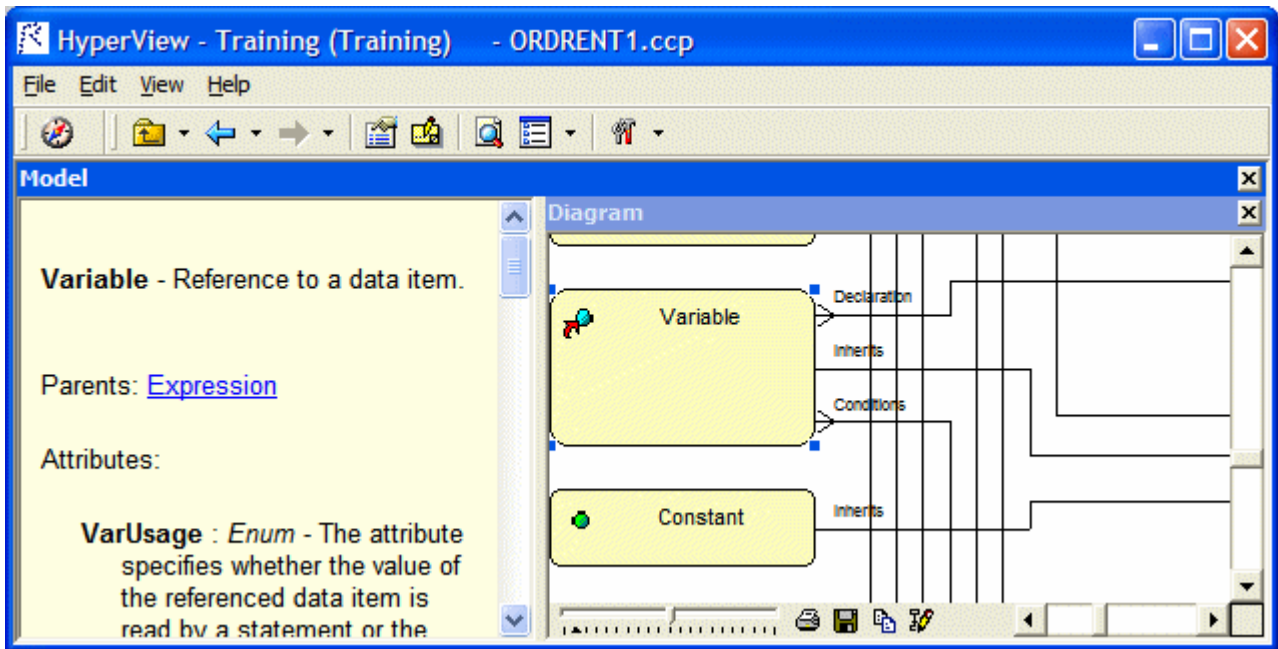
Using the Bird's Eye Pane

The Bird's Eye pane lets you quickly identify the location of a code construct relative to the entire program and any included files. Click a construct in the Source or Context panes to view its location in the Bird's Eye pane.




Using the Model Reference Pane

The Model Reference pane displays the Interactive Analysis metamodel in text and diagram form. Check the model reference embedded in the advanced search utility for the data types and definitions of construct attributes you use in searches.




Performing Advanced Searches

The Interactive Analysis Advanced Search facility distills the parse tree metamodel into a series of prompts that help you build complex filters for construct searches. It also offers predefined filters that you can use to search for dead code, hard-coded constants, nested conditionals, file and screen reads and writes, program calls, SQL ports, and more.

 **Note:** Certain performance-intensive searches, with criteria such as Contains or Is Nested In, may be restricted to the master user.


You typically use the batch advanced search facility, embedded in the Code Search pane, to find constructs in all files of a given type in your project. But you can also use standalone advanced search to find constructs in a single file.

 **Note:** The Change Analyzer tab opens a scoped version of the Advanced Search facility for Change Analyzer. For Change Analyzer usage, see *Analyzing Projects* in the product documentation set.

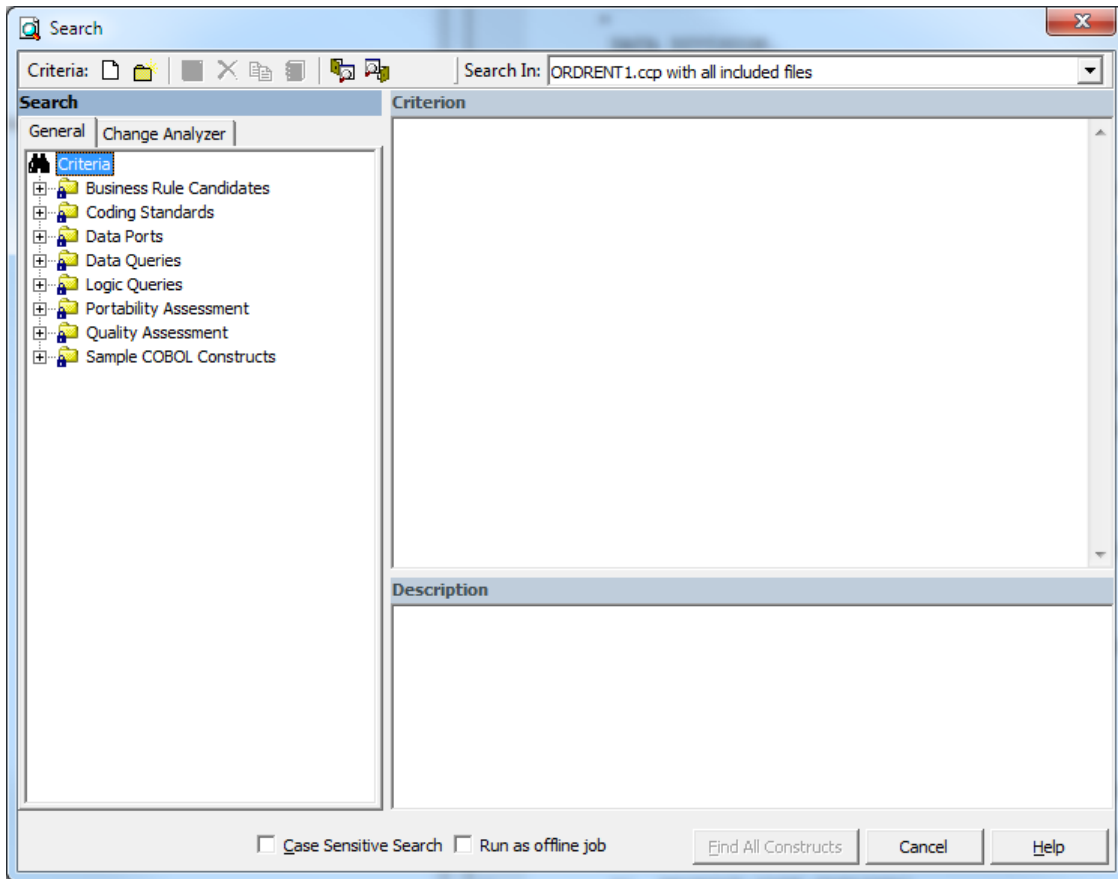
Defining a Search Filter


The Advanced Search facility uses the attributes and relationships of a construct to define the conditions you set in your search for the construct. As an example of how the facility works, let's look at how you might define a search filter for the conditions a program uses to validate the variable EIBRESP.

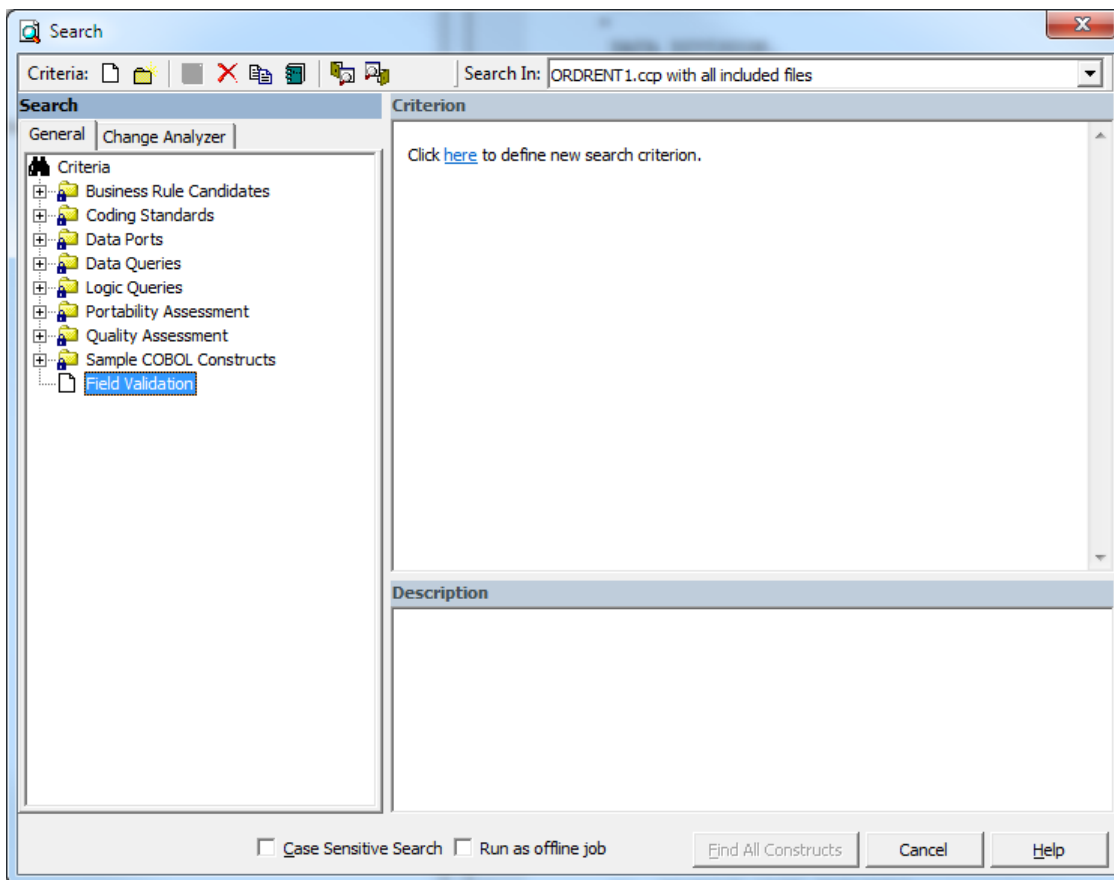
1. Select the file you want to search in the Objects pane, then choose **Edit > Find**. The Search window opens.

 **Note:** Skip this step if you are using Code Search to perform a batch search.

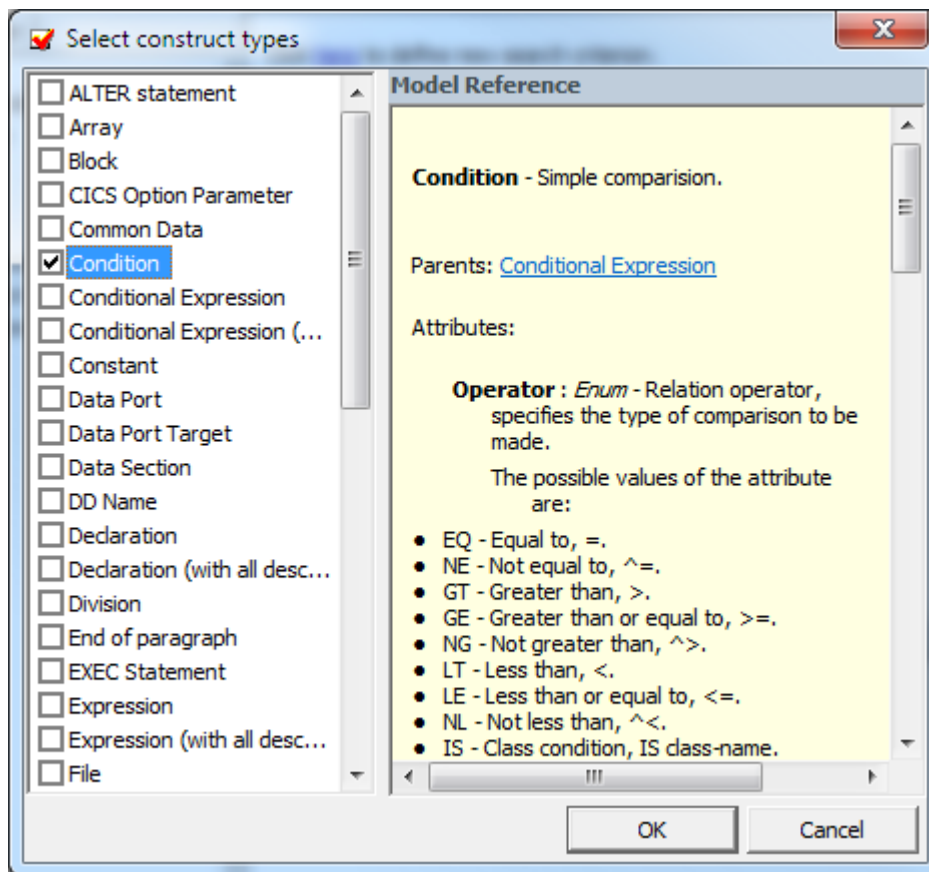
2. Click the General tab.



3. In the General tab, select a folder for the new criterion, then click the  button on the tool bar. The New Criterion dialog opens. Enter Field Validation in the text field and click **OK**. Interactive Analysis creates the Field Validation criterion in the selected folder.



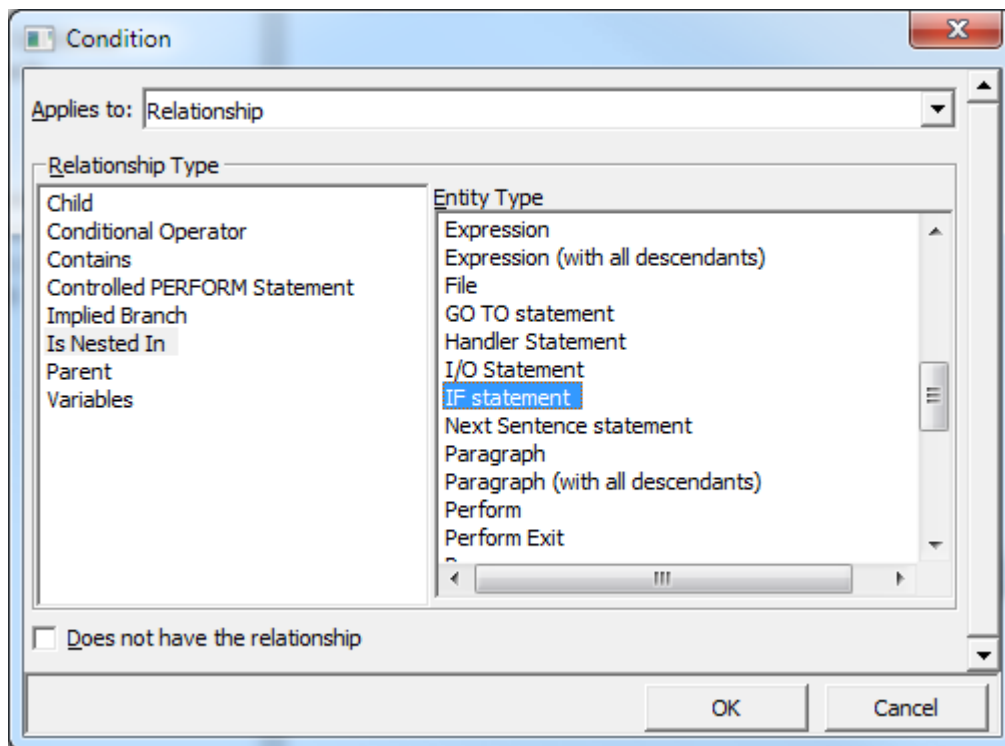
4. Click the **here** link in the righthand pane of the Search window. The Select Construct Types window opens.



5. Select **Condition** in the list of constructs in the lefthand pane. In the righthand pane, review the definition of a condition in the parse tree metamodel, then click **OK**. Interactive Analysis adds the condition construct to the filter definition in the righthand pane of the Search window:

Find All Condition

6. Click the **All** link. The Condition window opens.



7. In the Condition window, choose:

- Relationship in the **Applies to** drop-down.
- Is Nested In in the **Relationship Type** list box.
- If Statement in the **Entity Type** list box.

Click **OK**. Interactive Analysis adds the relationship to the filter definition in the righthand pane of the Search window:

```
Find Condition
which is nested in any IF statement
```

8. Click the **any** link. The Condition window opens. In the Condition window, choose:

- Relationship in the **Applies to** drop-down.
- Contains in the **Relationship Type** list box.
- Perform in the **Entity Type** list box.

Click **OK**. Interactive Analysis adds the relationship to the filter definition in the righthand pane of the Search window:

```
Find Condition
which is nested in IF statement which contains any
Perform
```

9. Click the **any** link. The Condition window opens. In the Condition window, choose:

- Attribute in the **Applies to** drop-down.
- Caption in the **Name** list box.
- Like in the **Operations** drop-down.

Enter *ERROR* in the **Values** field and click **OK**. Interactive Analysis adds the attribute to the filter definition in the righthand pane of the Search window:

```
Find Condition
which is nested in IF statement which contains
Perform such that Caption Like "*ERROR*"
```



Note: Except for the # symbol, you can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). Consult the Model Reference pane for attribute values.

10. Click the **is nested in** link. In the pop-up menu, choose **And**. The Condition window opens. In the Condition window, choose:

- Relationship in the **Applies to** drop-down.
- Contains in the **Relationship Type** list box.
- Variable in the **Entity Type** list box.

Click **OK**. Interactive Analysis adds the relationship to the filter definition in the righthand pane of the Search window:

```
Find Condition
which is nested in IF statement which contains
Perform such that Caption Like "*ERROR*"
and
which contains any Variable
```

11. Click the **any** link. The Condition window opens. In the Condition window, choose:

- Attribute in the **Applies to** drop-down.
- Caption in the **Name** list box.
- Like in the **Operations** drop-down.


Enter *EIB* in the **Values** field and click **OK**. Interactive Analysis adds the attribute to the filter definition in the righthand pane of the Search window:

```
Find Condition
which is nested in IF statement which contains
Perform such that Caption Like "*ERROR*"
and
which contains Variable such that Caption Like "*EIB*"
```



Note: To edit an element in the filter definition, select its link and choose **Edit** in the pop-up menu. To delete an element from the filter definition, select its link and choose **Delete** in the pop-up menu.

12. Enter a description of the filter definition in the Description pane.

13. Click  on the tool bar to save the filter definition.

14. Click **Find All Constructs** to start the search.



Note: Check **Run as Offline Job** to run the search in background mode. You will see a message *Code Search job was successfully submitted to the Queue.*

Executing Advanced Searches

You can execute an advanced search in standalone mode, against the file or construct selected in the Source or Context pane, or in batch mode, in the embedded search pane in Code Search.

1. In the **Search In** drop-down on the tool bar, choose:

- **Source file name with all included files** if you want to execute the search against the selected source file and any included files.
- **Source file name only** if you want to execute the search against the selected source file only.
- **Selected Construct - Construct** if you want to execute the search against the construct selected in the Source or Context panes.

2. Choose the criterion for the search you want to execute from the list of criteria in the lefthand pane.

3. Select **Case Sensitive Search** if you want the search to match case.

4. Select **Find All Constructs**. Interactive Analysis displays a dialog box with the results of the search. Click **OK**. The dialog and Search window are dismissed. Interactive Analysis displays the results in a list named Last Search Results in the Internal category in Code Search.

Advanced String Attribute Operations

The advanced string attribute operations can be set when you execute Code Search and when you have defined a search filter.



Note: The operations are available only for string objects.

You can execute a Code Search using different criteria. When you set a condition to search string attributes, you can specify the following operations:

Like / Not Like

Performs a wildcard (mask) match where ? (question mark) represents any single character and * (asterisk) represents any character sequence.

Example:

- Select **Name** from the Name list, then select **Like** from the Operations drop-down list, and then write * DAY in the values field. The new criterion returns all names ending with "DAY".
- If the search criterion is: "Name" like "???DAY", the search finds all 6 character long names ending with "DAY".

Regular Expression

Performs regular expression matching. This function uses syntax specific to the RDBMS.

Example (Oracle only): "Name" matches "^Ste(v|ph)en", will match names which begin with "Ste" and end with "en" with either "v" or "ph" in between, so both "Steven" and "Stephen" will be selected.

On Microsoft SQL Server databases, you need to install an additional package on the SQL Server machine to get regular expression support. This package is provided by Microsoft.



Note: There are significant differences in regular expression syntax between the Oracle and MS SQL Server implementations. For more information check their corresponding documentation.



Note: Due to the nature of the regular expressions, utilization of a database index is not possible for their evaluation, therefore the performance for such queries is expected to be relatively slower.

Length is Less than / Length is Greater than

Compares attribute value lengths.

Example: "Name" length is "6", finds all 6 character long names.

Match other attribute value/ Do not match other attribute value

Compares two attribute values.

Example: "Name" matches attribute "JobName", finds all names matching the "JobName" as well.

Contain other attribute value/ Do not contain other attribute value

Checks if an attribute value contains another attribute value.

Example: "Name" contains attribute "JobName", finds all names which also contain "JobName" value.

Partial Attribute match


Partial Match compares part of an attribute value with another part of another attribute value.

Example: "Name" pos 10-12 matches pos 5-7 of "JobName", finds all names where the value from character position 10 to 12 matches the "JobName" value from character position 5 to 7.

Working with Advanced Search Criteria



This section describes common tasks you perform with advanced search criteria.

Creating a Search Criterion


To create a search criterion, select the folder for the new criterion in the General tab, then click the  button on the tool bar. The New Criterion dialog opens. Enter the name of the criterion in the text field and click **OK**.

Editing a Search Criterion


To edit a search criterion, select the criterion, then select the link for the element you want to change in the filter definition. Choose **Edit** in the pop-up menu to edit the element, or **Delete** to delete the element.

 **Tip:** To restore the original definition of a predefined search criterion, select the criterion and click the  button on the tool bar.


Copying a Search Criterion


To copy a search criterion, select the criterion and click the  button on the tool bar. The New Criterion dialog opens. Enter the name of the new criterion in the text field and click **OK**.

Saving a Search Criterion


To save a search criterion, select the criterion and click the  button on the tool bar. The criterion is available to every project in the workspace.


Deleting a Search Criterion


To delete a search criterion, select the criterion and click the  button on the tool bar. You are prompted to confirm the deletion. Click **OK**.

 **Note:** You cannot delete a predefined search criterion.


Creating a Folder for Search Criteria

To create a folder for search criteria, click the  button on the tool bar. The New Folder dialog opens. Enter the name of the new folder in the text field and click **OK**. The new folder appears in alphabetical order in the tree in the lefthand pane of the window. Drag-and-drop search criteria to move them to the folder. You can create folders within folders.


To copy a folder and all its contents, select the folder and click the  button on the tool bar. The New Folder dialog opens. In the text field, enter text to be prepended to the folder name and to the names of each of its subfolders and search criteria, and click **OK**.


To modify a folder name, click in the name area for the folder to make the name editable, enter the new name, and press **Enter**. To delete a folder, select it and click the  button on the tool bar. You are prompted to confirm the deletion. Click **OK**.

Saving a Search Criterion as HTML

To save a search criterion in an HTML file, select the criterion and click the  button on the tool bar. The Save Criteria dialog opens, where you can specify the name and folder for the HTML file.

Exporting and Importing a Search Criterion

To export a search criterion to an XML file, select the criterion and click the  button on the tool bar. The Export Criteria dialog opens, where you can specify the name and folder for the XML file.

To import a search criterion, click the  button on the tool bar. The Import Criteria dialog opens, where you can select the criterion you want to import.

Staging Program Analysis with Code Search

Use the Code Search pane to stage the results of a batch search as input for subsequent tasks: viewing source and context, colorizing sliced code, extracting business rules, and the like. A system analyst, for example, might use Code Search to create a list of programs impacted by a change request, then create a project that contains only the source files for those programs.

Code Search searches are executed against the current project. You can also execute searches against the results of a previous search.




Note: Certain performance-intensive Code Search searches, with criteria such as Contains or Is Nested In, may be restricted to the master user.

Getting Started in Code Search


Code Search is project related. It executes a search over all verified files in the project.

This section describes a sample use of Code Search that should help you get oriented in the tool. Assume that you want to create business rules for segments of code that write to a file:

1. In the **Interactive Analysis** window, if the **Code Search** pane is not visible, click **View > Code Search**. This displays the **Code Search** pane in the **Interactive Analysis** window.
2. In the **Code Search** pane, click the **Top** view link. The Top view opens.
3. Double-click the **Rule Finder** category. The Rule Finder view opens.
4. On the tool bar, click **Add** (). This opens the **New list** dialog box.
5. In the **Enter new list name** field, type `File Writes` and then click **OK**. An empty list named **File Writes** is added to the Rule Finder view.
6. Click **Advanced Search**. This opens the **Search** dialog box.
7. In the top left, select the file type you want to execute the search against. In this case **COBOL File**.
8. In the **Search In** list, select **All Objects**.
9. In the Criteria tree, expand the **Data Ports** folder, then expand the **Files** folder. Select the predefined **All File Writes** criterion and then click **Find All Constructs**.



Note: You can modify the predefined search criterion or create your own search criterion as you do in the Advanced Search facility.

10. Interactive Analysis displays a list of files that match the search criteria in the Code Search pane. The count column displays the number of constructs found in the file.
11. Click **Execute** () and then click **Create Rules** from the displayed list. This opens the **Rules** dialog box. Which displays the following message:
You are about to generate templated business rules into a specified rule set. Continue?
12. Click **Yes** to continue. This opens the **Create Rules From List** dialog box.
13. Select an existing **Business Function** from the list or create a new one by clicking **New**:
 - a) If you choose **New**, the **Business Function** dialog box opens.
 - b) In the **Business Area** field, select a business area from the list or type a new one.
 - c) In the **Name** field, type a name.

- d) (Optional) In the **Technical Description** field, type a technical description of the function.
 - e) (Optional) In the **Business Description** field, type a business description of the function.
 - f) Click **OK** to create the business function and to return to the **Create Rules From List** dialog box.
- The newly created **Business Function** is selected.

14.Select an existing **Rule Set** from the list or create a new one by clicking **New**:

- a) If you choose **New**, the **Create Rule Set** dialog box opens.
- b) The **Name** field has an auto-generate name, if you want you can type your own name.
- c) (Optional) In the **Technical Description** field, type a technical description of the function.
- d) (Optional) In the **Business Description** field, type a business description of the function.
- e) Click **OK** to create the rule set and to return to the **Create Rules From List** dialog box.

15.(Optional) Click the **Business Description Template** tab, and then type a description.

16.(Optional) Click the **Technical Description Template** tab, and then type a description.

17.Click **OK**.


This creates the rule.

18.To view the new rule, click **View > Rules** This opens the **Rules** pane which contains the new rule created. In the **Grouping** tab you can see a tree displaying all the instances that match the rule created.

Understanding the Code Search Pane

Interactive Analysis lists offer convenient ways to navigate program code and record the results of program analyses. Use the hierarchy of views in the Code Search pane to display Interactive Analysis lists and perform other list-related tasks:

- The Top view displays the folders, or *categories*, for lists.
- The Category view displays the lists in the selected category.
- The List view displays source files with non-zero counts in the selected list.
- The File view displays each construct in the list for the selected source file. Click a construct in the list to navigate to it in other Interactive Analysis panes.

Each view gives you access to the next-level view in the hierarchy. Click the label for a view to navigate to it. Double-click an item in the view to drill down to the next view in the hierarchy. Click the  button on the tool bar to navigate to the parent of the current view.

Working with Categories




The Top view of the Code Search pane displays the categories for Interactive Analysis lists. Categories are displayed only for lists you can create for the source file type selected in the Objects pane.

You can use predefined categories or create your own. All predefined categories are shared. Use the General and Internal categories for unenumerated tasks.



Note: Interactive Analysis puts the results of standalone advanced searches in a predefined list called Last Search Results in the Internal categories.

To create a category, select the parent category and click **Code Search > Add**. The New category dialog opens, where you can specify the new category name. A list with the category name is automatically created in the category. To edit the category name, click the name and type over it in the text box.

By default, user-defined categories are not shared. To share a user-defined category, select it and click the  button on the tool bar. A  symbol indicates that the category is shared. Click the  button again to turn sharing off.



Note: Sharing a user-defined category automatically makes its parent categories shared. Unsharing a user-defined category automatically makes its children unshared.


Working with Lists

The Category view of the Code Search pane displays the lists in the selected category. You can use predefined lists or create your own. Use Code Search to populate lists.



Creating a List

To create a list, double-click the category for the list in the Top view and choose **Code Search > Add**. The New List dialog opens, where you can specify the name of the new list.

Deleting a List




To delete a list, select it and click the  button on the tool bar. You are prompted to confirm the deletion. Click **OK**.

Copying a List


To copy a list, select it and click the  button on the tool bar. In the Top view, select the category you want to copy the list into and click the  button. The Copy List dialog opens, where you can enter the name of the new list. You can copy a list into the same category.

Sharing a List


By default, lists are not shared. To share a list, select it and click the

 button on the tool bar. A  symbol indicates that the list is shared. Click the  button again to turn sharing off.

Adding a Construct to a List Manually

To add a construct to a list, select a source file in the List view, then choose a construct in the Source or Context panes. Click the  button on the tool bar to add the construct to the list of constructs for the selected file.



Deleting a Construct from a List

To delete a construct from a list, select the construct in the File view and click the  button on the tool bar. You are prompted to confirm the deletion. Click **OK**.

Executing Code Search Searches

Code Search is project related. It executes a search over all verified files in the project. All categories, criteria and lists with results are global options for all repository projects and show the results for the current search. After completing the search, the result is saved in the List View. The last executed search can be used as a starting point for the next search and can be accumulated with the new one by checking **Accumulate search results** in the search view. Otherwise the previous content from the list will be lost.

Use a Code Search to generate Code Search lists. You can modify the predefined search criteria or create your own criteria as needed. You can use a construct list itself to narrow the search.

1. Select a list, then click the  button next to the  button on the tool bar. In the drop-down, choose the type of object you want to execute the search against. The Search window opens.
2. In the **Search In** drop-down, select:
 - **All Sources** to execute the search against all the sources of the specified type in the project.
 - A list if you want to execute the search only against sources of the specified type with non-zero counts in the selected list. If you select **Rule Finder.File Writes**, for example, the search will be

executed only against sources of the specified type with non-zero counts in the File Writes list in the Rule Finder folder.




3. Choose the criterion for the search you want to execute from the list of criteria in the lefthand pane. Modify the predefined criteria or create your own criteria as needed.
4. Select **Accumulate Search Results** if you want the results of the search to be added to the results of the previous search. Otherwise, the new results overwrite the previous results.
5. Select:
 - **Find All Constructs** if you want Interactive Analysis to create for each searched file a list of all the constructs found by the search.
 - **List Sources Only** if you want Interactive Analysis simply to indicate for each searched file (with the notation >0) whether the construct of interest exists in the file.

Interactive Analysis displays a dialog box with the results of the search operation. Click **OK**. The Search window is dismissed and the results of the search are displayed in the List view.

Using Construct Lists to Narrow a Code Search

Once you have generated a construct list in Code Search, you can use the list itself as a condition of a search. That is, you can use the search results to refine a subsequent search.

Suppose you have performed a Code Search for duplicate paragraphs in COBOL programs, storing the results in a list called Matched Paragraphs in the General folder. The next step might be to search for all PERFORM statements that reference the duplicate paragraphs. The example below shows how you might use a Code Search list of duplicate paragraphs as a condition of the search.

1. Select a list, then click the  button next to the  button on the tool bar. In the drop-down, choose the type of object you want to execute the search against. The Search window opens.
2. Select a folder for the new criterion in the General tab, then click the  button on the tool bar. The New Criterion dialog opens. Enter `Perform Duplicates` in the text field and click **OK**. Interactive Analysis creates the Perform Duplicates criterion in the selected folder.
3. Click the **here** link in the righthand pane of the Search window. The Select Construct Types window opens.
4. Select **Perform** in the list of constructs in the lefthand pane, then click **OK**. Interactive Analysis adds the condition construct to the filter definition in the righthand pane of the Search window:

```
Find All Perform
```

5. Click the **All** link. The Condition window opens. Choose:
 - Relationship in the **Applies to** drop-down.
 - Called Paragraph in the **Relationship Type** list box.
6. Click **OK**. Interactive Analysis adds the relationship to the filter definition in the righthand pane of the Search window:

```
Find Perform  
which has related any Paragraph (as Called Paragraph)
```


7. Click the **any** link. The Condition window opens. Choose:
 - List in the **Applies to** drop-down.
 - General.Matched Paragraphs in the **Name** drop-down.

You can filter out list constructs by selecting **Does not belong to the list** in the Condition window.


8. Click **OK**. Interactive Analysis adds the list to the filter definition in the righthand pane of the Search window:

```
Find Perform  
which has related Paragraph (as Called Paragraph)  
which is in list  
General.Matched Paragraphs
```

9. Execute the search. Interactive Analysis displays a dialog box with the results of the search operation. Click **OK**. The Search window is dismissed and the results of the search are displayed in the List view.

 **Tip:** As a practical matter, you would probably also want to search for GO TO statements that reference the duplicate paragraphs. You could simply add the results of this search to the results of the search for PERFORM statements by using the **Accumulate Search Results** button.

Searching for Similar Constructs in Code Search


To search for similar constructs, select a construct in the Source or Context panes, then select a list. Click the  button on the tool bar. The New Criterion dialog opens. Enter a criterion name or accept the default.

The Search window opens with a predefined search criterion that will generate a list of constructs similar to the selected construct. The new criterion is added to the General tab of the advanced search facility. Edit the criterion if necessary, then execute the search.

Extracting Business Rules in Code Search



To extract business rules for multiple constructs, use the following method:

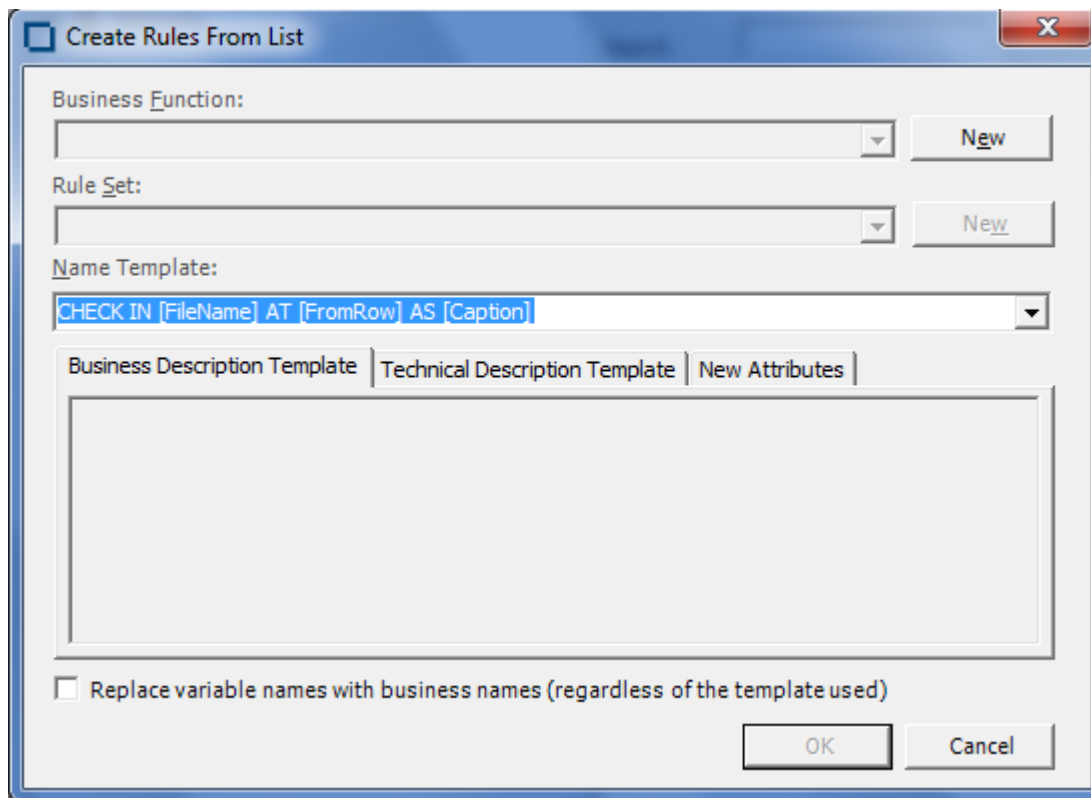
- Generate rules for listed constructs in a specified rule set. When you choose this option, you batch process rule attributes in the generation step.

 **Tip:** As long as the Rules pane is open, you can generate a rule for a single construct by selecting the construct in the File view and choosing **Rule > Create Rule**.

Generating Business Rules in a Specified Rule Set



Follow the instructions below to generate rules in a specified rule set with predefined attributes. Code Search creates a rule for each construct in the Code Search list.

1. Select a list in the Rule Finder category, then click the  button next to the  button on the tool bar and choose **Create Rules** in the drop-down menu. The Select Method dialog opens.
2. In the Select Method dialog, select **templated rules into a specified rule set** and click **OK**. The Create Rule Set window opens.
3. In the **Business Function** drop-down, select the business function for the rule set. The current business function is displayed. To create a new business function, click **New**.
4. In the **Name** field, enter the name of the rule set for the rules you want to create and click **OK**. The Create Rules From List window opens.







5. In the **Name Template** combo box, select a template for the rule name attribute from the drop-down, or enter a template manually. Templates derive from the list in the Rule Defaults tab in the Project options for Business Rule Manager. If you enter a template manually in Code Search, the new template is added to the list.
6. On the Business Description Template and Technical Description Template tabs, enter templates for the description and business description attributes, respectively.
7. On the New Attributes tab, specify the value for each attribute you want to set in the selected rules.
8. Select the checkbox at the bottom if you want to replace variable names with business names (regardless of the template used).
9. When you are satisfied with your entries in the Create Rules From List window, click **OK**. Interactive Analysis displays an informational message that tells you how many rules it created. Click **OK**.

Generating an Impact Subtree Report in Code Search

To generate an impact subtree report for all the constructs in a list, select the list in the Impact Report category, then click the  button next to the  button on the tool bar and choose **Create Impact SubTree Report** in the drop-down menu. A Save As dialog opens, where you can specify the name, location, and file type of the report.



Marking and Colorizing Constructs in Code Search

The Components pane lets you create a self-contained program, called a *component*, from the sliced code or simply generate a Interactive Analysis list of sliced constructs for further analysis. You can mark and colorize the list constructs in the Interactive Analysis Source pane.

To mark and colorize list constructs in the Source pane, select the list and click the  button on the tool bar. To mark and colorize sliced constructs in a single file, select the file in the List view and click the  button. To mark and colorize a single construct, select it in the File view and click the  button. Click the  button again to turn off marking and colorizing.





Creating Projects in Code Search

You can create a project based on the results of a Code Search analysis.

1. Select a list, then click the  button next to the  button on the tool bar and choose **Create Project** in the drop-down menu. The New Project dialog opens. Enter the name of the new project in the text field and click **OK**.
2. The Change Magnitude window opens. Select **Automatically calculate change magnitude** if you want Code Search to set change magnitudes for the listed source files based on the ranges specified in the fields below the check box. Modify the ranges as needed. The Code Search settings will override any existing change magnitudes for the files.
If you want COBOL source files with 6 to 10 constructs to have a Large change magnitude, for example, set the range for Medium to less than 6 and the range for Large to less than 11.
3. When you are satisfied with your choices, click **OK**.

Generating Metrics and Detailed Reports

Use Code Search to show the distribution of list instances across a project and view the details for each instance. The following reports are available:

- For each list in the selected category, the Metrics Report shows the number of list items in each program. To generate a Metrics Report, select a category in the Top view, then click the  button next to the  button on the tool bar and choose **Metrics report** in the drop-down menu.
- For each source file in the selected list, the Details Report shows the constructs in the list, their type, and their location in the file. You can customize the report to include any Interactive Analysis attribute related to the constructs. To generate a Details Report, select a list in the Folder view, then click the  button next to the  button on the tool bar and choose **Detailed report** in the drop-down menu. In the Select Attributes dialog, choose the construct attributes you want to view in the report.

Running Code Search Reports in the Main Window

COBOL Analyzer lets you select and run a set of Code Search queries from the Repository Browser. You can execute these sets of queries (Code Search reports) on a selection of files that belong to different projects and different object types. The selected queries can be predefined and/or created by you.

To run a report:


1. Select object and right-click in the Repository Browser or select objects and click **Reports > Code Search Reports > Code Search Reports**.
2. Click **Code Search Reports > Code Search Reports**.

From the Reports window you can select to view and run a report from the following three report types:

Migration The reports in this group are: Portability Assessment, Migrate Net Express to Visual COBOL, Migrate IBM PL/I to Open PL/I and Migrate Enterprise COBOL to 6.1.

Quality The reports in this group are: Quality Assessment and Performance Optimization.

Custom This contains the previously created custom reports (if any).

To view a report and edit it if necessary, click 

To run a report, click 

Creating a Custom Code Search Report

To create a custom report:

1. Select objects from the Repository Browser.



Note: You can select objects from different projects with different object types. If you don't select files but a project, the report will be run for all programs in it.

2. Select **Code Search Reports > Code Search Reports...** from the Reports menu or from the context menu in the Repository Browser.
3. When the Reports window opens, click **Create New Custom Report**.
4. From the dropdown menu at the top of the Code Search Reports window, you can select **<New Report>** or previously-created reports (if any).
5. Select categories and subcategories of queries from the tree structure in the Code Searches tab and/or the Repository Queries tab. Note that when you select a category, you automatically select all its subcategories.

The **Only show queries for project/object types** is checked by default and the queries displayed in the tree structure are only the ones for the project/object types that you have selected. If you uncheck this option, all possible queries will be displayed.

When **Only show selected queries for saved reports** is checked and you have selected an already saved report from the list at the top, only the queries selected for it will be displayed. If you are creating a new report, all queries are shown in the tree.

6. Click the **Preview Report** button to add categories to the report, add queries to a category or change the order of the queries.
7. In the **Report name** field write a name for the custom report or use the default one. The report name must be unique and not used for any other report.
8. Select a save location for the report. The default one is the `Output\CodeSearchReports` folder of the workspace.
9. From the options below you can choose if you want to generate an HTML report and if you want to make the report available to other users.
10. If you select to generate an HTML report, you can also select to generate a summary report which will show the differences between an older report and a new one.
11. Click **Save** to save the report.
12. Click **Run** to save and run the report.



Note: The report is generated in a folder with the same name as the report. It has a date/time stamp so as to avoid overwriting.



Note: The last ten reports that were run will be saved and you can access them from the Repository Browser by right-clicking and then selecting **Code Search Reports > Code Search Reports > Custom**. Running the report a report from the list of previously-created reports will show results but no report file will be generated.

Working with Saved Reports

You can edit, delete or make copies of already created and saved reports from the Code Search Reports window.

To make edits to a saved report, select one from the list, make any necessary changes in the list of selected queries, report name, save location and any of the other options.

You can also choose to overwrite the changes when running the report by selecting the checkbox next to the **Save** button.

To create a copy of a report but use a different name, click **Copy Report**.

To delete a saved report click **Delete Selected Report**.

Code Search Report Results

After you run a report, the output is saved in the selected location and the Code Search Reports window closes and the Queue Processor pops up. To view the report results, double-click the Activity Log line

which shows that the report was successfully executed or from the **View** menu select **Code Search Report Results**


The results are displayed in a tree view. Double-clicking a file in the tree, opens the source editor pane.

To open Interactive Analysis for a file, right-click it and select **Interactive Analysis**.

The Code Search Report Results window lets you view previously run reports by selecting them from the drop-down list at the top. The report titles contain information about the execution date and time. You can also choose to re-run a report, generate an HTML report and/or export for the Eclipse plugin.

To re-run a report, click .

To create a report, click .

To export for the Eclipse or Visual Studio plugins, click .

Storing Code Search Reports

You can store a predefined set of Code Search queries and a previously generated report as a custom report which can be later used to generate a Code Search report. These can be shared or private. The predefined reports need to be added in the `HCSearch.xml` file and will be loaded in the database as the standard queries.

Running Code Search Reports in Interactive Analysis

You can run Code Search reports in Interactive Analysis by clicking **Selected Object** or **All Objects** at the top.

When the report has been executed, a message will pop up asking if you want to load the results. Click **Yes** and the results will be loaded in the Code Search pane.

Analyzing Impact Traces

An *impact trace* describes the flow of data within and between programs in an application: how data items exchange values, use each other in computations, and so forth. If you are extracting business rules, for example, you might want to view an impact trace to make sure that a segment encapsulates all of the business logic for a candidate rule. Or you might want to use a scoped trace to help locate candidates that exhibit a particular type of relationship.

You can analyze the entire tree for a trace, or a consolidated tree that displays only the ports and leaf nodes traversed in the trace. Options let you filter out unwanted relationships or variables. You can save trace results and restore them in a later session.



Note: Projects must have been verified with the **Enable Impact Report** and **Enable Data Element Flow** options set in the Project Verification options.

Generating an Impact Trace

The Interactive Analysis Impact pane displays a hierarchical view and diagram of an impact trace. Both views show the flow of data to or from a *startup item*.

1. In the Interactive Analysis Source or Context pane, select the construct that contains the startup item or items and choose **View > Impact**. The Impact pane opens.
2. Choose:
 - **Impact > Backward Analysis** to view the flow of data into the startup items.
 - **Impact > Forward Analysis** to view the flow of data out of the startup items.



Note: Once a trace is generated, click the ← or → buttons on the tool bar to reverse the direction of the trace.

3. Choose:

- **Impact > Detailed Analysis** to generate the entire tree for a trace.
- **Impact > Consolidated Analysis** to generate a consolidated tree that displays only the ports and leaf nodes traversed in the trace. Only the subtrees for ports are displayed.

Interactive Analysis displays the trace results. Click the plus sign (+) next to an item to expand its hierarchy. Click the minus sign (-) to collapse the hierarchy.

4. Click the Diagrammatic View tab to display the impact trace in a diagram.

5. Select an item in the trace and repeat the procedure to start a trace for that item. The new trace is displayed below the previous trace.

6. To delete a trace, select its startup item and choose **Impact > Remove Root**. To clear the pane, choose **Impact > Remove All**.

Understanding the Hierarchical View

In the hierarchical view, each line in the trace contains a symbol indicating the type of relationship the affected data item has with the previous item and a label showing the details of the relationship. Options give you complete control over the information displayed in the label. The table shows the meaning of the typefaces used in the labels.

Typeface	Meaning
bold	An item that has not been visited by the trace.
gray	A repeated item.
blue	A repeated item at a different location.
bold blue	A repeated item at a different location that has not been visited by the trace.
red	An item filtered out of the trace.

Understanding the Diagram View













In the diagram view, each box in the trace contains the data item name and the name of the program it belongs to (unless the parent item belongs to the same program). Relationship lines contain the name of the relationship. Options let you display the memory offset and size of a data item. Place your cursor over a variable for a moment to display the memory offset and size in a tool tip. The table shows the meaning of the colors used in the diagram.





Color	Meaning
red outline	Startup item.
blue background	An item that has not been visited by the trace.
black outline	An item filtered out of the trace.

Data Flow Relationships

The table below describes the intraprogram and interprogram relationships detected by the data flow analysis tools.

Relationship	Definition	Type	Icon	Description
calls	N/A	interprogram	N/A	A parameter passed in a call to another program.

Relationship	Definition	Type	Icon	Description
cast	MOVE A TO B with data conversion	intraprogram		A data item moved to a data item of a different type.
common area transitions	N/A	interprogram	N/A	For Unisys 2200 Cobol, a common-storage data area item passed in a call to another program. Perform Unisys Common-Storage Area Analysis must be set in the project verification options.
comp	STRING A ... INTO B	intraprogram		An arbitrary computation. The result is produced by applying complex rules to the argument, such as STRING.
comp+	ADD A TO B	intraprogram		An addition-like operation: ADD, SUBTRACT, or corresponding parts of COMPUTE.
comp*	MULTIPLY A BY B	intraprogram		A multiplication-like operation: MULTIPLY, DIVIDE, or corresponding parts of COMPUTE.
comp@	MOVE ARRAY (IDX) TO A	intraprogram		An operation with array elements.
cond	IF A = B ...	intraprogram		Comparison of data items with a symmetric relationship.
cond*	IF A * X = B ...	intraprogram		Comparison of a multiple of a data item with another data item.
const cond	IF A = 1 ...	intraprogram		Comparison of a data item with a constant.
const.move	MOVE 1 TO B	intraprogram		A constant moved into a data item.
const.comp	ADD 1 TO B	intraprogram		An arithmetic operation with constants.
const.init	03 A ... VALUE 1	intraprogram		A data item initialized by a constant.
DMS records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via Unisys DMS database records.
files	N/A	interprogram	N/A	Data communication via files. Traced only when corresponding JCL, ECL, FCT, or CSD files are verified.
files in jobs	N/A	interprogram	N/A	Data flow in JCL datasets when <i>files</i> is selected.
input port	N/A	intraprogram		A data item in which data is received.

Relationship	Definition	Type	Icon	Description
move	MOVE A TO B	intraprogram		A data item moved to a data item of the same type.
network records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via network records.
output port	N/A	intraprogram		A data item from which data is sent.
screens	N/A	interprogram	N/A	Data sent to a screen by one program and received in a screen by another.
screen definitions	N/A	interprogram	N/A	Data flow in screen fields when <i>screens</i> is selected.
start	N/A	intraprogram		The startup item in an Impact pane consolidated analysis.
used	MOVE ... TO A ... MOVE A TO ...	intraprogram		A value assigned in a statement used as an argument in another statement.

Setting Impact Pane User Preferences

Use the Interactive Analysis > Impact tab of the User Preferences window to control the level of detail provided in the label for each relationship type in the trace: whether to show or hide the locations of variables in source, their memory offsets and sizes, and so forth. You can also use these options to specify the level of detail to include in an Impact Subtree Report.

1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Interactive Analysis > Impact tab.
2. Click the tab for the type of object whose relationship labels and report attributes you want to specify.
3. In the **Root** field, edit the template for the startup item label. Items preceded by a backslash are replaced with attributes reported in the Subtree Report. You can modify unescaped text as necessary.
4. In the Relationship pane, select a relationship, then edit the template for its label in the **Backward Analysis** or **Forward Analysis** fields as appropriate. Items preceded by a backslash are replaced with attributes reported in the Subtree Report. You can modify unescaped text as necessary.
5. Click **More** in the Report Attributes pane. In the Report Attributes dialog, select the attributes you want to include in the Subtree Report.
6. Choose **Display Business Names** if you want to display business names in the impact trace rather than original identifiers.

Setting Impact Pane Project Options: Processing Tab

Use the **Impact > Processing** tab of the Project Options window to specify the variables to filter out of the trace, whether nested data items or redefines are treated as startup items, and whether recursive impacts or controlled dependency relationships are included in the trace.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Impact > Processing** tab.
2. Choose any combination of:
 - **Add to Roots Data Item Subitems** if you want the trace to include, as startup items, data items in nested declarations of the startup items.
 - **Add to Roots Data Item Redefines** if you want the trace to include, as startup items, data items that redefine the selected startup items.

- **Process Immediate Fields Individually** if you want the trace to treat each subitem of the startup item as a root. The startup item is not included in the trace.
- **Self-Dependencies** if you want the trace to include recursive impacts on data items.
- **Controlled Conditions** if you want the trace to include controlled dependency relationships for data items, such as that between A and B in IF ... A THEN MOVE... TO B. In this example, B depends on A, since B is assigned a value under the control of a condition that depends on A. Choose:

- **Include Implicit Conditions** if you want the trace to include relationships derived from implicit control conditions. In the following example, statement (1) is explicitly controlled by the condition A > 0 because it is nested in the IF statement. Statement (2) is implicitly controlled by the condition because, on the one hand, there is no surrounding IF statement but, on the other hand, control may not reach the statement if the condition is not satisfied:

```
IF A > 0 THEN
MOVE A TO B (1)
ELSE
GOBACK.
MOVE B TO C. (2)
```



- **Include Calculation of Conditions** if you want the trace to show how the controlling data item is calculated.













3. In the Variable Name Filters pane, select the patterns that match the names of the variables you want to filter out of the impact trace. Recognized patterns are listed in the pane. Add patterns as necessary. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).



Setting Impact Pane Project Options: Relationships Tab

Use the **Impact > Relationships** tab of the Project Options window to specify the intraprogram and interprogram data flow relationships to include in the trace. The table below describes the intraprogram and interprogram relationships detected by the data flow analysis tools.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Impact > Relationships** tab.
2. In the Intraprogram pane, place a check mark next to each intraprogram data flow relationship you want the trace to include.
3. In the Interprogram pane, place a check mark next to each interprogram data flow relationship you want the trace to include.
4. Remove the check mark next to **Show All Used** if you want the trace to include only used nodes that are necessary to its calculation.

Relationship	Definition	Type	Icon	Description
calls	N/A	interprogram	N/A	A parameter passed in a call to another program.
cast	MOVE A TO B with data conversion	intraprogram		A data item moved to a data item of a different type.
common area transitions	N/A	interprogram	N/A	For Unisys 2200 Cobol, a common-storage data area item passed in a call to another program. Perform Unisys Common-Storage Area Analysis must be set in the project verification options.
comp	STRING A ... INTO B	intraprogram		An arbitrary computation. The result is produced by applying complex rules to the argument, such as STRING.

Relationship	Definition	Type	Icon	Description
comp+	ADD A TO B	intraprogram		An addition-like operation: ADD, SUBTRACT, or corresponding parts of COMPUTE.
comp*	MULTIPLY A BY B	intraprogram		A multiplication-like operation: MULTIPLY, DIVIDE, or corresponding parts of COMPUTE.
comp@	MOVE ARRAY (IDX) TO A	intraprogram		An operation with array elements.
cond	IF A = B ...	intraprogram		Comparison of data items with a symmetric relationship.
cond*	IF A * X = B ...	intraprogram		Comparison of a multiple of a data item with another data item.
const cond	IF A = 1 ...	intraprogram		Comparison of a data item with a constant.
const.move	MOVE 1 TO B	intraprogram		A constant moved into a data item.
const.comp	ADD 1 TO B	intraprogram		An arithmetic operation with constants.
const.init	03 A ... VALUE 1	intraprogram		A data item initialized by a constant.
DMS records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via Unisys DMS database records.
files	N/A	interprogram	N/A	Data communication via files. Traced only when corresponding JCL, ECL, FCT, or CSD files are verified.
files in jobs	N/A	interprogram	N/A	Data flow in JCL datasets when <i>files</i> is selected.
input port	N/A	intraprogram		A data item in which data is received.
move	MOVE A TO B	intraprogram		A data item moved to a data item of the same type.
network records	N/A	interprogram	N/A	For Unisys 2200 Cobol, data communication via network records.
output port	N/A	intraprogram		A data item from which data is sent.
screens	N/A	interprogram	N/A	Data sent to a screen by one program and received in a screen by another.
screen definitions	N/A	interprogram	N/A	Data flow in screen fields when <i>screens</i> is selected.


Relationship	Definition	Type	Icon	Description
start	N/A	intraprogram		The startup item in an Impact pane consolidated analysis.
used	MOVE ... TO A ... MOVE A TO ...	intraprogram		A value assigned in a statement used as an argument in another statement.


Setting Impact Pane Project Options: Impact Reports Tab

Use the **Impact > Impact Reports** tab of the Project Options window to exclude repeated items from the Impact Subtree Report and to limit the depth of the trace in all reports.



1. Choose **Options > Project Options**. The Project Options window opens. Click the **Impact > Impact Reports** tab.
2. Select **Output duplicates in the SubTree report** if you want to include repeated items in the Impact Subtree Report.
3. Select **Limit Impact Depth** if you want to limit the depth of the trace reported, then specify the maximum depth of the trace in the **Max Depth** combo box.

Exporting and Importing an Impact Trace


To export an impact trace in XML format, select its startup item and click the  button on the tool bar. A Save As dialog opens, where you can specify the name and location of the impact trace.

To import a saved impact trace, click the  button on the tool bar. An Open dialog appears, where you can select the trace to import.

Generating a Subtree Report

A Subtree Report shows the trace for a subtree in XML format or in a database. To generate a Subtree Report, click the  button next to the  button on the tool bar and choose **SubTree Report** in the drop-down menu. A Save As dialog opens, where you can specify the name, location, and file type of the report.



The following table shows the columns displayed in the database report format. You can control the level of detail displayed in the report in the Impact Pane User Preferences.

 **Note:** The attributes listed in the table are a superset of the attributes listed in the Impact Pane User Preferences. Database IDs refer to IDs generated for the database report format.



Attribute	Description
CallStack	Not used.
Cnt	Not used.
Col	Number of the source column that contains the beginning of the affected data item.
Context	Internal ID of the context for the affected data item.
DeclHCID	Internal ID of the declaration for the affected data item.
Depth	The depth of the trace.
Direction	The direction of the trace, forward or backward.
Duplicate	Indicates that the affected data item is repeated in the trace.
EntityType	Construct type of the affected data item.

Attribute	Description
Excluded	Indicates that the affected data item is filtered from the trace.
File	Name of the file that contains the affected data item.
FromSel	Character offset from the start of the file to the beginning of the affected data item, counting from 0.
HCID	Internal ID of the affected data item.
ID	Database ID of the affected data item.
JobName	Name of the job that contains the affected data item.
JobStepID	Database ID of the job step that contains the affected data item.
Keyword	The operation on the affected data item.
Ln	Number of the source line that contains the affected data item.
Name	Name of the affected data item.
ObjName	Name of the object that contains the affected data item.
ObjType	Type of the object that contains the affected data item.
Offset	Memory offset of the affected data item.
OS	Memory offset and size in bytes of the affected data item, separated by a colon (:).
PortFlag	Port type of the affected data item: I for input, O for output, N for not a port.
PortName	Name of the port referenced in the operation identified by the Keyword.
PrevID	Database ID of the affected data item's parent node.
RelType	Data flow relationship type for the affected data item.
RootID	Database ID of the root node of the tree.
Size	Size in bytes of the affected data item.
SourceID	Database ID of the file that contains the affected data item.
SourceText	Text of the source line that contains the affected data item.
SourceType	Source type of the file that contains the affected data item.
StepName	Name of the job step that contains the affected data item.
StepNum (also Step)	Number of the job step that contains the affected data item.
ToSel	Character offset from the start of the file to the end of the affected data item, counting from 0.

Generating a Pane Report

A Pane Report shows the entire tree for a trace in XML format. To generate a Pane Report, click the  button next to the  button on the tool bar and choose **Pane Report** in the drop-down menu. A Save As dialog opens, where you can specify the name, location, and file type of the report.

Generating an Affected Code Report (COBOL Only)

An Affected Code Report shows code that would be impacted by changing a data item's definition or usage (COBOL only). To generate an Affected Code Report, click the  button next to the  button on the tool bar and choose **Affected Code Report** in the drop-down menu. A Save As dialog opens, where you can specify the name, location, and file type of the report.

Analyzing Program Control Flows

Control flows describe the processing paths in a program. Call flows, decision flows, and statement flows each offer a different way of understanding the procedures in a program. Interactive Analysis offers the following tools for analyzing program control flows:

- The Program Control Flow pane displays a diagram that shows the call flow for paragraphs in a COBOL program, subroutines in an RPG program, or procedures in a PL/I or Natural program.
- The Execution Path pane displays a hierarchical view and diagram that show the conditions that determine the flow of control in a COBOL or PL/I program.
- The Flowchart pane displays a diagram that shows the flow of control between statements in a COBOL paragraph or PL/I procedure, or between steps in a job or JCL procedure.
- The Animator lets you step through COBOL or Natural code displayed in a Interactive Analysis pane.

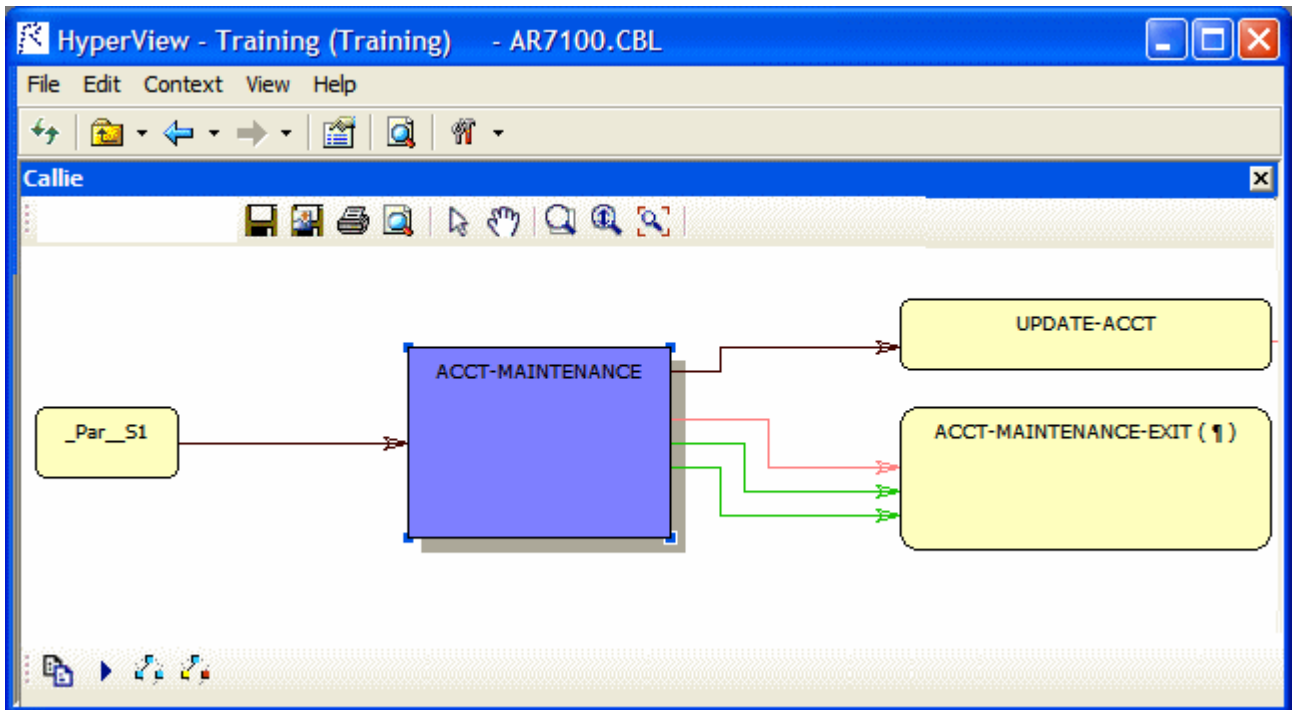
Using the Program Control Flow Pane

The Program Control Flow pane displays a diagram that shows the call flow for paragraphs in a COBOL program, subroutines in an RPG program, or procedures in a PL/I or Natural program. The flow is traced from a *seed* item to a called external program, subroutine, or procedure (if any).


Call relationship lines are color-coded as specified in the table below.

Color	Relationship
Green	GO TO
Red	Fall Thru
Brown	PERFORM
Blue	External call

The toolbar at the top of the pane displays icons used in the main Diagrammer window. For Diagrammer usage, see *Analyzing Projects* in the product documentation set.



Drawing the Diagram


To display a call flow diagram in the Program Control Flow pane, select the seed item in the Source or Context panes. The diagram is automatically redrawn when you select a new seed item in the Source or Context pane, unless the new seed already appears in the current diagram. To redraw the diagram in that case, select the new seed item in the diagram and click the  button on the toolbar at the bottom of the pane.

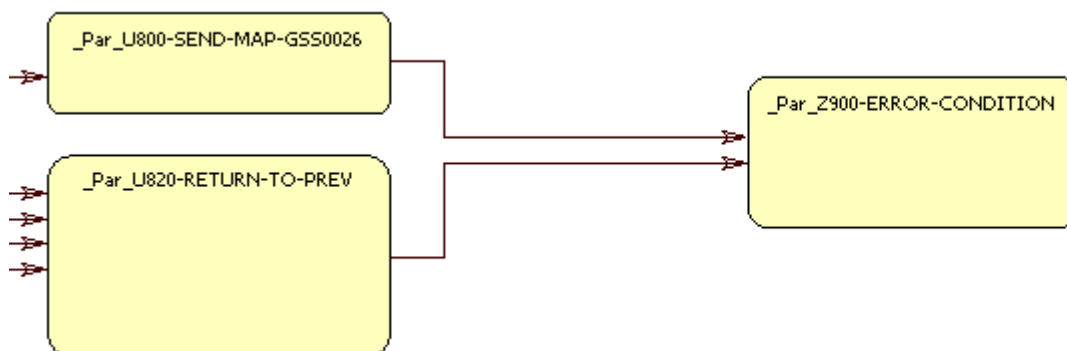
Choosing the Diagram View

The Program Control Flow pane offers two views of program control flows, a subgraph view and a subtree view. The figures below show the same paragraphs in both views.

SubGraph View

The SubGraph view offers a cyclic representation of the information in the diagram. Items are drawn once. Relationship lines cross. SubGraph views are often easier to understand than subtree views. Click

SubGraph mode  on the toolbar at the top of the pane to select the SubGraph view.

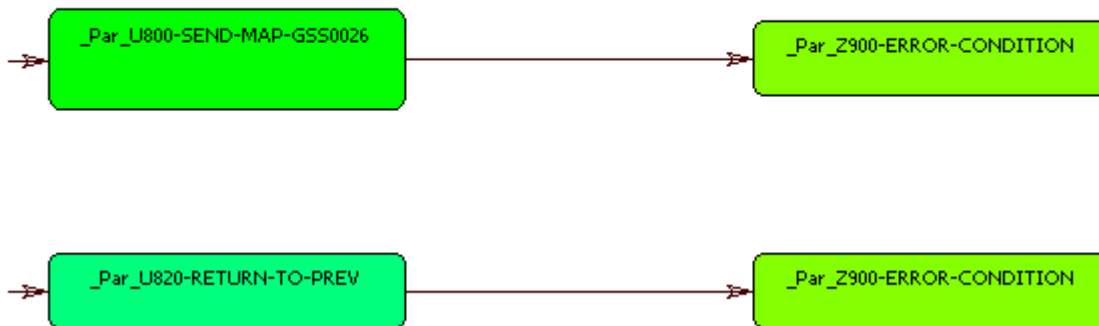


SubTree View

The SubTree view offers a linear representation of the information in the diagram. Items are represented as colored nodes and are drawn as many times as necessary. If two nodes are of the same color, they represent the same item and clicking on them would take you to the same location in the source code.

Relationship lines do not cross. Use this view if too many intersecting lines make a SubGraph view hard to read.

To select the SubTree view, click **SubTree mode**  on the toolbar at the top of the pane.




Selecting Items

Select an internal item in a call flow diagram to navigate to it in the Source and Context panes. For Cobol, select an external program to display it in the Source and Context panes. For PL/I and Natural, select an external procedure and choose **Switch to Procedure** in the right-click menu to display it in the Source and Context panes. Select a relationship line to navigate to the call in the Source and Context panes.

Setting Program Control Flow User Preferences

Use the Interactive Analysis > Program Control Flow tab of the User Preferences window to specify the default box color for items in subgraph views and the color of the selected item in subgraph views. You can also specify the number of visited nodes that Program Control Flow displays in different colors in subtree views. For both views, you can specify that business names replace original identifiers in the diagram.

1. Click **Tools > User Preferences**. The User Preferences window opens. Click the Interactive Analysis > Program Control Flow tab.
2. The current default background color of the box representing an item in the subgraph view is displayed in the **Default Box Color** drop-down. The current background color of the box representing the selected item in the subgraph view is displayed in the **Selected Box Color** drop-down. Click the adjacent  button to edit the color of the boxes.
3. In the **Track Length** combo box, enter the number of visited nodes in the subtree view that Program Control Flow displays in different colors.
4. Select **Display Business Names** to display business names rather than original identifiers in the Program Control Flow pane.

Setting Program Control Flow Project Options

Use the Program Control Flow tab of the Project Options window to control the depth of the flow trace: how many calling and called nodes in a sequence the diagram displays to and from the seed item. You can also limit the relationship types the diagram displays and filter out items by name.

1. Choose **Options > Project Options**. The Project Options window opens. Click the Program Control Flow tab.
2. In the Connections pane, select the relationship types you want to view in the diagram. Select:
 - **Show GO TO** to show GO TO relationships.
 - **Show Performs** to show PERFORMS relationships.
 - **Show External Calls** to show external call relationships.
 - **Show Follows** to show Fall Thru relationships.
 - **Merge Duplicate Links** to show a single relationship line rather than multiple lines when constructs are connected by identical call relationships.
3. In the Neighborhood Size pane, use the **Call Depth** and **Caller Depth** sliders to specify the depth of the flow trace: how many calling and called nodes in a sequence, respectively, the diagram displays to and from the seed item.



Note: If Program Control Flow loads too slowly in subgraph mode, set these options to values lower than 5.

4. In the Paragraph Name Filters pane, select the pattern that matches the names of items you want to exclude from the diagram. The recognized patterns are listed in the Paragraph Name Filters pane. Add patterns as needed.

Using the Execution Path Pane


The Execution Path pane displays a hierarchical view and diagram of the conditions that determine the control flow in a program. Each view traces the control flow from the first involved condition to the *seed* construct selected in the Source or Context pane. The seed construct can be either a paragraph or statement. Use an execution path to ensure that a code segment encapsulates all of the business logic for a candidate business rule.

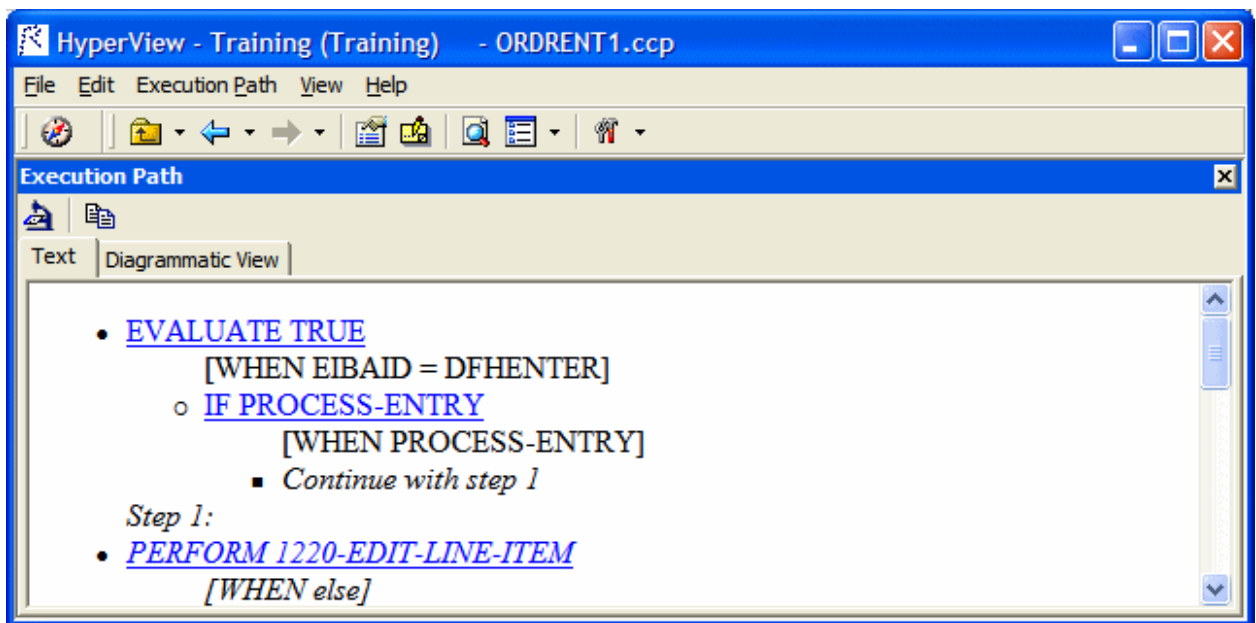


Note: For a COBOL program, the Execution Path tool may give incorrect results if the program contains a paragraph that is used in multiple PERFORM statements or in some combination of PERFORM and GO TO statements.

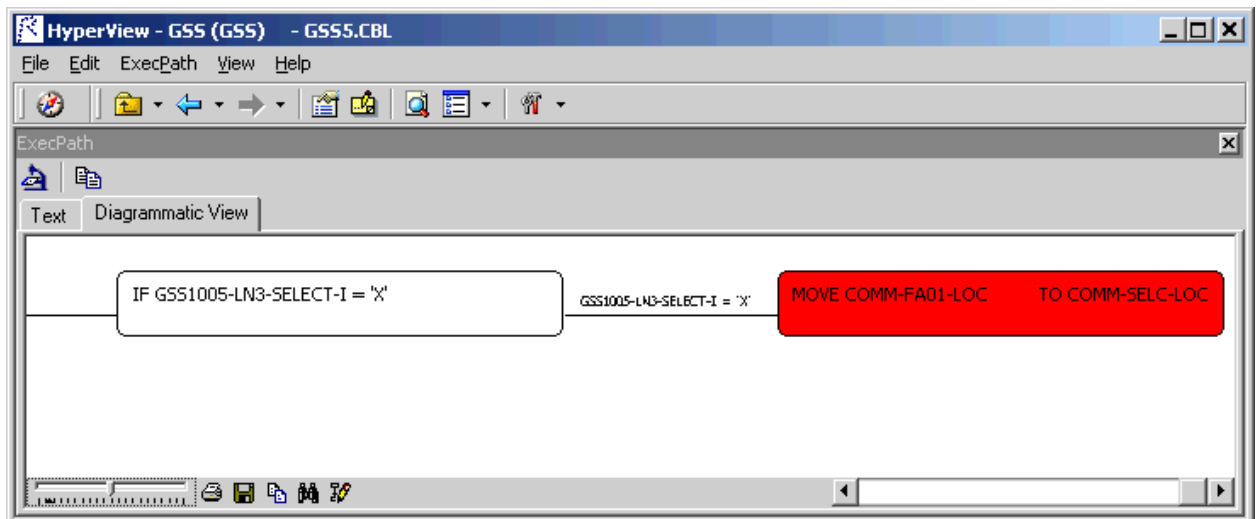
1. Select the seed construct in the Source or Context pane.
2. Choose **Execution Path > Analyze Selected Construct**. Interactive Analysis displays the execution path for the seed construct



Tip: Click the  button on the tool bar to copy the execution path to the clipboard with all formatting preserved.



3. Click the Diagrammatic View tab to display the execution path in a diagram. The seed construct is displayed in red. The tool bar displays icons used in the common diagrammer. For more information, see *Analyzing Projects* in the product documentation set.



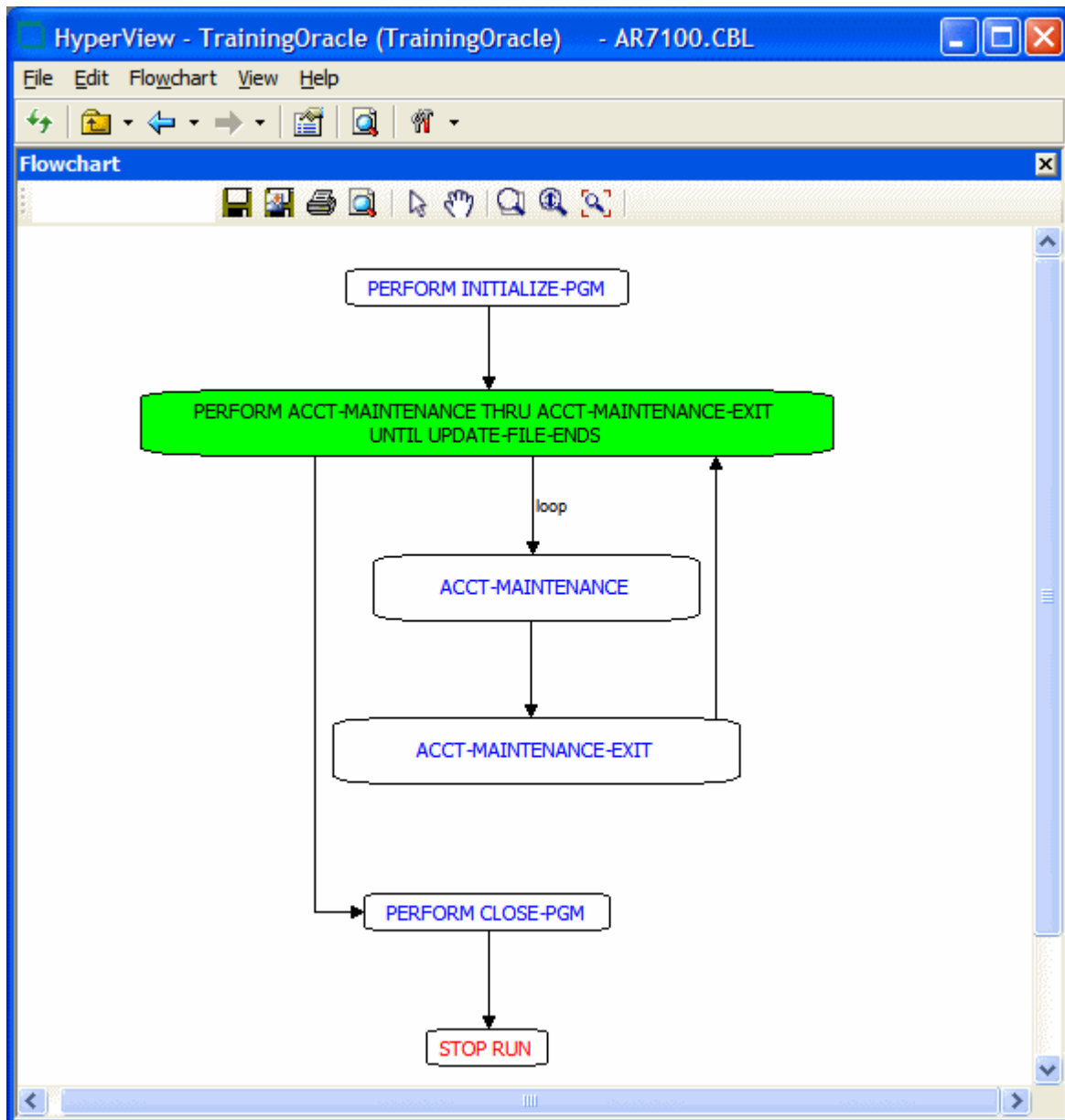
Using the Flowchart Pane

The Flowchart pane displays a diagram that shows the flow of control between statements in a Cobol paragraph or PL/I procedure, or between steps in a job or JCL procedure. The label of the box representing an item in the flow contains the text of the item.

PERFORM statements are displayed with a blue label. GO TO, RETURN, and STOP RUN statements are displayed with a red label. Conditions or conditional statement are displayed in boxes with a green background. Captions along relationship lines show the conditions or blocks that determine the flow. Select a statement or condition to navigate to it in the Source and Context panes.

Job step relationships are collapsed in a group. Double-click the maximize button for the group to expand the relationships. Double-click the minimize button for the group to collapse the relationships.

Options control grouping of same-level statements and whether business names are displayed. The toolbar at the top of the pane displays icons used in the main Diagrammer window. For Diagrammer usage, see *Analyzing Projects* in the product documentation set.



Drawing the Diagram

To display a flowchart diagram in the Flowchart pane, select the paragraph, procedure, job, or JCL procedure you want to analyze in the Source or Context panes. The diagram is automatically redrawn when you select a new startup item in the Source or Context pane.

Setting Flowchart User Preferences


Use the Interactive Analysis > Flowchart tab of the User Preferences window to control grouping of same-level statements and whether business names are displayed.

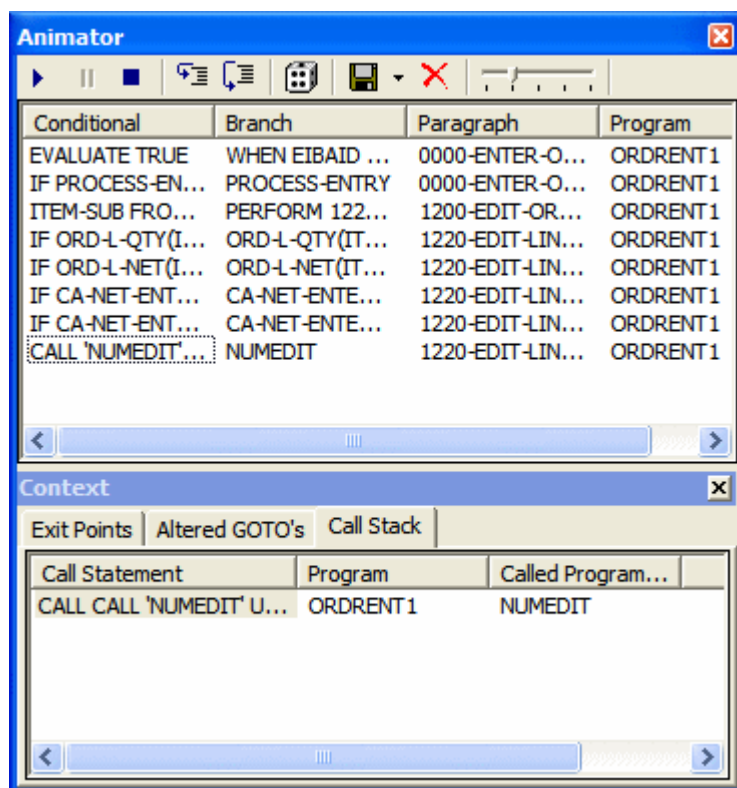
1. Choose **Tools > User Preferences**. The User Preferences window opens. Click the Interactive Analysis > Flowchart tab.


2. Select **Group Same Level Names** to collapse same-level statements in a group. Double-click the maximize button for the group to expand the statements. Double-click the minimize button for the group to collapse the statements.
3. Select **Display Business Names** to display business names rather than original identifiers in the Flowchart pane.


Using the Animator


The Animator lets you step through the code displayed in a Interactive Analysis pane. You can choose program branches yourself, or have the animator choose them randomly.

 **Tip:** The Animator is especially useful in tracking execution paths in Program Control Flow subgraph views.






1. Select the construct you want to start the animation from in the Source or Context pane.
2. Click the  button on the tool bar if you want Animator to choose logical branches randomly.









 **Tip:** Use the slider at the right of the tool bar to set the speed of the animation.

3. Click the  button on the tool bar to start the animation. If you are choosing program branches yourself, Animator stops at each condition it encounters and displays a pop-up menu that prompts you to choose a logical branch. Click the branch you want to go down, or click **Break** to pause the animation.

Interactive Analysis populates the top part of the Animator window with each condition it encounters, the logical branch that you or the Animator chose, and the paragraph that contains the condition.

4. In the Context pane at the bottom of the window, click the appropriate tab to view exit points, altered GO TOs, or the call stack for the paragraph currently being executed. You can hide the Context pane by choosing the **Animator > Show Context**. Choose it again to show the Context pane.

 **Tip:** Click the  button on the tool bar to step through the code manually. Each time you click the button the Animator steps into the next piece of program code. Click the  button if you want to step through the code manually but step over PERFORM statements.

5. Click the  button on the tool bar to pause the animation. Click the  button to restart the animation. Click the  button to stop the animation.
6. To save the animation results:
 - To an Interactive Analysis list, click the  button next to the  button on the tool bar and choose **Save Trace As List** in the drop-down menu. You can view the list in the Internal category in Code Search.
 - To XML, click the  button next to the  button on the tool bar and choose **Save Trace As XML** in the drop-down menu. A Save As dialog opens, where you can specify the name and location of the results report.
7. Click the  to delete the results from the Animator window.

Setting Up a Glossary

Use the Glossary pane to create a dictionary, or *glossary*, of meaningful names for items of interest in your workspace: data elements, paragraphs, programs, screens, tables, and the like. A glossary is especially useful for assigning business names to identifiers: natural language names that make it easy to document and retrieve business rules. The field PLC_NUM, for example, might have the business name Policy Number.

The Glossary pane lets you assign business names to identifiers manually or in batch mode. You can auto-extract business names from screens, import business names into a glossary from a file, and much more.



Note:

- The scope for Glossary is the whole workspace.
- We recommend that any mass updates should be done by a master user while no one else is updating the Glossary.
- Mass update functions such as Generate Business Names, Propagate Business Names, and Import Business Attributes should not be executed by more than one master user at a time, from one Interactive Analysis instance.
- All mass delete operations (Delete All, Delete Manual, Delete Automatic and so on) are Workspace wide operations and we recommended that these be performed only by the master user.
- Users should not perform any update/delete functions while a mass update is in progress.

Understanding the Glossary Pane

A *term* is the name of an item of interest in your workspace. The Glossary pane automatically generates a list of terms for the file selected in the Objects pane and any included files. Glossary search facilities let you home in on terms of interest in the workspace.

The Glossary tool consists of two panes:

- The Terms pane displays the terms in the selected file, filtered by your choices in the workspace options for the Glossary pane.
- The Search Result pane displays the terms returned by the most recent Glossary search. The scope of the search is workspace-wide, unless you use the advanced search feature to filter explicitly by project.

Each pane displays the term type, the business name and description assigned to the term, and how the business name and description were assigned (Manual, Automatic, Extracted from Screen, or From Propagation). To hide a pane, choose **Glossary > View Terms** or **Glossary > View Search Result**, as appropriate. Deselect the menu item to show the pane again.

To synch the Terms pane with other Interactive Analysis panes, choose **Glossary > Synchronize with other views**. Select a term in the Terms pane to navigate to the term in other Interactive Analysis panes. Deselect **Glossary > Synchronize with other views** to turn off synchronization.



Note: Turning off synchronization may improve Glossary performance for very large workspaces.

Searching for Terms

Glossary search facilities let you home in on terms of interest in your workspace:

- Use a *simple search* to perform coarse-grained searches for terms.
- Use an *advanced search* to perform narrowly targeted searches for terms.


The scope of all searches is workspace-wide, unless you use the advanced search feature to filter explicitly by project. Results are returned in the Search Result pane. To clear the Search Result pane, choose **Glossary > Clear Search Result**.





Using the Simple Search Facility

Use a *simple search* to perform coarse-grained searches for terms. Follow the instructions below to use the simple search facility.

1. Enter the text for the search (or select the text of a recent search) in the **Search** combo box on the tool bar. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).



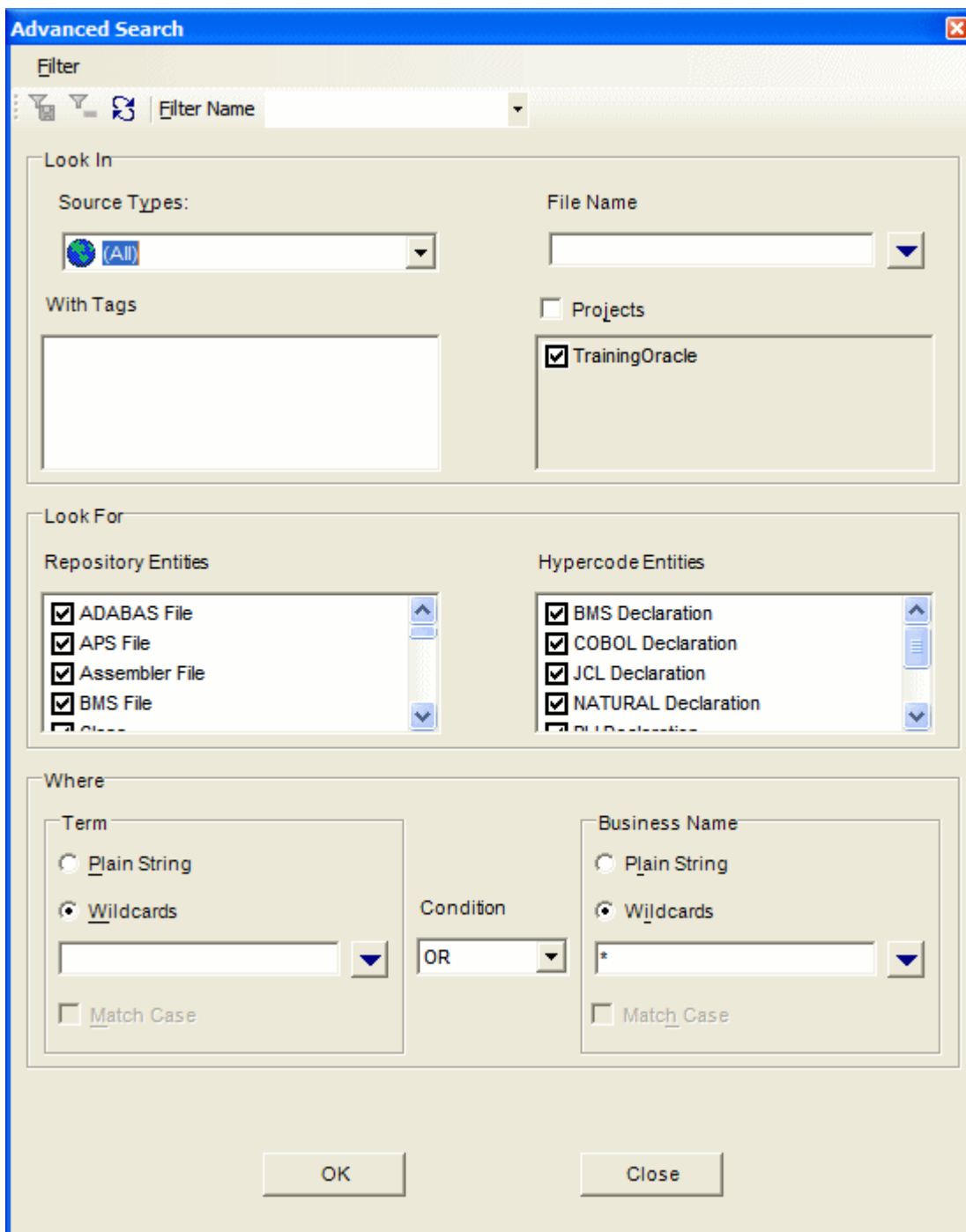
Note: Select a column heading and click the  button on the tool bar, then enter text in the Search combo box to scroll intelligently to items in that column. The tool matches text as you type.

2. Click the  button on the tool bar to match business names only. Click both the  button and the  button to match terms and business names (search terms are ORed).
3. Click the  button on the tool bar. The results of the search are displayed in the Search Result pane.

Using the Advanced Search Facility

Use an *advanced search* to perform narrowly targeted searches for terms. Follow the instructions below to use the advanced search facility.

1. Choose **Glossary > Advanced Search**. The Advanced Search dialog opens.




2. You can create a new search filter or edit an existing search filter:

- To create a new search filter, enter the name of the filter in the **Filter Name** combo box.
- To edit an existing search filter, select the filter in the **Filter Name** combo box.



Note: To save a search filter, choose **Filter > Save Current Filter**. To delete a search filter, choose **Filter > Delete Filter**. To restore a filter to its initial values, choose **Filter > Reset Values**.


3. To filter on source file type, select the type in the **Source Type** drop-down.
4. To filter on source file name, select the **File Name** check box, then select the matching pattern for the name in the combo box below. Enter a new matching pattern by typing in the combo box. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). Click the  button

next to the drop-down to display a pop-up menu of elements you can insert in the pattern. Click an element to insert it. The matching pattern is saved when you save or execute the filter.

5. To filter on tags, select each tag you want to filter on in the With Tags pane. The search will return matching repository objects that have been assigned the tag and matching HyperCode constructs contained in repository objects that have been assigned the tag.
6. To filter on projects, select the **Project** check box, then select the projects in the list pane below.
7. To filter on repository object type, select each object type you want to filter on in the Repository Entities pane.
8. To filter on HyperCode construct type, select each construct type you want to filter on in the Hypercode Entities pane.



Note: Both repository and Hypercode entities are filtered by your choices in the workspace options for the Glossary pane.

9. To filter on the text for a term and/or business name, enter a matching pattern for the text in the combo boxes for terms and/or business names in the Where pane. You can enter plain character strings or wildcard patterns.
 - a) Click the radio button for the type of pattern matching you want to use.
 - b) For wildcard patterns, click the  button next to the drop-down to display a list of elements you can insert in the pattern. Click an element to insert it. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). The matching pattern is saved when you save or execute the filter.
 - c) In the **Condition** drop-down, select **AND** if you want the search terms to be ANDed, or **OR** if you want the search terms to be ORed.
 - d) Select the **Match Case** check box if you want the search to match case.
10. Click **OK** to execute the filter. The Glossary tool displays the search results in the Search Result pane.



Note: If you execute another advanced search with the same criteria for the term or business name, the contents of the Search Result pane act as a filter. If you don't want to filter on existing results, choose **Glossary > Clear Search Result** to clear the Search Result pane.

Creating a List of Search Results

To create a Code Search list containing search results, choose **Glossary > Add to Code Search List**. Specify the list name in the **New list** dialog box and click **OK**. The list is displayed in the Glossary category in Code Search.

Assigning Business Attributes

You can assign business names manually or in batch mode. You can also export terms and business attributes from the glossary for one workspace for import into the glossary of another.

Assigning Business Names Manually

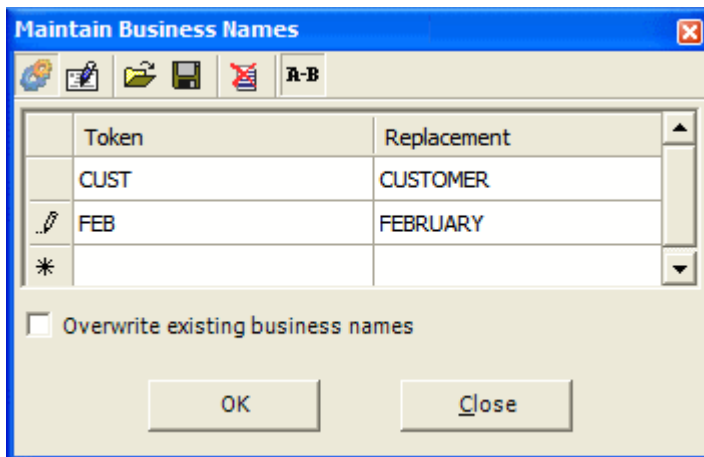
To assign a business name and business description to an object, select the object and choose **Set Business Attributes** in the right-click menu. A dialog box opens, where you can enter the business name and business description. To unassign a business name or business description, simply delete the value in the dialog box.



Assigning Business Names in Batch Mode





Ordinarily, you will want to assign business names in batch mode rather than manually. In the batch interface, you enter replacement text for each token or string you want to change in a term. For the string NUM, for example, you would enter the replacement text Number. You can also modify business names themselves in batch mode.


A token is an element in a program identifier delimited by a hyphen (-) or underscore (_). In the identifier WS01-CUST-FIELD, for example, there are three tokens: WS01, CUST, and FIELD.

1. Choose **Glossary > Generate/Modify Business Names**. The Maintain Business Names dialog opens.



2. Click the  button on the toolbar to enable business name generation.
3. Click the  button on the toolbar to enable token or string replacement. The button is a toggle.
4. In the **Token/String** field, enter the token or string you want to replace. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). In the **Replacement** field, enter the replacement string. Repeat the procedure for each token or string you want to replace.

 **Note:** Click the  button on the toolbar to clear the dialog. Click the  button on the toolbar to save the list of entries to a file. Click the  button on the toolbar to load a list of entries into the dialog.

5. Select **Overwrite existing business names** if you want existing business names to be overwritten by the matching text.
6. Click **OK**. The Glossary tool automatically generates business names for each term that contains the token or string, replacing the string with the text you specified.
7. To modify business names, click the  button on the toolbar, then repeat the procedure described above for string replacement.

Extracting Business Names from Screens

One of the most powerful features of COBOL Analyzer lets you *auto-extract screen labels* as business names for program variables. For each screen field with a label immediately to its left in the Screen pane, the Glossary tool generates a business name for the program variable whose value is displayed in the field. If the screen field has the label **Customer Number**, for example, Glossary assigns the business name "Customer Number" to the variable whose value is displayed in the field.

To extract business names from all the screens in the workspace, choose **Glossary > Extract From All Screens**. To extract business names only from the screen selected in the Source or Context panes, choose **Glossary > Extract From Current Screen**.

Importing and Exporting Business Attributes

You can export terms and business attributes from the glossary for one workspace for import into the glossary of another. You can also import terms and business attributes propagated programmatically. Term (source), business name (replacement), and business description (description) must appear in XML file format as follows:

```
<glossary>
  <item source="CUST_INFO" replacement="CUSTOMERINFORMATION"
    description="customer information"/>
</glossary>
```

```
...  
</glossary>
```

To export business attributes, choose **Glossary > Export Business Attributes**. A save as dialog opens, where you can specify the name and location of the file to export.

To import business attributes, choose **Glossary > Import Business Attributes**. An open dialog displays, where you can specify the file to import. You are prompted to confirm whether you want to overwrite existing business attributes. Select **Overwrite existing business attributes** if you want to replace existing business attributes as well as add new ones. Click **OK**.

Propagating Business Names

You can *propagate* the business name for one identifier to a related identifier with an identical meaning in the data flow. Business names can be propagated in either direction of the relationship and transitively:

- For the statements MOVE A to B and MOVE B to C, if A has a business name, it is propagated to B and C.
- For the statements MOVE A to B and MOVE B to C, if C has a business name, it is propagated to B and A.

No business name is assigned to a variable if it receives values from variables with different business names:

```
Working-Storage Section.  
10 Cust-No PIC 9(7).  
10 Ord-No PIC 9(7).  
10 Display-No PIC 9(7).  
  
Procedure Division.  
...  
Move Cust-No to Display-No.  
Display Display-No.  
Move Ord-No to Display-No.  
Display Display-No.  
...
```

If Cust-No has the business name Customer Number, and Ord-No has the business name Order Number, no business name is assigned to Display-No, since it receives values from variables with different business names.

To propagate business names from all existing business names, choose **Glossary > Propagate All**. To propagate business names from selected business names, choose a single business name in the Terms pane or multiple business names in the Search Result pane and choose **Glossary > Propagate Selected**. To propagate only from business names assigned in a specific way, choose **Glossary > Propagate <Assignment Method>**.

Deleting Business Attributes

To delete business attributes for a given term, select the term in the Terms or Search Result pane and choose **Set Business Attributes** in the right-click menu. The Set Business Attributes dialog opens, where you can delete the values in the **Business Name** and **Description** fields.

To delete all business names, right-click in the Terms or Search Result pane and choose **Delete Business Names > Delete All Business Names** in the pop-up menu. To delete only business names assigned in a specific way, right-click in the Terms or Search Result pane and choose **Delete Business Names > Delete <Assignment Method>** in the pop-up menu.

To delete all business descriptions, right-click in the Terms or Search Result pane and choose **Delete All Descriptions** in the pop-up menu.



Note: Mass delete operations are workspace-wide, and should normally be performed by the master user only.

Setting Glossary Workspace Options

Use the Glossary tab of the Workspace Options window to filter the objects and constructs available in the Terms pane and the Advanced Search dialog, and to specify the types of data flow relationships for which business names are propagated.

1. Choose **Options > Workspace Options**. The Workspace Options window opens. Click the **Glossary** tab.
2. Click the **Entities** tab, then the **Hypercode** tab. In the Source Type pane, select the source file type whose constructs you want to include, then select the constructs in the HyperCode Entities pane.
3. Click the **Repository** tab. Select each type of repository object you want to include.
4. Click the **Propagation** tab. Select each data flow relationship you want to propagate business names for.



Note: Click **Select All** to select all the items on a tab. Click **Select All** again to deselect all the items.

Refreshing a Glossary

You need to refresh a glossary if, during the current Glossary session, you assign a business name to an object in another tool and want the change to be reflected in the glossary. To refresh a glossary, choose **Glossary > Refresh Glossary**.

Generating Glossary Reports

The Glossary tool offers two reports:

- The Terms report prints the contents of the Terms pane. To generate a Terms report, choose **Glossary > Terms Report**.
- The Search report prints the contents of the Search Result pane. To generate a Search report, choose **Glossary > Search Report**.

Extracting Business Rules

Much of the code in a legacy application, by some estimates as much as 80%, is devoted to tasks with only a marginal relationship to the business logic of the application. Business rule extraction lets you separate the application's business logic from data transfer and other tasks.


That makes the application easier to understand, document, maintain, and modernize. It also makes it easier to determine the overlap between legacy applications on the one hand, and any gaps in their functionality on the other.

Business Rule Manager (BRM) lets you batch edit rule attributes; create custom attributes suited to your particular needs; match input/output data elements with the business names you've assigned to them in your project glossary; and much more.


Understanding Business Rules


A *business rule* is a named container for program code that performs a discrete task in a business process. The rule identifies and documents the *code segment* that performs this task. A business rule named Calculate Date Difference, for example, might consist of this segment:

```
COMPUTE WS-DATE-VARIANCE =  
WS-C-CARD-DATE-CCYMM - WS-TODAYS-DATE-CCYMM.
```

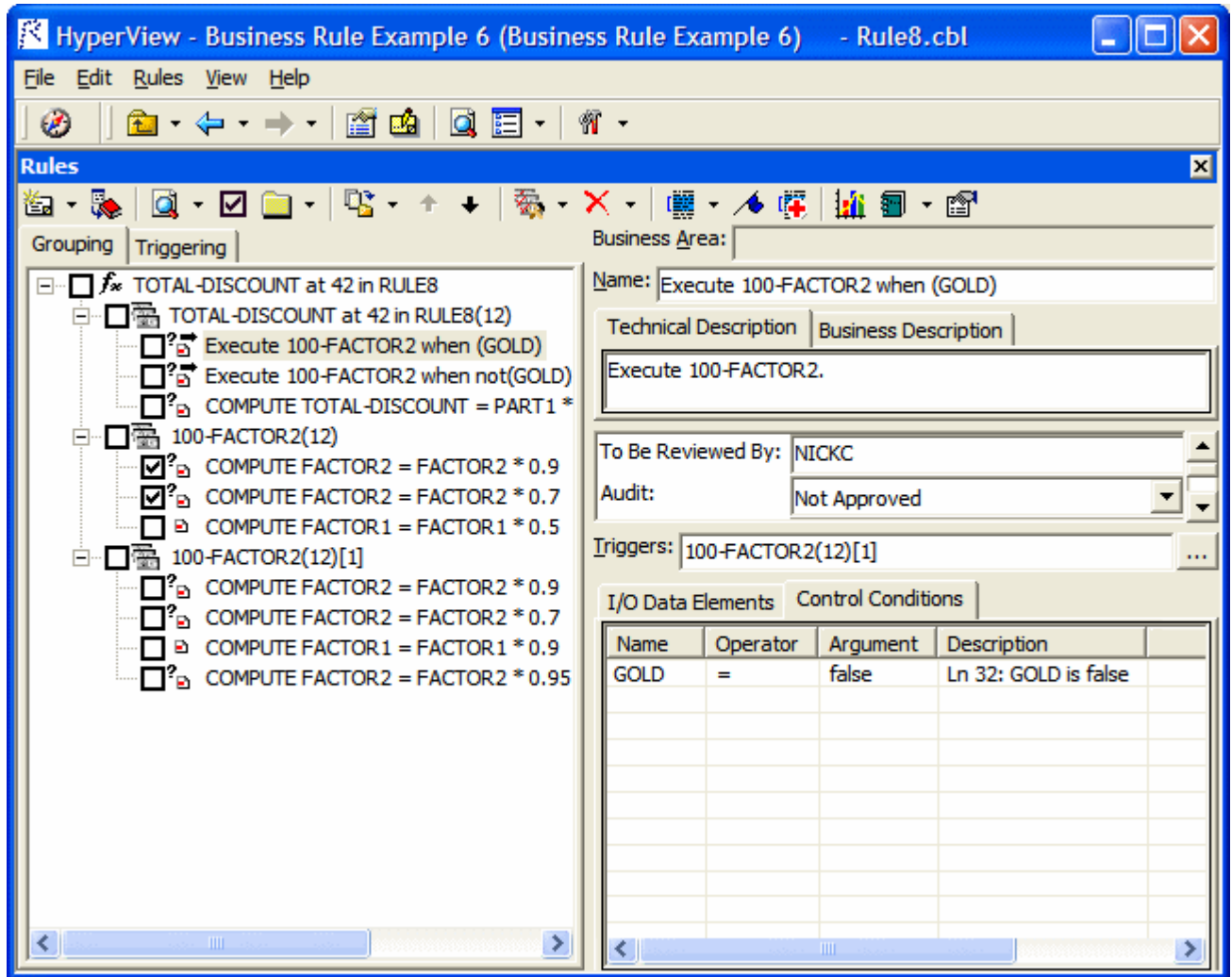
A business rule can contain one segment only. The same segment can be assigned to different rules. Rules are identified in the Rules pane with a  symbol.

How Business Rules Are Organized

You organize business rules in *rule sets*. A rule set typically consists of rules executed together, such that if one rule is executed, all are executed (the statements in a paragraph, for example). Think of a rule set as a business process, like Validate Input or Update File. Rule sets are identified in the Rules pane with a  symbol.

You organize rule sets, in turn, in *business functions*. A business function typically defines the order in which processes are triggered. Think of a business function as a use case, like Insert Customer or Update Order. Business functions are identified in the Rules pane with a  symbol.

Business function names must be unique in the workspace. The same rule set can exist in multiple business functions; the same rule can exist in multiple rule sets. The figure below shows the Interactive Analysis Rules pane with a typical hierarchy of business functions, rule sets, and business rules.



How to Assign Segments to Business Rules

You create business rules from code segments identified manually in source. Use a Code Search list to generate business rules in a specified rule set with predefined attributes.

Understanding Business Rule Attributes

Business rule attributes define the characteristics of the rule: its name, classification, status, business description, and the like. You can view rule attributes in the righthand pane of the Rules window.



Tip: Use the Technical Description, Business Description, Audit, Classification, Status, and Transition attributes for a business function or rule set to indicate that a value applies for all the rules included in the business function or rule set.

You can propagate rule attribute values to one or more rules, and you can use a variety of methods to batch edit rule attributes:

- Code Search lets you generate business rules in a specified rule set with predefined attributes.
- The search facility in the Rules pane lets you locate existing rules with common characteristics, which you can then modify in bulk with the Change Rules feature.

You can modify attribute characteristics and valid values, and you can create custom attributes suited to your particular needs.

Understanding Triggers, I/O Data Elements, and Control Conditions

In addition to the attributes of a rule, the righthand pane of the Rules window displays the input/output data elements of the rule, its control conditions, and whether it triggers a rule set:

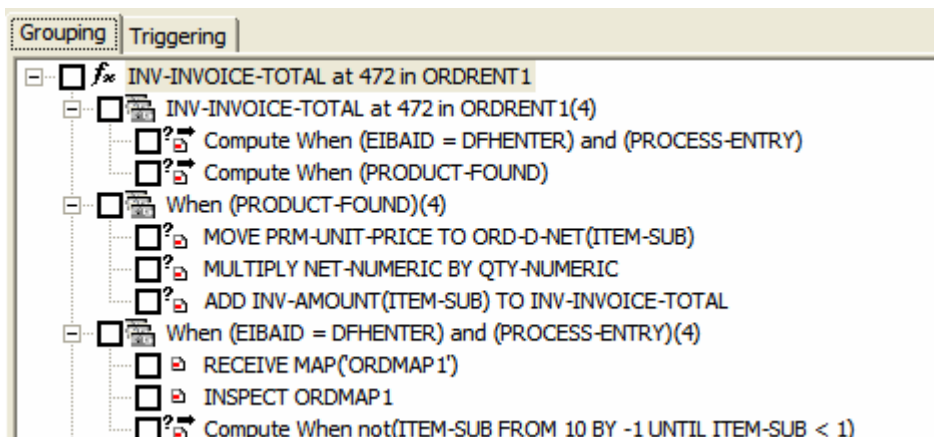
- *I/O data elements* identify the input, output, and input/output fields for a rule.
- *Control conditions* identify the conditions that govern the execution of a rule. A rule may be executed, for example, only when the value of the State Field is California. Rules that execute conditionally are identified in the Rules pane with a symbol.
- A *trigger* is a rule that causes another business process to be started. After the triggered process completes, the triggering process may either continue or terminate. The Valid Phone Number rule in the rule set Validate Input might trigger the rule set Update File, for example. Triggering rules are identified in the Rules pane with a symbol.

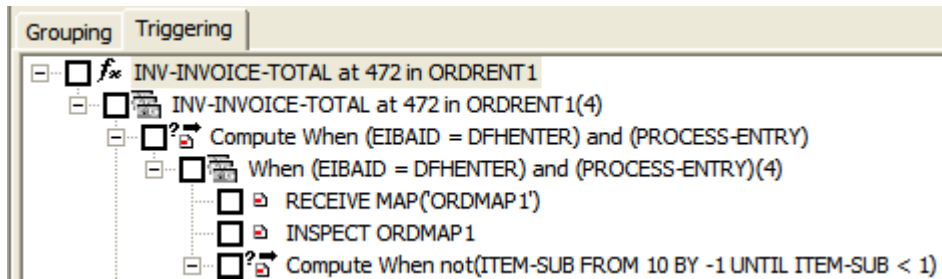
Understanding the Rules Pane Display

The lefthand portion of the Rules pane displays the business functions in your workspace in tree form:

- The Grouping tab lists rules in the order you created them manually, organized by business function and rule set.
- The Triggering tab lists rules in a hierarchy of triggers, in which each triggering rule can be a parent to triggering rules that depend on it in program logic.

The righthand pane displays the attributes and other characteristics of the business function, rule set, or rule selected in the lefthand pane.






Extracting Business Rules Manually

The following instructions describe how to assign a rule segment at the same time as you create a rule. You can also assign a segment after creating a rule.

1. In the Source pane, select the segment you want to assign to a business rule. In the Rules pane, select the business function or rule set (if any) for the new rule, then choose **Create Rule** in the **Rules** menu. The Create Business Rule dialog opens.

 **Tip:** You can select a code segment as construct or text. Select a segment as text when you want to select either more or less code than a construct contains. To select a segment as text, click-and-drag from the first line of the segment to the last line of the segment. The segment is highlighted in blue. To select a segment as construct, click inside the construct in the Source pane. The segment is highlighted in yellow.

2. In the **Business Function** drop-down, select the business function for the rule. The current business function is displayed. To create a new business function, click **New**, then follow the instructions for creating business functions.
3. In the **Rule Set** drop-down, select the rule set for the rule. The current rule set is displayed. To create a new rule set, click **New**, then follow the instructions for creating rule sets.
4. Specify the rule's attributes:
 - In the **Name** field, enter the name of the rule.
 - On the Business Description tab, enter the business description for the rule.
 - On the Technical Description tab, enter the technical description for the rule.
 - On the Attributes tab, specify the remaining attributes of the rule.
5. Click **OK**.

Performing Basic Tasks in the Rules Pane

The lefthand pane of the Rules window lists every business rule in the current project, organized by business function and rule set. To see their attributes in the righthand pane, select:

- A business function.
- A rule set. If a rule set is triggered by a rule, the trigger is listed in the lower portion of the righthand pane.
- A rule to view its attributes and its segment in the Interactive Analysis Source and Context panes.

Creating Business Functions

To create a business function, choose **Rules > Create Business Function**. The Business Function dialog opens. In the Business Function dialog, enter the attributes of the new business function and click **OK**.

Creating Rule Sets

To create a rule set, select the business function for the rule set in the lefthand pane and choose **Rules > Create Rule Set**. The Create Rule Set dialog opens, with the current business function displayed in the

Business Function drop-down. Select another business function if necessary, then enter the attributes of the new rule set and click **OK**.


Selecting Rules for Batch Processing

There are four ways to select rules for Batch Processing:

Default selection (using only the mouse) It marks the current node with all its child nodes. In case of reused rule set, it selects all instances. In the Triggering pane, all nodes down the triggering chains are checked.

Check a rule to make it a candidate for batch processing. Check a rule set to select all rules in it. Check a business function to select all the rule sets and rules in it.

To check all rules, click **Rules > Select All**. Click **Rules > Unselect All** to clear the selection. To invert the current selection, so that unselected rules are selected and selected rules are unselected, click **Rules > Invert Selection**.

 **Note:** Click **Rules > Selected Only** to display only the checked business functions, rule sets, or rules (and their parents or children) in the Rules pane.

Using the Shift key **Shift +** click marks only the current node.


Using the Ctrl key **Ctrl +** click marks the current node and all its sibling nodes.

Using Ctrl + Shift keys **Ctrl + Shift +** click marks the current node with all its child nodes. In the Triggering pane, the check does not affect the triggering chains.

Moving or Copying Rules or Rule Sets


To move or copy rules, place a check mark next to the rules and click **Rules > Copy/Move Selected Rules**. The Copy/Move Selected Rules dialog opens, where you can specify the rule set for the rules and whether you want to move or copy them.

To move or copy rule sets, check the rule sets and click **Rules > Copy/Move Selected Rule Sets**. The Copy/Move Selected Rule Sets dialog opens, where you can specify the business function for the rule sets and whether you want to move, copy, or reuse them. For copied rule sets, a number in curly brackets (*{n}*) is appended to the name to guarantee uniqueness. For reused rule sets, no change is made to the name.


 **Note:** If you attempt to delete a reused rule set, you are prompted to confirm whether you want to delete every instance or only the selected instance.

Copying a Rule or Rule Set with a Different Name

To copy a rule with a different name, select the rule and choose **Rules > Copy/Move**. The Copy/Move dialog opens, where you can specify the rule set for the rule and the new name of the rule.

 **Note:** If you select **Move** rather than **Copy** in this dialog, the rule is moved unchanged to the new location.

To copy a rule set with a different name, select the rule set and choose **Rules > Copy/Move**. The Copy/Move dialog opens, where you can specify the business function for the rule set and the new name of the rule set. You can also specify a technical description or business description. The rules in the rule set are not copied to the new location.

 **Note:** If you select **Move** rather than **Copy** in this dialog, the rule set and all its rules are moved unchanged to the new location.

Deleting Rules, Rule Sets, and Business Functions

To delete a rule, rule set, or business function, select it and choose **Rules > Delete**. You are prompted to confirm the deletion. Click **OK**. You cannot delete a non-empty rule set or business function. To undo the deletion, choose **Rules > Undo Delete**.

Changing the Order of Rules and Rule Sets

To move a rule up in the sequence for its rule set, select it and choose **Rules > Move Up in Sequence**. To move a rule down in the sequence for its rule set, select it and choose **Rules > Move Down in Sequence**. You can also use these menu choices to move a selected rule set up or down in the sequence for its business function.



Tip: For reporting purposes, it's usually best to list rules in order of execution.

Exporting Rules, Rule Sets, and Business Functions

To export rules, rule sets, or business functions to an ERD file, for import to another workspace or a third-party tool, select the items you want to export and choose **Rules > Rule Export**. You are prompted to continue. Click **Yes**. The Export Objects dialog opens, where you can specify the name and folder for the ERD file.



Note: Only the parents of exported items are exported. Children are not. That is, if you export a business function, only the business function is exported. If you export a rule set, only the rule set and its business function are exported.

Importing Rules, Rule Sets, and Business Functions

To import rules, rule sets, or business functions from an ERD file, choose **Rules > Rule Import**. The Import Rules dialog opens, where you can specify the name and folder for the ERD file.

Parents are also imported. If the parent for an imported item already exists in the workspace and has different children, the item is added to the existing children.



Note: If the segment for an imported rule does not exist in the workspace, the segment validity attribute for the rule is set to False.



Assigning Segments to Existing Rules

To assign a segment to an existing rule (whether or not a segment has already been assigned to it), select the rule, then select the segment in the Source pane. Choose **Rules > Assign segment**.

Deleting a Segment from a Rule

To delete a segment from a rule, select the rule and choose **Rules > Delete segment**.

Flagging Segments in Source

To place a  symbol in the Interactive Analysis Source pane display next to each line in the assigned segment, click the  button on the tool bar.

Viewing the Activity Log

To view a chronological record of your activities for a rule, select the rule and choose **Rules > Activity Log**.

Editing Rule Attributes

Before a rule will be useful to other members of your organization, you need to define its attributes: what task the rule performs, whether it has been approved, the last time it was validated, and so forth. You can view rule attributes in the righthand pane of the Interactive Analysis Rules window.



Tip: Use the Technical Description, Business Description, Audit, Classification, Status, and Transition attributes for a business function or rule set to indicate that a value applies for all the rules included in the business function or rule set. Use the Business Area attribute of a business function to classify business functions as members of a still broader category of activity, such as Customer Management.

You can propagate rule attribute values to one or more rules, and you can use a variety of methods to batch edit rule attributes:

- Code Search lets you generate business rules in a specified rule set with predefined attributes.
- The search facility in the Rules pane lets you locate existing rules with common characteristics, which you can then modify in bulk with the Change Rules feature.

You can modify attribute characteristics and valid values, and you can create custom attributes suited to your particular needs.

Editing System Attributes

The names and characteristics of system attributes cannot be modified. The values for many of these attributes are set programmatically.

Editing the Name Attribute

You specify the name of a business function, rule set, or rule when you create it. To edit the Name attribute, enter a new name in the **Name** field.

Editing the Business Area Attribute

The Business Area attribute of a business function identifies the business activity of which the business function is a part, such as Customer Management. To edit the Business Area attribute, enter appropriate text in the **Business Area** field.

Editing the Technical Description Attribute

The Technical Description attribute identifies the role of a rule in program logic. To edit the Technical Description attribute, enter appropriate text in the **Technical Description** tab.

Editing the Business Description Attribute

The Business Description attribute identifies the business role of a rule. To edit the Business Description attribute, enter appropriate text in the **Business Description** tab.

Editing the To Be Reviewed By Attribute

The To Be Reviewed By attribute identifies the business analyst responsible for reviewing and approving a rule. To edit the To Be Reviewed By attribute, enter appropriate text in the **To Be Reviewed By** field.

Editing the Program Attribute

The Program attribute identifies the program that contains the rule segment. The value is supplied when you assign a segment to a rule.

Editing the Segment Validity Attribute

Refreshing or editing source code may result in a mismatch between rules and segments. Lines of code may have been added or deleted during the refresh or edit, causing a rule to no longer to be synchronized with its segment after re-verification.

The Segment Validity attribute specifies whether a rule segment is valid. The value is supplied when a segment is validated or invalidated.

Editing the Last Validation Time Attribute

The Last Validation Time attribute identifies the date and time a rule segment was last validated. The value is supplied when a segment is validated.

Editing User-Defined Attributes

User-defined attributes can be renamed, defined with different values, or deleted altogether. Their default names and values are described below.

Editing the Audit Attribute

Newly created rules are considered unapproved, pending a decision that the rule represents valid business logic. The Audit attribute identifies whether the rule has been audited and approved. To edit the Audit attribute, choose one of the following from the **Audit** drop-down:

- Not Approved if the rule has not been accepted as valid business logic.
- Approved if the rule has been accepted as valid business logic.

Editing the Classification Attribute

The Classification attribute identifies the programming task a rule performs. To edit the classification attribute, choose one of the following from the **Classification** drop-down:

- I/O if the rule performs an input or output function.
- Calculation if the rule performs a calculation.
- Security if the rule performs a security function.
- Decision if the rule resolves a decision.
- Validation if the rule performs validation.
- Flow if the rule performs a data flow function.

Editing the Status Attribute

The Status attribute identifies the status of a rule in a code extraction or other project. To edit the Status attribute, choose one of the following from the **Status** drop-down:

- (none) if the rule has no extraction status.
- Extracted if the rule has been extracted, but not accepted or rejected.
- Working if the rule is still being extracted.
- Accepted if the rule has been accepted.
- Rejected if the rule has been rejected.

Editing the Transition Attribute

The Transition attribute identifies the status of a rule in an application redevelopment project. To edit the Transition attribute, choose one of the following from the **Transition** drop-down:

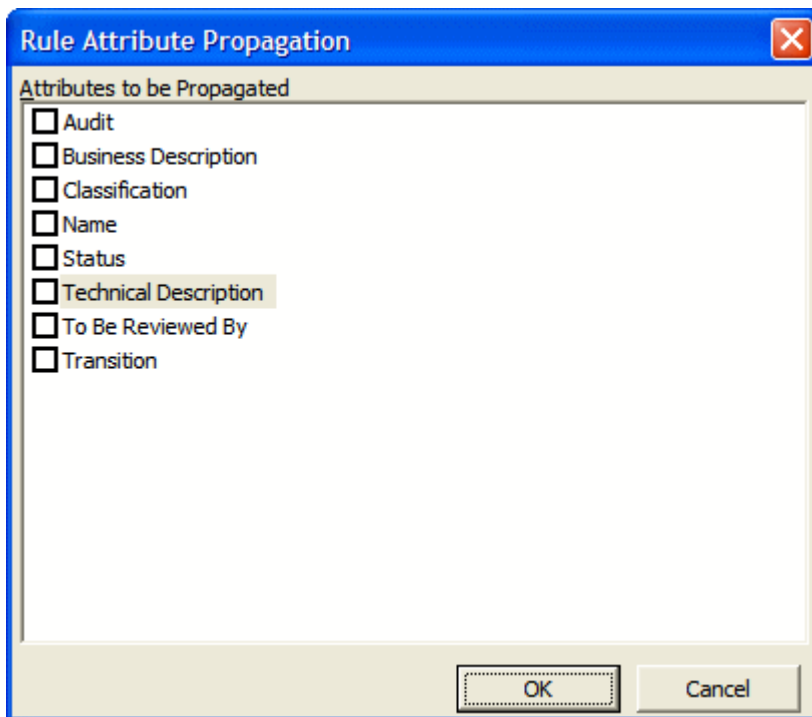
- **Valid as is**, if the rule is valid for redevelopment.
- **Complete**, if the rule has been redeveloped.
- **Obsolete**, if the rule is obsolete.

- **Requires modification**, if the rule requires modification for redevelopment.
- **Duplicate**, if the item is a duplicate of a rule being used for redevelopment.

Propagating Rule Attributes

Follow the steps below to propagate one or more attribute values for a rule to another rule or to a group of rules:


1. Select the rules to which you want to propagate attribute values.
2. Click the rule whose attribute values you want to propagate and choose **Rules > Propagate from Highlighted to Selected**. You are prompted to continue. Click **Yes**. The Attribute Propagation window opens.




3. In the Attribute Propagation window, place a check mark next to each attribute whose value you want to propagate, then click **OK**.

Identifying Triggered Rule Sets

A *trigger* is a rule that causes another business process to be started. After the triggered process completes, the triggering process may either continue or terminate. The Valid Phone Number rule, for example, in the rule set Validate Input might trigger the rule set Update File.

The **Triggers** field for a rule identifies the rule set the rule triggers. Triggering rules are identified in the Rules pane with a  symbol. A rule set can be triggered by multiple rules. A rule can trigger the rule set it belongs to.

If a rule set is triggered by a rule, the trigger is listed in the lower portion of the right-hand pane for the rule set. Double-click the name of a trigger in the list to navigate to the corresponding rule in the lefthand pane.

1. Select the triggering rule and click the  button next to the **Triggers** field in the right-hand pane. The **Select Triggered Rule Set** window opens.



Note: Alternatively, you can access the **Select Triggered Rule Set** functionality from the context menu in the Rules pane from the drop-down menu of the toolbar button **Change Rule Attributes**.

2. Select Triggered Rule Set. To create a new rule set for the rule to trigger, click **New**, then follow the instructions for creating a rule set.

Identifying I/O Data Elements

I/O data elements identify the input, output, and input/output fields in the code for a rule. You can specify the fields manually.

Use the autoretrieve feature to match I/O data elements with the business names you've assigned to them in your project glossary. A business name identifies the business function of a variable in natural language, and is invaluable in documenting and retrieving business rules. The field PLC_NUM, for example, might have the business name Policy Number.



Tip: Use the I/O Data Elements tab for a business function or rule set to specify the data elements for all the rules included in the business function or rule set.

Identifying I/O Data Elements Manually

Follow the instructions below to identify I/O data elements manually.


1. Select a business function, rule set, or rule, then right-click in the I/O Data Elements tab and choose **Create** in the pop-up menu. The I/O Data Element dialog opens.

The screenshot shows a dialog box titled "I/O Data Element". It has a blue title bar with a close button (X) on the right. The dialog contains the following fields and controls:

- Name:** A text input field.
- Kind:** A group box containing three radio buttons: I/O, Input, and Output.
- Business Name:** A group box containing two radio buttons: Auto and Manually. Below the radio buttons is a text input field.
- Description:** A large text area.
- Buttons:** "OK" and "Cancel" buttons at the bottom right.


2. In the **Name** field, enter the name of the data element as it appears in the code.
3. In the Kind group box, choose:
 - **I/O** for an input and output field.
 - **Input** for an input field.
 - **Output** for an output field.
4. In the Business Name area, select:
 - **Auto** if you want to automatically retrieve the business name for the I/O element from your project glossary.
 - **Manually** if you want to enter the business name for the I/O element by hand. Enter the name in the field below the radio button.

5. In the **Description** field, enter a description of the I/O element.
6. Click **OK** to dismiss the dialog and return to the Rules window. The specified field is listed in the I/O Data Elements tab in the righthand pane of the window.
7. Repeat this procedure for each field you want to add to the rule.

 **Tip:** To edit a field in the table, select the field and choose **Edit** in the right-click menu. The I/O Data Element dialog opens, where you can make the necessary changes. To delete a field, select it and choose **Delete** in the right-click menu.


Autodetecting I/O Data Elements

To autodetect I/O data elements, select a rule or several rules, then right-click in the I/O Data Elements tab and choose **Detect I/O Data Elements** in the pop-up menu. You are prompted to continue. Click **Yes**.

 **Note:** Alternatively, you can do this by selecting **Detect I/O Data Elements** from the context menu in the Rules pane or from the drop-down menu of the toolbar button **Change Rule Attributes**.

When the autodetect process is complete, a dialog box displays the number of autodetected I/O elements. Click **OK**. Click the I/O Data Elements tab in the right-hand pane of the Rules window to view the autodetected I/O elements.

To roll back automatic detection, select a rule or several rules, then right-click in the I/O Data Elements tab and choose **Undo I/O Detection** in the pop-up menu.

 **Note:** Alternatively, you can do this by selecting **Undo Last I/O Detection** from the context menu in the Rules pane or from the drop-down menu of the toolbar button **Change Rule Attributes**.

 **Note:** When you verify the project, set **Enable Impact Report** and **Enable Data Element Flow** in the project verification options to enable the autodetect method for I/O data elements.


The method is based on a set of heuristics designed to produce reasonable output. Results may need to be verified by a detailed impact analysis. For more information on the heuristics, contact support services.


Retrieving Business Names for I/O Data Elements

To match I/O data elements with the business names you've assigned to them in your project glossary, select a business function, rule set, or rule, then right-click in the I/O Data Elements tab and choose **Populate I/O Business Names** in the pop-up menu. You are prompted to continue. Click **Yes**.


When the autoretrieve process is complete, a dialog box displays the number of autoretrieved I/O business names. Click **OK**. Click the I/O Data Elements tab in the righthand pane of the Rules window to view the autoretrieved business names.

Identifying Control Conditions

Control conditions identify the conditions that govern the execution of a rule. A rule may be executed, for example, only when the value of the State field is California. Rules that execute conditionally are identified in the Rules pane with  symbol.

 **Tip:** Use the Control Conditions tab for a business function or rule set to specify the control conditions for all the rules included in the business function or rule set.

Use the autoreplace feature to replace variable names in control conditions with the business names you've assigned to them in your project glossary. A business name identifies the business function of a variable in natural language, and is invaluable in documenting and retrieving business rules. The field PLC_NUM, for example, might have the business name Policy Number.

 **Tip:** Move a control condition up or down in the Control Conditions tab by selecting it and choosing **Move Up** or **Move Down** in the right-click menu.

Specifying Control Conditions

Follow the instructions below to specify control conditions for a rule.

1. Select a business function, rule set, or rule, then right-click in the Control Conditions tab and choose **Create Control Condition** in the pop-up menu. The Control Condition dialog opens.



Note: Alternatively, you can do this by selecting **Create Control Condition** from the context menu in the Rules pane or from the drop-down menu of the toolbar button **Change Rule Attributes**.

2. In the **Name** field, enter the name of the data item evaluated by the condition as it appears in the code.
3. In the **Operator** drop-down, choose the operator for the comparison that determines whether the condition is met.
4. In the **Argument** field, enter the constant value to which the data item is compared. The value may be the name of another field.
5. In the **Description** field, enter a description of the control condition.
6. Click **OK** to dismiss the dialog and return to the Rules window. The specified condition is listed in the Control Conditions tab in the righthand pane of the window.



Tip: To edit a field in the table, select the field and choose **Edit** in the right-click menu. The Control Condition dialog opens, where you can make the necessary changes. To delete a field, select it and choose **Delete** in the right-click menu.

7. Repeat this procedure for each control condition you want to add to the rule.

Autoreplacing Variable Names in Control Conditions with Business Names

To autoreplace variable names in control conditions with their business names in a project glossary, select a business function, rule set, or rule, then right-click in the Control Conditions tab and choose **Substitute Business Names** in the pop-up menu. You are prompted to continue. Click **Yes**.

When the autoreplace process is complete, a dialog box displays the number of autoreplaced variable names. Click **OK**.

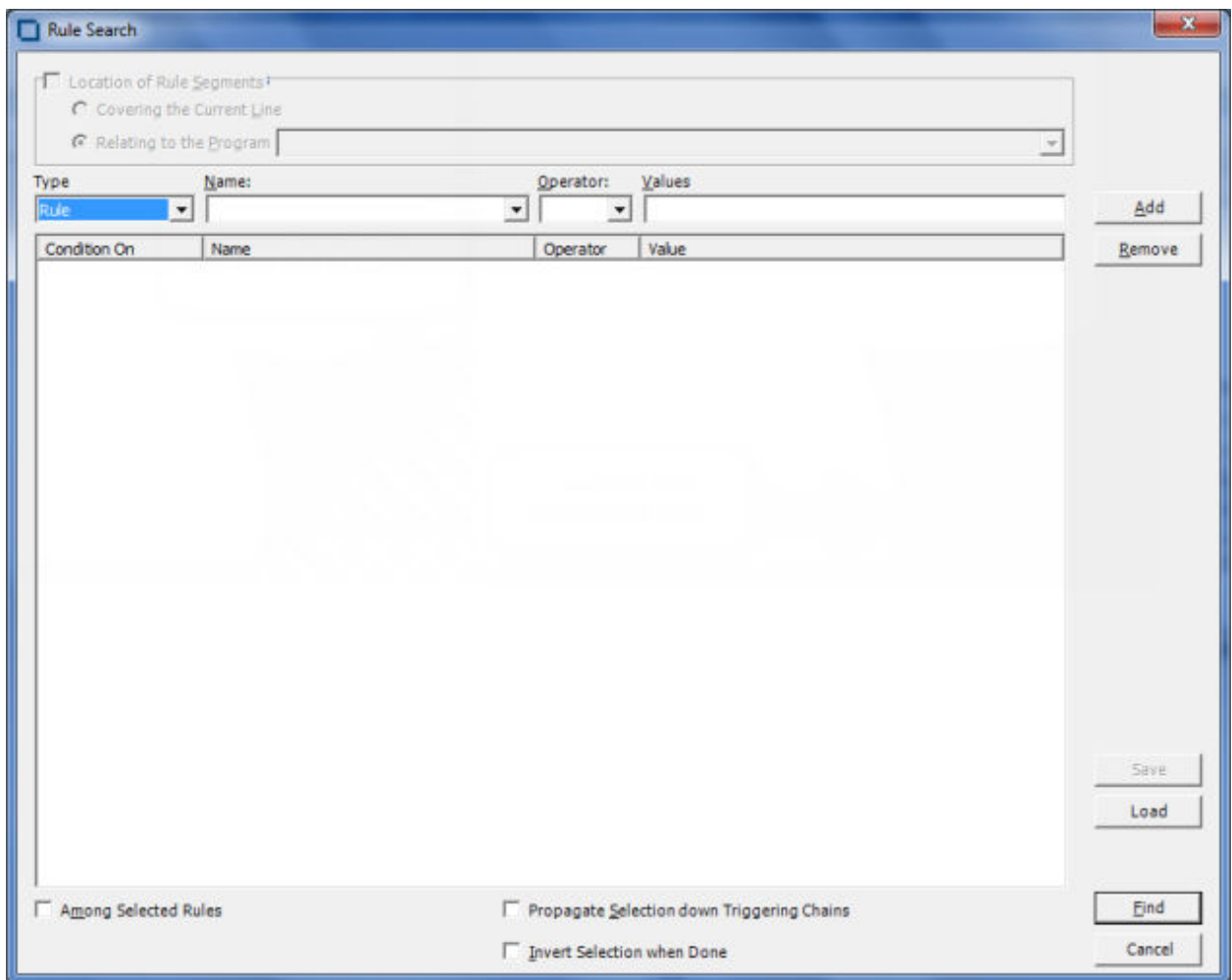
Searching for Rules

Use the search facility in the Rules pane to find rules based on segment location and/or attribute values. Use the Change Rules feature to batch edit the rules returned by the search.



Note: There is also a simple search option, under the Rules toolbar, that selects business functions, rules sets and rules that match the search string. It is case-sensitive and is available for the Grouping tab only, however results appear as checked in the Triggering tab as well.

1. Select a business function, rule set, or rule and choose **Rules > Start Search**. The Rule Search window opens.



2. To search for rules by segment location, select **Location of Rule Segments**, then choose:
 - **Covering the Current Line** if you want to find rules with a segment that contains the line of code selected in the Source pane.
 - **Relating to the Program** if you want to find rules related to a given program. Select the program in the adjacent drop-down.
3. To search for rules by attribute value, select:
 - The type of object that contains the attribute in the **Type** combo box.
 - The attribute in the **Name** drop-down.
 - The operator for the attribute value comparison in the **Operator** drop-down.
 - The value of the attribute in the **Value** drop-down.

Click **Add**. The search criterion is added to the list box below the drop-downs. Repeat this step for each attribute you want to search by. Select a criterion and click **Remove** to delete the criterion from the list box.
4. To search only the selected rules, select **Among Selected Rules**.
5. To find rules in the triggering hierarchy for the selection, select **Propagate Selection down Triggering Chains**.
6. To invert the search, so that rules not identified by the search criterion are found, choose **Invert Selection when Done**.
7. To save the search criteria you have chosen, click **Save**. This button is active only when the list of conditions is not empty. The criteria are saved to a file whose name contains the current date and time.

Example: SearchCriterion_2014-Jan-16_04-24-22.BRM.xml. By default it is saved in the workspace directory.

8. To load saved search criteria, click **Load**, select the file where the search criteria are saved and click **Open**.
9. When you are satisfied with your entries in the Rule Search window, click **Find**. The search facility automatically selects the check box for each rule returned by the search.

Batch Processing Rule Attributes

Use the Change Rules feature to batch process rule attributes. You can select batch-processing candidates manually, or use the search facility to select candidates automatically.



Note: Attributes of I/O data elements and control conditions can only be used to replace a Business Description or Technical Description. If there are multiple instances of I/O data elements or control conditions, they are displayed in the Business Description or Technical Description on separate lines.

1. Select the check box for each rule you want to batch process. If you are batch processing rules returned by a search, the check boxes are already selected.
2. Click **Rules > Change Rule Attributes**. You are prompted to continue. Click **Yes**. The Change Rules window opens.
3. In the **Attribute** drop-down, select the attribute you want to batch edit. In the **Find What** field, enter a pattern for the search text. In the **Replace With** field, enter a pattern for the replacement text. You can use any combination of text and bracketed attribute names. For attributes with enumerated values, click the arrow button next to the **Find What** and **Replace With** fields to display the values in a drop-down. For the remaining attributes, click the arrow button to display the available attributes in a drop-down. Select an item in a drop-down to populate the adjacent field.
4. When you are satisfied with your entries in the Change Rules window, click **Replace**. BRM modifies the selected rules. Click **Close** to dismiss the window and return to the Rules pane.

Usage Example

Suppose you wanted to use the Change Rules feature to rename two rules:

```
Rule 1 of program ABC
Rule 2 of program XYZ
```

as follows:

```
Rule 1 extracted from the ABC program
Rule 2 extracted from the XYZ program
```

1. Check Rule 1 of program ABC and Rule 2 of program XYX.
2. Choose **Rules > Change Rule Attributes**. You are prompted to continue. Click **Yes**. The Change Rules window opens.
3. In the **Select Attribute** field, choose the name attribute.
4. In the **Find What** field, enter:

```
of program [Program]
```
5. In the **Replace With** field, enter:

```
extracted from the [Program] program
```

Substituting a Business Name for an I/O Element in an Attribute Value

You can substitute a business name for an I/O element in an attribute value by appending "/bus" to the attribute name in either the **Find What** or **Replace With** fields in the Change Rules window.

Suppose the Catalog-Master variable has the business name Master Catalog. Now you want to replace the existing Business Description for the Rewrite Catalog-Master rule with the text "Rewrite Master Catalog."

You would select the Business Description attribute in the **Select Attribute** field and enter in the **Replace With** field:

[Name / bus]

Synchronizing Sources

Suppose a rule set in the current project is triggered by a rule in another project. When you *synchronize sources*, BRM includes the program that contains the segment for the triggering rule in the current project. To synchronize sources, choose **Rules > Sync Sources**.

Validating Rule Segments after Refreshing or Editing Code

Refreshing or editing source code may result in a mismatch between a rule and its segment. Lines of code may have been added or deleted during the refresh or edit, causing a rule to no longer to be synchronized with the segment after reverification.

BRM automatically handles invalidated segments based on your instructions in the Business Rule Manager > Automatic Rule Validation tab of the Project options window. Use the procedure below when you need to override the specified handling for a rule.

1. Select the rules you want to validate and choose **Rules > Validate**. You are prompted to continue. Click **Yes**. The Automatic Rule Validation dialog opens.
2. In the Actions for invalid segments pane, select:
 - **Leave segment, set Valid to false** if you want BRM to keep the invalid segment with the rule, but set the Segment Validity attribute of the rule to Invalid.
 - **Delete segment, set Valid to true** if you want BRM to delete the invalid segment from the rule and set the Segment Validity attribute of the rule to Valid.
 - **Try to match** if you want BRM to resynchronize the segment with the rule.
3. Place a check mark next to **Collapse spaces in strings** if you have added spaces to strings in segments and want BRM to ignore the spaces.
4. Click **OK**.

When the validation process is complete, a dialog box displays the number of validated rules. Click **OK**. The Segment Validity attribute for the rules is updated.

Limitations on Automatic Rule Validation

Automatic rule validation will fail to resynchronize a segment if the segment has been modified in any way during the refresh or edit (if a line has been inserted in the segment code, for example, or if the text of a line has been changed). You must re-synchronize these segments manually. Blank lines are OK.

Note, too, that if the code for a segment is duplicated in a program, the autovalidation method will synchronize the rule with the instance of the duplicated code nearest to its previous location, whether or not that instance was originally associated with the rule. The rule will be the same, but the program context may be different from the one you intended. Here, too, you will have to recreate the rule manually.

Setting Business Rule Manager Project Options

Project options for Business Rule Manager determine:

- The rule name templates available when you use Code Search to create rules in a specified rule set.
- How BRM handles invalidated rule segments.
- Whether BRM automatically includes triggered rule sets when it performs a rule export.

Setting Options on the Rule Defaults Tab

Use the **Business Rule Manager > Rule Defaults** tab of the Project options window to define the rule name templates available when you use Code Search to create rules in a specified rule set.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Business Rule Manager > Rule Defaults** tab.
2. The Rule Defaults tab displays the rule name templates available when you use Code Search to create rules in a specified rule set. Add templates as necessary.

Defining Rule Name Templates

Use the Business Rule Manager > Rule Defaults tab of the Project options window to define the rule name templates available when you use Code Search to create rules in a specified rule set.

You can use any combination of text and bracketed property names in a rule name template. So if you are creating rules for data ports, the entry:

```
File Write [Caption]
```

might produce the following rule names:

```
File Write REWRITE CATALOG-MASTER
File Write REWRITE FCOST-MASTER-REC
File Write WRITE DATASET('INVOICE')
```

Valid properties are any property of a construct, plus:

- Program: the name of the program that contains the construct.
- FileName: the name of the source file that contains the construct.
- FromRow: the number of the first line in the source that contains the construct.
- Caption: the label of the construct in the parse tree.

You can substitute a business name for an I/O element in a property value by appending "/bus" to the property name. If the CATALOG-MASTER variable has the business name MASTER CATALOG:

```
File Write [Caption/bus]
```

produces the output:

```
File Write REWRITE MASTER CATALOG
```

Setting Options on the Automatic Rule Validation Tab

Refreshing or editing source code may result in a mismatch between a rule and its segment. Lines of code may have been added or deleted during the refresh or edit, causing a rule to no longer be synchronized with the segment after reverification. BRM automatically handles invalidated segments based on your instructions in the **Business Rule Manager > Automatic Rule Validation** tab of the Project options window.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Business Rule Manager > Automatic Rule Validation** tab.
2. In the Actions for invalid segments pane, select:
 - **Leave segment, set Valid to false** if you want BRM to keep the invalid segment with the rule, but set the Segment Validity attribute of the rule to Invalid.
 - **Delete segment, set Valid to true** if you want BRM to delete the invalid segment from the rule and set the Segment Validity attribute of the rule to Valid.
 - **Try to match** if you want BRM to resynchronize the segment with the rule.
3. Place a check mark next to **Collapse spaces in strings** if you have added spaces to strings in segments and want BRM to ignore the spaces.

Setting Options on the Rule Export Tab

Use the **Business Rule Manager > Rule Export** tab of the Project options window to specify whether BRM automatically includes triggered rule sets when it performs a rule export.

1. Choose **Options > Project Options**. The Project Options window opens. Click the **Business Rule Manager > Rule Export** tab.
2. In the Close Dependencies pane, select:
 - **Only Selected** if you want BRM to export only the selected rules.
 - **Including Triggered Rule Set** if you want BRM to include triggered rule sets when it performs the rule export.

Setting Activity Log Options

Importing, exporting and deleting rules with enabled activity log options affects the performance and might take too long. This is the reason why these options are disabled by default. If you need to have an activity log report during import, export or deletion, go to **Options > Project Options > Business Rule Manager > Activity Log Options** and select the option that you need depending on the type of activity log record that you need:


- **Add a log record during export/import**
- **Add a detailed log record during delete**


Customizing Rule Attributes

Use the COBOL Analyzer Administration tool to modify characteristics and valid values of user-defined attributes, and to create custom attributes suited to your particular needs. Custom attributes are available to each workspace you create in the product.

The attribute definitions are stored in a file with a name of the form *.Repstry.xml. You can create multiple custom attribute files if you want, and switch back and forth between them as needed. COBOL Analyzer applies the most recently saved definitions to your business rules.

Choose **Rules > Extended Rule Attributes** to open a limited version of this tool in which you can modify or delete the values for character-type attributes. If you delete values in use in existing business functions, rule sets, or rules, the default value of the attribute is restored.

 **Note:** You must upgrade workspaces after customizing rule attributes. For upgrade instructions, see the installation manual for your product.

1. In the COBOL Analyzer Administration window, choose **Administer > Edit Rule Attributes**.
 - If you have never customized attributes, the Extended Rule Attributes window opens automatically with the default attributes.
 - If you have already customized attributes, and **Administer > Use Latest File** is not selected, an Open dialog appears where you can select the *.Repstry.xml file that contains the customized attributes you want to modify. The Extended Rule Attributes window opens.
 - If you have already customized attributes, and **Administer > Use Latest File** is selected, the Extended Rule Attributes window opens with the customized attributes for the *.Repstry.xml file you last saved in the window.
2. Click the Business Function, Rule Set, or Business Rule tab as appropriate.
3. To modify an existing attribute, select the attribute in the lefthand pane, then edit the characteristics or values you want to modify in the righthand pane. To remove an attribute, select it and click **Remove**.
4. To create a new attribute, click **Add**, then edit the characteristics of the new attribute as necessary. The new attribute is added to the list in the lefthand pane.
 -  **Tip:** To add a value in the Values pane, go to the end of any line containing a value, press Enter to create a new line, and type in the new value.
5. When you are satisfied with your entries, click **OK**. A Save As dialog opens, where you can specify the name and location of the *.Repstry.xml file. COBOL Analyzer applies the definitions to your business functions, rule sets, or rules.



Tip: To switch to the definitions in a different **.Repstry.xml file, deselect **Administer > Use Latest File**, open the file in the Edit Rule Attributes window, and click **OK** (whether or not you make changes to the file). Then save the file in the Save As dialog.

You must upgrade workspaces after customizing rule attributes. For upgrade instructions, see the installation manual for your product.






Generating Reports

BRM reports make it easy to develop the kind of detailed specification you'll need for a code extraction or redevelopment project. The following reports are available:

- The Business Rules Report lists the business function, rule set, segments, attributes, data elements, and control condition definitions of selected rules.
- The Process Outline Report shows the call context in which rules are executed for the selected program.
- The Coverage Report shows the percentage of program logic, including blank lines and comments, that contains rule segments. It applies to all rules.


Generating Business Rule Reports

Business Rule Reports list the business function, rule set, segments, attributes, data elements, and control condition definitions of selected rules. Options let you filter out rules from the report and control the attributes displayed in the report.

1. Select the business functions, rule sets, or rules you want to report on, click the  button on the tool bar, and choose **Business Rule Report (selected rules)** in the pull-down menu. The Configure Report Options window opens.
2. Select:
 - **Report rules with technical description** if you want only rules with a technical description to be included in the report.
 - **Report rules with business description** if you want only rules with a business description to be included in the report.
 - **Report all rules with all description types** if you want all rules to be included in the report.
3. Place a check mark next to **Print additional rule attributes** if you want to display other rule attributes in the report.
 - In the righthand pane, select each attribute you want to display in the report and click the  button to move the attribute into the lefthand pane. Attributes appear in the report in the order they are listed in the lefthand pane. Click the  button to move all the attributes into the lefthand pane.
 - In the lefthand pane, select an attribute and click the  button to move the attribute back into the right pane. Click the  button to move all the attributes into the righthand pane.
4. Click **OK**. The Business Rule Report window opens.


Generating Process Outline Reports

Process Outline Reports show the context in which rules are executed. The report lists every paragraph in the selected program, the call flow, or *process outline*, of the paragraph, every rule with a segment that starts in the paragraph, and the segment itself.

To generate a Process Outline Report, select the program you want to report on, click the  button on the tool bar, and choose **Process Outline Report (selected program)** in the pull-down menu. The Process Outline Report window opens.

Generating Coverage Reports

The Coverage Report shows the percentage of program logic (including blank lines and comments) that contains rule segments. The report lists the programs in the project, the total number of lines of code they contain, the number of lines of code with program logic, the number of business rule segments the program contains, and the percentage of program logic that contains business rule segments. It applies to all rules.





To generate a Coverage Report, click the  button on the tool bar and choose **Coverage Report (all rules)** in the pull-down menu. The Coverage Report window opens.

Using the Batch Duplicate Finder

Use the Batch Duplicate Finder (BDF) to perform sophisticated comparisons of similarly structured paragraphs. You can also use it to analyze paragraphs or procedures across workspaces, and to compare entire programs for their similarity.

Finding Duplicates

Follow the instructions below to find duplicate paragraphs with BDF. Make sure you verify the workspace files you plan to compare before running the tool.

1. If BDF is not already open, double-click the file `\<CA Home>\Bin\BDF.exe`. If BDF is already open, choose **Options > Startup Options**. In either case, the BDF scope window opens.
 2. In the BDF scope window, specify the scope of the comparison:
 - In the **Select Model** drop-down, select the object model for the workspace(s) to be analyzed.
 - In the **Select Unit** drop-down, select the unit of comparison: Paragraph/Procedure or Program.
 - Select **Compare Across Two Workspaces** if you want to perform the comparison across workspaces.
 - Select **Show Paragraph List** to view a list of paragraphs in the workspace(s) to be analyzed. You must choose this option if you want to select and mark the paragraphs to compare. Otherwise, all paragraphs are marked for comparison.
-  **Tip:** For extremely large workspaces, deselect **Show Paragraph List** to avoid the performance cost of loading paragraphs in the BDF main window.
3. Click **OK**. The BDF main window opens.
 4. Choose **File > Open Workspace**. The Open Workspace dialog opens. Select the workspace (.rwp) for the comparison, then click **Open**.
-  **Note:** If you are performing the comparison across workspaces, the Open Workspace dialog appears again. Select the second workspace and click **Open**.
5. The Select Projects window opens. Select the projects for the comparison, then click **OK**.
-  **Note:** If you are performing the comparison across workspaces, the Select Projects window appears again. Select the projects for comparison from the second workspace and click **Open**.
6. If you are performing the comparison in a single workspace, BDF loads the units of comparison in the List pane. If you are performing the comparison across workspaces, BDF loads the units of comparison for the first workspace at the top of the List pane with a value of A in the Workspace column, and for the second workspace at the bottom of the List pane with a value of B in the Workspace column.
-  **Tip:** Click an item to view its source in the Text pane. Select the appropriate choice in the **View** menu to show or hide a pane.
7. Set BDF search options.
 8. Select and mark the items you want to compare, then choose **File > Find Duplicates**.

BDF writes the results to the output file specified in the BDF search options. The output file renders the results in tab-delimited text. Use a spreadsheet editor to view and manipulate the results.

Marking Items for Comparison

For small projects, you can mark the items you want to compare manually. For larger projects, you can mark the items in batch mode, by defining a mask with the Select Paragraphs or Select Programs feature.

Marking Items Manually

Place a check mark next to an item to mark it. To mark all the items in a section, choose **List > Check All**. To unmark all the items in the section, choose **List > Uncheck All**.

To mark selected items in a section, select the items and choose **List > Check Selected**. To unmark selected items in the section, choose **List > Uncheck Selected**.



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

Marking Items in Batch Mode

To mark items in batch mode, choose **List > Select Paragraphs** or **List > Select Programs**. The BDF Define Mask window opens.

Enter the mask for the items you want to select in the **Paragraph name Like** field, then click **OK**. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA).

Deselect **Paragraph Action** or **Program Action** to select items that do not match the mask. To mark the items selected with the mask, choose **Check Selected** in the **List** menu.

Setting BDF Search Options

BDF search options control the types of comparison BDF performs, the columns it displays in the output, the name and location of the output file, and how it handles copybooks.

1. Choose **Options > Search Options**. The BDF Options window opens.
2. In the BDF Options window, choose **General Analysis** if you want to compare items on the basis of the minimum number of executable statements they contain. Enter the number in the **Minimum Paragraph Size** or **Minimum Program Size** field. An entry of 0 means that all items satisfy the test.
3. Choose the type of comparison you want to perform. If a pair of paragraphs or programs fails to satisfy the specified value for the comparison, the pair is excluded from the report. Select:
 - **Paragraph Analysis** or **Program Analysis** if you want to compare the statements in the pair for their minimum similarity on the Levenshtein edit distance metric. Select **Perform Exact Text Match** if you want to compare the entire text of the pair (including paragraph names, parameter names, comments, and blank lines) for exact matches.



Tip: Select **Compact Report** if you want the output to show items with 100% similarity in a numbered group, rather than with a line for each pair.

CUSTINQ1.ccp	1400-SEND-CUSTOMER-MAP	DCE3.CCP	1400-SEND-MENU
CUSTINQ1.ccp	1400-SEND-CUSTOMER-MAP	INVMENU.ccp	1400-SEND-MENU
DCE3.CCP	1400-SEND-MENU-MAP	INVMENU.ccp	1400-SEND-MENU

2 CUSTINQ1.ccp	1400-SEND-CUSTOMER-MAP
2 DCE3.CCP	1400-SEND-MENU-MAP
2 INVMENU.ccp	1400-SEND-MENU-MAP

- **Common Part Analysis** if you want to compare the statements in the pair for their minimum similarity on the longest common sequence (LCS) metric.

4. For either type of comparison, select Perform **Data Flow Analysis** if you want to perform an additional test to determine whether similar items perform data flow operations with the same pattern.
5. In the **Output File Name** field, enter the pathname of the BDF output file. In the **Directory for temporary files** field, enter the pathname of the directory for temporary files.
6. Specify copybook handling. Select:
 - **Exclude paragraphs in all copybook to copybook comparisons** if you want to prevent comparison of a paragraph in a copybook against any paragraph from any copybook, including the same copybook.
 - **Disable same copybook comparison** if you want to prevent comparison of a paragraph in a copybook against any paragraph in the same copybook.
7. In the Report Columns group box, select:
 - **Copybook** to include the Copybook column in the output.
 - **Coordinates** to include source code coordinates in the output.
 - **Number of statements** to include the number of statements in an item in the output.
 - **Number of common statements** to include the number of common statements in an item in the output.

Understanding Paragraph or Program Analysis

Select **Paragraph Analysis** or **Program Analysis** in the BDF search options if you want to compare items for their minimum similarity on the Levenshtein edit distance metric, namely, the number of operations needed to transform one item into the other, where an operation is an insertion, deletion, or substitution of a single character. The formula for this comparison type is:

$$\text{output} = 100 - (d/\text{Max}(a,b)) * 100$$

where d is the Levenshtein edit distance, a is the length of item A, and b is the length of item B.



Note: Only the statements themselves, not their parameters, are considered in performing the comparison.

Enter the minimum similarity in the **Minimum Similarity** field. A value of 100 means that only items containing exactly the same statements in exactly the same order satisfy the test. Enter 0 to disable the test.



Tip: You cannot use this comparison type to find paragraphs that CONTAIN or are INCLUDED in other paragraphs.

Understanding Common Part Analysis

Select **Common Part Analysis** in the BDF search options if you want to compare items for their minimum similarity on the longest common sequence (LCS) metric, namely, the number of characters in the longest sequence of characters the items have in common. The formula for this comparison type is:

$$\text{output} = (L/\text{Min}(a,b)) * 100$$

where L is the LCS, a is the length of item A, and b is the length of item B.



Note: Only the statements themselves, not their parameters, are considered in performing the comparison.

Enter the minimum similarity in the **Minimum Similarity** field. A value of 100 means that only items containing exactly the same statements in exactly the same order satisfy the test. Enter 0 to disable the test.



Tip: Use this comparison type to narrow the items of interest to those with common character sequences.

Understanding Data Flow Analysis

For either Paragraph/Program Analysis or Common Part Analysis, select **Perform Data Flow Analysis** in the BDF search options to perform an additional test to determine whether similar items perform data flow operations with the same pattern. In data pattern analysis, if item A contains:

```
MOVE A TO B  
MOVE B TO C
```

and item B contains:

```
MOVE X TO Y  
MOVE Y TO Z
```

their data pattern is regarded as identical. Whereas if item A contains:

```
MOVE A TO B  
MOVE B TO C
```

and item B contains:

```
MOVE X TO Y  
MOVE X TO Z
```

their data pattern is regarded as different.

Creating Components

Introducing Logic Analyzer

The Logic Analyzer tool offers a variety of advanced algorithms for *slicing* logic from program source: all the code you need for a computation, for example, or the code required to "specialize" a program based on the value of a variable. You can create a self-contained program, called a *component* from the sliced code or simply generate a list of sliced constructs for further analysis. You can mark and colorize the constructs in the Interactive Analysis Source pane.

Componentization Methods

The supported componentization methods slice logic not only from program executables but associated include files as well. Dead Code Elimination is an optimization tool built into the main methods and offered separately in case you want to use it on a standalone basis.



Note: Component Maker does not follow CALL statements into other programs to determine whether passed data items are actually modified by those programs. It makes the conservative assumption that all passed data items are modified. That guarantees that no dependencies are lost.

Structure-Based Componentization

Structure-Based Componentization lets you build a component from a range of inline code, COBOL paragraphs, for example. Use traditional structure-based componentization to generate a new component and its *complement*. A complement is a second component consisting of the original program minus the code extracted in the slice. Component Maker automatically places a call to the new component in the complement, passing it data items as necessary.

For COBOL programs, you can generate *parameterized slices*, in which the input and output variables required by the component are organized in group-level structures. These standard object-oriented data interfaces make it easier to deploy the transformed component in modern service-oriented architectures.



Tip: You typically repeat structure-based componentization in incremental fashion until all of the modules you are interested in have been created. For COBOL programs, you can avoid doing this

manually by specifying multiple ranges in the same extraction. Component Maker automatically processes each range in the appropriate order.

Computation-Based Componentization

Computation-Based Componentization lets you build a component that contains all the code necessary to calculate the value of a variable at a point in the program where it is used to populate a report attribute or screen. As with structure-based componentization, you can generate parameterized slices that make it easy to deploy the transformed component in distributed architectures.

For COBOL programs, you can use a technique called *blocking* to produce smaller, better-defined parameterized components. Component Maker will not include in the slice any part of the calculation that appears before the blocked statement. Fields from blocked input statements are treated as input parameters of the component.

Domain-Based Componentization

Domain-Based Componentization lets you "specialize" a program based on the values of one or more variables. The specialized program is typically intended for reuse "in place," in the original application, but under new external circumstances.

After a change in your business practices, for example, a program that invokes processing for a "payment type" variable could be specialized on the value PAYMENT-TYPE = "CHECK". Component Maker isolates every process dependent on the CHECK value to create a functionally complete program that processes check payments only.

Two modes of domain-based componentization are offered:

- In *simplified mode*, you set the specialization variable to its value anywhere in the program *except* a data port. The value of the variable is "frozen in memory." Operations that could change the value are ignored.
- In *advanced mode*, you set the specialization variable to its value at a data port. Subsequent operations can change the value, following the data and control flow of the program.

Use the simplified mode when you are interested only in the final value of a variable. Use the advanced mode when you need to account for data coming into a variable.

Dead Code Elimination (DCE)

Dead Code Elimination is an option in each of the main component extraction methods, but you can also perform it on a standalone basis. For each program analyzed for dead code, standalone DCE generates a component that consists of the original source code minus any unreferenced data items or unreachable procedural statements.



Note: Use the batch DCE feature to find dead code across your project. If you are licensed to use the Batch Refresh Process (BRP), you can use it to perform dead code elimination across a workspace.

Entry Point Isolation

Entry Point Isolation lets you build a component based on one of multiple entry points in a legacy program (an inner entry point in a Cobol program, for example). Component Maker extracts only the functionality and data definitions required for invocation from the selected point.

Entry Point Isolation is built into the main methods as an optional optimization tool. It's offered separately in case you want to use it on a standalone basis.

Componentization Outputs

The first step in the componentization process, called *extraction*, generates the following outputs:

- The source file that comprises the component.

- An abstract repository object, or *logical component*, that gives you access to the source file in COBOL Analyzer.
- A Interactive Analysis list of sliced constructs, which you can mark and colorize in the Interactive Analysis Source pane.



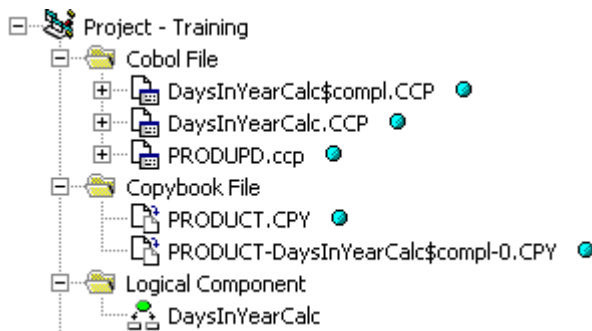
Note: Sliced data declarations are not marked and colorized.

The second step, called *conversion*, registers the source files in your repository, creating repository objects for the generated components and their corresponding copybooks.

Component Maker lets you execute the extraction and conversion steps independently or in combination, depending on your needs:

- If you want to analyze the components further, transform them, or even generate components from them, you will want to register the component source files in your repository and verify them, just as you would register and verify a source file from the original legacy application.
- If you are interested only in deploying the components in your production environment, you can skip the conversion step and avoid cluttering your repository.

The figure below shows how the componentization outputs are represented in the Repository Browser after conversion and verification of a COBOL component called DaysInYearCalc. PRODUPD is the program the component was extracted from.



Component Maker Basics

Component Maker is a Interactive Analysis-based tool that you can invoke from within Interactive Analysis itself:

- Start the tool in Interactive Analysis by selecting the program you want to slice in the COBOL Analyzer Repository Browser and choosing **Analyze > Interactive Analysis**. In the Interactive Analysis window, choose **View > Components**.




Note: Choose **View > Logic Analyzer** if you are using Logic Analyzer.

The Components pane consists of a hierarchy of views that let you specify the logical components you want to manipulate:



- The Types view lists the types of logical components you can create.
- The List view displays logical components of the selected type.
- The Details view displays the details for the selected logical component in two tabs, Properties and Components. The Properties tab displays extraction properties for the logical component. The Components tab lists the files generated for the logical component.



Getting Started in the Components Pane


You do most of your work in Component Maker in the Components pane. To illustrate how you extract a logical component in the Components pane, let's look at the simplest task you can perform in Component Maker, Dead Code Elimination (DCE).



 **Note:** The following exercise deliberately avoids describing the properties and options you can set for DCE. See the relevant help topics for details.

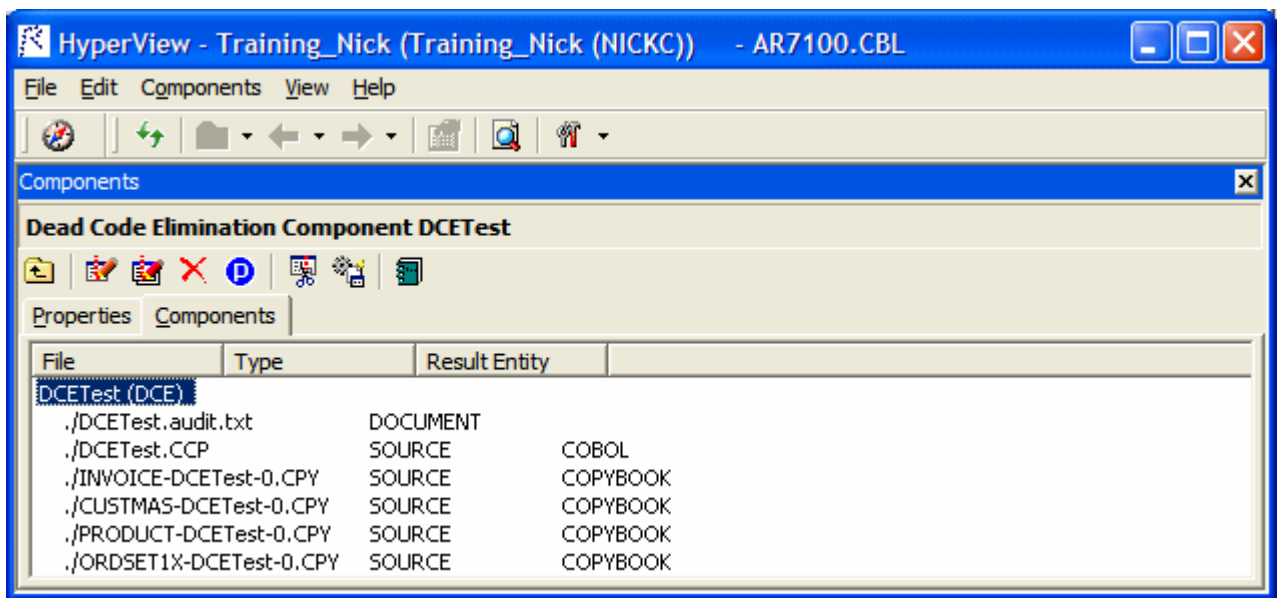
1. In the Components pane, double-click **Dead Code Elimination**. The DCE pane opens. This view shows the DCE-based logical components created for the programs in the current project.

 **Tip:** Click the  button on the tool bar to restrict the display to logical components created for the selected program.



2. Select the program you want to analyze for dead code in the Interactive Analysis Objects pane and click the  button. To analyze the entire project of which the program is a part, click the  button.
3. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new components to the list of components. If you selected batch mode, Component Maker creates a logical component for each program in the project, appending `_n` to the name of the component.
4. Double-click a component to edit its properties. The **Component of program** field contains the name of the selected program.
5. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.

 **Note:** This field is shown only for COBOL programs.



6. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
7. Click the  button on the tool bar to navigate to the list of components, then repeat the procedure for each component you want to extract.
8. In the list of components, select each component you want to extract and click the  button on the tool bar. You are prompted to confirm that you want to extract the components. Click **OK**.
9. The Extraction Options dialog opens. Set extraction options as described in the relevant help topic. When you are satisfied with your choices, click **Finish**.
10. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** to view the errors or warnings in the Activity Log. Otherwise, click **No**.
11. Assuming the extraction executed without errors, the view shown in the figure below opens. Click the Components tab to display a list of the component source files that were generated for the logical component and an audit report if you requested one. Click an item in the list to view the read-only text for the item.




Creating Components



To create a component, select the program you want to slice in the Interactive Analysis Objects pane. In the Types view, select the type of logical component you want to create and click the  button on the tool bar. (You can also click the  button in the List or Details view.) A dialog opens where you can enter the name of the new component in the text field. Click **OK**.

Extracting Components

To extract a single logical component, select the component you want to extract in the List view and click the  button on the tool bar. To extract multiple logical components, select the type of the components you want to extract in the Types view and click the  button. You are prompted to confirm that you want to continue. Click **OK**.


 **Tip:** Logical components are converted as well as extracted if the **Convert Resulting Components to Legacy Objects** is set in the Component Conversion Options pane.

Converting Components

To convert a single logical component, select the component you want to convert in the List view and click the  button on the tool bar. To convert multiple logical components, select the type of the components you want to convert in the Types view and click the  button. You are prompted to confirm that you want to continue. Click **OK**.

Deleting Components

To delete a logical component, select it in the List view and click the  button on the tool bar.

 **Note:** Deleting a logical component does not delete the component and copybook repository objects. You must delete these objects manually in the Repository Browser.


Viewing the Text for Generated Files

To view the read-only text for a generated file, click the file in the list of generated files for in the Components tab.







Tip: You can also view the text for a generated file in the COBOL Analyzer main window. In the Repository Browser Logical Component folder, click the component whose generated files you want to view.

Restricting the Display to Program-Related Components

To restrict the display to logical components of a given program, select the program and click the  button on the tool bar. The button is a toggle. Click it again to revert to the generic display.

Working with Interactive Analysis Lists

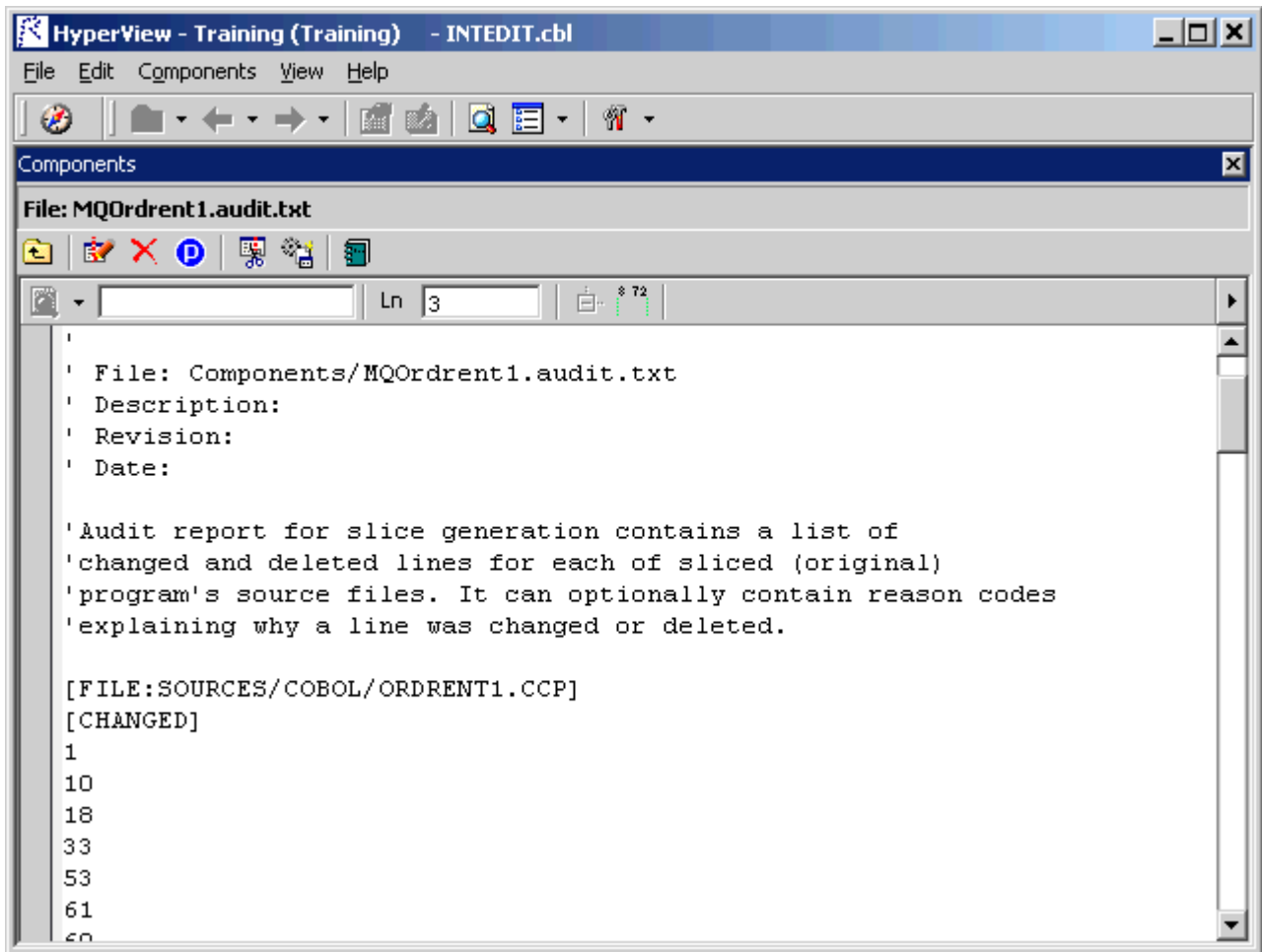
When you extract a logical component, Component Maker generates a Interactive Analysis list of sliced constructs. The list has the same name as the component. You can view the list in the Logic Analyzer folder in Code Search.

To mark and colorize sliced constructs in the list, select the list in Code Search and click the  button on the tool bar. To mark and colorize sliced constructs in a single file, select the file in the List view and click the  button. To mark and colorize a single construct, select it in the File view and click the  button. Click the  button again to turn off marking and colorizing.

Viewing Audit Reports

An *audit report* contains a list of changed and deleted lines in the source files (including copybooks) from which a logical component was extracted. The report has a name of the form `<component>.audit.txt`. Click the report in the Components tab to view its text.


An audit report optionally includes reason codes explaining why a line was changed or deleted. A reason code is a number keyed to the explanation for a change (for example, reason code 12 for computation-based componentization is RemoveUnusedVALUES).



Generating Coverage Reports

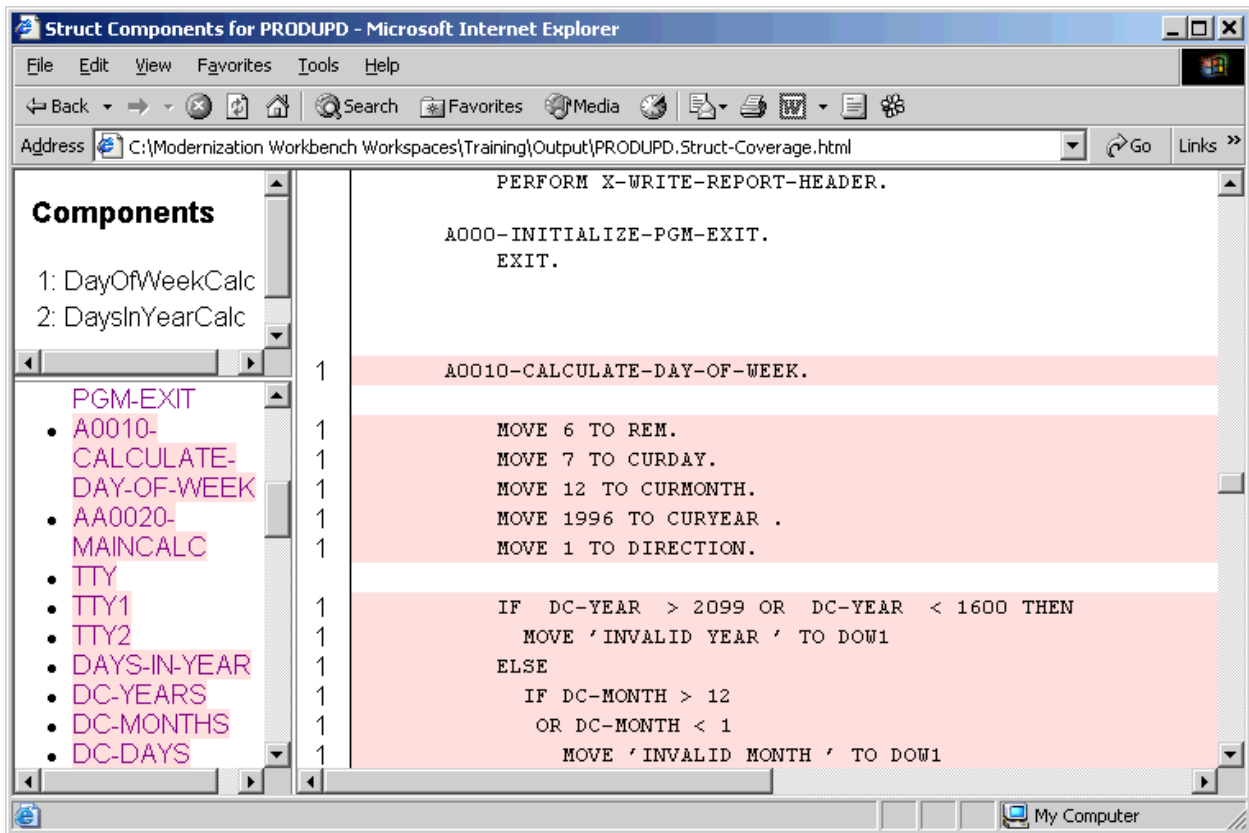
A *coverage report* shows the extent to which a source program has been "componentized":

- The top-left pane lists each component of a given type extracted from the program.
- The bottom-left pane lists the paragraphs in the program. Click on a paragraph to navigate to it in the righthand pane.
- The righthand pane displays the text of the program with extracted code shaded in pink. The numbers to the left of the extracted code identify the component to which it was extracted.

To generate coverage reports, click  on the Component Maker tool bar. The reports are listed in the Generated Document folder in the Repository Browser. Report names are of the form *<program>-<method>-Coverage*. Double-click a report to view it in a Web browser.



Note: Reports are created for each program in the current project.



Setting Component Maker Options


It's a good idea to become familiar with the component extraction options before beginning your work in Component Maker. Each extraction method has a different set of options, and each set differs for the supported object types. Extraction options are project-based, so they apply to every program in the current COBOL Analyzer project.

You can set Component Maker extraction options in the standard Project Options window or in the extraction options dialog that opens when you create a component. To open the standard Project Options window, choose **Options > Project Options**. In the Project Options window, click the **Component Maker** tab.

Setting General Options


The table below describes the Component Maker General extraction options.

Option	Language	Description
Add Program Name as Prefix	COBOL, Natural, PL/I, RPG	Prepend the name of the sliced program to the component name you specified when you created the component, in the form <i><program> \$<component></i> .
Generate Slice	COBOL, Natural, PL/I, RPG	Generate both a Interactive Analysis list of sliced constructs and a component.
Keep Legacy Copybooks	COBOL, RPG	Do not generate modified copybooks for the component. Modified copybooks have names of the form <i><copybook>-<component>-n</i> , where <i>n</i> is a number ensuring the uniqueness of the copybook

Option	Language	Description
		name when multiple instances of a copybook are generated for the same component.  Note: Component Maker issues a warning if including the original copybooks in the component would result in an error.
Keep Legacy Includes	PL/I	Do not generate modified program include files for the component. The layout and commentary of the sliced program is preserved.
Keep Legacy Macros	PL/I	Do not expand macros for the component. The layout and commentary of the sliced program is preserved.
Preserve Legacy Includes	Natural	Do not generate modified program include files for the component.
Rename Program Entries	COBOL	Prepend the name of the component to inner entry points, in the form <code><component>-<entrypoint></code> . This ensures that entry point names are unique and that the COBOL Analyzer parser can verify the component successfully. Unset this option if you need to preserve the original names of the inner entry points.


Setting Interface Options


The table below describes the Component Maker Interface extraction options.

Option	Language	Description
Blocking	COBOL	If you are performing a parameterized computation-based extraction and want to use blocking, click the More button. A dialog opens, where you can select the blocking option and the types of statements you want to block.  Note: Choose Use Blocking from Component Definitions if you want to block statements in a Interactive Analysis list.
Create CICS Program	COBOL	Create COMMAREAS for parameter exchange in generated slices.
Generate Parameterized Components	COBOL	Extract parameterized slices.

Setting Optimize Options

The table below describes the Component Maker Optimize extraction options.


Option	Language	Description
No changes	Cobol, Natural, RPG	Do not remove unused data items from the component.
Preserve Original Paragraphs	Cobol	Generate paragraph labels even for paragraphs that are not actually used in the source code (for example, empty paragraphs for which there are no PERFORMs).  Note: This option also affects refactoring. When the option is set, paragraphs in the

Option	Language	Description
		same "basic block" are defragmented separately. Otherwise, they are defragmented as a unit.
Remove Redundant NEXT SENTENCE	Cobol	<p>Remove NEXT SENTENCE clauses by changing the bodies of corresponding IF statements, such that:</p> <pre>IF A=1 NEXT SENTENCE ELSE . . . END-IF.</pre> <p>is generated as:</p> <pre>IF NOT (A=1) . . . END-IF.</pre>
Remove/Replace Unused Fields with FILLERS	Cobol, Natural, RPG	<p>Remove unused any-level structures and replace unused fields in a used structure with FILLERS. Set this option if removing a field completely from a structure would adversely affect memory distribution.</p> <p> Note: If you select Keep Legacy copybooks in the General component extraction options, Component Maker removes or replaces with FILLERS only unused inline data items.</p>
Remove Unreachable Code	Cobol, RPG	Remove unreachable procedural statements.
Remove Unused Any-Level Structures	Cobol, Natural, RPG	<p>Remove unused structures at any data level, if all their parents and children are unused. For the example below, D, E, F, and G are removed:</p> <pre>DEFINE DATA LOCAL 1 #A 2 #B 3 #C 2 #D 3 #E 3 #F 1 #G</pre>
Remove Unused Level-1 Structures	Cobol, Natural, RPG	<p>Remove only unused level-1 structures, and then only if all their children are unused. If, in the following example, only B is used, only G is removed:</p> <pre>DEFINE DATA LOCAL 1 #A 2 #B 3 #C 2 #D 3 #E 3 #F 1 #G</pre>
Replace Section PERFORMs by Paragraph PERFORMs	Cobol	Replace PERFORM section statements by equivalent PERFORM paragraph statements.

Option	Language	Description
Roll-Up Nested IFs	Cobol	Roll up embedded IF statements in the top-level IF statement, such that: <pre>IF A=1 IF B=2</pre> is generated as: <pre>IF (A=1) AND (B=2)</pre>

Setting Document Options

The table below describes the Component Maker Document extraction options.

Option	Language	Description
Comment-out Sliced-off Legacy Code	COBOL, RPG	Retain but comment out unused code in the component source. In the Comment Prefix field, enter descriptive text (up to six characters) for the commented-out lines.
Emphasize Component/Include in Coverage Report	COBOL, Natural, PL/I, RPG	Generate a Interactive Analysis list of sliced constructs and colorize the constructs in the Coverage Report.
Generate Audit Report	COBOL	Generate an audit report.
Generate Support Comments	COBOL, RPG	Include comments in the component source that identify the component properties you specified, such as the starting and ending paragraphs for a structure-based COBOL component.
Include Reason Codes	COBOL	Include reason codes in the audit report explaining why a line was changed or deleted.  Note: Generating reason codes is very memory-intensive and may cause crashes for extractions from large programs.
List Options in Component Header and in Separate Document	COBOL, RPG	Include a list of extraction option settings in the component header and in a separate text file. The text file has a name of the form <code><component>.BRE.options.txt</code> .
Mark Modified Legacy Code	COBOL, RPG	Mark modified code in the component source. In the Comment Prefix field, enter descriptive text (up to six characters) for the modified lines.
Print Calculated Values as Comments	COBOL	For domain-based component extraction only, print the calculated values of variables as comments. Alternatively, you can substitute the calculated values of variables for the variables themselves.
Use Left Column for Marks	COBOL, RPG	Place the descriptive text for commented-out or modified lines in the lefthand column of the line. Otherwise, the text appears in the righthand column.

Setting Component Type-Specific Options

Component type-specific extraction options determine how Component Maker performs tasks specific to each componentization method.


Setting Structure-Based Type-Specific Options

The table below describes the Component Maker structure-based type-specific extraction options.

Option	Language	Description
Dynamic Call	COBOL	Generate in the complement a dynamic call to the component. The complement will call a string variable that must later be set outside the complement to the name of the component.
Ensure Consistent Access to External Resources	COBOL	Monitor the integrity of data flow in the ranges you are extracting. If you select this option, for example, an extraction will fail if an SQL cursor used in the component is open in the complement.
Range Only	COBOL	Do not generate a complement. You must set this option to generate parameterized slices.
Restrict User Ranges to PERFORMed Ones	COBOL	Do not extract paragraphs that do not have a corresponding PERFORM statement. This option is useful if you want to limit components created with the Paragraph Pair or Section methods to PERFORMed paragraphs.
Suppress Errors	COBOL	Perform a "relaxed extraction," in which errors that would ordinarily cause the extraction to fail are ignored, and comments describing the errors are added to the component source. This option is useful when you want to review extraction errors in component source.

Setting Computation-Based Type-Specific Options



The table below describes the Component Maker computation-based type-specific extraction options.

Option	Language	Description
Generate HTML Trace	COBOL	Generate an HTML file with an extraction trace. The trace has a name of the form <i><component>.trace</i> . To view the trace, click the logical component for the extraction in the Repository Browser Logical Component folder. Double-click the trace file to view it in a Web browser.
Statement	COBOL	Perform statement-based component extraction.
Variable	COBOL	Perform variable-based component extraction.  Note: Even if you select variable-based extraction, Component Maker performs statement-based extraction if the variable you slice on is not an input variable for its parent statement: that is, if the statement writes to rather than reads from the variable.

Setting Domain-Based Type-Specific Options

The table below describes the Component Maker domain-based type-specific extraction options.


Option	Language	Description
Maximum Number of Variable's Values	COBOL	The maximum number of values to be calculated for each variable. Limit is 200. The lower the

Option	Language	Description
		maximum, the better performance and memory usage you can expect.
Maximum Size of Variable to Be Calculated	COBOL	Maximum size in bytes for each variable value to be calculated. The lower the maximum, the better performance and memory usage you can expect.
Multiple Pass	COBOL	Evaluate conditional logic again after detecting dead branches. Because the ELSE branch of the first IF below is dead, for example, the second IF statement can be resolved in a subsequent pass: <pre>MOVE 0 TO X. IF X EQUAL 0 THEN MOVE 1 TO Y ELSE /p> MOVE 2 TO Y. IF Y EQUAL 2 THEN... ELSE...</pre> <p> Note: Multi-pass processing is very resource-intensive, and not recommended for extractions from large programs.</p>
Remove Unused Assignments	COBOL	Exclude from the component assignments that cannot affect the computation (typically, an assignment after which the variable is not used until the next assignment or port).
Replace Variables by Their Calculated Values	COBOL	Substitute the calculated values of variables for the variables themselves. Alternatively, you can print the values as comments. <p> Note: Notice how the options in Remove Unused Assignments and Replace Variables by Their Calculated Values can interact. If both options are set, then the first assignment in the following fragment will be removed:</p> <pre>MOVE 1 TO X. DISPLAY X. MOVE 2 TO X.</pre>
Single Pass	COBOL	Evaluate conditional logic in one pass.
VALUEs Initialize Data Items	COBOL	Set variables declared with VALUE clauses to their initial values. Otherwise, VALUE clauses are ignored.

Setting Component Conversion Options

The table below describes the Component Maker Component Conversion extraction options.

Option	Language	Description
Convert Resulting Components	COBOL	Convert as well as extract the logical component.
Keep Old Legacy Objects	COBOL	Preserve existing repository objects for the converted component (copybooks, for example). If you select this option, delete the repository object for the component itself before performing the extraction, or the new component object will not be created.

Option	Language	Description
Remove Components after Successful Conversion	COBOL	Remove logical components from the current project after new component objects are created.
Replace Old Legacy Objects	COBOL	Replace existing repository objects for the converted component.  Note: This option controls conversion behavior even when you perform the conversion independently from the extraction. If you are converting a component independently and want to change this setting, select Convert Resulting Components to Legacy Objects, specify the behavior you want, and then deselect Convert Resulting Components to Legacy Objects.

Extracting Structure-Based Components

Structure-Based Componentization lets you build a component from a range of inline code, COBOL paragraphs, for example. Use traditional structure-based componentization to generate a new component and its *complement*. A complement is a second component consisting of the original program minus the code extracted in the slice. Component Maker automatically places a call to the new component in the complement, passing it data items as necessary.

Alternatively, you can generate *parameterized slices*, in which the input and output variables required by the component are organized in group-level structures. These standard object-oriented data interfaces make it easy to deploy the transformed component in modern service-oriented architectures.

Understanding Ranges

When you extract a structure-based component from a program, you specify the *range* of code you want to include in the component. The range varies: for COBOL programs, a range of paragraphs; for PL/I programs, a procedure; for RPG programs, a subroutine or procedure.



Tip: You typically repeat Structure-Based Componentization in incremental fashion until all the modules you are interested in have been created. For COBOL programs, you can avoid doing this manually by specifying multiple ranges in the same extraction. Component Maker automatically processes each range in the appropriate order. No complements are generated.

Specifying Ranges for COBOL Programs

For COBOL programs, you specify the paragraphs in the range for structure-based component extraction in one of three ways:

- Select a paragraph PERFORM statement to set the range to the performed paragraph or paragraphs. Component Maker includes each paragraph in the execution path between the first and last paragraphs in the range, except when control is transferred by a PERFORM statement or by an implicit RETURN-from-PERFORM statement.
- Select a pair of paragraphs to set the range to the selected paragraphs. You are responsible for ensuring a continuous flow of control from the first to the last paragraph in the range.
- Select a section to set the range to the paragraphs in the section.



Note: For traditional structure-based COBOL components, Component Maker inserts in the complement the labels of the first and last paragraphs in the range. The first paragraph is replaced in the complement with a CALL statement followed by a GO TO statement. The last paragraph is always empty.

The GO TO statement transfers control to the last paragraph. If the GO TO statement and its target paragraph are not required to ensure correct call flow, they are omitted.

Specifying Ranges for PL/I Programs

For PL/I programs, the range you specify for structure-based component extraction is an internal procedure that Component Maker extracts as an external procedure. The slice contains the required parameters for global variables.

Specifying Ranges for RPG Programs

For RPG programs, the range you specify for structure-based component extraction is a subroutine or procedure to extract as a component.

Understanding Parameterized Slices

For COBOL programs, you can generate *parameterized slices*, in which the input and output variables required by the component are organized in group-level structures. The component contains all the code required for input/output operations.

To extract a parameterized slice, select the **Generate Parameterized Components** option in the extraction options dialog. Note that you cannot generate a complement for a parameterized COBOL slice.



Note: For parameterized structure- and computation-based componentization of COBOL programs, you must select the **Perform Program Analysis** and **Enable Parameterization of Components** options in the project verification options.

COBOL Naming Conventions

- Component input structures have names of the form BRE-INP-<STRUCT-NAME>. Input fields have names of the form BRE-I-<FIELD-NAME>.
- Component Output structures have names of the form BRE-OUT-STRUCT-NAME. Output fields have names of the form BRE-O-<FIELD-NAME>.

Parameterization Example

The example below illustrates how Component Maker generates parameterized slices. Consider a COBOL program that contains the following structures:

```
WORKING-STORAGE SECTION.  
  01 A  
    03 A1  
    03 A2  
  01 B  
    03 B1  
    03 B2  
    03 B4
```

Suppose that only A1 has been determined by Component Maker to be an input parameter, and only B1 and B2 to be output parameters. Suppose further that the component is extracted with input and output data structures that use the default names, BRE-INP-INPUT-STRUCTURE and BRE-OUT-OUTPUT-STRUCTURE, respectively, and with the default Optimization options set. The component contains the following code:

```
WORKING-STORAGE SECTION.  
  01 A  
    03 A1  
    03 A2  
  01 B  
    03 B1  
    03 B2  
    03 B4
```


```


LINKAGE SECTION.
  01 BRE-INP-INPUT-STRUCTURE
    03 BRE-I-A
      06 BRE-I-A1
01 BRE-OUT-OUTPUT-STRUCTURE
  03 BRE-O-B
    06 BRE-O-B1
    06 BRE-O-B2
PROCEDURE DIVISION
  USING BRE-INP-INPUT-STRUCTURE BRE-OUT-OUTPUT-STRUCTURE .
BRE-INIT-SECTION SECTION.
  PERFORM BRE-COPY-INPUT-DATA .
  .....
  ....(Business Logic)....
  .....
*COBOL Analyzer added statement
  GO TO BRE-EXIT-PROGRAM.
BRE-EXIT-PROGRAM-SECTION SECTION.
  BRE-EXIT-PROGRAM.
  PERFORM BRE-COPY-OUTPUT-DATA .
  GOBACK.
BRE-COPY-INPUT-DATA .
  MOVE BRE-I-A TO A.
BRE-COPY-OUTPUT-DATA .
  MOVE B TO BRE-O-B.

```


Extracting Structure-Based COBOL Components

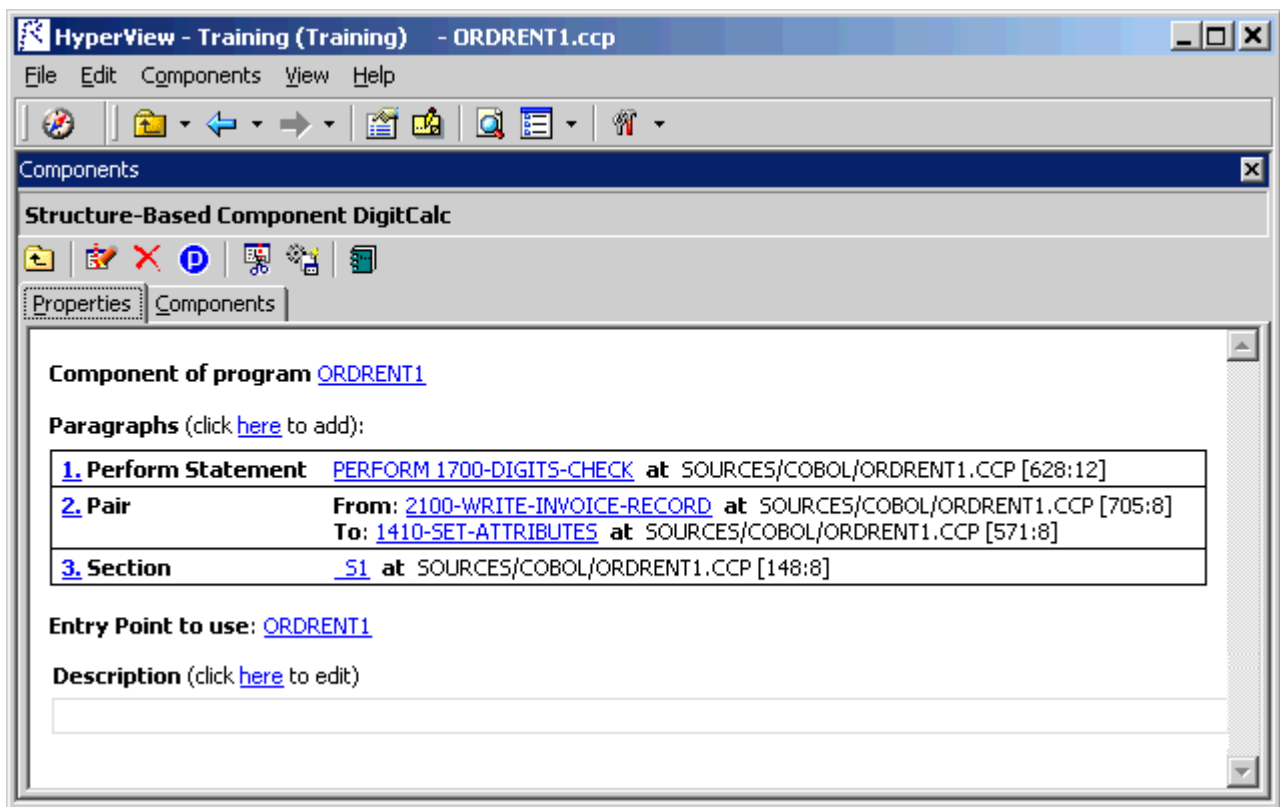
Follow the instructions below to extract structure-based COBOL components.


1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. In the **Paragraphs** field, click the **here** link. Choose one of the following methods in the pop-up menu:
 - **Paragraph Perform** to set the range to the paragraph or paragraphs performed by the selected PERFORM statement. Select the PERFORM statement in the Source pane, then click the link for the current selection and choose **Set** in the pop-up menu.
 - **Pair of Paragraphs** to set the range to the selected paragraphs. Select the first paragraph in the pair in the Source pane, then click the link for the current selection in the **From** field and choose **Set** in the drop-down menu. Select the second paragraph in the pair, then click the link for the current selection in the **To** field and choose **Set** in the pop-up menu.

 **Tip:** You can set the **From** and **To** fields to the same paragraph.

 - **Section** to set the range to the paragraphs in the section. Select the section in the Source pane, then click the link for the current selection and choose **Set** in the pop-up menu.


 **Note:** To delete a range, select the link for the numeral that identifies the range and choose **Delete** in the pop-up menu. To unset a PERFORM, paragraph, or section, click it and choose **Unset** in the pop-up menu. To navigate quickly to a PERFORM, paragraph, or section in the source, click it and choose **Locate** in the pop-up menu.
3. Repeat this procedure for each range you want to extract. You can use any combination of methods. The figure below shows how the properties tab might look for a multi-range extraction.





4. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.
5. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
6. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
7. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
8. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Extracting Structure-Based PL/I Components

Follow the instructions below to extract structure-based PL/I components.


1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. Select the program entry point in the Source pane. In the **Point** field, click the link for the current selection and choose **Set** in the pop-up menu.



 **Note:** To unset an entry point, click it and choose **Unset** in the pop-up menu. To navigate quickly to an entry point in the source, click it and choose **Locate** in the pop-up menu.
3. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.

4. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
5. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
6. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Extracting Structure-Based RPG Components

Follow the instructions below to extract structure-based RPG components.

1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. Select the subroutine or procedure you want to slice in the Source pane. In the **Point** field, click the link for the current selection and choose **Set** in the pop-up menu.

 **Note:** To unset an entry point, click it and choose **Unset** in the pop-up menu. To navigate quickly to an entry point in the source, click it and choose **Locate** in the pop-up menu.
3. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
4. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
5. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
6. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Extracting Computation-Based Components

Computation-Based Componentization lets you build a component that contains all the code necessary to calculate the value of a variable at a point in the program where it is used to populate a report attribute or screen. You can generate parameterized computation-based slices that make it easy to deploy the transformed component in distributed architectures.

Understanding Variable-Based Extraction

When you perform a computation-based extraction, you can slice by statement or by variable. What's the difference? Suppose you are interested in calculations involving the variable X in the example below:

```
MOVE 1 TO X
MOVE 1 TO Y
DISPLAY X Y.
```

If you perform statement-based extraction (if you slice on the statement `DISPLAY X Y`), all three statements will be included in the component. If you perform variable-based extraction (if you slice on the variable X), only the first and third statements will be included. In variable-based extraction, that is, Component Maker tracks the dependency between X and Y, and having determined that the variables are independent, excludes the `MOVE 1 to Y` statement.



Note: If you slice on a variable for a COBOL component, you must select **Variable** in the Component Type Specific options for computation-based extraction.

Understanding Blocking

For COBOL programs, you can use a technique called *blocking* to produce smaller, better-defined parameterized components. Component Maker will not include in the slice any part of the calculation that appears before the blocked statement. Fields from blocked input statements are treated as input parameters of the component.

Consider the following fragment:

```
INP1 .
  DISPLAY "INPUT YEAR (1600-2099)" .
  ACCEPT YEAR .
  CALL 'PROG' USING YEAR .
  IF YEAR > 2099 OR YEAR < 1600 THEN
    DISPLAY "WRONG YEAR" .
```

If the CALL statement is selected as a block, then both the CALL and ACCEPT statements from the fragment are not included in the component, and YEAR is passed as a parameter to the component.



Tip: Specify blocking in the blocking dialog accessed from the Interface options pane.

Understanding Parameterized Slices

For COBOL programs, you can generate *parameterized slices*, in which the input and output variables required by the component are organized in group-level structures. The component contains all the code required for input/output operations.

To extract a parameterized slice, select the **Generate Parameterized Components** option in the extraction options dialog. Note that you cannot generate a complement for a parameterized COBOL slice.



Note: For parameterized structure- and computation-based componentization of COBOL programs, you must select the **Perform Program Analysis** and **Enable Parameterization of Components** options in the project verification options.

COBOL Naming Conventions

- Component input structures have names of the form BRE-INP-<STRUCT-NAME>. Input fields have names of the form BRE-I-<FIELD-NAME>.
- Component Output structures have names of the form BRE-OUT-STRUCT-NAME. Output fields have names of the form BRE-O-<FIELD-NAME>.

Parameterization Example

The example below illustrates how Component Maker generates parameterized slices. Consider a COBOL program that contains the following structures:

```
WORKING-STORAGE SECTION.
  01 A
    03 A1
    03 A2
  01 B
    03 B1
    03 B2
    03 B4
```


Suppose that only A1 has been determined by Component Maker to be an input parameter, and only B1 and B2 to be output parameters. Suppose further that the component is extracted with input and output data structures that use the default names, BRE-INP-INPUT-STRUCTURE and BRE-OUT-OUTPUT-


STRUCTURE, respectively, and with the default Optimization options set. The component contains the following code:

```
WORKING-STORAGE SECTION.  
  01 A  
    03 A1  
    03 A2  
  01 B  
    03 B1  
    03 B2  
    03 B4  
LINKAGE SECTION.  
  01 BRE-INP-INPUT-STRUCTURE  
    03 BRE-I-A  
      06 BRE-I-A1  
  01 BRE-OUT-OUTPUT-STRUCTURE  
    03 BRE-O-B  
      06 BRE-O-B1  
      06 BRE-O-B2  
PROCEDURE DIVISION  
  USING BRE-INP-INPUT-STRUCTURE BRE-OUT-OUTPUT-STRUCTURE.  
BRE-INIT-SECTION SECTION.  
  PERFORM BRE-COPY-INPUT-DATA.  
  .....  
  ....(Business Logic)....  
  .....  
  *COBOL Analyzer added statement  
  GO TO BRE-EXIT-PROGRAM.  
BRE-EXIT-PROGRAM-SECTION SECTION.  
  BRE-EXIT-PROGRAM.  
  PERFORM BRE-COPY-OUTPUT-DATA.  
  GOBACK.  
BRE-COPY-INPUT-DATA.  
  MOVE BRE-I-A TO A.  
BRE-COPY-OUTPUT-DATA.  
  MOVE B TO BRE-O-B.
```

Extracting Computation-Based COBOL Components

Follow the instructions below to extract computation-based COBOL components.

1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. Select the variable or statement you want to slice on in the Source pane. In the **Point** field, click the link for the current selection and choose **Set** in the pop-up menu.


 **Note:** If you slice on a variable, you must select **Variable** in the Component Type Specific options for computation-based extraction.

To unset a variable or statement, click it and choose **Unset** in the pop-up menu. To navigate quickly to a variable or statement in the source, click it and choose **Locate** in the pop-up menu.

3. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.
4. If you plan to specify **Use Blocking from Component Definitions** in the Interface options, select the list of statements to block in Code Search, then click the link for the current selection in the **Block statements** field and choose **Set** in the drop-down menu.




Note: Choose **Show** to display the current list in Code Search. Choose **(none)** to unset the list. For Code Search usage, see *Analyzing Programs* in the product documentation set.

5. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
6. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
7. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
8. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Extracting Computation-Based Natural Components


Follow the instructions below to extract computation-based Natural components.

1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. Select the variable or statement you want to slice on in the Source pane. In the **Point** field, click the link for the current selection and choose **Set** in the pop-up menu.



Note: If you slice on a variable, you must select **Variable** in the Component Type Specific options for computation-based extraction.

To unset a variable or statement, click it and choose **Unset** in the pop-up menu. To navigate quickly to a variable or statement in the source, click it and choose **Locate** in the pop-up menu.

3. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
4. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
5. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
6. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Extracting Domain-Based Components

Domain-Based Componentization lets you "specialize" a program based on the values of one or more variables. The specialized program is typically intended for reuse "in place," in the original application but under new external circumstances.

After a change in your business practices, for example, a program that invokes processing for a "payment type" variable could be specialized on the value `PAYMENT-TYPE = "CHECK"`. Component Maker isolates every process dependent on the `CHECK` value to create a functionally complete program that processes check payments only.

Two modes of domain-based componentization are offered:

- In simplified mode, you set the specialization variable to its value anywhere in the program except a data port. The value of the variable is "frozen in memory." Operations that could change the value are ignored.

- In advanced mode, you set the specialization variable to its value at a data port. Subsequent operations can change the value, following the data and control flow of the program.

Use the simplified mode when you are interested only in the final value of a variable, or when a variable never receives a value from outside the program. Use the advanced mode when you need to account for data coming into a variable (when the variable's value is repeatedly reset, for example). The next two sections describe these modes in detail.



Tip: Component Maker lets you set the specialization variable to a range of values (between 1 and 10 inclusive, for example) or to multiple values (not only CHECK but CREDIT-CARD, for example). You can also set the variable to all values not in the range or set of possible values (every value but CHECK and CREDIT-CARD, for example).

Understanding Program Specialization in Simplified Mode

In the simplified mode of program specialization, you set the specialization variable to its value anywhere in the program except a data port. The value of the variable is "frozen in memory." The table below shows the result of using the simplified mode to specialize on the values CURYEAR = 1999, MONTH = 1, CURMONTH = 12, DAY1 = 4, and CURDAY = 7.

Source Program	Specialized Program	Comment
<pre>INP3. DISPLAY "INPUT DAY". ACCEPT DAY1. MOVE YEAR TO tmp1. PERFORM ISV. IF DAY1 > tt of MONTHS (MONTH) OR DAY1 < 1 THEN DISPLAY "WRONG DAY".</pre>	<pre>INP3. DISPLAY "INPUT DAY". MOVE YEAR TO tmp1. PERFORM ISV. IF 0004 > TT OF MONTHS(MONTH) THEN DISPLAY "WRONG DAY" END-IF.</pre>	<p>ACCEPT removed.</p> <p>No changes in these statements (YEAR is a "free" variable).</p> <p>Value for DAY1 substituted. The 2nd condition for DAY1 is removed as always false. END-IF added.</p>
<pre>MAINCALC. IF YEAR > CURYEAR THEN MOVE YEAR TO INT0001 MOVE CURYEAR TO INT0002 MOVE 1 TO direction ELSE MOVE YEAR TO INT0002 MOVE 2 TO direction MOVE CURYEAR TO INT0001.</pre>	<pre>MAINCALC. IF YEAR > 1999 THEN MOVE YEAR TO INT0001 MOVE 1999 TO INT0002 MOVE 1 TO direction ELSE MOVE YEAR TO INT0002 MOVE 2 TO direction MOVE 1999 TO INT0001.</pre>	<p>Value for CURYEAR substituted.</p>
<pre>MOVE int0001 TO tmp3. MOVE int0002 TO tmp4. IF YEAR NOT EQUAL CURYEAR THEN PERFORM YEARS.</pre>	<pre>MOVE int0002 TO tmp4. IF YEAR NOT = 1999 THEN PERFORM YEARS.</pre>	<p>Component Maker removes the first line for tmp3, because this variable is never used again. Value for CURYEAR substituted.</p>
<pre>IF MONTH > CURMONTH THEN MOVE MONTH TO INT0001 MOVE CURMONTH TO</pre>		<p>Value for MONTH substituted, making the condition (1>12) false, so Component Maker removes the IF branch and then the whole conditional statement as such.</p>

Source Program	Specialized Program	Comment
INT0002 MOVE 1 TO direction		
ELSE MOVE MONTH TO INT0002 MOVE 2 TO direction MOVE CURMONTH TO INT0001.	MOVE 0001 TO INT0002 MOVE 2 TO direction MOVE 0012 TO INT0001.	The three unconditional statements remain from the former ELSE branch. Value for CURMONTH substituted.
IF MONTH NOT EQUAL CURMONTH THEN PERFORM MONTHS.	PERFORM MONTHS.	The condition is true, so the statement is made unconditional.
IF DAY1 > CURDAY THEN MOVE DAY1 TO INT0001 MOVE CURDAY TO INT0002 MOVE 1 TO direction		This condition (4>7) is false, so Component Maker removes the IF branch and then the whole conditional statement as such.
ELSE MOVE DAY1 TO INT0002 MOVE 2 TO direction MOVE CURDAY TO INT0001.	MOVE 4 TO INT0002 MOVE 2 TO direction MOVE 0007 TO INT0001.	The three unconditional statements remain from the former ELSE branch. Values for DAY1 and CURDAY substituted.
IF day1 NOT EQUAL CURDAY THEN PERFORM DAYS.	PERFORM DAYS.	The condition is true, so the statement is made unconditional.

Understanding Program Specialization in Advanced Mode

In the advanced mode of program specialization, you set the specialization variable to its value at a data port: any statement that allows the program to receive the variable's value from a keyboard, database, screen, or other input source. Subsequent operations can change the value, following the data and control flow of the program. The table below shows the result of using the advanced mode to specialize on the values MONTH = 1 and DAY1 = 4.

Source Program	Specialized Program	Comment
INP1. DISPLAY "INPUT YEAR (1600-2099)". ACCEPT YEAR. IF YEAR > 2099 OR YEAR < 1600 THEN DISPLAY "WRONG YEAR".	INP1. DISPLAY "INPUT YEAR (1600-2099)". ACCEPT YEAR. IF YEAR > 2099 OR YEAR < 1600 THEN DISPLAY "WRONG YEAR".	No changes in these statements (YEAR is a "free" variable).
INP2. DISPLAY "INPUT MONTH". ACCEPT MONTH. IF MONTH > 12 OR MONTH < 1 THEN DISPLAY "WRONG MONTH".	INP2. DISPLAY "INPUT MONTH". MOVE 0001 TO MONTH.	ACCEPT is replaced by MOVE with the set value for MONTH. With the set value, this IF statement can never be reached, so Component Maker removes it.

Source Program	Specialized Program	Comment
<pre>INP3. DISPLAY "INPUT DAY". ACCEPT DAY1. MOVE YEAR TO tmp1. PERFORM ISV. IF DAY1 > tt of MONTHS (MONTH) OR DAY1 < 1 THEN DISPLAY "WRONG DAY".</pre>	<pre>INP3. DISPLAY "INPUT DAY". MOVE 0004 TO DAY1. MOVE YEAR TO tmp1. PERFORM ISV. IF 0004 > TT OF MONTHS(MONTH) THEN DISPLAY "WRONG DAY" END-IF.</pre>	<p>ACCEPT is replaced by MOVE with the set value for DAY1.</p> <p>No changes in these statements (YEAR is a "free" variable).</p> <p>The 2nd condition for DAY1 is removed as always false. END-IF added.</p>

Understanding Program Specialization Lite

Ordinarily, you must turn on the **Perform Program Analysis** option in the project verification options before verifying the Cobol program you want to specialize. If your application is very large, however, and you know that the specialization variable is never reset, you can save time by skipping program analysis during verification and using the simplified mode to specialize the program, so-called "program specialization lite."


Component Maker gives you the same result for a lite extraction as it would for an ordinary domain extraction in simplified mode, with one important exception. Domain extraction lite cannot calculate the value of a variable that depends on the value of the specialization variable. Consider the following example:


```
01 X Pic 99.
01 Y Pic 99.
...
MOVE X To Y.
IF X = 1
  THEN ...
  ELSE ...
END-IF.
...
IF Y = 1
  THEN ...
  ELSE ...
END-IF.
```

If you set X to 1, both simplified mode and domain extraction lite resolve the IF X = 1 condition correctly. Only simplified mode, however, resolves the IF Y = 1 condition.


Extracting Domain-Based COBOL Components


Follow the instructions below to extract domain-based COBOL components.


1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. In the **Data Item Value** field, click the **here** link. Choose one of the following methods in the pop-up menu:
 - **HyperCode List** to set the specialization variable to the constant values in a list of constants.
 - **User Specified Value(s)** to set the specialization variable to a value or values you specify.
3. Select the specialization variable or its declaration in the Source pane. Click the link for the current selection in the **Data Item** field and choose **Set** in the drop-down menu. For advanced program specialization, you can enter a structure in **Data Item** and a field inside the structure in **Field**.



 **Note:** To delete an entry, select the link for the numeral that identifies it and choose **Delete** in the pop-up menu. To unset an entry, click it and choose **Unset** in the pop-up menu. To navigate quickly to a variable or declaration in the source, click it and choose **Locate** in the pop-up menu.

4. In the **Comparison** field, click the link for the current comparison operator and choose:
 - **equals** to set the specialization variable to the specified values.
 - **not equals** to set the specialization variable to every value but the specified values.
5. If you chose **HyperCode List**, select the list of constants in Code Search, then click the link for the current selection in the **List Name** field and choose **Set** in the drop-down menu.

 **Note:** Choose **Show** to display the current list in Code Search. Choose **(none)** to unset the list. For Code Search usage, see *Analyzing Programs* in the product documentation set.
6. If you chose **User Specified Value(s)**, click the **here** link in the **Values** field. Choose one of the following methods in the pop-up menu:
 - **Value** to set the specialization variable to one or more values. In the **Value** field, click the link for the current selection. A dialog opens where you can enter a value in the text field. Click **OK**.


 **Note:** Put double quotation marks around a string constant with blank spaces at the beginning or end.
 - **Value Range** to set the specialization variable to a range of values. In the **Lower** field, click the link for the current selection. A dialog opens where you can enter a value for the lower range end in the text field. Click **OK**. Follow the same procedure for the **Upper** field.


 **Note:** For value ranges, the specialization variable must have a numeric data type. Only numeric values are supported.
7. Repeat this procedure for each value or range of values you want to set and for each variable you want to specialize on. For a given specialization variable, you can specify the methods in any combination. For a given extraction, you can specify simplified and advanced modes in any combination.

 **Note:** To delete a value or range, select the link for the numeral that identifies it and choose **Delete** in the pop-up menu.
8. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.
9. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
10. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
11. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
12. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.


Extracting Domain-Based PL/I Components


Follow the instructions below to extract domain-based PL/I components.


 **Note:** Not-equals comparisons and value ranges are not supported in PL/I.

1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. In the **Data Item Value** field (to set a single specialization variable) or the **Value for Data Item List** (to set a list of specialization variables), click the **here** link. Choose one of the following methods in the pop-up menu:


- **HyperCode List** to set the specialization variable(s) to the constant values in a list of constants.
 - **User Specified Value(s)** to set the specialization variable(s) to a value or values you specify.
3. If you are setting:
 - A single specialization variable, select the specialization variable or its declaration in the Source pane. Click the link for the current selection in the **Data Item** field and choose **Set** in the drop-down menu. For advanced program specialization, you can enter a structure in **Data Item** and a field inside the structure in **Field**.
 - A list of specialization variables, click the link for the current selection and choose the list of variables or declarations to use in the pop-up menu.



 **Note:** To delete an entry, select the link for the numeral that identifies it and choose **Delete** in the pop-up menu. To unset an entry, click it and choose **Unset** in the pop-up menu. To navigate quickly to a variable or declaration in the source, click it and choose **Locate** in the pop-up menu.
 4. If you chose **HyperCode List**, select the list of constants in Code Search, then click the link for the current selection in the **List Name** field and choose **Set** in the drop-down menu.

 **Note:** Choose **Show** to display the current list in Code Search. Choose **(none)** to unset the list. For Code Search usage, see *Analyzing Programs* in the product documentation set.
 5. If you chose **User Specified Value(s)**, click the **here** link in the **Values** field. Choose one of the following methods in the pop-up menu:
 - **Value** to set the specialization variable to one or more values. In the **Value** field, click the link for the current selection. A dialog opens where you can enter a value in the text field. Click **OK**.

 **Note:** Put double quotation marks around a string constant with blank spaces at the beginning or end.

 - **Value Range** to set the specialization variable to a range of values. In the **Lower** field, click the link for the current selection. A dialog opens where you can enter a value for the lower range end in the text field. Click **OK**. Follow the same procedure for the **Upper** field.

 **Note:** For value ranges, the specialization variable must have a numeric data type. Only numeric values are supported.
 6. Repeat this procedure for each value or range of values you want to set and for each variable you want to specialize on. For a given specialization variable, you can specify the methods in any combination. For a given extraction, you can specify simplified and advanced modes in any combination.

 **Note:** To delete a value or range, select the link for the numeral that identifies it and choose **Delete** in the pop-up menu.
 7. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
 8. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
 9. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
 10. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Eliminating Dead Code

Dead Code Elimination (DCE) is an option in each of the main component extraction methods, but you can also perform it on a standalone basis. For each program analyzed for dead code, DCE generates a component that consists of the original source code minus any unreferenced data items or unreachable procedural statements. Optionally, you can have DCE comment out dead code in Cobol and Natural applications, rather than remove it.



Note: Use the batch DCE feature to find dead code across your project. If you are licensed to use the Batch Refresh Process (BRP), you can use it to perform dead code elimination across a workspace.

Generating Dead Code Statistics

Set the **Perform Dead Code Analysis** option in the project verification options if you want the parser to collect statistics on the number of unreachable statements and dead data items in a program, and to mark the constructs as dead in Interactive Analysis. You can view the statistics in the Legacy Estimation tool, as described in *Analyzing Projects* in the product documentation set.



Note: You do not need to set this option to perform dead code elimination in Component Maker.

For COBOL programs, you can use a DCE coverage report to identify dead code in a source program. The report displays the text of the source program with its "live," or extracted, code shaded in pink.

Understanding Dead Code Elimination

Let's look at a simple before-and-after example to see what you can expect from Dead Code Elimination.

Before:




```
WORKING-STORAGE SECTION.  
  
01 USED-VARS.  
05 USED1 PIC 9.  
  
01 DEAD-VARS.  
05 DEAD1 PIC 9.  
05 DEAD2 PIC X.  
  
PROCEDURE DIVISION.  
  
FIRST-USED-PARA.  
MOVE 1 TO USED1.  
GO TO SECOND-USED-PARA.  
MOVE 2 TO USED1.  
  
DEAD-PARA1.  
MOVE 0 TO DEAD2.  
  
SECOND-USED PARA.  
MOVE 3 TO USED1.  
STOP RUN.
```

After:

```
WORKING-STORAGE SECTION.  
  
01 USED-VARS.  
05 USED1 PIC 9.  
  
PROCEDURE DIVISION.  
  
FIRST-USED-PARA.  
MOVE 1 TO USED1.  
GO TO SECOND-USED-PARA.  
  
SECOND-USED PARA.  
MOVE 3 TO USED1.  
STOP RUN.
```

Extracting Optimized Components

Follow the instructions below to extract optimized components for all supported languages.

1. Select the program you want to analyze for dead code in the Interactive Analysis Objects pane and click the  button. To analyze the entire project of which the program is a part, click the  button.
2. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new components to the list of components. If you selected batch mode, Component Maker creates a logical component for each program in the project, appending `_n` to the name of the component.
3. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.
4. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
5. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
6. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
7. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.



Performing Entry Point Isolation

Entry Point Isolation lets you build a component based on one of multiple entry points in a legacy program (an inner entry point in a Cobol program, for example) rather than the start of the Procedure Division. Component Maker extracts only the functionality and data definitions required for invocation from the selected point.

Entry Point Isolation is built into the main methods as an optional optimization tool. It's offered separately in case you want to use it on a stand-alone basis.

Extracting a COBOL Component with Entry Point Isolation

Follow the instructions below to extract a COBOL Component with entry point isolation.

1. Select the program you want to slice in the HyperView Objects pane and click the  button. A dialog opens where you can enter the name of the new component in the text field. Click **OK**. Component Maker adds the new component to the list of components. Double-click the component to edit its properties.
2. In the **Entry Point to use** field, click the link for the current selection and choose the entry point you want to use in the pop-up menu. To unset an entry point, click it and choose **Unset** in the pop-up menu.
3. In the **Description** field, click the **here** link to open a text editor where you can enter a description of the component. The description appears in the box below the **Description** field in the Properties tab and in the Description property for the logical component repository object.
4. Click the  button on the tool bar to start extracting the logical component. You are prompted to confirm that you want to continue. Click **OK**.
5. The Extraction Options dialog opens. Set options for the extraction and click **Finish**.
6. Component Maker performs the extraction. You are notified that the extraction is complete. If the extraction completed without errors or warnings, click **OK** to continue. If the extraction completed with

errors or warnings, click **Yes** in the notification dialog to view the errors or warnings in the Activity Log. Otherwise, click **No**.

Technical Details

This appendix gives technical details of Component Maker behavior for a handful of narrowly focused verification and extraction options; for COBOL parameterized slice generation; and for COBOL arithmetic exception handling.

Verification Options

This section describes how a number of verification options may affect component extraction. For more information on the verification options, see *Preparing Projects* in the product documentation set.

Override CICS Program Terminations

Select **Override CICS Program Terminations** in the project verification options if you want the parser to interpret CICS RETURN, XCTL, and ABEND commands in Cobol files as not terminating program execution.

If the source program contains CICS HANDLE CONDITION handlers, for example, some exceptions can arise only on execution of CICS RETURN. For this reason, if you want to see the code of the corresponding handler in the component, you need to check the override box. Otherwise, the call of the handler and hence the handler's code are unreachable.

Support CICS HANDLE Statements

Select **Support CICS HANDLE statements** in the project verification options if you want the parser to recognize CICS HANDLE statements in Cobol files. EXEC CICS HANDLE statements require processing to detect all dependencies with error-handling statements. That may result in adding extra paragraphs to a component.

Perform Unisys TIP and DPS Calls Analysis

Select **Perform Unisys TIP and DPS Calls Analysis** in the project verification options if you are working on a project containing Unisys 2200 Cobol files and need to perform TIP and DPS calls analysis.

This analysis tries to determine the name (value of the data item of size 8 and offset 20 from the beginning of form-header) of the screen form used in input/output operation (at CALL 'D\$READ', 'D\$SEND', 'D\$SEND', 'D\$SEND1') and establish the repository relationships ProgramSendsMap and ProgramReadsMap between the program being analyzed and the detected screen.

For example:

```
01 SCREEN-946 .
  02 SCREEN-946-HEADER .
    05 FILLER PIC X(2)VALUE SPACES .
    05 FILLER PIC 9(5)COMP VALUE ZERO .
    05 FILLER PIC X(4)VALUE SPACES .
    05 S946-FILLER PIC X(8) VALUE '$DPS$SWS '
    05 S946-NUMBER PIC 9(4) VALUE 946 .
    05 S946-NAME PIC X(8) VALUE 'SCRN946 ' .
CALL 'D$READ USING DPS-STATUS, SCREEN-946 .
```

Relationship ProgramSendsMap is established between the program and screen 'SCRN946'.



Note: Select **DPS routines may end with error** if you want to perform call analysis of DPS routines that end in an error.

Perform Unisys Common-Storage Analysis

Select **Perform Unisys Common-Storage Analysis** in the project verification options if you want the system to include in the analysis for Unisys Cobol files variables that are not explicitly declared in CALL

statements. This analysis adds implicit use of variables declared in the Common Storage Section to every CALL statement of the program being analyzed, as well as for its PROCEDURE DIVISION USING phrase. That could lead to superfluous data dependencies between the caller and called programs in case the called program does not use data from Common Storage.

Relaxed Parsing

The **Relaxed Parsing** option in the workspace verification options 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.

For code verified with relaxed parsing, Component Maker behaves as follows:

- Statements included in a component that contain errors are treated as CONTINUE statements and appear in component text as comments.
- Dummy declarations for undeclared identifiers appear in component text as comments.
- Declarations that are in error appear in component text as they were in the original program. Corrected declarations appear in component text as comments.
- Commented-out code is identified by an extra comment line: "COBOL Analyzer assumption".

PERFORM Behavior for Micro Focus COBOL

For Micro Focus COBOL applications, use the PERFORM behavior option in the workspace verification options window to specify the type of PERFORM behavior the application was compiled for. You can select:

- **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 non-recursive PERFORM behavior, a COBOL program can contain PERFORM mines. In informal terms, a PERFORM mine is a place in a program that can contain an exit point of some active but not current PERFORM during program execution.

The program below, for example, contains a mine at the end of paragraph C. When the end of paragraph C is reached during PERFORM C THRU D execution, the mine "snaps" into action: control is transferred to the STOP RUN statement of paragraph A.

```
A.  
  PERFORM B THRU C.  
  STOP RUN.  
B.  
  PERFORM C THRU D.  
C.  
  DISPLAY 'C' .  
  * mine  
D.  
  DISPLAY 'D' .
```

Setting the compiler option to allow non-recursive PERFORM behavior where appropriate allows the COBOL Analyzer parser to detect possible mines and determine their properties. That, in turn, lets Component Maker analyze control flow and eliminate dead code with greater precision. To return to our example, the mine placed at the end of paragraph C snaps each time it is reached: such a mine is called stable. Control never falls through a stable mine. Here it means that the code in paragraph D is unreachable.

Keep Legacy Copybooks Extraction Option

Select **Keep Legacy Copybooks** in the General extraction options for COBOL if you want Component Maker not to generate modified copybooks for the component. Component Maker issues a warning if including the original copybooks in the component would result in an error.

Example 1:

```
[COBOL]
01 A PIC X.
PROCEDURE DIVISION.
COPY CP.
[END-COBOL]
[ COPYBOOK CP.CPY]
STOP RUN.
DISPLAY A.
[END-COPYBOOK CP.CPY]
```

For this example, Component Maker issues a warning for an undeclared identifier after Dead Code Elimination.

Example 2:

```
[COBOL]
PROCEDURE DIVISION.
COPY CP.
STOP RUN.
P.
[END-COBOL]
[ COPYBOOK CP.CPY]
DISPLAY "QA is out there"
STOP RUN.
PERFORM P.
[END-COPYBOOK CP.CPY]
```

For this example, Component Maker issues a warning for an undeclared paragraph after Dead Code Elimination.

Example 3:

```
[COBOL]
working-storage section.
copy file.
PROCEDURE DIVISION.
p1.
  move 1 to a.
p2.
  display b.
  display a.
p3.
  stop run.
[END-COBOL]
[ COPYBOOK file.cpy]
01 a pic 9.
01 b pic 9.
[END-COPYBOOK file.cpy]
```

For this example, the range component on paragraph p2 looks like this:

```
[COBOL]
WORKING-STORAGE SECTION.
  COPY FILE1.
  LINKAGE SECTION.
  PROCEDURE DIVISION USING A.
[END-COBOL]
while, with the option turned off, it looks like this:
[COBOL]
WORKING-STORAGE SECTION.
  COPY FILE1-A$RULE-0.
  LINKAGE SECTION.
  COPY FILE1-A$RULE-1.
[END-COBOL]
```

That is, turning the option on overrides the splitting of the copybook file into two files. Component Maker issues a warning if that could result in an error.

How Parameterized Slices Are Generated for COBOL Programs

The specifications for input and output parameters are:

- Input

A variable of an arbitrary level from the LINKAGE section or PROCEDURE DIVISION USING is classified as an input parameter if one or more of its bits are used for reading before writing.

A system variable (field of DFHEIB/DFHEIBLK structures) is classified as an input parameter if the **Create CICS Program** option is turned off and the variable is used for writing before reading.

- Output

A variable of an arbitrary level from the LINKAGE section or PROCEDURE DIVISION USING is classified as an output parameter if it is modified during component execution.

A system variable (a field of DFHEIB/DFHEIBLK structures) is classified as an output parameter if the **Create CICS Program** option is turned off and the variable is modified during component execution.

- For each input parameter, the algorithm finds its first usage (it does not have to be unique, the algorithm processes all of them), and if the variable (parameter from the LINKAGE section) is used for reading, code to copy its value from the corresponding field of BRE-INPUT-STRUCTURE is inserted as close to this usage as possible.
- The algorithm takes into account all partial or conditional assignments for this variable before its first usage and places PERFORM statements before these assignments.

If a PERFORM statement can be executed more than once (as in the case of a loop), then a flag variable (named BRE-INIT-COPY-FLAG-[<n>] of the type PIC 9 VALUE 0 is created in the WORKING-STORAGE section, and the parameter is copied into the corresponding variable only the first time this PERFORM statement is executed.

- For all component exit points, the algorithm inserts code to copy all output parameters from working-storage variables to the corresponding fields of BRE-OUTPUT-STRUCTURE.

Variables of any level (rather than only 01-level structures together with all their fields) can act as parameters. This allows exclusion of unnecessary parameters, making the resulting programs more compact and clear.

For each operator for which a parameter list is generated, the following transformations are applied to the entire list:

- All FD entries are replaced with their data descriptions.
- All array fields are replaced with the corresponding array declarations.
- All upper-level RENAMES clauses are replaced with the renamed declarations.
- All upper-level REDEFINES clauses with an object (including the object itself, if it is present in the parameter list) are replaced with a clause of a greater size.
- All REDEFINES and RENAMES entries of any level are removed from the list.
- All variable-length arrays are converted into fixed-length of the corresponding maximal size.
- All keys and indices are removed from array declarations.
- All VALUE clauses are removed from all declarations.
- All conditional names are replaced with the corresponding data items.

Setting a Specialization Variable to Multiple Values

For Domain-Based Componentization, Component Maker lets you set the specialization variable to a range of values (between 1 and 10 inclusive, for example) or to multiple values (not only CHECK but CREDIT-CARD, for example). You can also set the variable to all values not in the range or set of possible values (every value but CHECK and CREDIT-CARD, for example).

Component Maker uses multiple values to predict conditional branches intelligently. In the following code fragment, for example, the second IF statement cannot be resolved with a single value, because of the two

conflicting values of Z coming down from the different code paths of the first IF. With multiple values, however, Component Maker correctly resolves the second IF, because all the possible values of the variable at the point of the IF are known:

```
IF X EQUAL Y
  MOVE 1 TO Z
ELSE
  MOVE 2 TO Z
DISPLAY Z.
IF Z EQUAL 3
  DISPLAY "Z=3"
ELSE
  DISPLAY "Z<>3"
```

Keep in mind that only the following COBOL statements are interpreted with multiple values:

- COMPUTE
- MOVE
- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

That is, if the input of such a statement is defined, then, after interpretation, its output can be defined as well.

Single-Value Example:

```
MOVE 1 TO Y.
MOVE 1 TO X.
ADD X TO Y.
DISPLAY Y.
IF Y EQUAL 2 THEN...
```

In this fragment of code, the value of Y in the IF statement (as well as in DISPLAY) is known, and so the THEN branch can be predicted.

Multiple-Value Example:

```
IF X EQUAL 0
  MOVE 1 TO Y
ELSE
  MOVE 2 TO Y.
ADD 1 TO Y.
IF Y = 10 THEN... ELSE...
```

In this case, Component Maker determines that Y in the second IF statement can equal only 2 or 3, so the statement can be resolved to the ELSE branch.

The statement interpretation capability is available only when you define the specialization variable "positively" (as equalling a range or set of values), not when you define the variable "negatively" (as not equalling a range or set of values).

Arithmetic Exception Handling

For COBOL, the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements can have ON SIZE ERROR and NOT ON SIZE ERROR phrases. The phrase ON SIZE ERROR contains an arithmetic exception handler.

Statements in the ON SIZE ERROR phrase are executed when one of the following arithmetic exception conditions take place:

- The value of an arithmetic operation result is larger than the resultant-identifier picture size.
- Division by zero.

- Violation of the rules for the evaluation of exponentiation.

For MULTIPLY arithmetic statements, if any of the individual operations produces a size error condition, the statements in the ON SIZE ERROR phrase is not executed until all of the individual operations are completed.

Control is transferred to the statements defined in the phrase NOT ON SIZE ERROR when a NOT ON SIZE ERROR phrase is specified and no exceptions occurred. In that case, the ON SIZE ERROR is ignored.

Component Maker specialization processes an arithmetic statement with exception handlers in the following way:

- If a (NOT) ON SIZE ERROR condition occurred in some interpreting pass, then the arithmetic statement is replaced by the statements in the corresponding phrase.
- Those statements will be interpreted at the next pass.

Using the Batch Refresh Process

Using the Batch Refresh Process

The COBOL Analyzer (CA) Batch Refresh Process (BRP) lets you register and verify source files in batch mode. You typically use this process when sources on the mainframe have changed, and you need to synchronize the modified sources with the sources you are working with in CA. You can also use BRP to perform analysis and reporting functions.

Understanding the Batch Refresh Process

Provides an overview of the Batch Refresh Process.

The Batch Refresh Process (BRP) is a utility that supports the synchronization of sources from a mainframe or enterprise server with the CA repositories representing those sources. BRP is installed with the CA server.

BRP is responsible for updating the workspace with sources provided to it from the mainframe and verifying all unverified sources. Optionally, BRP can be configured to run any required source code pre-processing, as well as certain analysis and reporting functions.

When sources are updated to a workspace, CA determines whether or not to load the file. When the name of the incoming file matches the name of a file currently in the workspace, the two files are compared. If they are different, the incoming file will replace the existing file. If they are the same, no change is made. If the incoming file does not currently have a match in the workspace, the file is added to the default project. The default project is a project with the same name as the workspace. If this project does not exist, it is automatically created.


Updating a source in a workspace causes that source to be invalidated. Any sources that are dependent upon the updated file will also be invalidated. For example, an update to a copybook will cause all the COBOL files that use the copybook to become invalidated. That, in turn, will cause all JCL files that execute the programs in the source files to become invalidated. Once the update phase is completed, all invalidated and unverified sources in the workspace will be verified.

Configuring the Batch Refresh Process

The Batch Refresh Process is installed with the CA server. For each workspace it processes, BRP refers to an initialization file containing configuration settings. Use the BRP Configurator in the COBOL Analyzer Administration tool on the CA server to modify the initialization file settings.

1. In the CA Administration tool, choose **Administer > Configure BRP**. The BRP Configurator opens.

2. In the Current BRP Configurations pane, choose the BRP configuration file you want to edit and click **Edit**.

 **Note:** If the BRP configuration you want to edit is not listed in the pane, click **Find** to locate the file in the file system.

3. To create a new configuration, click **Add**. A Select Workspace dialog box opens, where you can specify the workspace (.rwp) file you want to configure for BRP.

 **Note:** To copy a configuration, select it and click **Copy**. To delete a configuration, select it and click **Delete**.


4. The main BRP Configurator window opens, with a tab for each configuration task:

- On the General tab, set basic BRP initialization values (required).
- On the User Exit tab, identify any user exits you have created to extend or modify BRP functionality (optional).
- On the Advanced tab, enable BRP support for IMS Analysis, Executive Report, and WebGen (optional).

5. When you are satisfied with your entries on each tab, click **OK**.

Configuring General Settings

Set required BRP initialization values on the General tab. The settings are described in the table below.

Setting	Description
CA Install Path	Specifies the path of the CA installation folder.
BRP Install Path	Specifies the path of the BRP installation folder. This folder must contain the Reports, PreparedSources, Staging, Utility, and Working folders. Use override parameters for folders in a different location.
Workspace Path	Specifies the path of the folder for the workspace to be refreshed. This folder is at the same level as the workspace (.rwp) file.
Location of Source Files	<p>Specifies the path of the folder that contains all of your sources to register. Then running the BRP for the first time will register and analyze all sources in the specified folder and subfolders. After the initial registration, running the BRP will update the sources the same way as the Source Synchronization feature.</p> <p>After you have specified the location and run BRP, the value of the option will be erased, so if you need to register more files from a different folder, make sure you specify the new location for this option again.</p> <p> Note: This option is only for in-place registration.</p>
Obsolete Processing	Check this box to turn on obsolete processing. Obsolete processing automatically determines which source files are no longer part of a "live" application and moves them to another project. Sources are determined to be obsolete by virtue of being absent from the set of incoming sources for a BRP run.
Obsolete Project	If Obsolete Processing is selected, specifies the project to which obsolete source files will be moved.
Obsolete Exclusions File	If Obsolete Processing is selected, specifies a text file that lists files that should be ignored during obsolete processing. This mechanism is intended to avoid having

Setting	Description
	CA generated or provided files classified as obsolete. For example, CA-provided system copybooks or DASDL-generated copybooks. The text file should be formatted with a single file name per line. This mechanism is also useful when there are sources that are particularly difficult to provide on an ongoing basis or if a source is generated during runtime in the application.

Configuring User Exits

Identify user exits you have created to extend or modify BRP functionality on the User Exit tab. A user exit is a point in the standard BRP processing when a user-supplied set of commands is executed. Typically the commands execute utilities that accomplish tasks ranging from source code pre-processing to specialized report generation.

Understanding Exits

There are seven user exits in BRP. Each is named and corresponds to a major division of processing, or step, in a BRP run. The names are listed below in the order they are executed:

- Setup
- Init (Initialization)
- Staging
- Update
- Verification
- Generation
- End

With the exception of the Setup and End user exits, each is executed as the very first task of the corresponding BRP step. For example, in the Generation step the Generation user exit is executed followed by executive report generation and WebGen generation.

There are some essential tasks that occur during the Setup step that make it impractical for user exit execution to be first. The Setup step is where the main BRP log is opened and all parameter values are generated, if necessary, and checked for validity. The Setup user exit occurs after the log file is created, but before parameter values are generated and checked.

The End user exit occurs at the very end of the BRP run. There are no tasks that occur after it other than closing the main BRP log file.

Which user exit should be used to execute a particular piece of functionality depends upon the task that needs to be accomplished. For example, source code pre-processing usually needs to occur prior to the sources being loaded into the workspace. This would make the Staging or Update user exits ideal. However, it is best to do source code pre-processing once all sources are in a single spot and are guaranteed to have proper file extensions. That would eliminate the Staging user exit, since it is during Staging that file extensions are added, if necessary. Therefore, the best place to execute source code pre-processing utilities is the Update user exit.

Other common uses of user exits are to run specific reporting or analysis functions. These typically require that the verification step has been completed. Therefore, the Generation user exit will typically work best for these situations.

Configuring Exits

Configuring a user exit involves two separate tasks:

- Creating a BRP-enabled utility to accomplish the task at hand.
- Pointing the user exit to that utility.

Creating a BRP-enabled utility is a non-trivial task. Guidelines and information on this subject can be found in the section *Producing Utilities for BRP*. Use the User Exit tab of the BRP Configurator to point a user exit to the corresponding BRP-enabled utility.



Note: In the default configuration the Generation user exit is configured and provides a useful example.

BRP contains anchor points for all seven user exits. The DOS batch file should be named for the user exit it corresponds to and it should be located in the BRP Utilities folder. Be sure to specify a full and complete path to the DOS file. Relative paths may not work properly in this context.

The DOS batch file must contain the actual commands that the user exit will execute. This also provides the opportunity to do more than one task in any given user exit.

BRP checks any enabled user exit INI file parameter value for validity during the Setup step. If the value does not point to an existing file, BRP will quit with a severe error.

Configuring Advanced Settings

Configure settings on the Advanced tab to improve verification performance and enable support for IMS Analysis, Executive Report, and WebGen. The settings are described in the table below.

Setting	Description
Launch Stand Alone Queue Processor (Conversion Only)	Check this box to launch queue processors during the verification step. The value of the option determines how many additional queue processors will be launched after verification finishes and will be used for database loading only.
Number of extra Queue Processors	If Launch standalone Queue Processor is selected, click the arrow buttons to specify the number of additional Queue Processors you want to launch.
Generate PCF	Generate the project control file (PCF) before updating the sources and/or after verification.
Drop indices	Controls the dropping of indices before verifying. Auto - drops the indices only if WS objects count is above 256 Yes - always drop indices No - do not drop indices
Clean Queue before processing	All items in the queue will be deleted when the Queue Processor is launched.
Wait for Queue to become empty before processing	If there are items to be processed in the Queue when the Queue Process is launched, the batch refresh process will wait until the Queue is cleared.
Wait until Queue is processed	Check this box to force BRP to wait until the Queue Processor(s) queue is empty.
Timeout in minutes	If Wait until Queue is processed is selected, click the arrow buttons to specify the time in minutes BRP should wait for the count on the Processor(s) queue to change. If the count does not change within the specified time, BRP resumes. Sixty minutes is recommended.
Run IMS Analysis	Check this box to enable IMS Analysis.
Restore indices	Restore indices after verification
Run WebGen	Check this box to enable WebGen.

Setting	Description
Run Executive Report	Check this box to enable Executive Report.
Report Folder	Specifies the folder to store the Executive Report in. Use the Browse button to locate the folder.
Debug	Check this box to enable debug mode, in which additional messages are written to the BRP log.

BRP Logging

At the beginning of every BRP run a timestamp value is generated, consisting of the date and time. That timestamp is used throughout the run in order to uniquely identify and group the logs and information generated. Each run creates log files as well as other pieces of information. The timestamp for the run is added to the beginning of the filename for each log. All logs and information files are typically written to the Reports folder of the BRP install directory.

The main BRP log is, by default, named BRP log.txt, although the name can be altered by changing the command that is found in the runBRP.bat file in the BRP install directory. If more than one BRP installation is present, add the name of the workspace being refreshed to the main BRP log name.


The main BRP log contains basic information on when each major step of the run starts and finishes, as well as any relevant summary or diagnostic information. This is the log to check to determine whether the BRP run completed successfully or not. A run to completion will result in the last message in the log indicating the process finished successfully. Log messages marked "ERROR" should be reviewed. These are problems that were encountered, but they are not bad enough to cause BRP to abend. Messages marked as "SEVERE" are issues encountered that required BRP to abend. These should be investigated and corrected.

In addition to the main BRP log, the Update Log.txt and Verify Log.txt are also generated. These, as their names indicate, document the results of the update and verification steps respectively. The update log contains an entry for each file that is added to the workspace, whether by virtue of being different (updated) or new (added). Files that are processed during update that have unknown file extensions will also be documented in this log. Files that are processed and rejected because they are not different from the version in the workspace are not documented. The verification log lists the verification status of each file that is processed during verification. Summary statistics appear at the end of the log.

The remainder of the logs and files that are generated during a BRP run are there to provide in-depth information for troubleshooting if there is a problem during the run. If there is a problem that requires the attention of support services, please be sure to include all the logs and files from a run. Sorting the file names in the Reports directory by name will naturally group them together.

Preparing Files for Batch Refresh Processing

Organize your sources in folders in a structure that reflects your enterprise organization and place those folders under the prepared sources folder (...\Workspaces\BRP_ws\PreparedSources). The BRP process will not remove the folders after registering the files, allowing you to keep adding or changing files in the structure.

 **Note:** If your files have very long file names (248 characters for file name and 260 characters for absolute path + file name), we recommend moving your BRP working folder closer to the root directory. For example, move C:\Workspaces\BRP_Training\ to C:\BRP_Training\ or even C:\BRP_T. This is especially applicable for files with nested folder structure as part of their file names.

Using Batch Refresh Process with homonyms in a workspace

Your workspace can contain more than one copy of source file, program, or Java class that have the same name loaded in a workspace. Two files that have the same name, or homonym, must be differentiated

within the workspace. That means they must be loaded from different folders and that their paths become a component of its identifying name.



Note: Homonyms are only supported in workspaces that have been created with COBOL Analyzer 3.5 or later. Homonyms are not supported in workspaces that have been upgraded from earlier versions of COBOL Analyzer.

If two copies of the file `Dispatch.cbl` exist within a new workspace, then they have to be loaded from different directories, for example:

```
loadlib\dir1\Dispatch.cbl
```

The following is treated as a distinct and different object:

```
loadlib\dir2\Dispatch.cbl
```

Both of these files can create the same program, in this case called `dispatch`.

When using the Batch Refresh Process (BRP) to update or add files to a workspace, the files are placed in the `PreparedSources` folder which is located under the BRP folder for that workspace.



Important: The folder structure under the `PreparedSources` folder needs to match the folder structure used to load the original file, if the file is to be replaced.

If a new version of the file is to be added to the workspace, then this needs to go into a different folder. In addition, new files placed in the original folder structure will be added to the workspace and will not replace files that have the same name but were loaded from a different folder.

If the same version of a copybook is used by different projects, it must only be loaded and refreshed from the first folder structure. Otherwise, the workspace will have duplicate versions of the same copybook.

For example, the following file would replace one of the two previously loaded copies of `Dispatch.cbl`:

```
<BRP_folder>\PreparedSources\loadlib\dir2\Dispatch.cbl
```

whereas the following would be treated as new files:

```
<BRP_folder>\PreparedSources\loadlib\dir2\newprog1.cbl
```

```
<BRP_folder>\PreparedSources\loadlib\dir3\Dispatch.cbl
```

Enabling Parallel Verification


Parallel verification typically improves verification performance for very large workspaces by using multiple execution agents, called *Queue Processor*, to process source files concurrently. You can start any number of Queue Processors on the local machine, remote machines, or some combination of local and remote machines. You can run parallel verification online in the COBOL Analyzer or in batch mode with the Batch Refresh Process (BRP).




Important: When you run parallel verification on more than one machine, you need to make sure that workspace and project verification options are set identically on each machine. The easiest way to do this is to log in as the same Windows user on each machine. Alternatively, you can define a default option set that is automatically assigned to every user in the environment who has not explicitly defined a custom option set. See the related topics for more information on option sets.


You enable parallel verification in three steps:

- Select the parallel verification method and the minimum number of concurrent Queue Processors on the **Verification > Parallel Verification** tab of the Workspace Options.
- Start the Queue Processors on the local and/or remote machines. If you start fewer than the minimum number of Queue Processors specified on the Parallel Verification tab, the verification process starts the needed Queue Processors automatically on the local machine.
- Verify the workspace online in the COBOL Analyzer or in batch mode using the Batch Refresh Process (BRP).

 **Note:** Verification results are reported in the Activity Log History window. They are not reported in the Activity Log itself (for online verification) or BRP log files (for batch verification). You can also use a Verification Report to view the results.

Follow the instructions below to launch Queue Processors and to specify the type of work they perform. You can launch multiple Queue Processors on the same machine. Once the minimum number of Queue Processors has been started, you can launch them at any point in the verification process.

1. In the COBOL Analyzer Administration window, choose **Administer > Launch Queue Processor**. The Launch Queue Processor window opens.
2. In the **Serve workspace** combo box, specify the workspace to be processed.
3. In the Processing Mode pane, select any combination of:
 - **Conversion** to perform operations used to generate a Interactive Analysis construct model.
 - **Verification** to perform verification operations.
 - **Code Search Query** to perform Code Search searches in offline mode.
4. Select **Produce Log File** to generate a log file for parallel verification. The log file has a name of the form `<workspace_name>HCC.<random_number>.log` and is stored at the same level as the workspace (.rwp) file.
5. Click **OK**. The product launches the Queue Processor. Click the  button on the Windows toolbar to view the Queue Processor window.

 **Note:** Once verification has started, you can change the processing mode for a Queue Processors by selecting the appropriate choice in the **Processing** menu in the Queue Processor window.


Executing the Batch Refresh Process

The BRP Configurator creates a runBRP.bat file and saves it to the location specified in the BRP Install Path configuration option. Executing this batch file will start a BRP run.


The batch file executes the runBRP.exe executable file with appropriate parameters. The command format is as follows:

```
runBRP.exe <INI file> <log file>
```

where *INI file* is the path to the BRP initialization file and *log file* is a path to the main BRP log file.

 **Note:** The workspace is locked while BRP runs. It cannot be accessed by users. In the event of BRP failure, you can unlock the workspace by choosing **Administer > Unlock Workspace** in the Administration tool.

A full BRP run will produce several detailed log files in addition to the main BRP log. These detail files will always be written to the Reports folder. The main BRP log also is written to the Reports folder by default.

 **Note:** Running multiple BRP processes simultaneously on the same workspace is not supported.

Adding Source File Extensions

It is recommended that source files coming into BRP have proper file extensions already in place. In some cases, however, this is not possible and BRP can add them if needed. There is no need to configure initialization file parameters to use the functionality.

To have BRP add the file extensions, you must separate the sources, by type, into separate folders in the PreparedSources directory. Each folder must be named for the source type it contains and the source type name must correspond to CA source type names.

CA source type names can be determined by examining the folder names found in the Sources folder of a workspace directory. If the workspace already contains a source of a particular type, there will be a folder in the Sources directory corresponding to that source type. For example, Cobol files are found in the Cobol folder. The precise file extension that is added for any particular source type is determined by the configuration of the Registration Extensions tab in the target workspace's workspace options. The first

defined file extension for each source type will be the extension that is added by BRP. For example, Cobol File has three default file extensions listed: .cbl, .cob, and .ccp. Since .cbl is listed first, that is the extension used by BRP. The order that these values appear in the workspace options can be changed by removing extensions and adding them back in.

Note that file extensions are added onto the file without regard for any currently existing file extension if this functionality is used. For example, if the files in a folder named Cobol currently have a .txt extension (which is commonly added by some mainframe FTP applications), each file would end up having an extension like .txt.cbl. Various source file naming conventions include multiple "dots" in the source name. Since this scenario is unpredictable and varies widely, it is risky and impractical to have BRP strip any possible existing file extensions.

If there is a mix of sources with and without file extensions, BRP can handle this. Any files with proper extensions should be placed in the PreparedSources directory directly, as normal. Any files that need extensions should be dealt with as described above.

BRP Logging

At the beginning of every BRP run a timestamp value is generated, consisting of the date and time. That timestamp is used throughout the run in order to uniquely identify and group the logs and information generated. Each run creates log files as well as other pieces of information. The timestamp for the run is added to the beginning of the filename for each log. All logs and information files are typically written to the Reports folder of the BRP install directory.

The main BRP log is, by default, named BRP log.txt, although the name can be altered by changing the command that is found in the runBRP.bat file in the BRP install directory. If more than one BRP installation is present, add the name of the workspace being refreshed to the main BRP log name.

The main BRP log contains basic information on when each major step of the run starts and finishes, as well as any relevant summary or diagnostic information. This is the log to check to determine whether the BRP run completed successfully or not. A run to completion will result in the last message in the log indicating the process finished successfully. Log messages marked "ERROR" should be reviewed. These are problems that were encountered, but they are not bad enough to cause BRP to abend. Messages marked as "SEVERE" are issues encountered that required BRP to abend. These should be investigated and corrected.

In addition to the main BRP log, the Update Log.txt and Verify Log.txt are also generated. These, as their names indicate, document the results of the update and verification steps respectively. The update log contains an entry for each file that is added to the workspace, whether by virtue of being different (updated) or new (added). Files that are processed during update that have unknown file extensions will also be documented in this log. Files that are processed and rejected because they are not different from the version in the workspace are not documented. The verification log lists the verification status of each file that is processed during verification. Summary statistics appear at the end of the log.

The remainder of the logs and files that are generated during a BRP run are there to provide in-depth information for troubleshooting if there is a problem during the run. If there is a problem that requires the attention of support services, please be sure to include all the logs and files from a run. Sorting the file names in the Reports directory by name will naturally group them together.

Producing Utilities for BRP

The following are guidelines for producing utilities for BRP. These guidelines apply for any utility. Currently these utilities are normally written by support services and partners.

Versioning

Each utility needs a version number. The version number should be the date of the last modification made to the utility, formatted as follows:

```
yyyymmdd
```

The version number must appear in the first line of the log file that the utility produces.

Logging

Log files are often the only way to get reliable data. The task of analyzing output can become easier when the log files are used and recording appropriate levels of output.

At a minimum log files need to contain:

- Utility name
- Utility version
- Parameter names and values
- Record of files modified/written (when appropriate)
- Record of individual changes made to modified files (when appropriate)

The log message format should be as follows:

```
hh:mm:ss<tab>message type<tab>message
```

Message types can include INFO, WARN, ERROR, SEVERE, or DEBUG. These are generally self-explanatory, but SEVERE should not be used unless there is an abend (in Perl the die() command). Add new message types if the situation calls for it. For example, BRP has a SETUP message type.

Source, Executable, and CFG Files

Utilities are produced by support services and partners and are delivered as a compiled executable with documentation and, if necessary, a CFG file.

Guidelines for BRP Utilities

This section focuses on guidelines for utilities that need to be "enabled" for BRP. The only difference is where input is coming from, output is going to, and how parameters are provided.

BRP and Non-BRP Modes

In general, any utility created for BRP should also be able to be run in a standalone manner; that is, it should run outside of and separate from BRP as well. Typically this means getting parameters from a CFG file. This is already being done for all pre-processing type utilities right now. There are occasional situations where this is not practical. The utility needs to be able to determine whether it is being executed in a BRP context or not. If the stand-alone mode requires a CFG file, the absence of a CFG file parameter can serve as a trigger for BRP-mode execution. Where this will not work, the first parameter of the utility should be "BRP" to trigger BRP-mode execution.

Using User Exits

There are several user exit points in BRP. At different user exits potential input files are in different places and output requirements are different as well. Knowing which user exit a utility is going to be run from is crucial. It is recommended that support services be consulted regarding which user exit to employ for a particular task. The majority of user exit utilities are source code pre-processors and all use the Update user exit.

Parameter Data

Parameter data typically comes from any of three general sources: command line, CFG file, or DOS environment variables. The first two are straightforward. DOS environment variables are easily acquired by capturing the output of the DOS set command with the following line of Perl code:

```
$dos_env_vars_str = `set`;
```



Note: The special characters preceding and following the word "set" are not single quote characters; they are "backtick" characters.

The parameter values that drive a BRP run are made available to a user exit via DOS environment variables. BRP generates a DOS batch file that contains commands to set DOS environment variables.

The user exit command is added to the end of the generated batch file and the batch file is executed using the backtick operator in Perl. The backtick operator executes a DOS command (in BRP the path to a batch file) in a shell "nested" inside of the shell of the BRP executable. The environment variables set up for a user exit only exist during the execution of that user exit. The environment variable commands are re-generated and run for each user exit.

In general, the format of parameter names and values should be standardized. BRP job parameters are of the form:

```
Parameter Name = Parameter Value
```

DOS environment variables and CFG file parameters are formatted in the same way. Command line parameters should follow the same standard. In general, command line parameters need to override the same named parameter from a CFG file or DOS environment. This allows a way to alter behavior in cases where the user may not have direct control over all the values.

Logging

In addition to the general logging guidelines, the name of the log file and where it is written need to be addressed in BRP-mode utilities.

The log file name pattern is:

```
(timestamp)UtilityName Log.txt
```

where *UtilityName* is obvious and *timestamp* is a BRP environment parameter (BRP_TIMESTAMP) that identifies all logs for a BRP run.

Input/Output

Input and output locations will change depending on what files are needed and which user exit the utility is run from. Most utilities (source code pre-processing) will be running from the Update user exit.

The sources coming into the BRP process will be in the following path:

```
BRP_STAGINGDIR\BRP_TIMESTAMP
```

where *BRP_STAGINGDIR* is a full path referring to the Staging folder of a BRP install and *BRP_TIMESTAMP* holds the timestamp value for the current BRP run.

Output sources must be written back to this same location. However, to maintain integrity should the user-exit utility fail or otherwise not finish, it is recommended that output sources be written to the BRP Working folder (BRP_WORKINGDIR) and only when processing is completed should they then be copied back to the proper output location. A subfolder should be created in the Working folder for this purpose using the following format:

```
timestamp_UtilityName
```

This naming convention is required.

Returning Values

BRP determines the return state of a user exit by examining all the output written to the "console" (STDOUT in Perl terminology) by the commands executed by the user exit. The examination is done after the user exit completes execution and control returns back to the BRP run. If there is no output BRP assumes the user exit commands completed successfully. If there is any output found BRP assumes there was a SEVERE level error and will immediately stop the run.

BRP will include any output it finds in a SEVERE level message in the main BRP log. Any user exit executed utility should be sure to make effective use of this behavior. User exits do not have any knowledge of what commands or utilities they are executing. Therefore a message written to the console should contain the utility or command name along with an appropriately brief message. The details behind a utility failure can be included in the utility's own log.

BRP Environment Parameters

BRP parameter values are split into two groups. Ones prefixed with "BRP_" are for BRP specific values. Those prefixed with "EXT_" are for source file extension definitions. The tables below list all variables that are set by BRP for use by user exits along with a short description. Any path value will be fully qualified unless otherwise noted.

The following parameter uniquely identifies a BRP run.

Name	Description
BRP_TIMESTAMP	Timestamp value that uniquely identifies a BRP run and the logs that are generated during that run.

The following parameters are the paths to log files from BRP runs. Note that all the references to specific jobs are default settings only. There are very few cases where these exact jobs will not be used, but they do exist.

Name	Description
BRP_APPLYOBSOLETEPCFLOGFILE	Log from ApplyPCF.bj job for applying the BRP_OBSOLETEPCF file.
BRP_BWGLOGFILE	Log from BWG.exe (Batch WebGen).
BRP_CREATEBEGINPCFLOGFILE	Log from CreatePCF.bj job. This is run at the beginning of the BRP run and creates the file BRP_BEGINPCF.
BRP_CREATEENDPCFLOGFILE	Log from CreatePCF.bj job. This is run just before verification and creates the file BRP_ENDPCF.
BRP_EXECEPORTLOGFILE	Log for the ExecutiveReport.bj job.
BRP_GETTEXTLOGFILE	Log for the GetExtensions2.mbu job. This is run at the beginning of the BRP and creates the file BRP_FILEEXTFILE. See the section below on extension values for more information.
BRP_IMSANALYSISLOGFILE	Log for the IMS Analysis.bj job.
BRP_UPDATELOGFILE	Log for the UpdateOnly.bj job.
BRP_VERIFYLOGFILE	Log for the VerifyOnly2.bj job.

The following parameters are for the various PCF files that are generated and used during a BRP run.

Name	Description
BRP_BEGINPCF	Generated at the beginning of a BRP run. Used for many purposes in BRP including determining obsolete sources.
BRP_ENDPCF	Generated toward the end of a BRP run, after the BRP_OBSOLETEPCF is applied and before verification.
BRP_OBSOLETEPCF	Generated during a Master BRP run if BRP_OBSOLETEPROCESSING is set to 1. This will shift sources missing from the current incoming set of files to the project specified in BRP_OBSOLETEPROJECT.

The following parameters are data files that BRP uses, flags that turn certain processing on or off, BRP install folders, and other various values.

Name	Description
BRP_DROPIND	Flag (1/0) that drops database indexes to improve verification and IMS Analysis performance.

Name	Description
BRP_LAUNCHHCC	Flag (1/0) that launches the Queue Processor to improve verification performance.
BRP_WAITHCC	Time in minutes to wait for Queue Processor to respond.
BRP_BRPLOGFILE	Main log file for a BRP run.
BRP_BRPINSTALLPATH	Path where BRP is installed. This is specified in the BRP initialization file. BRP will derive the path values for the six BRP folders (PreparedSources, Reports, Staging, Utilities, and Working) based off this path if they are not specified in the initialization file.
BRP_PREPAREDSOURCESDIR	Path where sources coming into BRP start off. In cases where a utility is in place to handle getting the sources off a server or mainframe this is the location where those sources are copied to.
BRP_REPORTDIR	Path to the folder where all log files are written to. Other resource files created during a BRP run are also written here including all PCF files, file extension data file and all generated user exit batch files.
BRP_RMWINSTALLPATH	Path to the install folder for CA.
BRP_STAGINGDIR	Folder where sources reside for updating to the workspace. Sources will actually be in a subfolder that is named with the BRP timestamp and not the Staging folder directly. This is also where the majority of source pre-processing utilities will look for inputs and write outputs.
BRP_UTILITIESDIR	Path to directory that contains all the executables the BRP will need along with extra resource files and the static user exit batch files.
BRP_WORKINGDIR	Folder user exit utilities should use for any work they need to perform.
BRP_WORKSPACEDIR	Path to the target workspace.
BRP_FILEEXTFILE	File containing the file extension definitions for the target workspace.
BRP_LASTRUNFILE	Text file containing the timestamp of the last BRP run that completed execution.
BRP_OBSOLETEEXCLUSIONSFILE	File that lists any files that should be excluded from obsolete processing. Typically this includes files generated by CA (ex. DASDL copybooks), but is often used for client-specific sources as well.
BRP_BRPRUNTYPE	Must be set to "Master".
BRP_DEBUG	Flag (1/0) that will increase the amount of messaging written to the main BRP log. Typically this is always set to 1.
BRP_LASTRUNTIMESTAMP	The timestamp value of the last BRP run that completed execution.
BRP_OBSOLETEPROCESSING	Flag (1/0) that turns obsolete processing on or off. When it is turned on the incoming set of files will be compared against the set of file currently in the target workspace. Any files currently in the workspace, but not in the

Name	Description
	incoming set of files will be moved to an obsolete project (named in the BRP_OBSOLETEPROJECT parameter).
BRP_OBSOLETEPROJECT	Name of a project where obsolete sources will be moved to.
BRP_SITE	Documentation parameter that is set in the BRP initialization file. The value here will be written to the beginning of the main BRP log file. It is used mainly for support purposes to ensure that initialization and log files produced by a client match up.

The following parameters contain the command that will be executed by BRP. Typically this will specify a static DOS batch file (as opposed to the generated batch file BRP generates for each user exit). The static batch file is used so that multiple commands can be executed in a single user exit. These parameters will only exist if the user exit is being used. Below is a complete list of all user exit parameters, but it will be rare to see them all at once. They are listed in the order they would be executed in a BRP run. There is one user exit for each major step of the BRP. They always are the first task that is done in each step.

Name	Description
BRP_USEREXIT_SETUP	
BRP_USEREXIT_INIT	
BRP_USEREXIT_STAGING	
BRP_USEREXIT_UPDATE	
BRP_USEREXIT_VERIFICATION	
BRP_USEREXIT_GENERATION	
BRP_USEREXIT_END	

The following parameters contain information on the source file extensions that are valid for each legacy file type for the target workspace. The exact parameters that will be here depend upon what options are activated for the target workspace.

Name	Description
Parameter Names	The general format of the parameter names is EXT_ <i>type</i> , where <i>type</i> is the name of the corresponding directory in the workspace Sources folder. Note that in the past this name is not necessarily the same as the type name found in a PCF file. For example, PL/I included sources are contained in the Sources folder PLIInclude, but that source type is named "PLINC" in PCF files.
Parameter Values	The extension values are separated by a single space and will be in the same order they appear in the workspace options window. Example: EXT_COBOL = cbl cob ccp C74

Testing

Testing user exit utilities can be challenging. The easiest way to do this is to use one of the DOS batch files generated for each enabled user exit during a BRP run. These files will contain all the parameters and the values can be changed to suit the needs of the testing requirements. Replace the last command in the file with whatever command is necessary. In a Perl context we would use:

```
perl -d myUtility.pl
```

Note that a DOS command window will not execute one of these generated batch files when they have same filename the BRP run assigns (for example, (timestamp)UserExit.bat). This is due to some intrinsic

interpretation of the leading "(timestamp)" in the file name. Simply delete this portion of the file name and the batch file will work normally.

Executing Batch Scripts

Use the Batch Refresh and Verification (Brave) utility to execute batch scripts. `Brave.exe` is located in `[COBOL_Analyzer_Installation_Directory]\Bin`.

The batch scripts usually have at least three base parameters. They have to be in this order:

```
<batch script> <log file> <workspace>
```

where *batch script* is the name of the script to run (.bj file), *log file* is the path to the main BRP log file and *workspace* is the name of the workspace.



Important: The second parameter after the batch file is always the log file that is written to by the Brave utility to show the progress that has been made. If this parameter is omitted, then the workspace file is treated as a log file and will be corrupted. A new workspace connection has to be built if this happens.



Note: The parameters of the batch job are case-sensitive. If the wrong case is used, an error occurs.

The examples in this section illustrate how to run the scripts with `Brave.exe`. The examples can be adapted for use programmatically or in a batch file.

Example: Generating Reports

Follow the steps below to generate an Unresolved Report in Excel format. Refer to `ReferenceReport.bj` for argument details.

1. From a command prompt, enter the following command, substituting file names and paths as appropriate:

```
[COBOL_Analyzer_Installation_Directory]\Bin>Brave.exe "[COBOL_Analyzer_Installation_Directory]\Scripts\BRP\ReferenceReport.bj" "C:\UnresolvedLog.txt" "Workspace=C:\Workspaces\sdkworkspace.rwp" "Type=Unresolved" "File=C:\Workspaces\sdkworkspace\Output\UnresolvedReport.xls"
```

The command consists of:

- The path to `Brave.exe`.
 - The path to the `ReferenceReport.bj` file.
 - The path to the output log file generated on execution of the command.
 - The path to the workspace.
 - The type of reference report to generate.
 - The path to the output report. The format of the report depends on the extension.
2. Check the output log file for errors or warnings. Here is the log file for the command:

```
Batch Registration and Verification. Version 2.1.02.2860 (build 2.1.02.2860)
```

```
Date: 8/8/2008 Computer: D620-JEREMYW
Cmd: "[COBOL_Analyzer_Installation_Directory]\Scripts\BRP\ReferenceReport.bj" "C:\UnresolvedLog.txt" "Workspace=C:\Workspaces\sdkworkspace.rwp" "Type=Unresolved" "File=C:\Workspaces\sdkworkspace\Output\UnresolvedReport.xls"
Job: [COBOL_Analyzer_Installation_Directory]\Scripts\BRP\ReferenceReport.bj

13:43:13 >Open C:\Workspaces\sdkworkspace.rwp
13:43:15 >Report Unresolved C:\Workspaces\sdkworkspace\Output\UnresolvedReport.xls
13:43:23 C:\Workspaces\sdkworkspace\Output\UnresolvedReport.xls has been prepared
```

```
13:43:23 >Close
13:43:24 ---Finished---
```

Example: Executing Repository Queries

Follow the steps below to execute a Repository Exchange Protocol (RXP) query. RXP is an XML-based API that you can use to interact with application-level information in the workspace repository. Refer to RXP.bj for argument details.

1. From a command prompt, enter the following command, substituting file names and paths as appropriate:

```
C:\Program Files\COBOL Analyzer\Bin>Brave.exe "C:\Program Files\COBOL
Analyzer\Scripts\BRP\RXP.bj" "C:\QueryLog.txt" "Workspace=C:\Workspaces\
sdkworkspace.rwp" "RXP=C:\Program Files\COBOL Analyzer\Scripts\BRP\RXP\
Repository.rxp" "Query=Used Sources" "Output=C:\Workspaces\sdkworkspace
\Output\U
sedSources.xml"
```

The command consists of:

- The path to Brave.exe.
 - The path to the RXP.bj file.
 - The path to the output log file generated on execution of the command.
 - The path to the workspace.
 - The path to the .rxp file containing RXP queries.
 - The query to execute in the .rxp file, "Used Sources".
 - The path to the output file. The format of the file depends on the extension.
2. Check the output log file for errors or warnings. Here is the log file for the command:

```
Batch Registration and Verification. Version 2.1.02.2860 (build
2.1.02.2860)

Date: 8/8/2008 Computer: D620-JEREMYW
Cmd: "C:\Program Files\COBOL Analyzer\Scripts\BRP\RXP.bj" "C:
\QueryLog.txt" "Workspace=C:\Workspaces\sdkworkspace.rwp" "RXP=C:\Program
Files\COBOL Analyzer\Scripts\BRP\RXP\Repository.rxp" "Query=Used Sources"
"Output=C:\Workspaces\sdkworkspace\Output\UsedSources.xml"
Job: C:\Program Files\COBOL Analyzer\Scripts\BRP\RXP.bj

14:03:32 >Open C:\Workspaces\sdkworkspace.rwp
14:03:33 Cuter .ExecuteRXP (Prm.RXP, Prm.Query, Prm.Output, Prm.Project)
14:03:34 File C:\Workspaces\sdkworkspace\Output\UsedSources.xml has been
prepared
14:03:34 >Close
14:03:34 ---Finished---
```

Example: Creating Diagrams

Follow the steps below to generate Call Map diagrams in EMF format for every program in a workspace. Refer to DiagramTS.bj for argument details.

1. From a command prompt, enter the following command, substituting file names and paths as appropriate:

```
C:\Program Files\COBOL Analyzer\Bin>Brave.exe "C:\Program Files\COBOL
Analyzer\Scripts\BRP\DiagramTS.bj" "C:\DiagramLog.txt" "Workspace=C:\
Workspaces\sdkworkspace.rwp" "Scope=Call Map" "Pattern=C:\Workspaces
\sdkworkspac
e\Output\*.emf"
```

The command consists of:

- The path to Brave.exe.
- The path to the DiagramTS.bj file.
- The path to the output log file generated on execution of the command.
- The path to the workspace.
- The diagram scope, "Call Map".
- The pattern for naming the generated diagrams, consisting of the output folder, file name pattern, and extension. The format of the diagrams depends on the extension.

2. Check the output log file for errors or warnings. Here is the log file for the command:

```
Batch Registration and Verification. Version 2.1.02.2860 (build
2.1.02.2860)

Date: 8/8/2008 Computer: D620-JEREMYW
Cmd: "C:\Program Files\COBOL Analyzer\Scripts\BRP\DiagramTS.bj" "C:
\DiagramLog.txt" "Workspace=C:\Workspaces\sdkworkspace.rwp" "Scope=Call
Map" "Pattern=C:\Workspaces\sdkworkspace\Output\*.emf"
Job: C:\Program Files\COBOL Analyzer\Scripts\BRP\DiagramTS.bj

13:22:41 >Open C:\Workspaces\sdkworkspace.rwp
13:22:41 >Diagram Quick * "Call Map" "C:\Workspaces\sdkworkspace\Output
\*.emf"
Destination directory is C:\Workspaces\sdkworkspace\Output
Diagrams have been generated successfully
13:23:06 >Close
13:23:06 ---Finished---
```

Example: Performing an Advanced Search

Follow the steps below to search for all declarations of computational data items in a workspace. Refer to ClipperSearch.bj for argument details.

1. From a command prompt, enter the following command, substituting file names and paths as appropriate:

```
COBOL_Analyzer_Installation_Directory\Bin>Brave.exe "[COBOL
Analyzer_Installation_Directory]\Scripts\BRP\ClipperSearch.bj" "C:
\ClipperSearchLog.txt" "Worksp
ace=C:\Workspaces\sdkworkspace.rwp" "Criteria=General:Data Queries
\Computational
Data" "Model=COBOL" "ListName=Miscellaneous" "Category=General"
```

The command consists of:

- The path to Brave.exe.
- The path to the ClipperSearch.bj file.
- The path to the output log file generated on execution of the command.
- The path to the workspace.
- The path to the search criterion in the Interactive Analysis Advanced Search tool, including the tab name and folder names.
- The Interactive Analysis model for the source files to be searched, "COBOL".
- The Code Search list where the search results will be displayed.
- The Code Search category that contains the list.

2. Check the output log file for errors or warnings. Here is the log file for the command:

```
Batch Registration and Verification. Version 2.1.02.2860 (build
2.1.02.2860)

Date: 8/8/2008 Computer: D620-JEREMYW
Cmd: "COBOL_Analyzer_Installation_Directory\Scripts\BRP\ClipperSearch.bj"
"C:\ClipperSearchLog.txt" "Workspace=C:\Workspaces\sdkworkspace.rwp"
"Criteria=General:Data Queries\Computational Data" "Model=COBOL"
>ListName=Miscellaneous" "Category=General"
```

```

Job: C:\Program Files\COBOL Analyzer\Scripts\BRP\ClipperSearch.bj

10:33:25 >Open C:\Workspaces\sdkworkspace.rwp
10:33:26 Cuter .ClipperSearch (Prm.Criterion, Prm.Model, Prm.ListName,
Prm.Category, Prm.Accumulate)
10:33:27 (success) 236 construct(s) found.
10:33:27 >Close
10:33:27 ---Finished-

```

Using Batch Scripts

Use the batch job scripts supplied with COBOL Analyzer in BRP user exits or on a standalone basis. The scripts are located in `\<CA Home>\Scripts\BRP`.

Only scripts recommended for use by clients are described. Unless otherwise specified, tool options set in the COBOL Analyzer govern the behavior of the scripts. The *default project* referred to in the descriptions is the project with the same name as the workspace.

The *notification file* available in some scripts summarizes job execution. A sample notification file for the Verify.bj script follows. The notification file indicates that eight source files were verified successfully and two were verified with errors.

```

Date: 10/28/2009
Workspace C:\Workspaces\Training
Status of the Verification step:
successful - 8
with errors - 2
failed - 0

```

AddNew.bj

Action

Register new source files only. Use:

- Register.bj to register new source files and refresh updated source files.
- UpdateOnly.bj to refresh updated source files only.
- Refresh.bj to register and verify new and updated source files.
- Verify.bj to verify registered source files.

Syntax

```
AddNew LogFile Workspace Dir [Entity] [Project] [Detailed]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Dir	Staging directory for incoming source files.

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.
Project	Project. Default is the default project.

Optional Parameters	Description
Detailed	Log file.

AffectedCodeReport.bj

Action

Generate an Affected Code Report in MS Word format. The report shows code that would be impacted by changing a data item's definition or usage. The data item is called a *seed field*.

Syntax

```
AffectedCodeReport LogFile Workspace Model SearchPattern CriterionName
[Accumulate]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Model	Interactive Analysis model for the listed source files.
SearchPattern	Search criterion for the seed field.
CriterionName	Name of the search criterion for the seed field.
ListName	Name of the list. It can be [WS.User].ChangeAnalyzer.Working or [WS.User].ChangeAnalyzer.Affected, where [WS.User] is the name of the system where the Affected Code Report is created.
aCategoryName	Name of the category. For now it can only be ChangeAnalyzer.
outPutPath	Output Directory. If it is left blank, the report will be displayed instead of being saved to a file.

Optional Parameters	Description
Accumulate	Whether to append the results to existing results, True or False. Default is False.

AnalyzeProgram.bj

Action

Generate Interactive Analysis information for the workspace.

Syntax

```
AnalyzeProgram LogFile Workspace [Project] [Notify][Drop] [LaunchHHC]
[ExtraHHC] [StopHHC] [Wait]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Project	Project. Default is the default project.
Notify	Notification file.

Oracle Only Parameters	Description
Drop	<p>Whether to drop repository indexes. Specify:</p> <ul style="list-style-type: none"> • Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. • Yes, to drop repository indexes. • No, to not drop repository indexes. <p>Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.</p>
LaunchHHC	Whether to launch the Queue Processor, Yes or No. Launching the Queue Processor generally improves performance. Default is No.
ExtraHHC	If LaunchHHC is specified, the number of additional Queue Processors to launch.
StopHHC	Whether to stop the Queue Processor(s) when processing is complete, Yes or No.
Wait	<p>Whether to wait until the Queue Processor(s) queue is empty. Specify:</p> <ul style="list-style-type: none"> • Yes, to wait indefinitely. • No, to not wait. • The number of seconds to wait for the count on the queue to change. If the count does not change within the specified time, BRP resumes. Sixty minutes is recommended. 0 means no timeout.

ApplyPCF.bj

Action

Assign source files to projects based on a *project control file (PCF)*. A project control file identifies the projects to which source files belong. It specifies the path to the source files rather than just their names so as to avoid issues resulting from having files with homonymous names.

ApplyPCF.bj differs from SetProject.bj in that it does not allow you to assign source files to projects additively. Use CreatePCF.bj or Related.bj to create a project control file.

Syntax

```
ApplyPCF LogFile Workspace ProjectCF
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
ProjectCF	Project control file (.pcf).

BusinessRulesReport.bj

Action

Generate a Business Rules Report. The report lists the business functions, rule sets, segments, attributes, data elements, and control conditions of business rules in the workspace.

Syntax

```
BusinessRulesReport LogFile Workspace File [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Project	Project. Default is the default project.

BusinessRulesValidation.bj

Action

Indicate whether business rule segments no longer are valid. An invalid segment is out of synch with the rule, typically because lines of code have been added or deleted during a refresh or edit. Optionally, specify how to handle invalid segments.

Syntax

```
BusinessRulesValidation LogFile Workspace [Action] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Action	How to handle invalid segments. Specify: <ul style="list-style-type: none"> • leave, to retain the invalid segment but set the rule Segment Validity attribute to Invalid. • delete, to delete the invalid segment and set the rule Segment Validity attribute to Invalid. • valid, to resynchronize the segment, subject to the limitations described in <i>Analyzing Programs</i> in the product documentation set. The values are case-sensitive.
Project	Project. Default is the default project.

ChangeTraffic.bj

Action

Generate *change traffic metrics*. Change Traffic metrics compare the source files currently in a workspace with the source files in a *change unit*:

- At the tag level, the number of new, modified, and deleted files in the change unit.
- At the object level, the number of new, modified, deleted, and moved lines in the modified files, and the number of lines in the new and deleted files.



Important: An additional module required for generating the object level Change Traffic metrics is not included in the COBOL Analyzer installer. It is called diff and is a source comparison tool that has been developed as an open source module using GnuWin. As such, it is free to download, but is not something that Micro Focus can distribute.

Diff can create metrics to show differences between two source modules. These metrics are then gathered by `ChangeTraffic.bj` and loaded into CA via the Enterprise View core module `apm-core.jar`.

To download and install diff:

1. Go to <http://www.microfocus.com/docs/links.asp?ea=gnuwin>.
2. Go to the latest available version (currently it is 2.8.7.1) and select the executable to download. In this case it is `diffutils-2.8.7-1.exe`.
3. Run the executable to install diff.
4. Go to its installation directory. By default it is: `C:\Program Files (x86)\GnuWin32\bin`.
5. Copy `diff.exe` into the CA bin directory. By default it is: `C:\Program Files (x86)\Micro Focus\COBOL Analyzer\Bin`.


Once diff has been placed in the CA bin directory, `ChangeTraffic.bj` can be run as part of Batch Refresh Process to compare the files that are currently in a workspace against the files that are about to be loaded into CA.

The change unit comprises the new and modified source files, and optionally a *change unit description file*, named `Configuration.txt`. Use the change unit description file to:

- List the source files to be deleted, in the form `DELETED:filename`.
- Assign the change metrics a date other than the date the script is executed. Specify the date in the form `DATE: YYYY-MM-DD HH24:MI:SS`.

For example:

```
DATE: 2010-03-15 17:12:13
DELETED:MYPROG.CBL
DELETED:MYCOPY.CPY
```

 **Note:** Generating or importing change traffic metrics does not physically delete source files from the workspace. Use `ProcessChangeUnit.bj` with a change unit description file to delete source files from the workspace.

The output of `ChangeTraffic.bj` is an XML file that can be imported into the repository in online mode using the Enterprise View Configuration interface, or in batch mode using the following command:

```
java -jar avmcore-Panther.jar "workspace" "outputfile"
```

where *workspace* is the name of the workspace and *outputfile* is the XML output file.

 **Note:** Enterprise View must be installed before you run the import command.

Syntax

```
ChangeTraffic UDCFolderr SourceFolder XMLOutputFile ProjectName UDCName
```

Required Parameters	Description
UDCFolder	Dedicated folder for the change unit.
SourceFolder	\Sources folder for the workspace.
XMLOutputFile	Output file.
ProjectName	Name of tag for the application.
UDCName	Name of tag for the change unit. Use this argument to distinguish between change units. Enter an empty string if you do not want to specify a change unit tag.

CheckQueue.bj

Action

Check whether the Queue Processor(s) queue is empty.

Syntax

```
CheckQueue LogFile Workspace [Project] [Wait]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Project	Project. Default is the default project.

Oracle Only Parameters	Description
Wait	Whether to wait until the Queue Processor(s) queue is empty. Specify: <ul style="list-style-type: none"> • Yes, to wait indefinitely. • No, to not wait.

Oracle Only Parameters	Description
	<ul style="list-style-type: none"> The number of seconds to wait for the count on the queue to change. If the count does not change within the specified time, BRP resumes. Sixty minutes is recommended. 0 means no timeout.


ClipperDetails.bj

Action

Generate a Code Search Details Report. For each source file in the specified Code Search list, the report shows the constructs in the list, their type, and their location in the file. You can customize the report to include any Interactive Analysis attribute related to the constructs.

Syntax

```
ClipperDetails LogFile Workspace Model ListName Category FileName Attrs
[Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Model	Interactive Analysis model for the listed source files.
ListName	<p>Name of the list. The list is assumed to be owned by the current user. If it is owned by another user, append a vertical bar () and the user name.</p> <p> Note: Lists stay active in the workspace. As part of the verification process for an entity, any entry in a list for that entity is removed. The list will need to be recreated post verification. Any script that is used to create the list can be added to a post-verification user exit to the BRP process so they are correctly re-created as part of a BRP run.</p>
Category	List category.
FileName	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.
Attrs	String containing Interactive Analysis attributes to include in the report, separated by a vertical line ().

Optional Parameters	Description
Project	Project. Default is the default project.

ClipperMetrics.bj

Action

Generate a Code Search Metrics Report. For each list in the specified Code Search category, the Metrics Report shows the number of list items in each program in the workspace.

Syntax

```
ClipperMetrics LogFileWorkspace Model Category FileName [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Model	Interactive Analysis model for the listed source files.
Category	List category.
FileName	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Project	Project. Default is the default project.


ClipperMultiSearch.bj

Action

Execute a Code Search with multiple criteria. The results are displayed in the specified Code Search list.

Syntax

```
ClipperMultiSearch LogFileWorkspace Criteria Model ListName Category  
[Project] [Accumulate]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Criteria	File containing the search criteria, one per line. Specify the full path name of the criterion in the Interactive Analysis Advanced Search tool, including the tab name (General) and any folder names. For example, General:Coding Standards\MOVE Statements\Possible Data Padding. Follow the notation specified exactly.
Model	Interactive Analysis model for the listed source files.
ListName	Name of the list. The list is assumed to be owned by the current user. If it is owned by another user, append a vertical bar () and the user name.  Note: Lists stay active in the workspace. As part of the verification process for an entity, any entry in a list for that entity is removed. The list will need to be recreated post verification. Any script that is used to create the list can be added to a post-

Required Parameters	Description
	verification user exit to the BRP process so they are correctly re-created as part of a BRP run.
Category	List category.

Optional Parameters	Description
Project	Project. Default is the default project.
Accumulate	Whether to append the results to existing results, True or False. Default is False.


ClipperSearch.bj

Action

Execute a Code Search search. The results are displayed in the specified Code Search list.

Syntax

```
ClipperSearch LogFile Workspace Criterion Model ListName Category [Project]
[Accumulate]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Criterion	Full path name of the search criterion in the Interactive Analysis Advanced Search tool, including the tab name (General) and any folder names. For example, General:Coding Standards\MOVE Statements\Possible Data Padding. Follow the notation specified exactly.
Model	Interactive Analysis model for the listed source files.
ListName	Name of the list. The list is assumed to be owned by the current user. If it is owned by another user, append a vertical bar () and the user name.  Note: Lists stay active in the workspace. As part of the verification process for an entity, any entry in a list for that entity is removed. The list will need to be recreated post verification. Any script that is used to create the list can be added to a post-verification user exit to the BRP process so they are correctly re-created as part of a BRP run.
Category	Category of the list.

Optional Parameters	Description
Project	Project to execute search for.
Accumulate	Whether to append the results to existing results, True or False. Default is False.

CodeSearchReport.bj

Action

Generate a Code Search report.

Syntax

```
CodeSearchReport LogFile Workspace ReportName Project [Folder]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
ReportName	Name of the Code Search report.
Project	Project. Default is the default project.

Optional Parameters	Description
Folder	Output directory where the HTML report will be stored. If there is no value provided for Folder, the HTML report won't be created.

ComplexityReport.bj

Action

Generate a Complexity Metrics Report. The report shows complexity metrics for objects of the specified type.

Syntax

```
ComplexityReport LogFile Workspace File [Entity] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Entity	Entity type of objects to report on. Default is program.
Project	Project. Default is the default project.

CreatePCF.bj

Action

Create a *project control file (PCF)* for a workspace. A project control file identifies the projects to which source files belong. It specifies the path to the source files rather than just their names so as to avoid issues resulting from having files with homonymous names. Use ApplyPCF.bj or SetProject.bj to assign source files to projects based on the project control file.

Syntax

```
CreatePCF LogFile Workspace Out
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Out	Project control file (.pcf).

CreateWS.bj

Action

Create a workspace.

Syntax

```
CreateWS LogFile Workspace [DB]  
/* Oracle */DSN Schema Password [User]  
/* DB/2 */DSN Schema Password [User] [TableSpace] [IndexSpace]  
/* SQL Server */Server Database Password User
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
DB	* or database type: Oracle, DB2, or MSSQL. Default is *.

Oracle Required Parameters	Description
DSN	ODBC data source name (DSN) for the database that holds the repository.
Schema	Database schema name for the repository.
Password	Database password that gives access to the schema.

Oracle Optional Parameters	Description
User	Database user name that gives access to the schema.

DB/2 Required Parameters	Description
DSN	ODBC data source name (DSN) for the database that holds the repository.
Schema	Database schema name for the repository.
Password	Database password that gives access to the schema.

DB/2 Optional Parameters	Description
User	Database user name that gives access to the schema.
TableSpace	Name of the tablespace for the repository.
IndexSpace	Name of the tablespace for database indexes.

SQL Server Required Parameters	Description
Server	Name of the server.
Database	Name of the database.
Password	Password that gives access to the database.
User	User name that gives access to the database.

CRUDReport.bj

Action

Generate a CRUD Report. The report shows the data operations programs perform (Insert, Read, Update, or Delete) and the data objects on which the programs operate.

Syntax

```
CRUDReport LogFile Workspace File [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Project	Project. Default is the default project.

DBA.Cobol.bj

Action

Perform domain-based analysis of COBOL programs. Domain-based analysis "slices out" a specialized program based on the values of one or more variables.

An input file in CSV format identifies the slice parameters. Each line contains the following information:

```
ProgName, SliceName, DataItem, FileName, Row, Col, Comparison, Value, LowerValue, UpperValue
```

where:

- *ProgName* is the name of the program from which the slice will be extracted.
- *SliceName* is the name of the slice.
- *DataItem* is the name of the specialization variable.
- *FileName* is the name of the source file containing the specialization variable.
- *Row* is the row number of the specialization variable in the source file.
- *Col* is the column number of the specialization variable in the source file.
- *Comparison* is the comparison type: "equals" sets the specialization variable to the values specified in *Value*; "not equals" sets the specialization variable to every value but the values specified in *Value*.
- *Value* is the value to set the specialization variable to.
- If *Value* is omitted, *LowerValue* is the lower value of the range of values to set the specialization variable to.
- If *Value* is omitted, *UpperValue* is the upper value of the range of values to set the specialization variable to.

Multiple locations can be specified for a slice. Multiple conditions can be set for a location. All content is case-sensitive.

Input File Sample

```
DAYOFWEEK, Domain1, YEAR, DayOfWeek.cbl, 12, 12, equals, 2000, ,  
DAYOFWEEK, Domain1, YEAR, DayOfWeek.cbl, 12, 12, equals, , 2002, 2005  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 13, 12, equals, 4, ,  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 13, 12, equals, 5, ,  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 65, 19, equals, 5, ,  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 65, 19, equals, 6, ,  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 95, 15, equals, 7, ,  
DAYOFWEEK, Domain1, MONTH, DayOfWeek.cbl, 95, 15, equals, , 1, 3  
DAYOFWEEK, Domain2, YEAR, DayOfWeek.cbl, 81, 15, equals, , 2001, 2010  
GSS, Domain3, GSS1003-CMD-CODE-I, GSS.cbl, 186, 16, equals, "ENTER" , ,
```

Syntax

```
DBA.Cobol LogFile Workspace List [Options] [Export] [Notify]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
List	CSV file with slice parameters.

Optional Parameters	Description
Options	Options script file. Default values for options usually are acceptable. Contact support services for special needs.
Export	Destination folder for slices. Results normally are viewed in CA Component Maker.
Notify	Notification file.

DCE.bj

Action

Perform dead-code elimination (DCE) for programs in source files of the specified type. For each program analyzed for dead code, DCE generates a component that consists of the original source code minus any unreferenced data items or unreachable procedural statements.

Syntax

```
DCE LogFile Workspace Entity [Options] [Pattern] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Entity	Interactive Analysis model for the source files to be analyzed for dead code. Valid values are COBOL, PL 1, NATURAL, and NATSUBROUTINE.

Optional Parameters	Description
Options	Options script file. Default values for options usually are acceptable. Contact support services for special needs.
Pattern	Pattern for naming generated components. The pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the analyzed program. If the argument is omitted, component names are generated in the form BRE nn , where nn is an incrementing number.
Project	Project. Default is the default project.

DiagramCallie.bj

Action

Generate Program Control Flow Diagrams for the workspace. The diagrams show the call flow for paragraphs in a COBOL program, subroutines in an RPG program, or procedures in a PL/I or Natural program.

The *subgraph* mode offers a cyclic representation of the information in the diagram. Items are drawn once. Relationship lines cross. Subgraph views are often easier to understand than subtree views.

The *subtree* mode offers a linear representation of the information in the diagram. Items are drawn as many times as necessary. Relationship lines do not cross. Use this view if too many intersecting lines make a subgraph view hard to read.

Syntax

```
DiagramCallie LogFile Workspace [Pattern] [Mode] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Pattern	Pattern for naming the generated diagrams, consisting of the output folder, file name pattern, and extension. For example, D:*.emf. The file name pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the analyzed object. The format of the diagram depends on the extension: .bmp, .jpg, .vsd, .vdx, or .emf. Default is <code>WorkspaceFolder\Output\ObjectName.bmp</code> .
Mode	Mode of the diagram, subgraph or subtree.
Project	Project. Default is the default project.

DiagramFlowchart.bj

Action

Generate Flowchart Diagrams for the workspace. The diagrams show the flow of control between statements in a COBOL paragraph or PL/I procedure, or between steps in a job or JCL procedure.

Syntax

```
DiagramFlowchart LogFile Workspace [Pattern] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Pattern	Pattern for naming the generated diagrams, consisting of the output folder, file name pattern, and extension. For example, D:*.emf. The file name pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the analyzed object. The format of the diagram depends on the extension: .bmp, .jpg, .vsd, .vdx, or .emf. Default is <code>WorkspaceFolder\Output\ObjectName.bmp</code> .
Project	Project. Default is the default project.

DiagramProject.bj

Action


Generate a relationship flow diagram for the specified project. Use DiagramTS.bj to generate a relationship flow diagram for each workspace object of a specified type.

Syntax

```
DiagramProject LogFile Workspace Scope [Pattern] [Tag] [Layout] [Project]
[OptionSet] [ReportType]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Scope	Diagram scope (including user-defined scopes).

Optional Parameters	Description
Pattern	Pattern for naming the generated diagram, consisting of the output folder, file name pattern, and extension. For example, D:*.emf. The file name pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the project. The format of the diagram depends on the extension: .bmp, .jpg, .vsd, .vdx, or .emf. Default is <code>\WorkspaceFolder\Output\ProjectName.bmp</code> .
Tag	Tag used to black-box objects in the diagram. If related objects are contained in different black boxes, generates a Blackbox Interface Report. The report is stored in the folder location specified in <code>Pattern</code> , with the name <code>DiagramName-ProjectName-BlackBoxRpt.csv</code> . Report types are described in <code>ReportType</code> below.
Layout	Diagram layout: circular, hierarchical, orthogonal, symmetric, or tree.
Project	Project. Default is the default project.
OptionSet	Whether to use the current settings in the interactive Diagrammer tool for auto expand, project boundary entities, and potential incomplete composite relationships, or the default settings: <ul style="list-style-type: none"> *interactive, to specify the current settings. *default, to specify the default settings. Default is *default.
ReportType	Type of Blackbox Interface Report. A Blackbox Interface Report lists the relationships between objects in different black boxes. Specify: <ul style="list-style-type: none"> Full, to generate the report with a row for each relationship. A row contains:

Optional Parameters	Description
	<ul style="list-style-type: none"> The black box tag structure for the left and right objects in the relationship. The entity types and names of the left and right objects in the relationship. The relationship name. <ul style="list-style-type: none"> Distinct, to generate the report with a row for each blackbox interface (effectively collapsing the rows for relationships with the same interface). A row contains: <ul style="list-style-type: none"> The black box tag structure for the left and right objects in the relationship. The relationship name. The number of distinct objects with the left black box tag structure. The number of distinct objects with the right black box tag structure. The number of occurrences of the relationship. <p>Default is Full. Omitting the parameter is the same as specifying Full.</p> <p> Note: This parameter must be specified last in the list of parameters.</p>

DiagramTS.bj

Action

Generate a relationship flow diagram for each object of the specified type in the workspace. Specify a project to limit the diagrams to objects in the project. Use DiagramProject.bj to generate a relationship flow diagram for an entire project.

Syntax

```
DiagramTS LogFile Workspace Scope [Pattern] [Entity] [Tag] [Layout] [Project] [OptionSet]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Scope	Diagram scope (including user-defined scopes).

Optional Parameters	Description
Pattern	Pattern for naming the generated diagrams, consisting of the output folder, file name pattern, and extension. For example, D:*.emf. The file name pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the analyzed object. The format of the diagram depends on the extension: .bmp, .jpg, .vsd, .vdx, or .emf. Default is <code>WorkspaceFolder\Output\ObjectName.bmp</code> .
Entity	* or entity type of source files. Default is *.

Optional Parameters	Description
Tag	Tag used to black-box objects in the diagrams.
Layout	Diagram layout: circular, hierarchical, orthogonal, symmetric, or tree.
Project	Project. Default is the default project.
OptionSet	<p>Whether to use the current settings in the interactive Diagrammer tool for auto expand, project boundary entities, and potential incomplete composite relationships, or the default settings:</p> <ul style="list-style-type: none"> • *interactive, to specify the current settings. • *default, to specify the default settings. <p>Default is *default.</p>

EffortReport.bj

Action

Generate an Effort Estimation Report. The report compares source files based on weighted values for selected complexity metrics.

Syntax

```
EffortReport LogFile Workspace File [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Project	Project. Default is the default project.

ExecutiveReport.bj

Action

Generate an Executive Report. The report provides HTML views of application inventories that a manager can use to assess the risks and costs of supporting the application.

Syntax

```
ExecutiveReport LogFile Workspace Folder [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Project	Project. Default is the default project.

ExportDescriptions.bj

Action

Export object descriptions to an ERD file.

Syntax

```
ExportDescriptions LogFile Workspace ERD [Entity] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
ERD	ERD file.

Optional Parameters	Description
Entity	<p>* or entity type of the objects to export descriptions for. Default is *.</p> <p>Use the flag attribute of an entity type to specify all entity types with that flag, for example:</p> <pre>*LEGACY</pre> <p>which specifies all entity types with the LEGACY flag. For more on flags, see <i>Software Development Toolkit</i>, available from support services.</p>
Project	Project. Default is the default project.

ExportRules.bj

Action

Export business rules to an ERD file.

Syntax

```
ExportRules LogFile Workspace FileName [Detailed]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
FileName	ERD file.

ExportScreens.bj

Action

Export renderings for screens in the workspace.

Syntax

```
ExportScreens LogFile Workspace [Pattern] [Output] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Pattern	Pattern for naming screen renderings, consisting of the output folder, file name pattern, and extension. For example, D:*.rtf. The file name pattern may contain any valid symbols. An asterisk (*) is replaced with the name of the screen. The format of the renderings depends on the extension. Supported extensions are .rtf and .doc.
Output	Output folder if not specified in Pattern.
Project	Project. Default is the default project.

FortifyMBSExport.bj

Action

Exports a Mobile Build Session (MBS) using verified COBOL files. The MBS includes COBOL files and their respective copybooks. When created the MBS is saved to a single file with a .mbs file extension.

Syntax

```
FortifyMBSExport Workspace MbsFile [MbsDir] [Project] [Entity] [Status] [Cond] [Notify] [Wait] [LaunchHCC] [ExtraHCC] [StopHCC]
```

Required Parameters	Description
Workspace	Workspace directory or file.
MbsFile	MBS output file.

Optional Parameters	Description
<i>MbsFile</i>	Absolute path to the MBS output file. Default value is [<i>Workspace</i>]\Output\.
<i>Project</i>	Project context. This will use the default workspace project if it is not specified.
<i>Entity</i>	* or entity name to export.
<i>Status</i>	Verification status of source files to export: * Exports all source files. R Exports source files verified under the relaxed parsing option. E Exports source files that verified with an error. S Exports source files that verified successfully.
<i>Cond</i>	Condition on objects selection.
<i>Notify</i>	Notification file.

Oracle and DB2 Only Parameters	Description
<i>Wait</i>	If the HyperCodeConverter queue is empty: Yes Wait No Do not wait. timeout Specify a time in seconds before timeout occurs. If 0 seconds is specified then no timeout to wait.
<i>LaunchHCC</i>	[Yes No] - launch standalone HyperCodeConverter
<i>ExtraHCC</i>	n - launch extra HyperCodeConverters on finish
<i>StopHCC</i>	[Yes No] - stop HyperCodeConverter on finish

GenScreens.bj

Action

Generate screens from Device Description files.

Syntax

```
GenScreens LogFile Workspace [Entity] [Condition] [Project] [Notify]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.

Optional Parameters	Description
Condition	Scope of source. Use the Repository Exchange Protocol (RXP) to code the condition. For more information, see <i>Analyzing Projects</i> in the COBOL Analyzer documentation set. Default is project.
Project	Project. Default is the default project.
Notify	Notification file.

ImpactReport.bj

Action

Generate an Impact Subtree Report. The report shows the impact trace subtree for the specified data item occurrence in XML format or in a database.



Note: `ImpactReport.bj` has to be run against the usage of a variable and it will not run from the definition of a variable.

Syntax

```
ImpactReport LogFile Workspace Entity Name HCID FileName [Direction]
[Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Entity	Legacy source type, e.g. COBOL.
Name	Legacy source name.
HCID	HCID of the data item occurrence. It will only be set if the source file has been verified. Alternatively, use the following arguments: <ul style="list-style-type: none"> <i>Var</i>, to specify the variable name. <i>Source</i>, to specify the relative path of the source file containing the data item occurrence. It has the same value as the Source attribute in the General tab of the file properties. To see the value, right-click the object in the Repository pane, select Properties and then the General tab. <i>Ln</i>, to specify the line in the source file that has the variable usage that the report is to be run against. <i>Col</i>, to specify the column on the line in the source file that has the variable usage that the report is to be run against.
FileName	Output file. The format of the report depends on the extension. Supported extensions are .xml and .mdb.

Optional Parameters	Description
Direction	Direction of the trace:

Optional Parameters	Description
	<ul style="list-style-type: none"> F or 1, to specify a forward trace. B or 0, to specify a backward trace. Forward is the default.
Project	Project. Default is the default project.


ImpactReportFromList.bj

Action

Generate an Impact Subtree Report from a Code Search list. The report shows the impact trace subtrees for occurrences of data items in the list in XML format or in a database.

Syntax

```
ImpactReportFromList LogFile Workspace Model ListName Category Output
[Direction] [Split] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Model	Interactive Analysis model for the listed source files.
ListName	Name of the list. The list is assumed to be owned by the current user. If it is owned by another user, append a vertical bar () and the user name.  Note: Lists stay active in the workspace. As part of the verification process for an entity, any entry in a list for that entity is removed. The list will need to be recreated post verification. Any script that is used to create the list can be added to a post-verification user exit to the BRP process so they are correctly re-created as part of a BRP run.
Category	List category.
Output	Output file. The format of the report depends on the extension. Supported extensions are .xml and .mdb. When <code>Split</code> is set to Y, the path of the folder for .mdb output files.

Optional Parameters	Description
Direction	Direction of the trace: <ul style="list-style-type: none"> F or 1, to specify a forward trace. B or 0, to specify a backward trace. Forward is the default.
Split	Whether to use the <i>split program</i> method, Y or N. The split program method generates a separate .mdb output

Optional Parameters	Description
	file for each program that contains a data item occurrence in the list. N is the default.
Project	Project. Default is the default project.

ImportRules.bj

Action

Import business rules from an ERD file.

Syntax

```
ImportRules LogFile Workspace FileNames [Mode]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
FileNames	File name of ERD file, or a pattern to match ERD file names.

Optional Parameters	Description
Mode	<p>How to handle rules that have the same internal names as existing rules. Specify:</p> <ul style="list-style-type: none"> • Creating, to create rules with unique internal names, whether or not they have the same internal names. • Replacing, to replace existing rules whether or not they have been updated. • Updating, to replace existing rules only when they have been updated. <p>The update test is against the Last Validation Time attribute. Replacing is the default.</p>

IncludeReferences.bj


Action

Include referenced or referencing objects in a project. The project and the objects to be included must have been verified. Use this script to include in a project:

- All referenced objects.
- All referencing objects.
- Directly referencing objects only. If program A calls program B, and program B calls program C, A is said to directly reference B and indirectly reference C.

Syntax

```
IncludeReferences LogFile Workspace Cond Scope Project
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Cond	Type of object to include: <ul style="list-style-type: none"> Referenced, to include referenced objects. Referencing, to include referencing objects.
Scope	Scope of references to include: <ul style="list-style-type: none"> All, to include all referenced or referencing objects. Direct, to include directly referencing objects only.  Note: Do not omit this parameter when including referenced objects (when Cond=Referenced). Specify All.
Project	Project. Default is the default project.

Invalidate.bj

Action

Invalidate source files. Invalidating some or all of the source files in a workspace before reverifying can save time when you reverify very large workspaces.

Syntax

```
Invalidate LogFile Workspace [Entity] [Cond] [ObjList] [Detailed] [Drop]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.
Cond	Source files to invalidate. Use the Repository Exchange Protocol (RXP) to code the condition. For more information, see <i>Analyzing Projects</i> in the product documentation set.
ObjList	When Cond is not set, a control file with a list of source files to invalidate. Each line of the control file contains the following information: <pre>"EntityType" "EntityName"</pre> where: <ul style="list-style-type: none"> <i>EntityType</i> is the entity type of the source file to invalidate, COBOL, for example.

Optional Parameters	Description
	<ul style="list-style-type: none"> <i>EntityName</i> is the name of the source file to invalidate, <i>DayOfWeek.cbl</i>, for example.
Detailed	Log file.

Oracle Only Parameters	Description
Drop	<p>Whether to drop repository indexes. Specify:</p> <ul style="list-style-type: none"> Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. Yes, to drop repository indexes. No, to not drop repository indexes. <p>Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.</p>

InventoryReport.bj

Action

Generate an Inventory Report. The report shows high-level statistics for source file types in the current workspace: number of files of each type, whether verified, number of lines of code (verified files only), and the like.

Syntax

```
InventoryReport LogFile Workspace File [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

MFCobolCLink.bj

Action

Following a verification of the source files, run `MFCobolCLink.bj` to create relationships between COBOL programs and C programs, as well as between C programs and JCL programs.

Before the script is run, calls from COBOL to C functions appear as calls to unresolved program entry points. The calls from C functions to COBOL programs appear as calls to unresolved functions. The batch process scans the repository for these relationships and when it locates an appropriate match, it replaces the existing relationships with new ones that link the COBOL and C entities. For calls from COBOL to C there is a "Calls Program Entry Decision" relationship from the calling program to a new decision entity and the decision entity has a "Resolves to Function" relationship to the C function called from the COBOL program. For calls from C to COBOL there is a "Calls Decision" relationship from the calling C function to a

new decision entity and a "Resolves to Program Entry Point" from the decision to the COBOL program called from the C function.

MFCobolCLink.bj generates relationships between C and JCL programs in a similar way.

There are two options under **Options > Workspace Options > Verification > Settings > C File** that are related to this process.

The first option is called **Generate program entry points for main functions**. When checked, any C function with the name "main" will now generate an associated program entry point object whose name is the last component of the file name.

The second option is **Generate program entry points for filename functions**, which works in a similar way and is for C modules that do not have a main function.



Note: MFCobolCLink.bj needs to be re-run if any of the C, COBOL or JCL source files are modified and require verification.

Syntax

```
MFCobolCLink LogFile Workspace
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Populate.bj

Action

Populate a workspace from an ERD file.

Syntax

```
Populate LogFile Workspace ERD [Detailed]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
ERD	ERD file, or a file that contains a list of ERD files. If the latter, the list file name must be preceded by an @ symbol, for example, @ERDs.txt. Each line of the list file specifies the full path name of an ERD file.

Optional Parameters	Description
Detailed	Log file.

PortabilityAssessment.bj

Action

Generates Portability Assessment reports in the specified folder.

Syntax

```
PortabilityAssessment LogFile Workspace [Folder] [Project]
```


Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Folder	Output directory for the report.

Optional Parameters	Description
Project	Project. Default is the default project.

ProcessChangeUnit.bj

Action

Set a change date for complexity metrics other than the date source files are verified, and/or delete source files from the workspace. Specify the date and the files to be deleted in a *change unit description file*.

 **Note:** Execute this script *before* registering the source files.

Use SetChangeDate.bj to clear the change date from the system. Use ChangeTraffic.bj to generate *change traffic metrics*.

Syntax

```
ProcessChangeUnit LogFile Workspace File
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
File	Change unit description file. A flat file containing the change date, in the form DATE: YYYY-MM-DD HH24:MI:SS, and the files to be deleted, in the form DELETED:filename. For example: DATE: 2010-03-15 17:12:13 DELETED:MYPROG.CBL DELETED:MYPROG.CPY

QualityAssessment.bj

Action

Generates a Quality Assessment report in the specified folder.

Syntax

```
QualityAssessment LogFile Workspace [Folder] [Project]
```


Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Folder	Output directory for the report.

Optional Parameters	Description
Project	Project. Default is the default project.

ReferenceReport.bj

Action

Generate Reference Reports. The reports identify missing or unneeded files or objects in the workspace:

- An *Unresolved Report* identifies missing application elements.
- An *Unreferred Report* identifies unreferenced application elements.
- A *Cross-reference Report* identifies all application references.
- An *External Reference Report* identifies references in object-oriented applications to external files that are not registered in the workspace, such as .java, Java Archive (JAR), or C++ include files (assuming you have identified the locations of these files in the Workspace Verification options window for the source files). These references are not reported as unresolved in the Unresolved Report.

Syntax

```
ReferenceReport LogFile Workspace Type File [Entities] [Restrict] [Project]
```


Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Type	The type of report, Unresolved, Unreferred, CrossRef, or ExternalRef.
File	Output file. The format of the report depends on the extension. Supported extensions are .html, .htm, .xls, .rtf, .doc, .txt, and .csv.

Optional Parameters	Description
Entities	* or a comma-separated list of entity types to report on. Default is *.
Restrict	Whether to restrict references to the specified project, Yes or No. Default is Yes.
Project	Project. Default is the default project.

Refresh.bj

Action

Register and verify new source files, refresh and verify updated source files.

 **Note:** The Refresh2.bj variant also autoresolves decisions.

Syntax

```
Refresh LogFile Workspace StageDir [Project] [Notify] [Detailed] [Drop]
[LaunchHHC] [ExtraHHC] [StopHHC] [Wait]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
StageDir	Staging directory for incoming source files.

Optional Parameters	Description
Project	Project. Default is the default project.
Notify	Notification file.
Detailed	Log file.

Oracle Only Parameters	Description
Drop	<p>Whether to drop repository indexes. Specify:</p> <ul style="list-style-type: none"> • Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. • Yes, to drop repository indexes. • No, to not drop repository indexes. <p>Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.</p>
LaunchHHC	Whether to launch the Queue Processor, Yes or No. Launching the Queue Processor generally improves performance. Default is No.
ExtraHHC	If LaunchHHC is specified, the number of additional Queue Processors to launch.
StopHHC	Whether to stop the Queue Processor(s) when processing is complete, Yes or No.
Wait	<p>Whether to wait until the Queue Processor(s) queue is empty. Specify:</p> <ul style="list-style-type: none"> • Yes, to wait indefinitely. • No, to not wait. • The number of seconds to wait for the count on the queue to change. If the count does not change within the specified time, BRP resumes. Sixty minutes is recommended. 0 means no timeout.

Register.bj

Action

Register new source files, refresh updated source files. Use:

- AddNew.bj to register new source files only.
- UpdateOnly.bj to refresh updated source files only.
- Refresh.bj to register and verify new and updated source files.
- Verify.bj to verify registered source files.

Syntax

```
Register LogFile Workspace StageDir [Project] [Entity] [Drop]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
StageDir	Staging directory for incoming source files.

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.
Project	Project. Default is the default project.

Oracle Only Parameters	Description
Drop	Whether to drop repository indexes. Specify: <ul style="list-style-type: none">• Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed.• Yes, to drop repository indexes.• No, to not drop repository indexes. Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.

Related.bj

Action


Create a *project control file (PCF)* based on the relationships between source files. The source file on the left side of the relationship is called the *startup object*. The source file on the right side of the relationship is called the *target object*.

A project control file identifies the projects to which source files belong. Use ApplyPCF.bj or SetProject .bjto assign source files to projects based on a project control file.

Syntax

Related *LogFile* *Workspace* *Out* [*List*] [*Project*] [*Startup*] [*Target*] [*Include*]

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
Out	Project control file (.pcf).

Optional Parameters	Description
List	<p>A control file with a list of startup objects. Each line of the control file contains the following information:</p> <pre>"EntityType" "EntityName"</pre> <p>where:</p> <ul style="list-style-type: none"> • <i>EntityType</i> is the entity type of the startup object, COBOL, for example. • <i>EntityName</i> is the name of the startup object, DayOfWeek.cbl, for example.
Project	When <i>List</i> is not specified, the project containing the startup objects.
Startup	<p>When <i>List</i> is not specified, the entity type of the startup objects. Enclose multiple entity types in parentheses in a vertical-line-separated list, for example:</p> <pre>(COBOL COPYBOOK BMS)</pre> <p>Use the flag attribute of an entity type to specify all entity types with that flag, for example:</p> <pre>*LEGACY</pre> <p>which specifies all entity types with the LEGACY flag, the default. For more on entity flags, see <i>Software Development Toolkit</i>, available from support services.</p> <p> Note: Additional operators are available for special needs. Contact support services for details.</p>
Target	The entity type of the target objects. The notation is as for <i>Startup</i> . Default is (BMS PSB DBD CSD).
Include	<p>Whether to include source files related to other source files in relationships flagged R_USE, such as Cobol Includes Copybook File. Default is Yes. Specify NONE for No.</p> <p>Restrict the result to source files of given types by specifying the types, for example:</p> <pre>Include=(COBOL JCL)</pre> <p>The notation is as for <i>Startup</i>. For more on relationship flags, see <i>Software Development Toolkit</i>, available from support services.</p>

ResolveDecisions.bj

Action

Resolve decisions automatically.

Syntax

```
ResolveDecisions LogFile Workspace [Project] [Notify] [Drop] [LaunchHHC]  
[ExtraHHC] [StopHHC] [Wait]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Project	Project. Default is the default project.
Notify	Notification file.

Oracle Only Parameters	Description
Drop	Whether to drop repository indexes. Specify: <ul style="list-style-type: none">• Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed.• Yes, to drop repository indexes.• No, to not drop repository indexes. Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.
LaunchHHC	Whether to launch the Queue Processor, Yes or No. Launching the Queue Processor generally improves performance. Default is No.
ExtraHHC	If LaunchHHC is specified, the number of additional Queue Processors to launch.
StopHHC	Whether to stop the Queue Processor(s) when processing is complete, Yes or No.
Wait	Whether to wait until the Queue Processor(s) queue is empty. Specify: <ul style="list-style-type: none">• Yes, to wait indefinitely.• No, to not wait.• The number of seconds to wait for the count on the queue to change. If the count does not change within the specified time, BRP resumes. Sixty minutes is recommended. 0 means no timeout.

RestoreDecisions.bj

Action

Restore resolved decisions. Reverifying a file invalidates resolved decisions. Use RestoreDecisions.bj with a *decisions control file (DCF)* to restore resolved decisions. Use SaveDecisions.bj to create a decisions control file before reverifying.

Syntax

```
RestoreDecisions LogFile Workspace DecisionsCF
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
DecisionsCF	Decisions control file (DCF).

RXP.bj

Action

Execute a Repository Exchange Protocol (RXP) query. RXP is an XML-based API that you can use to interact with application-level information in the workspace repository. For more information, see *Analyzing Projects* in the product documentation set.

Syntax

```
RXP LogFile Workspace RXP [Query] [Output] [Project]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
RXP	File that contains RXP queries.

Optional Parameters	Description
Query	* or name of query to execute. Default is *.
Output	Output file. The format of the file depends on the extension. Supported extensions are .html, .htm, .xml, .xls, .rtf, .doc, .txt, and .csv.
Project	Project. Default is the default project.

SaveDecisions.bj

Action

Create a *decisions control file (DCF)* for a workspace. A decisions control file identifies the decisions in the workspace and the objects they have been resolved to. After reverification (which invalidates decisions), use RestoreDecisions.bj to restore the resolved decisions to the workspace.

Syntax

```
SaveDecisions LogFile Workspace DecisionsCF [Decisions] [Rels]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
DecisionsCF	Output file (.txt).

Optional Parameters	Description
Decisions	Type of decisions to include. Specify: <ul style="list-style-type: none">All, to include all decision types.Uncompleted, to include uncompleted decisions.Unresolved, to include unresolved decisions. Default is Unresolved.
Rels	Whether to include relationships in the DCF, Yes or No. Default is No.

SetChangeDate.bj

Action

Set a change date for complexity metrics other than the date source files are verified. Specify the date in the date parameter or in a *change unit description file*. Run the script without the date or file parameters to clear the change date from the system.



Note: Execute this script *before* registering the source files.

Use ProcessChangeUnit.bj to set a change date and delete source files from the workspace. Use ChangeTraffic.bj to generate *change traffic metrics*.

Syntax

```
SetChangeDate LogFile Workspace [Date] [File]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Date	Change date in the form YYYY-MM-DD HH24:MI:SS. For example, 2010-03-15 17:12:13.
File	Change unit description file. A flat file containing the change date, in the form DATE: YYYY-MM-DD HH24:MI:SS. For example: DATE: 2010-03-15 17:12:13

SetProject.bj

Action

Assign source files to projects based on a *project control file (PCF)*. A project control file identifies the projects to which source files belong. It specifies the path to the source files rather than just their names so as to avoid issues resulting from having files with homonymous names.

SetProject.bj differs from ApplyPCF.bj in that it allows you to assign source files to projects additively, without deleting their links to existing projects. Use CreatePCF.bj or Related.bj to create a project control file.

Syntax

```
SetProject LogFile Workspace ProjectCF [Incremental]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
ProjectCF	Project control file (.pcf).

Optional Parameters	Description
Incremental	Assign source files to projects additively, without deleting their links to existing projects. Use the argument with no value to specify additive assignment. Omit the argument to specify overwrite assignment.

TagCmd.bj


Action

Create, delete, and rename tags. Create and delete references between tags.

Syntax

```
TagCmd LogFile Workspace Operation Tags
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Required Parameters	Description
Operation	<p>Operation. Specify:</p> <ul style="list-style-type: none"> • create, to create tags. • delete, to delete tags. • rename, to rename tags. • link, to create references between tags. • unlink, to delete references between tags.
Tags	<p>Names of tags. For the create and delete operations, use a comma (,) to separate multiple tags. For the rename, link, and unlink operations, use a vertical bar () to separate paired tags, and a comma to separate multiple pairs.</p> <p> Note: Tag names are case-sensitive. Names containing commas or vertical bars may produce unexpected behavior.</p>

Unregister.bj

Action

Unregister source files.

Syntax

```
Unregister LogFile Workspace [Entity] [Cond] [ObjList] [Detailed] [Drop]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.
Cond	Source files to unregister. Use the Repository Exchange Protocol (RXP) to code the condition. For more information, see <i>Analyzing Projects</i> in the product documentation set.
ObjList	<p>When Cond is not set, a control file with a list of source files to unregister. Each line of the control file contains the following information:</p> <pre>EntityType EntityName</pre> <p>where:</p> <ul style="list-style-type: none"> • <i>EntityType</i> is the entity type of the source file to unregister, COBOL, for example. • <i>EntityName</i> is the name of the source file to unregister, DayOfWeek.cbl, for example.
Detailed	Log file.

Oracle Only Parameters	Description
Drop	<p>Whether to drop repository indexes. Specify:</p> <ul style="list-style-type: none"> • Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. • Yes, to drop repository indexes. • No, to not drop repository indexes. <p>Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.</p>

UpdateOnly.bj

Action

Refresh updated source files only, optionally based on a project control file (PCF). A project control file identifies the projects to which source files belong. Use:

- CreatePCF.bj or Related.bj to create a project control file.
- Register.bj to register new source files and refresh updated source files.
- Refresh.bj to register and verify new and updated source files.
- Verify.bj to verify registered source files.

Syntax

```
UpdateOnly LogFile Workspace StageDir [ProjectCF] [Notify] [Drop]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).
StageDir	Staging directory for incoming source files.

Optional Parameters	Description
ProjectCF	Project control file (.pcf).
Notify	Notification file.

Oracle Only Parameters	Description
Drop	<p>Whether to drop repository indexes. Specify:</p> <ul style="list-style-type: none"> • Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. • Yes, to drop repository indexes. • No, to not drop repository indexes. <p>Dropping repository indexes generally improves performance when a large number of files need to be</p>

Oracle Only Parameters	Description
	processed. Dropped indexes are restored when processing is complete. Default is No.

UpdateTrendingSnapshot.bj

Action

Update data for Enterprise View trending charts. EV trending charts display complexity metrics data for workspace objects per monthly period. To update the database table for trending data, run UpdateTrendingSnapshot.bj:

- After verifying a workspace for the first time.
- After reverifying updated objects in a workspace.

Syntax

```
UpdateTrendingSnapshot LogFile Workspace
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Upgrade.bj

Action

Upgrade a workspace. Upgrading a workspace synchronizes the workspace with a new CA configuration.

Syntax

```
Upgrade LogFile Workspace
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Verify.bj

Action

Verify registered source files. Use Refresh to register and verify new and updated source files.

Syntax

```
Verify LogFile Workspace [Entity] [Status] [Cond] [Project] [Notify] [Drop]
[LaunchHHC] [ExtraHHC] [StopHHC] [Wait]
```

Required Parameters	Description
LogFile	Output log file generated when the batch script is executed.
Workspace	Workspace file (.rwp).

Optional Parameters	Description
Entity	* or entity type of source files. Default is *.
Status	Verification status of source files to verify. Specify: <ul style="list-style-type: none"> • *, to verify all source files. • !, to verify unverified source files. • R, to verify source files verified under the relaxed parsing option. • E, to verify source files that verified with an error. • S, to verify source files that verified successfully.
Cond	Source files to verify. Use the Repository Exchange Protocol (RXP) to code the condition. For more information, see <i>Analyzing Projects</i> in the product documentation set.
Project	Project. Default is the default project.
Notify	Notification file.

Oracle Only Parameters	Description
Drop	Whether to drop repository indexes. Specify: <ul style="list-style-type: none"> • Auto, to let the script determine whether to drop repository indexes based on the number of files to be processed. • Yes, to drop repository indexes. • No, to not drop repository indexes. Dropping repository indexes generally improves performance when a large number of files need to be processed. Dropped indexes are restored when processing is complete. Default is No.
LaunchHHC	Whether to launch the Queue Processor, Yes or No. Launching the Queue Processor generally improves performance. Default is No.
ExtraHHC	If LaunchHHC is specified, the number of additional Queue Processors to launch.
StopHHC	Whether to stop the Queue Processor(s) when processing is complete, Yes or No.
Wait	Whether to wait until the Queue Processor(s) queue is empty. Specify: <ul style="list-style-type: none"> • Yes, to wait indefinitely. • No, to not wait. • The number of seconds to wait for the count on the queue to change. If the count does not change within

Oracle Only Parameters	Description
	the specified time, BRP resumes. Sixty minutes is recommended. 0 means no timeout.

Using Architecture Modeler

Introducing Architecture Modeler

Use Architecture Modeler to add support for unsupported languages and frameworks to COBOL Analyzer (CA). You can also use it to extend support for already supported languages and frameworks. Architecture Modeler's:

- Graphical user interface, with error checking and metamodel lookup, makes it easy to define new entity and relationship types in CA.
- Regular expression generator simplifies the task of specifying the search patterns the CA parser uses to extract entities and relationships from source code.

Based on your input, Architecture Modeler creates a plug-in that defines an extension, or *add-on*, to the CA metamodel. The add-on specifies the entity and relationship types of interest in the newly supported source files.

The plug-in also references a *universal parser* configuration file that defines the search patterns CA uses to extract entities and relationships from newly supported files. For text files, search patterns are mapped from regular expressions. For XML files, search patterns are mapped from XPath queries.

Here in schematic form are the tasks you perform in Architecture Modeler:

1. Load an existing metamodel.
2. Save the metamodel with the name of your extension.
3. Add a sample source file of the type you want to support in CA.
4. Define the entity types for the add-on.
5. Define the relationship types for the add-on.
6. Map search patterns for the entity and relationship instances you want the parser to extract.
7. Export the model to CA.
8. Reconfigure CA.

Before turning to these tasks, users who are new to CA may want to familiarize themselves with the concepts underlying the CA metamodel. Experienced users can skip straight to the tasks.



Note: Architecture Modeler currently supports the level-1, or *application-level*, metamodel only. Support for the level-2, or *object-level*, metamodel (Interactive Analysis) will be available in a future release.

Entity extension mode is currently not available for the following entities: EZT, CA7, JSP, JSX, NetronSPC, NetronSpec, TLD, TS, and WebConfig.



Important: If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise some operations might fail.

Opening Architecture Modeler

1. Open the COBOL Analyzer installation directory
2. Double-click `\bin\Architecture Modeler.exe`. The Architecture Modeler window opens.



Note:

- Choose **Tools > Confirmations** to specify whether you want to be prompted to confirm deletion of attributes, entities, relationships, or source files.
- Choose **Tools > Samples view font** to specify the font in which source file samples are displayed.

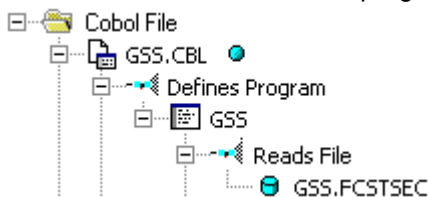


Important: If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise some operations might fail.

Understanding the Application-Level Metamodel

The *object model* for an application defines the relationships between the objects that comprise the application. These can be physical objects, like program source files or JCLs, or logical objects that identify abstract program elements: entry points, data stores, screens, jobs, and so forth.

The *relationships* between objects describe the ways in which they interact. In the figure below, the source file GSS.CBL defines the GSS program. The program, in turn, reads the data file GSS.FCSTSEC.



Relationships are conceived as having a left and right end. In the relationship CobolDefinesMap, for example, the left relationship end is DefinesMap, while the right relationship end is IsDefinedInCobol.

Each object or relationship the parser generates is modeled as an entity or relationship in an *Entity Relationship Diagram (ERD)*. Each entity or relationship is an instance of an entity or relationship *type*. The *application-level metamodel* defines these types.

Entity Type Properties

The properties of an entity type define its characteristics: its internal and external names, icon in COBOL Analyzer, flags, and so on. The table below shows the properties of an entity type.



Note: Only properties editable in Architecture Modeler are shown in the table.

Property	Description
Name	The internal name of the entity type: COBOL, for example.
Description	The display name of the entity type: Cobol File, for example.
Source Name	The name of the entity attribute that holds “reference-resolving” information for an entity: Source, for example. See the subtopics below for more information.
Display Name	The name of the entity attribute that holds the display name for the entity. A Java File’s display name derives from the value of its ClassName attribute, for example.
Flags	Flags that define the nature of the entity type: LEGACY, for example. See the subtopics below for more information.
Registration	The name of the class used to perform registration, unregistration, and invalidation tasks. Specify Default, unless the entity type should not have a folder in the project, in which case, specify NoRegistration.
Source Type	The type of “reference-resolving” information for the entity: File, for example. See the subtopics below for more information.

Property	Description
Icon	The name of the file that contains the entity type's icon for display: COBOL.gif, for example.

Source Name Property

The Source Name property of an entity type defines the name of the entity attribute that holds “reference-resolving” information for an entity. This attribute name is not the same for all entity types.

Consider a copybook. If a Cobol source file references a copybook, the parser creates a copybook object with the name referenced in the source file. And it does so whether or not the copybook actually exists in the CA workspace.

Later, when the system resolves references to the copybook, it looks in the Source attribute of the copybook for its location in the workspace file system. If the attribute is empty, the system flags the copybook as missing. The system knows it should look in the Source attribute because the Source Name property of the copybook entity type is set to “Source.”

For a program entry point, by contrast, the sourcename property is set to “HCID,” meaning that the system looks in the HCID (HyperCode ID) attribute when it attempts to resolve a CALL statement that references the entry point. An empty HCID attribute indicates that the called program does not exist in the workspace or has not been parsed.

Source Type Property

The Source Type property of an entity type defines the type of reference-resolving information for the entity: a file name in the case of a copybook, for example, a record in a table in the case of an entry point. The Source Type also determines whether CA deletes the reference-resolving item when the corresponding object is deleted from the workspace. The table below shows the types.

Type	Description
File	A source file: a copybook, for example. If the object is deleted, the source file is deleted.
Container	A file containing a list of source files: a TARGETMODEL, for example. If the object is deleted, the list file is deleted.
Folder	The folder for the Folder entity, which identifies an CA project. If the project is deleted, the folder is deleted.
Global	An item in a global file: a record in a table, for example. If the object is deleted, the item is not deleted.

Entity Flags

The Flags property of an entity type defines the nature of the entity: whether it is a physical source object or a logical object extracted from a source object, whether it defines a program, and so forth. An entity type can have multiple flags. A Cobol source file, for example, has LEGACY, PROGRAMCODE, and SEED flags. The table below shows the flags for an entity type.


 **Note:** Only flags selectable in Architecture Modeler are shown in the table.

Flag	Description
COMPOSITE	n/a
EXTRACT	An entity extracted from a seed entity. A program, for example, is extracted from a Cobol file. An extract entity must be unique in the repository.

Flag	Description
GENERATED	An entity generated in the transformation process, a TargetXML file, for example.
LEGACY	A source file that can be loaded in the repository, a Cobol file, for example.
KNOWLEDGE	A business function or logical component.
PROGRAMCODE	An entity that defines a program, a Cobol file, for example.
SEED	An entity from which another entity can be extracted. A Cobol file is the seed for a program, for example.
SYSTEM	An entity referenced in the application but not loaded in the repository, a data store, for example.

Entity Type Attributes

The attributes of an entity type define the characteristics of the entity instance: its name, the name of the source file from which it derives, its complexity metrics, and so forth. The default attributes and attribute values supplied by Architecture Modeler are sufficient for most entity types. You can add or delete attributes and edit their values as necessary.

 **Note:** You can specify additional default attributes in the file `\bin\ArchitectureModeler.exe.config`.

The properties of an attribute define its characteristics: its name, data type, whether the attribute value persists after invalidation, and so on. The table below shows the properties of an attribute.

Property	Description
Name	The internal name of the attribute: Name, for example.
Description	The display name of the attribute: Name, for example.
Type	For attributes other than Name, the data type of the attribute: CHAR, for example.
Size	For CHAR type attributes, the length of the attribute.
DefaultValue	Whether the attribute should be displayed with a value of "N/A" until it receives a value at verification: 0 for true, -1 for false.
Persistent	For an attribute of a LEGACY object, whether its value persists after the object is invalidated.

Relationship Type Properties

The properties of a relationship type define its characteristics: its name, flags, cardinalities, and so on. The table below shows the properties of a relationship type.

Property	Description
Name	The full relationship name: CobolDefinesMap, for example.
Flags	Flags that define the nature of the relationship type: PRODUCE, for example. See the subtopics below for more information.
Left Entity	The internal name of the entity on the left relationship end, COBOL, for example.
Left End Name	The internal name of the left relationship end: DefinesMap, for example.
Left End Description	The display name of the left relationship end: Defines Screen, for example.

Property	Description
Left End Cardinality	The cardinality of the left relationship, 1 or M. 1 means that the left entity can have one such relationship; M means that the left entity can have multiple such relationships. A Cobol file can define only one map, for example.
Right Entity	The internal name of the entity on the right relationship end, MAP, for example.
Right End Name	The internal name of the right relationship end: IsDefinedInCobol, for example.
Right End Description	The display name of the right relationship end: Is Defined In Cobol File, for example.
Right End Cardinality	The cardinality of the right relationship, 1 or M. 1 means that the right entity can have one such relationship; M means that the right entity can have multiple such relationships. A map can be defined in multiple Cobol files, for example.

Relationship Flags

The Flags property of a relationship type defines the nature of the relationship: whether the relationship is one between a seed and an extract entity, for example, or between extract entities or legacy entities. A relationship can have multiple flags. The FolderIncludesDbSchema relationship, for example, has the flags GROUPING and PRODUCES.

The relationship flag also determines how the product behaves when one of the entities in the relationship is invalidated, modified, or deleted. In a relationship between a seed and an extract entity, for example, deleting the seed requires deleting the extract. The table below shows the flags for a relationship type.

Flag	Description
GROUPING	A relationship between a folder entity, which identifies an CA project, and a legacy entity, FolderIncludesCobol, for example.
GENERATES	A relationship between a legacy entity and a generated entity, CopybookGeneratesTargetXml, for example. If the legacy entity is modified or deleted, the generated entity is deleted.
LINKS	Future use.
PRODUCES	A relationship between a seed and an extract entity, CobolDefinesProgram, for example. If the seed entity is modified or deleted, the extract entity is deleted.
REFERS	A relationship between extract entities, ProgramCallsProgramEntry, for example. If the left entity is invalidated or deleted, the right entity is deleted.
USES	A relationship between legacy entities, CobolIncludesCopybook, for example. If the right entity is modified or deleted, the left entity is invalidated, and every entity with a relationship of type PRODUCES with the left entity is deleted.

Defining an Extension with Architecture Modeler

Perform the following tasks to define a metamodel extension with Architecture Modeler:

1. Load an existing metamodel.
2. Save the metamodel with the name of your extension.
3. Add a sample source file of the type you want to support in CA.

4. Define the entity types for the add-on.
5. Define the relationship types for the add-on.
6. Map search patterns for the entity and relationship instances you want the parser to extract.
7. Export the model to CA.
8. Reconfigure CA.

Before turning to these tasks, users who are new to CA may want to familiarize themselves with the concepts underlying the CA metamodel. Experienced users can skip straight to the tasks.

Loading a Metamodel

You can load an existing metamodel to use as a shell for your extension, or create a new metamodel. Loading an existing metamodel is probably a better choice, because it ensures against duplicating existing entity and relationship names.

- To load an existing metamodel, choose **Model > Open**. An Open dialog is displayed, where you can browse for `\Model\Repository\Common.Repstry.xml` in the COBOL Analyzer installation directory. Click **Open** to load the metamodel.
- To create a new metamodel, choose **Model > New > Model**. Nothing is displayed in the **Architecture Modeler** window until you add files, entities, or relationships.

Saving the Extended Metamodel

Whether you load an existing metamodel or create a new one, you must save the extended metamodel to preserve your changes. To save an extended metamodel, choose **Model > Save**. A Save As dialog opens, where you can save the extension with a new or an existing file name. Click **Save** to return to the Architecture Modeler window.

Adding a Sample Source File

When you add a sample source file to Architecture Modeler, you define the entity type for the source file in the extended metamodel, and make available the code you will use to generate regular expressions. To add support for Objective C, for example, you would define an entity type that represents an Objective C source file, then add one or more Objective C source files to Architecture Modeler.

Choose a sample that contains the kinds of entities and relationships you want the parser to extract. If you want the parser to model calls to Objective C functions, make sure your sample contains a function call!



Note: Architecture Modeler supplies default values for the properties and attributes of the source file type you are adding. Except for the value of the Description property, which holds the entity type name displayed to the end user, the default values usually are sufficient. You can change the Description (and any other property or attribute values) when you edit the source file type.

1. Click the **Files** button in Architecture Modeler and choose **Add New** in the drop-down menu. The Add New File window opens.
2. In the **Name** field, enter the name of the entity type for the source file. For an Objective C source file, you might choose the name "OBJC" for the entity type.



Note: If you are extending support for an already supported source file, the name you specify cannot be the same as the name of the already supported source file. No entity type is created for the specified name. The extension is simply a mechanism for naming the universal parser configuration file that contains the search patterns with which you are extending support.

3. In the **Type** drop-down, select the type of the source file you are adding, XML or Text. If you select Text, click the **Format** button to view formatting options.



Note: Think of Text as any kind of file other than XML.

4. Select **as an extension to** if you are extending support for an already supported source file, then choose the source file type in the adjacent combo box.



Note: The Metrics and Name Pattern areas are disabled in entity extension mode.

5. In the Metrics group box, select each metric you want CA to calculate for the source file.
6. In the Source samples group box, click **Add new**. An Open dialog is displayed, where you can browse for the sample source file. Click **Open** to add the file. Repeat this procedure for each sample you want to add. To delete a file, select it and click **Delete**.
7. In the Name Patterns group box, click **Add new**. An editable text field is displayed. Click inside the field and enter the extension, including the dot (.), for the source file type. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). Click outside the field to add the extension. Repeat this procedure for each extension you want to add. To delete an extension, select it and click **Delete**.
8. When you are satisfied with your choices, click **OK**.



Architecture Modeler displays the new source file type in the Name area in the central pane of the window, and the text of the sample source file in the righthand pane. If more than one source file type is listed, click the name of the type to display the corresponding sample.





Note: Select a source file type and click **Files > Edit** to modify its file characteristics. Select a source file type and click **Files > Delete** to delete the type.

Specifying Formatting Options for Text Files

Comments, strings, and fixed formats may create "noise" in parser extractions. The more specific you are in defining how you want the parser to handle these and similar text file characteristics, the cleaner your results will be.

1. To specify formatting options for a text file, click the **Format** button in the Add New File or Edit File window. The Input Source Format window opens.
2. In the Input Source Format window, click the button  to sort the list by category, click the  button to sort the list alphabetically.
3. Edit the formatting options as necessary:
 - For Boolean values, choose from the drop-down list inside the value field.
 - For string values, enter the text of the string in the value field.
 - For the StringLiteralSymbol option, click the browse button in the value field. The Char Collection Editor window opens. Click the **Add** button to add a symbol for the start of a string literal. Architecture Modeler displays the new collection member in the lefthand pane. Select the member, then enter its value in the righthand pane. Repeat this procedure for each symbol you want to define.

To delete a collection member, select it and click **Remove**. Click the button  to sort the list by category, click the  button to sort the list alphabetically. Click **OK** to confirm your changes.

The table below shows the formatting options in alphabetical order. Option values are modified automatically depending on your choices in other fields.

Option	Description
EndingColumn	The last valid column of the source.
HasEndingColumn	Whether the source has an ending column.
HasLiteralContinuationSymbol	Whether the source has a literal continuation symbol.
HasMultiLineComment	Whether the source has multiline comments.
HasSingleLineComment	Whether the source has single line comments.
HasSingleLineCommentContinuation	Whether the source has single line comment continuations.

Option	Description
HasStartingColumn	Whether the source has a starting column.
HasStringLiteralSymbol	Whether the source has symbols used to start string literals.
IgnoreColumns	Whether to ignore column information before the starting column and after the ending column.
IgnoreComments	Whether to ignore comments.
IgnoreStringLiterals	Whether to ignore string literals.
MultiLineCommentEnd	Symbol for the end of a multiline comment.
MultiLineCommentStart	Symbol for the start of a multiline comment.
SingleLineCommentContinuationSymbol	Symbol for the continuation of a single line comment.
SingleLineCommentStart	Symbol for the start of a single line comment.
StartingColumn	The first valid column of the source.
StringLiteralContinuationSymbol	Symbol for the continuation of a string literal.
StringLiteralSymbol	Symbols for the start of string literals.


4. When you are satisfied with your choices in the Input Source Format window, click **OK**.

Defining Entity Types

Each object the parser generates is an instance of an entity *type*. You need to define these types in the extended metamodel. If you are adding support for Objective C, for example, you would define an entity type that represents an Objective C function.




For each entity type, you specify two kinds of information:

- The *properties* of the entity type define its characteristics: its internal and external names, icon in COBOL Analyzer, flags, and so on.
- The *attributes* of the entity type define the characteristics of the entity instance: its name, the name of the source file from which it derives, its complexity metrics, and so forth. The default attributes and attribute values supplied by Architecture Modeler are sufficient for most entity types. You can add or delete attributes and edit their values as necessary.



1. Click the **Entities** button in Architecture Modeler and choose **Add New** in the drop-down menu. The properties definition page of the Add New Entity window opens.
2. On the properties definition page, fill in the properties of the entity type. Architecture Modeler flags any errors with a  symbol.



Note: Do not enter spaces in the Name property.

3. When you are satisfied with your entries on the properties definition page, click the  button. The attributes definition page opens.
4. On the attributes definition page, add, edit, or delete attributes as necessary:
 - To add an attribute, click the **Add New** button. The Add New Attribute dialog opens, where you can specify the values for each property of the attribute. Architecture Modeler flags any errors with a  symbol. When you are satisfied with your entries, click **OK**.
 - To edit an attribute, click the **Edit** button. The Edit Attribute dialog opens, where you can specify the values for each property of the attribute. Architecture Modeler flags any errors with a  symbol. When you are satisfied with your entries, click **OK**.
 - To delete an attribute, click the **Delete** button.

5.

When you are satisfied with your entries on the attributes definition page, click the  button. Architecture Modeler flags any errors with a  symbol.



Note: The errors may not be immediately visible. If you are on the attributes definition page, errors may be flagged on the properties definition page, and vice versa.


Assuming there are no errors in your entries, Architecture Modeler saves the entity type definition, and displays the entity type in the list of entity types in the righthand pane of the window.



Note: Select an entity type and click **Entities > Edit** to modify its definition. Select an entity type and click **Entities > Delete** to delete the definition.

Defining Relationship Types

Each relationship the parser generates is an instance of a relationship *type*. You need to define these types in the extended metamodel. If you are adding support for Objective C, for example, you would define a relationship type that represents the relationship between an Objective C source file and an Objective C function.

1. Click the **Relations** button in Architecture Modeler and choose **Add New** in the drop-down menu. The Add New Relation window opens.
2. In the Add New Relation window, fill in the properties of the relationship type. Architecture Modeler flags any errors with a  symbol.



Note: Do not enter spaces in the Name property.

3. When you are satisfied with your entries, click **OK**.

Assuming there are no errors in your entries, Architecture Modeler saves the relationship type definition, and displays the relationship type in the list of relationship types in the righthand pane of the window.



Note: Select a relationship type and click **Relations > Edit** to modify its definition. Select a relationship type and click **Relations > Delete** to delete the definition.

Architecture Modeler Internal Language Functions Description

string.substr(pos1,pos2)

`string.substr(pos1,pos2)` – returns a substring, derived from `pos1` to `pos2` of `string`.

Examples:

```
expressionid.0.substr ( 5, 4 )
expressionid.0.substr(expressionid.0.length() - 5, 4 )
expressionid.0.substr( 4 , expressionid.0.length() - 5 )
expressionid.0.substr(expressionid.0.length() - 9 , expressionid.0.length() - 5 )
expressionid.sourcefilename.substr(1,5)
expressionid.sourcefilename.substr(expressionid.0.sourcefilename.length() - 11 , 3 )
expressionid.sourcefilename.substr( 0 , expressionid.0.sourcefilename.length() - 4 )
expressionid.sourcefilename.substr(expressionid.0.sourcefilename.length() - 15 )
```

string1.append(string2)

`string1.append(string2)` - appends `string2` to `string1`.

Examples:

```
expressionid.0.append( \"._abcd_\" )
expressionid.0.append(expressionid.0 )
```

```
expressionid.0.append(expressionid.0.sourcefilename )
expressionid.sourcefilename.append( \"_001\" )
expressionid.sourcefilename.append( expressionid.0 )
```

string.length()

string.length() - returns the length of the string.

Examples:

```
expressionid.sourcefilename.length()
expressionid.sourcefilename.length()-3
expressionid.sourcefilename.length()+13
expressionid.sourcefilename.length( )
```

concat(string1, string2)

concat(string1, string2) – returns concatenation of string1 and string2

Examples:

```
concat(\"ab_\", \"cd__\" )
concat( expressionid.0,expressionid.0)
concat( expressionid.0, \"_test_\" )
concat( \"_test_\", expressionid.0 )
concat( expressionid.sourcefilename , expressionid.0 )
concat(expressionid.0, expressionid.sourcefilename )
concat(expressionid.sourcefilename.substr(0,
expressionid.sourcefilename.length()-3), expressionid.0)
concat(expressionid.sourcefilename.append( \"...\"), expressionid.0)
concat(expressionid.sourcefilename, \"_abc_\" )
concat( \"_abc_\", expressionid.sourcefilename )
concat( \"_abc_\", concat( \"_efg_\", \"_xy_z_\" ) )
concat( concat( \"_efg_\", \"_xy_z_\"), \"_abc_\" )
concat( concat( \"_efg_\", concat(\"_val1_\", \"_val2_\")) , \"_abc_\" )
concat( concat(expressionid.sourcefilename.append( \"...\"), \"_val_\" ) ,
expressionid.0)
```

replace(string1,string2,string3)

replace(string1,string2,string3) – replaces all occurrences of string2 in string1 with string3.

Examples:

```
replace( \"ab_\", \"_\", \"__\" ) // equals \"ab__\"
replace( \"ab_\", \"ab\", \"cd\" ) // equals \"cd_\"
replace(expressionid.sourcefilename.substr(1,5), \"a\", \"b\" )
// replaces 'a' with 'b' in the first 5 letters of expressionid.sourcefilename
replace(expressionid.sourcefilename, '/' . '\\') // converts backslashes to slashes
```

getpath(string)

getpath(string) – returns a string, trimmed starting from the last occurrence of ‘\’ or ‘/’. Returns the absolute path as string before the last occurrence of \ or /.

Examples:

```
getpath(\"c:\path1\path2\filename.ext\") // returns c:\path1\path2\
getpath(\"..\path1\path2\filename.ext\") // returns ..\path1\path2\
getpath(expressionid.sourcefilename) // returns the path of the sourcefilename
```

getfilename(string)

getfilename(string) – .returns the filenames in paths as a string after the last occurrence of the \ or /?

Examples:

```
getfilename ("c:\path1\path2\filename.ext") // returns filename.ext
getfilename ("..\path1\path2\filename.ext") // returns filename.ext
getfilename (expressionid.sourcefilename) // returns filename of the
sourcefilename
```

unifypath(string)

`unifypath(string)` – returns transformed string by the following rules:

- Converting all backslashes to slashes.

Example:

`/path1/path2` is transformed into `\path1\path2`

- Removing duplicate slashes.

Example:

`//path1/\path2` is transformed into `\path1\path2`

- Removing references to current directory

Example:

`\.\path1/.\path2` is transformed into `\path1\path2`

- Removing references to parent directory (if not in the beginning) examples

Examples:

`\path1\..\path2` is transformed into `\path2`

`..\path1\..\path2\path3` is transformed into `..\path2\path3`

Mapping Regular Expressions to Text File Searches

The COBOL Analyzer parser extracts entity and relationship instances from text files using search patterns you define in Architecture Modeler. In schematic form, the parser uses these patterns to:

- Match relationship instances in the code.
- Form the names of entity instances.
- Form the names of relationship instances.

Architecture Modeler's regular expression generator simplifies the task of defining these patterns. Follow the steps below to map regular expressions to text file searches.

1. Click **Files** in Architecture Modeler. Architecture Modeler displays the defined source file types in the Name area in the central pane of the window, and the text of the corresponding sample in the right-hand pane. If more than one source file type is listed, click the name of the type to display the sample.
2. Generate the pattern the parser uses to match relationship instances:
 - a) Select the code of interest in the sample source file.
 - b) Choose **Pick** in the right-click menu.The Add Expression window opens.
3. Add an expression ID, e.g. "function" for the selected function call. All expression IDs appear in the middle-bottom left-hand side of the screen in a tree form.
4. Select the expression ID and choose **Edit** in the right-click menu. The Edit Regular Expression Mapping window opens. Expressions are represented in a hierarchical manner. Each expression contains its regular expression pattern and all direct children expressions.
5. Select the expression pattern and choose **Edit** in the right-click menu. The Edit Regular Expression window opens. The generated regular expression is displayed in the upper part of the left-hand pane in tree form. The expression does not require further editing.



Note: Do not manually edit the generated regular expression unless you have modified it inadvertently.

6. In the left-hand pane of the Edit Regular Expression window, define the regular expression the parser uses to form the names of entity instances:
 - a) Expand the tree for the generated regular expression.
 - b) Use a subexpression of the generated regular expression to match the names of entity instances. Follow the instructions in *Editing Subexpressions* to edit the subexpression.
 - c) Click **Apply Changes** after each modification.
7. In the right-hand pane of the Edit Regular Expression Mapping window, map the regular expression for the names of entity instances. You do not need to map a regular expression for the source file name:
 - a) In the Entity group box, select the entity type you want to match in the Type drop-down.
 - b) Drag-and-drop the subexpression for the names of entity instances from the expression tree to the cell beside the Name attribute. The subexpression is mapped to a variable with a name reflecting the order of grouped expressions in the expression tree, counting from the regular expression for the matched relationship. For example if $(\backslash w+)$ is the only other grouped expression besides the expression for the matched relationship, the mapped variable is named `%[ExpressionID].2%`.
 - c) Assuming you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename` in the cell beside the Source attribute. If you are defining another type of relationship, reference-resolving information may be contained in a different attribute.
 - d) You can use functions to help you resolve entities' relationship ends. These functions are: `substr`, `append`, `length`, `concat`, `replace`, `getpath`, `getfilename`, `unifypath`. For more details on the usage of these functions, read *Application Modeler Internal Language Functions Description*.
8. Click **Save** in the Entity group box. The entity mapping is added to the list of mappings in the Mapping List group box. Repeat this procedure for each entity mapping you want to define.
9. In the right-hand pane of the Edit Regular Expression Mapping window, map the regular expression for the names of relationship instances:
 - a) In the Relation group box, select the relationship type you want to map in the Type drop-down.
 - b) Drag-and-drop the subexpression for the left entity name from the expression tree to the cell beside the Left Entity Name attribute. If you are defining a relationship between a source file and an extracted object, enter the system variable `%[expressionid].sourcefilename%` in the cell beside the Left Entity Name attribute.
 - c) Drag-and-drop the subexpression for the right entity name from the expression tree to the cell beside the Right Entity Name attribute.
10. Click **Save** in the Relation group box. The relationship mapping is added to the list of mappings in the Mapping List group box.



Note: To edit a mapping definition, select it in the Mapping List group box and click **Edit**. Follow the procedure you used to define the mapping. To delete a mapping definition, select it in the Mapping List group box and click **Remove**.

11. When you finish editing your entries in the Edit Regular Expression Mapping window, click **Save**. Architecture Modeler displays the edited expression ID in the central pane of the window.



Note:

- Select the expression ID and choose **Edit** in the right-click menu to modify the pattern.
- Select it and choose **Delete** in the right-click menu to delete the pattern.

Editing Subexpressions

Editing subexpressions consists of two tasks:

- *Grouping and quantifying* subexpressions.
- *Providing Scope for recursive regular expressions*.

The subexpression for the entity you want to match must either be grouped or an exact match. Any other input is not accepted.

1. In the Edit Regular Expression dialog, edit the expression manually, or use the buttons to modify the expression.
 - a) Click **Convert to Exact match** to match the Item Value to the Matched Value sample only.
 - b) Click **Convert to Group** if you want to group the expression.
 - c) Click **Revert Changes** if you want to cancel any last changes or **Apply Changes** to save them.



Note: The subexpression for the entity you want to match must either be grouped or an exact match. Any other input is not accepted.

2. If you need to qualify the expression further, edit the regular expression in the **Item Value** field considering the following:
 - Plus equates to the quantifier +, which denotes one or more occurrence.
 - Star equates to the quantifier *, which denotes 0 or more occurrences.
 - Optional equates to the quantifier ?, which denotes 0 or 1 occurrence.
3. Click **Apply Changes** after each modification.
4. Click **OK** when you are done editing your entries in Edit Regular Expression dialog.

Architecture Modeler displays the edited subexpression in the expression tree.

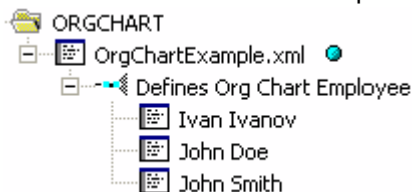
Mapping XPath Queries to XML File Searches

The CA parser extracts entity and relationship instances from XML using search patterns you define in Architecture Modeler. In schematic form, the parser uses these patterns to:

- Match relationship instances in the code.
- Form the names of entity instances:



- Form the names of relationship instances:



Architecture Modeler's XPath query generator simplifies the task of defining these patterns. Follow the steps below to map XPath queries to XML file searches.

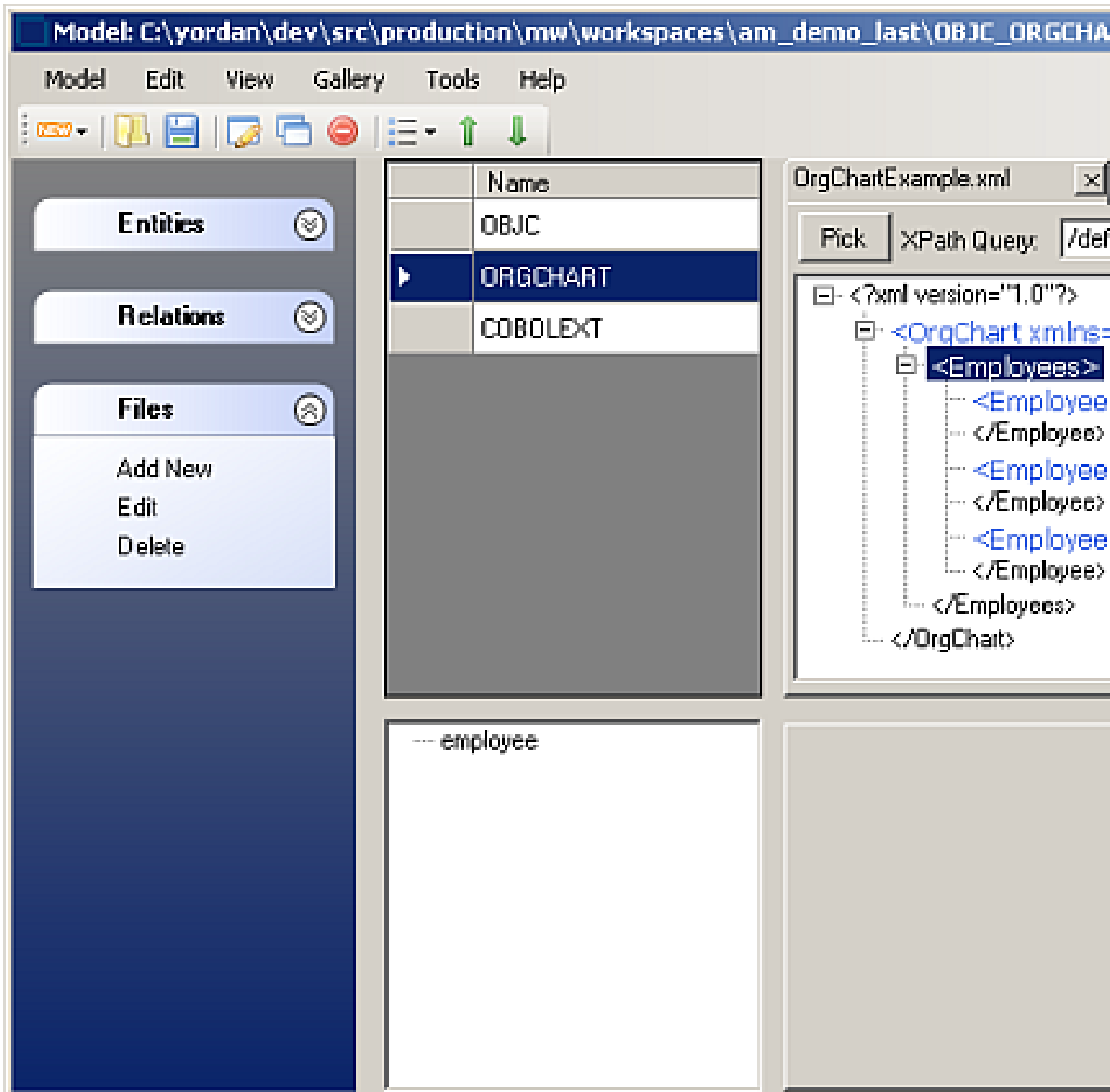
1. Click the **Files** button in Architecture Modeler. Architecture Modeler displays the defined source file types in the Name area in the central pane of the window, and the text of the corresponding sample in the right-hand pane. If more than one source file type is listed, click the name of the type to display the sample.
2. Define the XPath query the parser uses to match relationship instances:

- Select the node of interest in the sample, then use the right-click menu to form the XPath query. To match the relationship OrgChartDefinesOrgChartEmployee in the XML code shown below, the XPath query would be:

```
/def:OrgChart/def:Employees/child::def:Employee
```

To form the query, select the <Employees> node and choose **Select All <Employees> children elements > /child::def:Employee** from the right-click menu.

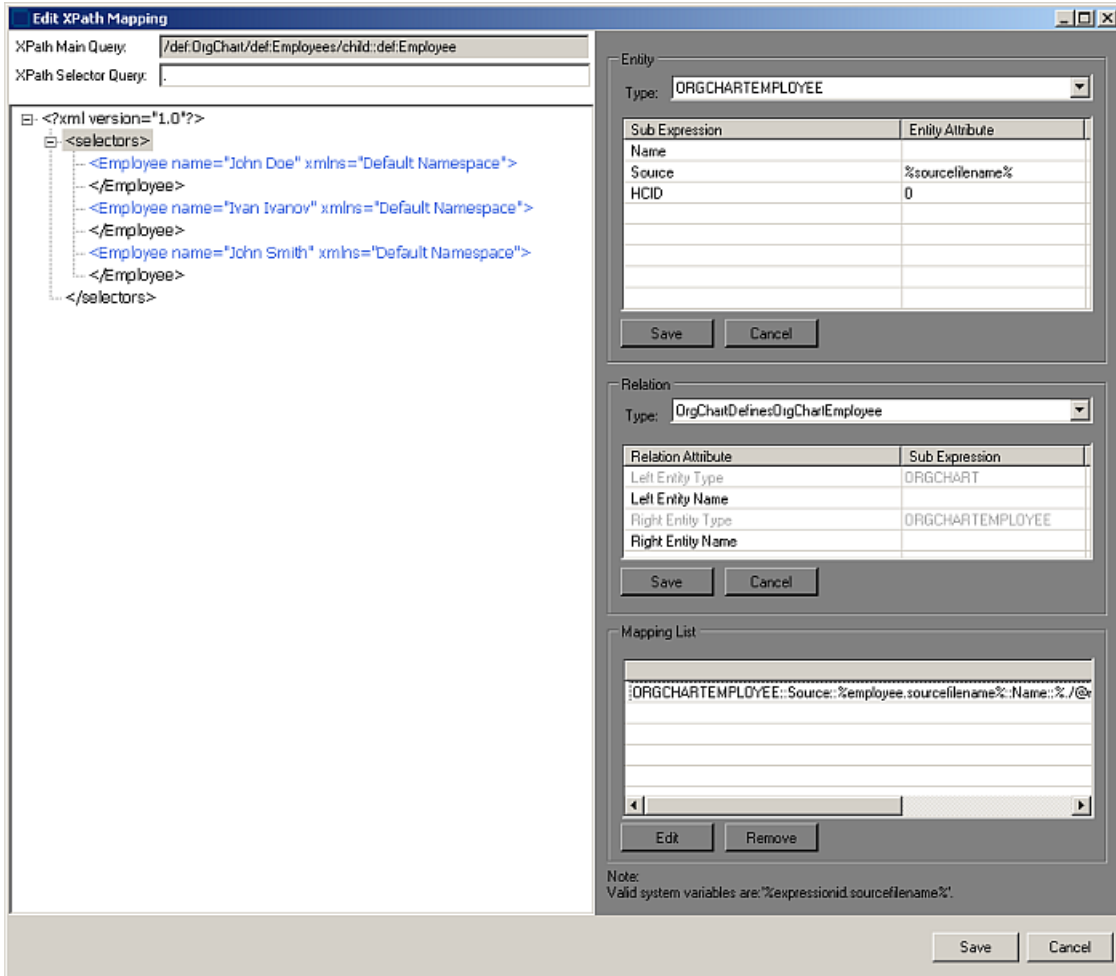
- The XPath query is displayed in the **XPath Query** field. When you are satisfied with the query, click **Pick**.



The Add Expression window opens.

3. Enter an expression ID, e.g. "employee" for the selected Employees xml element. All expression IDs are displayed in a tree on the middle-bottom left-hand side of the screen in.
4. Select the expression ID and choose **Edit** from the right-click menu. The Edit XPath Mapping window opens. The XPath Query for the matched relationship is displayed in the **XPath Main Query** field.
5. In the right-hand pane of the Edit XPath Mapping window, define the XPath query the parser uses to form the names of entity instances. You do not need to define a query for the source file name:
 - In the Entity group box, select the entity type you want to match in the **Type** drop-down, ORGCHARTEMPLOYEE in our example.

- Use a subexpression of the XPath query you created in step 2 to match an attribute of the node. If you want to form the entity name from the Name attribute, enter `./@name%` in the cell beside the Name attribute. That subexpression matches the employees "John Doe," "Ivan Ivanov," and "John Smith" in our example. If you are matching a different attribute, `title`, for example, enter `./@title%` in the cell next to the Name attribute.
 - Assuming you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename%` in the cell beside the Source attribute. If you are defining another type of relationship, reference-resolving information may be contained in a different attribute.
6. Click **Save** in the Entity group box. The entity mapping is added to the list of mappings in the Mapping List group box. Repeat this procedure for each entity mapping you want to define. The figure below shows the entity mapping for our example.



7. In the right-hand pane of the Edit XPath Mapping window, define the XPath query the parser uses to form the names of relationship instances:
- From the **Type** drop-down in the Relation group box, select the relationship type you want to map, `OrgChartDefinesOrgChartEmployee` in our example.
 - Map the subexpression for the left entity name. If you formed the entity name from the `name` attribute of the node, enter `./@name%` in the cell next to the Left Entity Name attribute. If you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename%` in the cell next to the Left Entity Name attribute.
 - Map the subexpression for the right entity name. If you formed the entity name from the `name` attribute of the node, enter `./@name%` in the cell beside the Right Entity Name attribute.
8. Click **Save** in the Relation group box. The relationship mapping is added to the list of mappings in the Mapping List group box. The figure below shows the relationship mapping for our example.



Note: To edit a mapping definition, select it in the Mapping List group box and click **Edit**. Follow the procedure you used to define the mapping. To delete a mapping definition, select it in the Mapping List group box and click **Remove**.

Edit XPath Mapping

XPath Main Query:

XPath Selector Query:

```
[-] <?xml version="1.0"?>
  [-] <selectors>
    -- <Employee name="John Doe" xmlns="Default Namespace">
    -- </Employee>
    -- <Employee name="Ivan Ivanov" xmlns="Default Namespace">
    -- </Employee>
    -- <Employee name="John Smith" xmlns="Default Namespace">
    -- </Employee>
  .. </selectors>
```

Entit

Type

Su

Na

So

HC

Rela

Type

Su

Le

Le

Rig

Rig

Map

OP

OP

←

Note:

Valid s

9. When you are satisfied with your entries in the Edit XPath Mapping window, click **Save**.

Architecture Modeler displays the search pattern for the matched relationship in the Match pattern area in the central pane of the window.



Note: Select the search pattern for the matched relationship and choose **Edit** in the right-click menu to modify the pattern. Select it and choose **Delete** from the right-click menu to delete the pattern.

Exporting the Extended Metamodel

Based on your input, Architecture Modeler creates a plug-in that defines an extension, or *add-on*, to the CA metamodel. The add-on specifies the entity and relationship types of interest in the newly supported source files.

To export the plug-in to CA:

1. Select the source file type for the extension in the **Name** area in the central pane of the Architecture Modeler window.
2. Choose **Export to CA** in the right-click menu.

A Browse dialog opens.

3. Browse for the Plugins folder in the COBOL Analyzer installation directory.
4. Click **OK** to export the extended metamodel. The plug-in has a name of the form `<SourceFileType>Plugin.xml`.



Important: If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise some operations might fail.

Reconfiguring COBOL Analyzer

CA After exporting an extended metamodel to COBOL Analyzer, you need to reconfigure CA for the new source file type. To reconfigure CA after plug-in export,

1. Double-click `\bin\rescan.exe` in the COBOL Analyzer installation directory.

The Configuration Manager window opens, with a check box in the Programming Languages folder for the new extension.

2. Select the check box and click **OK**.

Make sure to upgrade the configuration of any workspaces you want to use the new extension in.

Troubleshooting the Extended Metamodel

The plug-in for an extended metamodel references a *universal parser* configuration file that defines the search patterns CA uses to extract entities and relationships from newly supported files. CA issues the following errors for problems it encounters processing the universal parser configuration file at verification:

- *Error - 404 3 Error while processing XML file '%1'*. Logged when the Universal Parser configuration file is corrupt or missing.
- *Error - 407 3 Error while processing regular expression '%1'*. Logged when a regular expression defined in the Universal Parser configuration file is invalid.
- *Error - 409 3 Input source file '%1' not found*. Logged when the file to be parsed is not found.
- *Error - 411 3 Error creating expression with Id '%1'*. Logged when an expression with a given Id failed to be created.
- *Error - 412 3 Expression Id '%1' already exists*. Logged when an expression Id is not unique.
- *Error - 413 3 Error in attribute value '%1': Unexisting expression Id*. Logged when a non existing expression Id is found in an attribute value.
- *Error - 414 3 Error in attribute value '%1': Match not found*. Logged when a not existing match index is found in an attribute value.
- *Error - 415 3 Error in attribute value '%1': Match group not found*. Logged when a not existing match group was detected in an attribute value.

- *Error - 416 3 Error in attribute value '%1': Syntax error.* Logged when an attribute value is not syntactically correct.
- *Error - 417 3 Error in attribute value '%1': Type mismatch.* Logged when an attribute value contains type mismatching.

Using Galleries

In Architecture Modeller you can export or import model extensions to a common location known as Gallery.

1. To export to Gallery:
 - a) Select the extension you want to export and click **Gallery>Export**. The Export to Gallery window opens.
 - b) Write a short description for the Gallery item and click **OK**. The Select gallery for export window opens.
 - c) Browse to the current Gallery file location and click **OK** .
The selected model extension has been successfully exported to the selected gallery.
2. To Import from Gallery:
 - a) Click on **Gallery > Import** menu item. The Import from Gallery window appears.
 - b) Select the short description for the model extension you want to import from the gallery and click **OK**.

CA Web

Installing CA Web

The following steps assume you have created a workspace. See *Getting Started* for more information.

Typically, CA Web is installed together with COBOL Analyzer, by selecting the **Web Client** option in the installer. If you have already installed COBOL Analyzer but haven't installed CA Web with it, follow the steps below:

1. Install COBOL Analyzer. For instructions on how to do this see the *Installation Guide*.
2. Install JRE 7 (`jre-7u21-windows-i586.exe`). The JRE installer is provided with the CA installer in `CD1 - COBOL Analyzer\CA Web Server Pre-requisites\`.
3. Install Apache Tomcat (`apache-tomcat-6.0.16.exe`). The Apache Tomcat installer is provided with the CA installer in `CD1 - COBOL Analyzer\CA Web Server Pre-requisites\`. Follow the screen prompts and accept the defaults, except:
 - On the **Choose Components** screen, expand the **Tomcat** option and check **Service** to have Tomcat start automatically.
4. Stop the Apache Tomcat service from **Windows Control Panel > Administrative Tools > Services**.
5. Copy `CAWeb.war` from the Web Client folder in the COBOL Analyzer installation directory to `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\webapps`.
6. Start the Apache Tomcat service from **Windows Control Panel > Administrative Tools > Services**.

Deploying CA Web

After installing CA Web follow these steps:

1. In **Windows Control Panel > Administrative Tools > Services** start the Apache Tomcat service.

2. While you are still in **Windows Control Panel > Administrative Tools > Services**, right-click the COBOL Analyzer Web service and choose **Properties**. Go to the **Log On** tab, choose **This account** and enter the username and password of a user that has access to CA and the workspace.



Note: The logon user for the COBOL Analyzer Web service is also significant because the user preferences of this user will impact the types of repository and Hypercode objects that will be returned in the search. Diagrams shown in CA Web will also use the options set for this user.

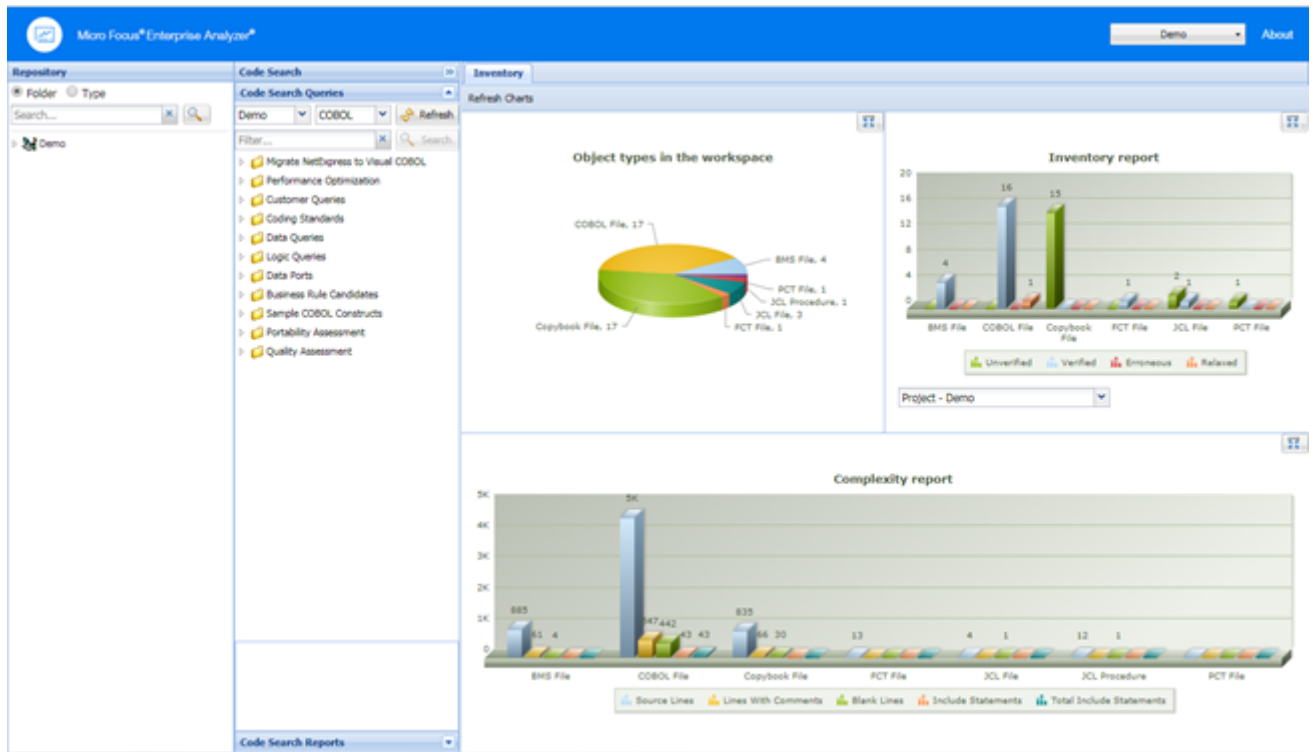
3. Restart the COBOL Analyzer Web service.
4. Open COBOL Analyzer Administration and select **Configure Web Service** from the **Administer** menu.
5. Click **Add** and point to your workspace `.rwp` file. The workspace that is checked as the default will be the workspace that is used with CA Web.
6. Click **Save**.
7. Go to `http://[servername]:8080/CAWeb`.

Troubleshooting

- Mozilla Firefox and Google Chrome might work better than Internet Explorer. With Internet Explorer you might need to click **Search** to actually execute the search instead of just pressing **Enter**.
- Do not use Compatibility View when using CA Web in Internet Explorer. It might cause problems with the **Go to source** feature.
- If you are unable to access the site, try re-deploying the `CAWeb.war` following these steps:
 1. Stop the Apache Tomcat service.
 2. Delete the CA Web folder from: `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\webapps`. Leave the `CAWeb.war` file.
 3. Delete all the files found in the following folders:
 - `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\webapps`
 - `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\temp`
 - `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\work\Catalina\localhost`
 4. Start the Apache Tomcat service.
 5. If you still cannot access the site from the server, check the Apache Tomcat Java configuration and make sure it is using JRE version 7 or higher.
- You might have to configure the Apache Tomcat service to use a user account that has access to CA and the workspace. To do so:
 1. Right-click the Apache Tomcat service and choose **Properties**.
 2. Go to the **Log On** tab, choose **This account** and enter your username and password.
 3. Restart the service.
- If you still do not see data on the site try the following in this order:
 1. Restart the COBOL Analyzer Web service.
 2. Delete the browser's temporary Internet files.
- If searches or queries are returning no results, refresh the Web page to check the connection status. If the status is Disconnected, you need to restart COBOL Analyzer Web service. To do so:
 1. Right-click the service and choose **Properties**.
 2. In the **Recovery** tab choose **Restart the Service** for the first, second, and subsequent failures.
 3. Click **OK** and restart the service.

Getting Started with CA Web

The starting page of CA Web consists of a few different sections - a banner, Repository pane, Code Search pane and results pane with the Inventory tab displayed in it.



Banner

Click **About** to see the CA Web version number.

The button in the top right shows the name of the current workspace. You can change the workspace by clicking this button and selecting a workspace from the list.

The choices you have depend on what workspaces you have configured from **Administer > Configure Web Service** in the COBOL Analyzer Administration.

Repository Pane

The Repository pane shows the contents of the repository for the current workspace. It has two views that you can switch between using the radio buttons:

Folder Displays the contents of the repository in the same folder structure as the one they have in the host environment. This allows for elements with the same name (homonyms) to be loaded and displayed.

Type Displays the contents of the repository by project, object type, and object relationship.

Each of the two tabs consists of the following elements:

Browser Shows the objects in the repository of the current workspace in a tree structure. Depending on the tab you have selected in the Repository pane, the contents are displayed either by type or by folder.

Search Lets you execute various searches in the repository of the current workspace.

To expand a node in the Repository pane tree, click ▾. By drilling down the nodes you can easily find programs called by or called to, copybooks used, data accessed, etc.. These elements are different

depending on the element chosen. They are all relationships at object level between programs, files, database tables, copybooks, and so forth.



Note: Unlike the COBOL Analyzer IDE, the CA Web Repository Browser can not expand relationships recursively for the root object. For example, expand a JCL file that uses a control card, then expand the **Used By** relationship for the control card. The starting JCL is displayed, but it cannot be expanded to see its relationships again.

Double-click an object and the source code will display in the pane on the right as a new tab alongside Inventory.

Right-click an object, and a context menu with the following menu items comes up: Query Repository, Diagrammer and Open.

Query

Shows all the queries available for the selected type of source.

Repository

- Related Java Packages
- Boundary Points (potential) - Points that could lead to requests resources outside of Java
- Boundary Points (resolved) - Points that lead to requests resources outside of Java
- Is Invoked By
- Related Java Files
- Dependent Sources
- Direct References (outgoing)
- Direct References (incoming) - This makes references to usage directly. A program is called through a Program Entry Point so this won't return the list of Called Programs, but the includes and copies.
- Is Called By
- Is CRUD By
- Used Data Stores
- Used Sources

Diagrammer

Creates a small graph. The information on the graph depends on the requested type of graph. For example, Call Map will show all the links between programs.

the following table displays the Diagrammer scopes:

Call Map	Program to program calls
CICS Flow	Interactions between transactions, programs, and screens
Job Flow	Process to program to data port to data store
Screen Flow	Program to screen send/receive
AS/400 Flow	DBFile links
Data Flow	Program to data
IMS Flow	Interactions between transactions, programs, screens and segments.
Job Executive Report	Job to program, sys-program and data store
Natural Flow	Natural links between programs, screens, etc.
Program Executive Report	All links coming from a program
Source Dependencies	Links between sources (copies, includes, etc.)
Unisys Flow	Unisys links

Open Opens the source file.

Code Search

From the Code Search pane you can execute queries. By default, the queries are the same as in COBOL Analyzer but you can create your own queries in the tool and add them to the Code Search tab.

In this pane you have two different views: Code Search Queries and Code Search Reports. From Code Search Queries you can execute queries defined in Code Search in CA and from Code Search Reports you can execute pre-defined reports such as QA report, coding standards report, migration assessment, etc.

From the top left dropdown list you can choose the project that you want to use as the scope for the Code Search Query.

The dropdown list next to it lets you choose the language of the sources.

Click **Refresh** to update the list depending on your choices.

Filter lets you search in the list of queries. You can then use the **Search** button to execute the query, or you can right-click a query and then select **Search**. All queries results appear in the Inventory pane.

Results Pane

When you start CA Web, this pane displays the Inventory tab with three different types of charts:

- Object types in the workspace - the types and number of elements in the workspace.
- Inventory report - the verification results for each type of element.
- Complexity report.

When you execute a query, the results from it appear in this pane as a new tab named after the query.

Doubleclick an element to see its source code. It will appear in a new view in this pane.

Doubleclick a result of a query to open the source code with the cursor on the corresponding line.

You can also right-click a variable and create an "Impact trace" for the variable. The impact trace will show you how the data flows to and from other variables, files, screens etc.

CA Web Reports

There are several reports that you can create and export from the Web Client. All report exports are started from the buttons at the bottom of the Results pane.

Export Table (Generic XML)	Exports the query results table in an XML format.
Export to Eclipse/Visual Studio	Exports the Code Search results in an XML format to be used by Enterprise Developer in Eclipse and Visual Studio.
Export to CSV	Exports the results to CSV format.

REST API and Jenkins Plugin

REST API and Jenkins Plugin

The REST API lets you administrate workspaces, run searches against the repository, and generate reports. A Jenkins plugin to use the REST API is included for easy integration with the popular continuous integration system.

REST API

The REST API library runs as an interactive client on your instance of COBOL Analyzer. This enables you to test actual requests and responses against your configuration, workspace, and files.

To access the list of available features in the interactive COBOL Analyzer API library, add `/docs` to the end of your base COBOL Analyzer URI, by default `http://localhost:1248/api/docs`.

GET : `http://localhost:1248/api`

Returns a list of the main APIs

- **GET** : `http://localhost:1248/api/management`

Returns a list of all available management API's

- **GET** : `http://localhost:1248/api/management/version`

Returns product details

- **GET** : `http://localhost:1248/api/workspaces`

Returns a list of all configured workspaces

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}`

Returns a list of all available workspace API's

- **POST** : `http://localhost:1248/api/workspaces/{workspaceName}?Path={workspaceDirectory}`

Register a new workspace

- **PUT** : `http://localhost:1248/api/workspaces/{workspaceName}?Path={workspaceDirectory}`

Update workspace directory

- **DELETE** : `http://localhost:1248/api/workspaces/{workspaceName}`

Remove workspace

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Inventory`

The report shows high-level statistics for source file types in the current workspace

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Projects`

List all available projects for the workspace

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CodeSearchReports`

List all available Code Search Reports

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CodeSearchReports/{reportName}`

Executes Code Search Report

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/SourceSync`

Synchronize sources

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CheckQueueEmpty`

Checks whether the queue is empty

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports`

List all available reports for the workspace

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports/Executive`

The report provides HTML views of application inventories that a manager can use to assess the risks and costs of supporting the application

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports/Unresolved`

An Unresolved Report identifies missing application elements

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports/CrossRef`

A Cross-reference Report identifies all application references

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports/Unreferred`

An Unreferred Report identifies unreferenced application elements

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/Reports/CRUD`

The report shows the data operations programs perform (Insert, Read, Update, or Delete) and the data objects on which the programs operate

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CodeSearch`

List all available mode

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CodeSearch/{model}`

List all available queries for the model

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CodeSearch/{model}/{criterion}`

Executing Code Search Searches

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/CAWeb`

Returns a link to the workspace on CA Web if configured

- **GET** : `http://localhost:1248/api/workspaces/{workspaceName}/FortifyMBSExport`

Export Fortify MBS

First Steps with the REST API

Before using the REST API or CA Jenkins Plugin make sure that the Micro Focus COBOL Analyzer Web service is running with the Logon User set to your Windows user.

To make a **GET** request, you can use any browser like Chrome, Firefox, or Internet Explorer. For the other HTTP requests such as **PUT**, **POST** or **DELETE** you can use any tool like Postman or Fiddler.

To register a new workspace, use either the **REST API** or **COBOL Analyzer Administration > Administer > Configure Web Service**.

If you want to use the REST API, you need to make a **POST** request with the name of the workspace and the Path of the workspace as a parameter: `http://localhost:1248/api/workspaces/NewWorkspace?Path=C:\MicroFocus\Workspaces`

```
{
  "status": "success",
  "message": "Workspace 'NewWorkspace' is registered successfully!"
}
```

Next you can list all registered workspaces: `http://localhost:1248/api/workspaces`

```
[
  {
    "name": "NewWorkspace",
    "directory": "D:\\MicroFocus\\Workspaces",
    "default": true,
    "url": "http://localhost:1248/api/workspaces/NewWorkspace"
  }
]
```

```
}  
]
```

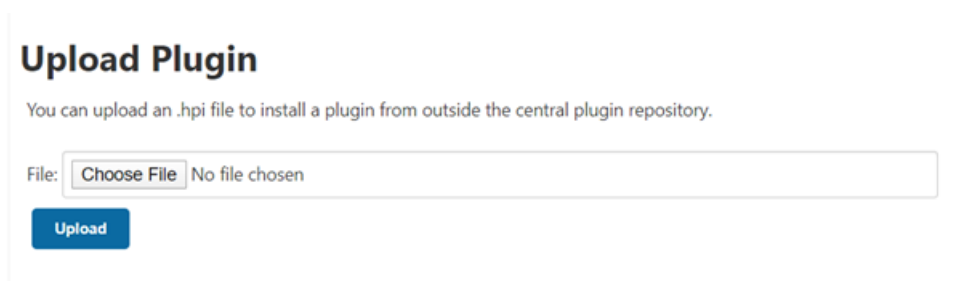
Then you can list all available APIs for the selected workspace: `http://localhost:1248/api/workspaces/NewWorkspace`

```
[  
  {  
    "url": "http://localhost:1248/api/workspaces/NewWorkspace/Projects",  
    "description": "List all available projects for the workspace"  
  },  
  {  
    "url": "http://localhost:1248/api/workspaces/NewWorkspace/Inventory",  
    "description": "The report shows high-level statistics for source file  
types in the current workspace",  
    "parameters": [  
      {  
        "name": "Format",  
        "description": "Report formats: json(default), html, htm, xls, rtf,  
doc, txt, csv",  
        "required": false,  
        "example": "format=txt"  
      }  
    ]  
  },  
  ...  
]
```

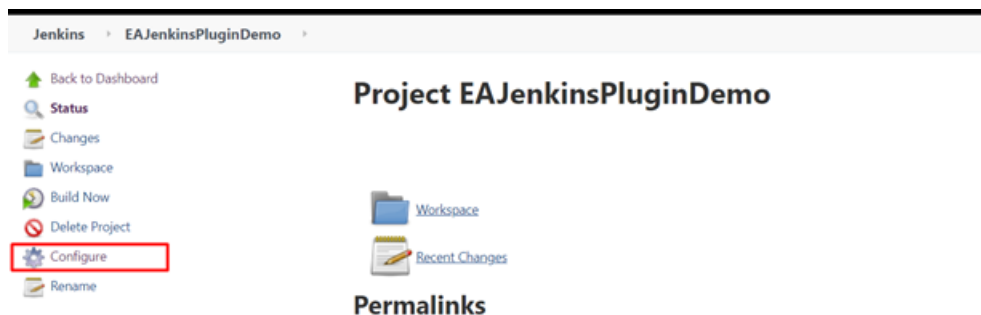
In the list of available APIs you will find the full details needed to use each API. Descriptions, parameters, supported formats, and so forth are all provided.

Getting Started with the EA Jenkins Plugin

1. Ensure that the Micro Focus COBOL Analyzer Web service is running with the Logon User set to your Windows user.
2. Install Jenkins with the default settings on your local machine: `https://jenkins.io/download/`
3. Go to **Manage Jenkins > Manage Plugins > Advanced > Upload Plugin**. Select the `EAJenkinsPlugin.hpi` from the CA installation `\Bin` directory.



4. Go to **Home Page > New Item > Create Freestyle Project**.
5. Go to **Project Configuration**.



Jenkins - EA/JenkinsPluginsDemo - #1 - Micro Focus Enterprise Analyzer

Micro Focus Enterprise Analyzer

Resources:
[View: Niagara Enterprise COBOL to EIT in Slack/Your Studio](#)

Inventory Report:

Type	source lines	quantity	unclassified	classified	errored	reused	missing
AssemblyPackage	11,174	100	0	100	0	0	0
COB File	288	24	0	24	0	0	0
COB2 File	11	11	0	11	0	0	0
COB2C File	1,101	21	0	21	0	0	0
Control Layout File	18	18	0	18	0	0	0
Controlcard File	181	4	0	4	0	0	0
COB File	1,108	4	0	4	0	0	0
COB File	118	0	0	0	0	0	0
COB File	68,818	0	0	0	0	0	0
Control Description	111	1	0	1	0	0	0
COB File	11	1	0	1	0	0	0
COB File	68	0	0	0	0	0	0
COB File	11,117	0	0	0	0	0	0
Natural Data Area	1,191	10	0	10	0	0	0
Natural COB File	1,184	10	0	10	0	0	0
Natural Map	1,180	10	0	10	0	0	0
Natural Subroutine File	24,211	10	0	10	0	0	0
Natural File	24,211	10	0	10	0	0	0
COB File	11	0	0	0	0	0	0
COB File	1,184	10	0	10	0	0	0
COB File	1,181	10	0	10	0	0	0
COB File	24,211	10	0	10	0	0	0

Product:
 Micro Focus Enterprise Analyzer
 6101278

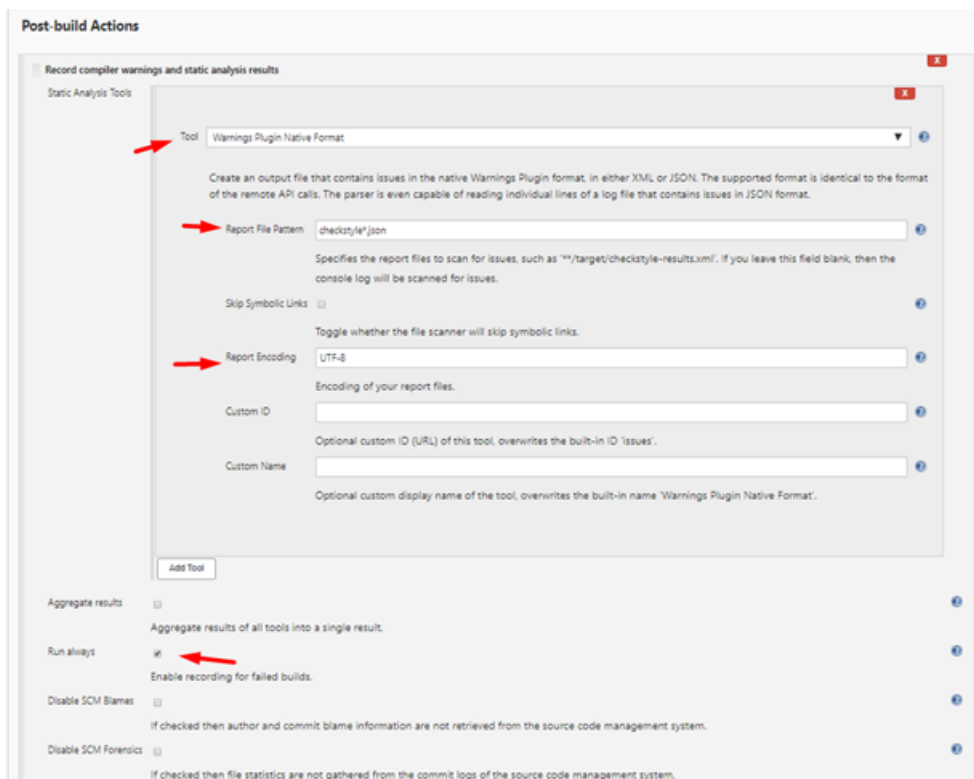
Adding the Next Generation Warnings Plugin

<https://plugins.jenkins.io/warnings-ng/>

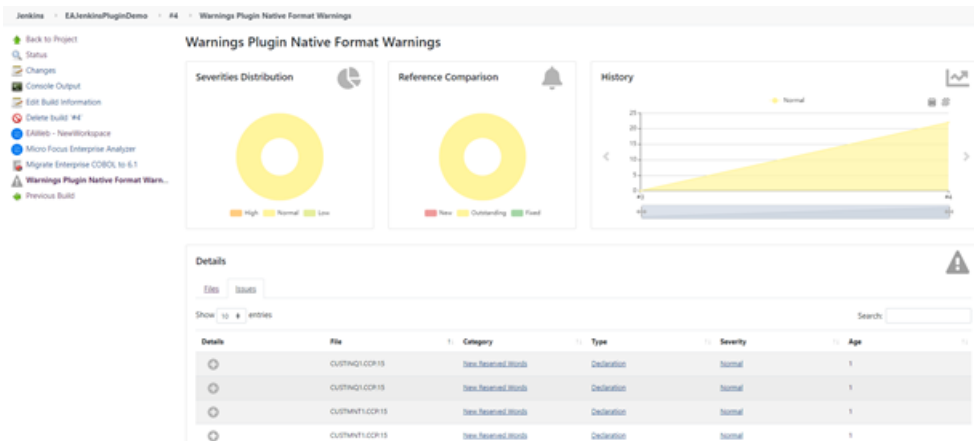
1. Go to **Manage Jenkins > Manage Plugins > Available > Search for "Warnings Next Generation"**.
2. Install the plugin and restart Jenkins.
3. Go to **Project Configuration**.
4. In **Post Build Action** select **Record compiler warnings and static analysis results**.



5. Set up the following configuration:



6. The plugin is now configured.



Support Notes

Supported Languages

The table below outlines the support Enterprise Analyzer provides for different languages based on platform. The level of support is as follows:

- **Basic:** System level objects, dependencies and metrics only
- **Intermediate:** Syntactical analysis within source objects
- **Advanced:** Support for field level data dependencies

Environment	Language	Level of Support		
		Advanced	Intermediate	Basic
Distributed Environment	Language			
	Micro Focus COBOL	✓		
	ACU COBOL	✓		
	RM/COBOL	✓		
	Java SE (1.3-1.7)		✓	
	Visual Basic 6.0		✓	
	Microsoft C, GNU C, Unisys		✓	
	Microsoft Visual C+ + 6		✓	
	JSP (2.2)			✓
	.NET Framework			✓
	C#			✓
	Visual Basic .NET			✓
	Oracle PL/SQL for Oracle Database			✓
	Application Frameworks			
	Spring Framework - Dependency Injection Support			✓
JEE - EJBs			✓	
Other Mainframe	Language			
	HP COBOL (II/XL)		✓	

Supported Features by Language

The table below shows COBOL Analyzer feature support by language. The remainder of the section provides usage notes and caveats for the major supported languages and related platforms. Make sure to check the *Release Notes* on the installation CD for late-breaking support information.



Note: In the table Y* means that HyperCode is available on a line-by-line basis only. Y** means that impact analysis context-sensitivity is available on an intraprogram basis only.

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Inventry Management											

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Registration	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Source Editor	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
-Syntax Highlighting	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
Verification	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
- Relaxed Parsing	Y	Y	Y	N	N	N	N	N	N	N	N
- Verification of DBCS	Y	Y	N	Y	N	N	N	N	N	N	N
Inventory Report	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Verification Report	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Orphan Analysis	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Batch Refresh Process	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Architecture Modelling											
Decision Resolution	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
- Decision Autoresolution	Y	Y	N	Y	N	N	N	N	N	N	N
Generic API Analysis	Y	Y	N	N	N	N	N	N	N	N	Y
Boundary Decision	N	N	N	N	Y	N	Y	Y	Y	N	N

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Analysis											
System-Level Analysis											
Diagrammer	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Complexity Metrics	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Effort Estimation	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Batch Application Viewer	Y	Y	Y	Y	N	N	N	N	N	N	N
CRUD Report	Y	Y	Y	Y	N	N	N	N	N	N	Y
Query Repository	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
System-Level Reporting											
Executive Report	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
- Possible Code Anomalies	Y	Y	Y	Y	Y	N	Y	N	Y	Y	N
- Prepackaged Code Anomalies	Y	N	N	N	N	N	N	N	N	N	N
- Custom with Code Search Queries	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N
Reference Reports	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
-Source Dependencies	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
-Calls	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
-Data Access	Y	Y	Y	Y	N	N	N	N	N	Y	N
-Screen Access	Y	Y	Y	Y	N	N	N	N	N	N	Y
Business Control											
Tag Manager	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Glossary	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
- Business Name Synchronization	Y	Y	Y	Y	N	N	N	N	N	N	N
- Propagate from Screen Fields	BMS	N	N	N	N	N	N	N	N	N	N
Source-Level Analysis											
Interactive Analysis	Y	Y	Y	Y	Y	N	Y	Y*	Y	N	Y*
-Source Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
- Context Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
-Code Search Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
-Model Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
- Program	Y	Y	Y	Y	N	N	N	N	N	N	N

Funcio nal Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Control Flow Pane (Call Diagra m)	-	Y	Y	N	N	N	N	N	N	N	N
Flowch art Pane	-	Y	N	N	N	N	N	N	N	N	N
Executi on Path Pane	-	Y	N	N	N	N	N	N	N	N	N
Animat or Pane	-	Y	N	N	N	N	N	N	N	N	N
-Bird's Eye Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
-Watch Pane	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
-Screen Pane	Y	Y	Y	Y	N	N	N	N	N	N	N
Data Field Analysi s											
Global Data Elemen t Flow	Y	Y	Y	Y	N	N	N	N	N	N	N
-Data View Pane	Y	Y	Y	Y	N	N	N	N	N	N	N
-Data Flow Pane	Y	Y	Y	Y	N	N	N	N	N	N	N
Impact Analysi s	Y	Y	Y	Y	N	N	N	N	N	N	N
- Intrapro gram	Y	Y	Y	Y	N	N	N	N	N	N	N
- Interpro gram	Y	Y	Y	Y	N	N	N	N	N	N	N

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
-Thru External Data	IMS, SQL	IMS, SQL	IMS, SQL	IMS, SQL	N	N	N	N	N	N	N
- Context Sensitivity	Y	N	Y**	N	N	N	N	N	N	N	N
Change Analyzer	Y	Y	Y	Y	N	N	N	N	N	N	N
Slicings											
Logic Analyzer	Y	Y	Y	Y	N	N	N	N	N	N	N
-Dead Code and Data Analysis	Y	Y	Y	Y	N	N	N	N	N	N	N
- Structure-Based Analysis	Y	Y	N	Y	N	N	N	N	N	N	N
- Computation-Based Analysis	Y	N	Y	N	N	N	N	N	N	N	N
- Domain-Based Analysis	Y	Y	N	N	N	N	N	N	N	N	N
Business Rules											
Business Rule Manager	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y
-Create Rules	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
-Create Code	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y

Functional Area/Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Search Rules											

C/C++ Support

COBOL Analyzer supports Microsoft C 6.0 and Microsoft Visual C++ 6.0. Follow the instructions in this section to register and verify C and C++ source files.

Registering C/C++ Files

Before registering C/C++ files, set:

- **Expand tabulation symbols** on the Registration > Source Files tab of the Workspace Options. Tabulation symbols are replaced with a corresponding number of spaces when you register the files. You must select this option if you want to view Interactive Analysis information for C or C++ programs.

Verifying C/C++ Files

Before verifying C/C++ files:

- On the **Verification > Settings** tab of the Workspace Options, list the folders for include files used by the application (either original folders or folders in CA, if the include files were registered).
 - 💡 **Tip:** You can also specify these folders in the verification options for a project, in which case CA looks only at the folders for the project.
- On the Verification > Settings tab of the Workspace Options, enter the parameters used to compile the application in the **C/C++ Parser Parameters** field. For example, the parameters would be `-D_DEBUG -D_UNICODE -D_WINDLL -D_DLL` for an MFC application.
 - 💡 **Tip:** You can also specify these parameters in the verification options for a project, in which case only the project parameters are used for verification.
- On the Verification tab of the Project Options, 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.
- On the Boundary Decisions tab of the Workspace Options, specify the resource types for method calls your application uses to interface with databases, message queues, or other resources.

COBOL Support

COBOL Analyzer supports the following COBOL versions:

- Micro Focus COBOL, V3.2 (level 10).
- .NET/JVM COBOL
- ACUCOBOL-GT®, Version 6.1.
- HP3000, HP COBOL II/XL.
- RM/COBOL

.NET/JVM COBOL

.NET COBOL and JVM COBOL are sometimes referred to as managed COBOL.

 **Restriction:** Object-oriented COBOL statements are not supported.


Separators Must Be Followed by Blanks

The Cobol parser assumes that every separator must be followed by a blank. If you index a variable with a separator that is not followed by a blank, `MY-VARIABLE(1,1)`, the parser may treat `(1,1)` as a numeric literal, especially when the program was compiled with the `DECIMAL POINT IS COMMA` option. To index a variable, use the format `MY-VARIABLE(1, 1)` or `MY-VARIABLE(1 1)`.

Copybooks in a Library

If the copybooks used in a Cobol program are in a library, and the library is referenced in a `COPY` statement with the format `COPY text-name IN library-name` or `COPY text-name OF library-name`, the parser looks first for a copybook named `library-name.text-name`, and if it does not exist, for a copybook named `text-name`. If `text-name` does not exist, the parser reports `library-name.text-name` as an unresolved reference.

It is your responsibility to prefix library member names with library names or filepaths and dot (`.`) separators: `dir1.dir2.member.cpy` represents the copybook `dir1/dir2/member`, for example. When the parser encounters a reference to a member, it first searches for the longest possible name, `dir1.dir2.member.cpy`, and if not found, then the shorter versions, `dir2.member.cpy` and `member.cpy`.

 **Note:** Unresolved references to library members are always reported with the longest name. This means that if you subsequently register a missing copybook with a short name, the referencing source file will not be invalidated. It's up to you to remember that the referencing source needs to be reverified.

How COBOL Analyzer Calculates COBOL Dead Code Statistics

This section provides details on how CA calculates COBOL dead code statistics.

Dead Statements

A dead statement is a statement that can never be reached during program execution. Only control flow analysis techniques (static analysis) are used for the detection of dead statements. Domain-based analysis is not performed.

Statements directly connected with dead statements are also considered to be dead. For instance, `EXEC CICS HANDLE` statements are dead when all `EXEC CICS` statements are dead or there are no `EXEC CICS` statements at all.

Dead Data Elements

Dead data elements are unused structures at any data level, all of whose parents and children are unused. Condition names (88-level items) are dead if unused.

Only user-defined data elements can be counted as dead. Data elements from system copybooks are never counted as dead.

Dead Constructs

A paragraph consisting solely of dead statements is a dead paragraph. A section consisting solely of dead paragraphs or that is empty is a dead section. The exception to this is the Configuration Section. Because there are no candidate dead constructs (statements or data elements) in the Configuration Section, this section is not processed and does not contribute to dead code metrics. A division is never considered dead.

A file description entry (FD) containing only dead data elements and not used in any file operation is a dead file description. A file section containing only dead file descriptions is a dead section. A `SELECT` statement referring to a dead file description is a dead construct.

A file-control paragraph consisting solely of dead SELECT statements is a dead paragraph. An input-output section consisting solely of dead file-control paragraphs is a dead section.

Dead Statements, Dead Data Elements, and Dead Lines from Copybooks

Dead statements and dead data elements from copybooks (that either start or end in a copybook) are counted in the Dead Statements, Dead Data Elements, and Dead Lines metrics. They are also counted separately in the Dead Statements from Includes, Dead Data Elements from Includes, and Dead Lines from Includes metrics.

If a copybook is included multiple times, then each instance of the copybook is considered to be an independent source file, and all dead constructs and dead lines from the copybook are counted as many times as they are identified as dead. For instance, if a copybook is included twice and both inclusions result in a dead data element, the result is Dead Data Elements from Includes=2 and Dead Lines from Includes=2 (assuming each dead data element occupies only one line of the included copybook). If the same copybook is included twice but only one instance results in a dead data element, then Dead Data Elements from Includes=1 and Dead Lines from Includes=1.

All “Dead from Includes” metrics are for the specified program only. These metrics do not include an analysis of the same copybook over the entire application.

Interactive Analysis Usage

In Interactive Analysis, all dead statements, dead paragraphs, dead sections, dead data declarations, dead files, and instances of dead files in statements will have the attribute Dead set to True.



Note: Not all language syntax phrases are represented in the Interactive Analysis model, so not all dead constructs contributing to dead lines can be identified using Code Search. In other words, Code Search can identify all dead data elements and all dead statements, but not necessarily all dead lines.

Special Handling of Cobol Program Complexity Metrics

- Abbreviated conditions are expanded before calculations.
- DECLARATIVEs content and other exception-handling statements are counted once, as ordinary statements.
- Handling of EVALUATE formats:

```
EVALUATE ... [ALSO ...] Conditional Statement  
WHEN ... [ALSO ...] Binary Decision, Conditional Statement  
WHEN OTHER Conditional Statement  
END-EVALUATE
```

- Handling of SEARCH formats:

```
SEARCH ... AT END ... Binary Decision  
WHEN ... Binary Decision, Conditional Statement  
WHEN ...AND... Binary Decision, Conditional Statement  
END-SEARCH
```

Possible Padding in MOVE Statements

The Advanced Search criterion for Possible Data Padding defects in MOVE statements (in which the internal representation of the source variable in the assignment is shorter than the internal representation of the destination variable) finds only MOVE statements involving at least one group data item. It does not find MOVE statements involving assignments of elementary data items.

ACUCOBOL Support


In COBOL Analyzer ACUCOBOL's displays of message boxes generate screen output relationships. If the message box has no title, it can appear as `program.display`. This happens when there are two `DISPLAY MESSAGE BOX` statements that do not specify screen.


Java Support

COBOL Analyzer supports Java SE 1.3-1.6.

Follow the steps below to analyze Java applications in COBOL Analyzer:


1. The Java SE Runtime Environment (JRE) needs to be installed on your machine. The current version of the JRE specified in the Windows registry will be used.

 **Note:** For the required JRE version, see the *Hardware and Software Requirements* section in the Installation Guide.


 **Note:** If your application was compiled with a previous version of the JRE, specify the path of the Java SE runtime library (`rt.jar`) used to compile the application in the **Enter classpath to JAR Files and/or path to external Java file root directories** field in **Workspace Options > Verification > Settings**.

2. Register Java sources (`.java`) in COBOL Analyzer.

3. Go to **Options > Workspace Options > Verification > Settings**, right-click in the **Enter classpath to JAR Files and/or path to external Java file root directories** field and choose **Add** from the context menu to specify JAR or Zip files containing class libraries referenced in the application. Alternatively, choose **Add Folder** from the context menu to specify the path of the folder containing the JAR or Zip files, then select **Include Jar/Zip Files From Directories**.


 **Note:** This option is also available in **Project Options > Verification > Settings**. The project option is checked before the workspace option, so setting it for the project effectively overrides the setting for the workspace.

4. If your application uses constructs that are incompatible with the latest Java version, specify the Java version it uses in the **Java Source Level** field in **Project Options > Verification**. You would set this option to 1.4, for example, if your application used "enum" as an identifier, since "enum" is a reserved word in Java 5.

 **Note:** This option affects only the syntax accepted by the parser. It does not change the version of the Java SE runtime library (`rt.jar`) used to parse Java files.

5. Specify the resource types for method calls your application uses to interface with databases, message queues, or other resources in **Workspace Options > Boundary Decisions**.

6. Verify the application source files.

 **Note:** COBOL Analyzer does not support WSDL files.

JSP Support

COBOL Analyzer provides application-level support for JSP 2.2. You must install the Java add-on to enable JSP support. For installation instructions, see the installation manual for your product.

Follow the steps below to analyze JSP applications in COBOL Analyzer:

1. Register JSP (`.jsp`), Tag Library Descriptor (`.tld`), and Web Config (`web.xml`) files in COBOL Analyzer.
2. If your JSP files reference Java types or packages defined in JAR files, external Java files, or Java files, specify a list of patterns used to resolve the location of the files in **Project Options > Verification > Settings** for the JSP file type.

3. Verify JSP and Tag Library Descriptor files. Web Config files (`web.xml`) do not need to be verified.

Resolving the Location of Java Types and Packages Referenced in JSP Files

When references are resolved to Java types from JPS files during verification, some references may not be resolved due to ambiguous names.

For example, `Id.java` in the `app1` folder is registered as `app1\com\company\Id.java` and `Id.java` in the `app2` folder is registered as `app2\com\company\Id.java`.

JSP cannot find the classes because there are now two possible definitions of the class `com.company.Id` in the workspace. If your JSP files reference this class, you need to specify which Java file defines the class in the **Java Classpath** field for the JSP file type in **Project Options > Verification > Settings**. Otherwise, the relationship between JSP and the class is marked as unresolved in the workspace.

Java Classpath contains a list of patterns used to resolve the location of Java types and packages referenced in JSP files. Each pattern describes the path to a Java file that defines the referenced type or package. In our case, the pattern might be `app1*`, which matches any Java source file registered in the workspace from the `app1` folder.

You can also match JAR files or external files referenced in Java applications but not registered in the workspace, as specified in the **Enter classpath to JAR Files and/or path to external Java file root directories** field for the Java file type in **Workspace Options > Verification > Settings**. The pattern `*\jre6\lib\rt.jar` would find the Java 6 run-time library, for example.

Right-click in the **Java Classpath** field and choose **Add** in the pop-up menu to enter a pattern. You can use asterisk symbol to match any sequence of characters.

Relationships that cannot be resolved by any of the specified patterns are marked as unresolved in the workspace. Once the classes are resolved, only the base name of the referenced class or package appears in the relationship (in the Repository Browser). Use the Unresolved Report in the Reference Reports window to see what is still unresolved and determine the patterns you need to add.

.NET Support

COBOL Analyzer supports Microsoft .NET Framework 3.5, Visual C# 3.0, and Visual Basic .NET 9.0. The table below shows the supported file types and their entity types in the repository. Register the entire folder for the project to avoid problems with duplicate file names.

File Type	Entity Type
.csproj, .vbproj	.Net Project
.cs, .vb	.Net File

PL/SQL Support

COBOL Analyzer supports Oracle PL/SQL for Oracle Database 11g, Release 1.

SQL Support

COBOL Analyzer supports DB2 UDB for z/OS, Version 8 (DCLGENs, SQL, DDL).

Renaming DCLGEN Include Files

In previous releases, DCLGEN include files with the same names as ordinary include files had to be renamed with a DCLGEN prefix and dot (.) separator so that both types of file could be registered: `ATTR.<valid extension>`, for example, and `DCLGEN.ATTR.<valid extension>`.

You don't have to rename these files anymore. In fact, if you have the files renamed already, you should now rename them back to not have the DCLGEN. prefix.



Note: Unresolved references to library members are always reported with the short name. This means that if you subsequently register a missing include file with a long name, the referencing source file will not be invalidated. It's up to you to remember that the referencing source needs to be reverified.

Prefixes for SQL Names

To maintain uniqueness of ERD entity names, CA specifies SQL names with an SQLID prefix, defined by a corresponding SET CURRENT SQLID statement. Names of Table objects are prefixed with CURRENT SQLID when it is set by a preceding SET CURRENT SQLID statement.

VB Support

COBOL Analyzer supports Microsoft Visual Basic 6.0. Follow the steps below to analyze VB applications in COBOL Analyzer.

1. Use the XDL.exe utility shipped with CA to generate Library Description files for ActiveX components referenced in VB Project files but not contained in the projects themselves. For more information, see "Generating Library Description Files."



Note: COBOL Analyzer provides Library Description files for ActiveX system libraries in the CA \Templates\VB directory.

2. Register the VB application in COBOL Analyzer. The following objects must be registered:
 - VB project files (.vbp)
 - System Library Description files (.xdl) in the CA \Templates\VB directory
 - Generated Library Description files (.xdl)
 - Source files (.bas, .cls, .ctl, .frm)
3. Specify the resource types for method calls your application uses to interface with databases, message queues, or other resources on the Workspace Options > Boundary Decisions tab.
4. Verify Library Description files and VB project files.



Note: Library Description files must be verified before VB project files. As long as you verify the entire project for your Visual Basic application, CA parses the source files in appropriate order, taking account of the dependencies between file types.

Bear in mind that dependencies between CA projects are not taken into account. If VB Project1 references VB Project2, VB Project2 must be verified first. To check for dependency-related verification errors, run verification iteratively, verifying only those files in each project that have not been verified successfully. If the number of unsuccessfully verified files does not decrease after a run, the remaining issues are not dependency-related.



Important: Before registering VB sources, see "Restrictions on Visual Basic Support" for limitations in COBOL Analyzer support that may affect your project.

Generating Library Description Files

ActiveX components referenced in VB Project files but not contained in the projects themselves must be registered in an CA project as Library Description files (.xdl). The following ActiveX component types may be referenced:

- DLLs (.dll)
- OCX controls (.ocx)
- TLBs (type libraries, .tlb)
- OLBs (object libraries, .olb)

COBOL Analyzer provides Library Description files for ActiveX system libraries in the CA \Templates\VB folder. Use the XDL.exe utility shipped with CA to generate Library Description files for the remaining ActiveX components.

1. Open a VB Project file and locate the first referenced ActiveX component. Referencing statements use a Reference= or Object= format:

```
Reference=* \G{6B263850-900B-11D0-9484-00A0C91110ED}#1.0#0#D:\WINNT
\System32\msstdfmt.dll#Microsoft
    Data Formatting Object Library
Object={5E9E78A0-531B-11CF-91F6-C2863C385E30}#1.0#0; MSFLXGRD.OCX
```

2. Run the XDL.exe utility. You can use a command prompt window or the graphical user interface provided with the utility:
 - In a command prompt window, enter `<CA Home>\Bin\XDL.exe component_name.file_extension "C:\XDLs\component_name.XDL"`
 - In the CA \bin directory, double-click XDL.exe. The XDL Report window opens. Choose **File > Open**, then browse to the location of the referenced file (.dll, .ocx, .tlb, or .olb) and click **Open**. In the XDL Report window, choose **File > Save As** and save the XDL file to a location outside the workspace.
3. Repeat these steps for each referenced Active X component and each VB Project file.

Restrictions on Visual Basic Support

The following constructs are not supported and are marked with the "Item was ignored by parser" message during verification:

- AppActivate
- ChDir
- ChDrive
- Close
- Deftype
- DeleteSetting
- Erase
- Error
- Event
- Get
- GoSub...Return
- GoTo
- Implements
- Input #
- Kill
- Line Input #
- Lock
- Lset
- MkDir
- Name
- Open
- Option Base
- Option Compare
- Option Private
- Put
- RaiseEvent
- Randomize
- Resume

Rmdir
Rset
SaveSetting
Seek
Stop
Time
Exit
Width #

Complexity Metrics

Complexity Metrics Overview

The tables in this section describe the supported complexity metrics for objects in the COBOL Analyzer (CA) repository. The tables are listed alphabetically by object type.



Note: CA uses a generic "Program" object for programs written in different languages. The tables break out program types by language. Look for COBOL Program, Natural Program, PL/I Program, and so forth.

ADL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Records	Number of records.
Sets	Number of sets.
Source Lines	Number of lines of source.

Assembler Copybook File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Assembler File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.
Unique System Macro Instructions	Number of unique system macro instructions.
Unique User Macro Instructions	Number of unique user macro instructions.

Assembler Programs

Provides details about the complexity metrics associated with Assembler Programs.

Metric	Description
Absolute Complexity	Number of Binary Decisions divided by the number of statements.
Binary Decisions	Number of branch conditional opcodes where (mask != 'FF').
Conditional Complexity	Number of Binary Decisions.
Conditional Statements	Number of branch conditional opcodes where (mask != 'FF').
Cyclomatic Complexity	Binary Decisions + 1.
Data Elements	DC, DS.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is the number of Unique Operators, N2 is the number of Operands, and n2 is the number of Unique Operands.
Entry Points	ENTRY,CSECT, plus user defined macros.
Error Estimate	$B = E^{(2/3)} / 3000$, where E is Programming Effort.
Executable Statements	Number of statements of kind computation, flow, internal calls, external calls, and some system macros; e.g. BAL, BAS BASM, BALR, BASR, BASSM CALL, SVC.
Extended Cyclomatic Complexity	Cyclomatic Complexity plus the number of Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: COBOL=77, Natural=52, PL/I=67, ASM=1. Estimate of the number of end-user business functions implemented by the program.
Go To Statements	The sum of unconditional and conditional Go To (If) statements: B,BR + Binary Decisions.
Inner Call Statements	BAL, BAS.

Metric	Description
Intelligent Content	$I = L * V$, where L is the Program Level and V is the Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines Of Code	Number of source lines in the program including copybooks.
Loop Statements	BCT, BCTR, BXH, BXLE.
Operands	Number of operands for computational statements.
Operators	Number of computational statements. Operators can have zero, one or two Operands.
Program Length	$N = N1 + N2$, where N1 is the number of Operators and N2 is the number of Operands.
Program Level	$L = 1 / D$, where D is the program Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is the Program Length and n is the program Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is the Program Volume and L is the Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Returning Calls	Inner Call Statements + External Calls [(BAL, BAS) + (CALL, SVC, BASM, BALR, BASR, BASSM)].
Unique operands	Number of unique operands.
Unique operators	Number of unique operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

BMS Copybook File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

BMS File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.

Metric	Description
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

C File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

C++ Class

Metric	Description
Comments Ratio	Number of Comments divided by Lines of Code.
Computational Statements	Number of statements performing arithmetic calculations.
Constructors	Number of constructors.
Data Members	Number of data members.
Depth of Inheritance Hierarchy	Maximum nesting of the inheritance hierarchy.
Destructors	Number of destructors.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Inherited Methods	Number of inherited methods.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume,

Metric	Description
	ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Member Functions	Number of member functions.
Number of Comments	Number of items containing text preceded by // or enclosed between /* and */.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Public Data Members	Number of public data members.
Public Member Functions	Number of public member functions.
Response for Class	Response for class.
Static Data Members	Number of static data members.
Static Functions	Number of static functions.
Static Member Functions	Number of static member functions.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

C++ File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).

Metric	Description
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

C++ Function

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Comments Ratio	Number of Comments divided by Lines of Code.
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where $n1$ is Unique Operators, $N2$ is Operands, and $n2$ is Unique Operands.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Exception Handling Statements	Number of exception handling statements.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Labels	Number of labels.

Metric	Description
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Local Data Items	Number of local data items.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.
Number of Comments	Number of items containing text preceded by // or enclosed between /* and */.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Static Data Items	Number of static data items.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Function Calls	Number of unique function calls.
Unique Member Function Calls	Number of unique member function calls.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.

Metric	Description
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

C++ Header File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

C++ Member Function

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Comments Ratio	Number of Comments divided by Lines of Code.
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where v(G) is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Error Estimate	$B = E^{(2/3)} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). v(G) for reduced graph without D-structured primes.
Exception Handling Statements	Number of exception handling statements.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.

Metric	Description
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Labels	Number of labels.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Local Data Items	Number of local data items.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.
Number of Comments	Number of items containing text preceded by // or enclosed between /* and */.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.

Metric	Description
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Static Data Items	Number of static data items.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Function Calls	Number of unique function calls.
Unique Member Function Calls	Number of unique member function calls.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

COBOL Copybook File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Data Elements	Number of declared data items (elementary structures and their fields).
Include Statements	Number of include statements: COPY, ++INCLUDE, –INC, EXEC SQL INCLUDE.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

COBOL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Resource Statement	Number of included resource statements.
Include Statements	Number of include statements: COPY, ++INCLUDE, –INC, EXEC SQL INCLUDE.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

COBOL Program

Metric	Description
3-Maintainability	3-MI = $171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, and LOC is Lines of Code.
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as COBOL INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on): IF, EVALUATE (number of WHEN except WHEN OTHER), PERFORM...TIMES, PERFORM...UNTIL, PERFORM...VARYING, PERFORM...VARYING...AFTER (number of AFTER phrases + 1), statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID (one decision per statement), GOTO...DEPENDING ON (number of alternatives), SEARCH (number of WHEN, AT END). IDMS: IF.
Computational Statements	Number of statements performing arithmetic calculations: ADD, SUBTRACT, DIVIDE, MULTIPLY, COMPUTE.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions, not including conditional GOTOs. IF, EVALUATE, SEARCH, PERFORM...UNTIL, PERFORM...VARYING...UNTIL, statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID. IDMS: IF.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields).
Dead Data Elements	Number of dead data elements in programs and used include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.

Metric	Description
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data Elements or Dead Statements. Also, source lines containing dead constructs.
Dead Lines from Includes	Number of dead lines in include files. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes. Also, source lines containing dead constructs.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points: PROCEDURE DIVISION, ENTRY.
Error Estimate	$B = E^{(2/3)} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All Procedure Division statements, plus CONTINUE and NEXT STATEMENT.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: COBOL=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Inner Call Statements	Number of statements that invoke Inner Procedures: PERFORM procedure-name.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: COBOL paragraphs (including nameless).
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations: OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START, SORT, MERGE, RETURN, RELEASE, ACCEPT, DISPLAY, STOP literal. SQL : INSERT, FETCH, SELECT, UPDATE, DELETE, EXECUTE. CICS : CONVERSE, SEND, SEND MAP, SEND TEXT, RECEIVE, RECEIVE MAP, RECEIVE TEXT, READQ, WRITEQ, DELETEQ, READ, READNEXT, READPREV, WRITE, REWRITE, DELETE. IDMS : ERASE, OBTAIN, GET, MODIFY, STORE.

Metric	Description
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
LINK Statements	Number of CICS LINK statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions: AND, OR.
Loop Statements	Number of repetitively executing statements: PERFORM...TIMES, PERFORM...UNTIL, PERFORM... VARYING PERFORM...VARYING...AFTER (# of AFTER + 1).
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as COBOL XCTL statements.
Number of Code Lines Without Copy	Number of lines of source code. Included files are not counted. Comments and blank lines are not counted.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Unique Operators.
Parameters	Number of COBOL Procedure Division USING...RETURNING parameters.
Pointers	Number of data elements declared as pointers. Data items with USAGE IS POINTER, PROCEDURE-POINTER.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Recursive Number Of Comments With Copy	Number of lines of source containing comments, plus the number of lines of source containing comments in

Metric	Description
	included files and any files they include. Inline comments placed on lines with statements are not counted.
Recursive Number Of Source Lines With Copy	Number of lines of source, plus the number of lines of source in included files and any files they include. Comments and blank lines are counted.
Response for Class	Response for class.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

COMS Configuration File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Control Cards File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Control Language File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.

Metric	Description
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

CSD File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

DASDL Include File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Database Description File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Database File

Metric	Description
Fields	Number of fields.
Key Fields	Number of key fields.
Omit Fields	Number of omit fields.
Records	Number of records.

Metric	Description
Select Fields	Number of select fields.

DBD File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Segments	Number of segments.
Source Lines	Number of lines of source, including blank lines and comments.

DDL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Columns	Number of columns.
Foreign Keys	Number of foreign keys.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Primary Keys	Number of primary keys.
Source Lines	Number of lines of source, including blank lines and comments.
Tables	Number of tables.

Device Description File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source.

DMSII DASDL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).

Metric	Description
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

DMSII Database

Metric	Description
Data Sets	Number of data sets.
Disjoint Data Sets	Number of disjoint data sets.
Global Data Items	Number of global data items.
Remapped Data Sets	Number of remapped data sets.
Sets	Number of sets.
Subsets	Number of subsets.

DMSII Dataset

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

DMS DDL File

Metric	Description
Areas	Number of areas.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Records	Number of records.
Sets	Number of sets.
Source Lines	Number of lines of source.

DPS File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source.

ECL Addstream

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

ECL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Number of Executes	Number of executes.
Number of Processor Calls	Number of processor calls.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

FCT File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Source Lines	Number of lines of source, including blank lines and comments.

Fujitsu COBOL Program

Metric	Description
3-Maintainability	$3-MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, and LOC is Lines of Code.
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on): IF, IF DB-EXCEPTION, EVALUATE (number of WHEN except WHEN OTHER), PERFORM...TIMES, PERFORM...UNTIL, PERFORM...VARYING, PERFORM...VARYING...AFTER (number of AFTER phrases + 1), statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID (one decision per statement), GOTO...DEPENDING ON (number of alternatives), SEARCH (number of WHEN, AT END). IDMS: IF.
Computational Statements	Number of statements performing arithmetic calculations: ADD, SUBTRACT, DIVIDE, MULTIPLY, COMPUTE.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions, not including conditional GOTOs. IF, IF DB-EXCEPTION, EVALUATE, SEARCH, PERFORM...UNTIL, PERFORM...VARYING...UNTIL, statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID. IDMS: IF.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields).
Dead Data Elements	Number of dead data elements in programs and used include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data

Metric	Description
	Elements or Dead Statements. Also, source lines containing dead constructs.
Dead Lines from Includes	Number of dead lines in include files. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes. Also, source lines containing dead constructs.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points: PROCEDURE DIVISION, ENTRY.
Error Estimate	$B = E^{**}(2/3) / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All Procedure Division statements, plus CONTINUE and NEXT STATEMENT.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: COBOL=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Inner Call Statements	Number of statements that invoke Inner Procedures: PERFORM procedure-name.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: COBOL paragraphs (including nameless).
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations: OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START, SORT, MERGE, RETURN, RELEASE, ACCEPT, DISPLAY, STOP literal. SQL : INSERT, FETCH, SELECT, UPDATE, DELETE, EXECUTE. CICS : CONVERSE, SEND, SEND MAP, SEND TEXT, RECEIVE, RECEIVE MAP, RECEIVE TEXT, READQ, WRITEQ, DELETEQ, READ, READNEXT, READPREV, WRITE, REWRITE, DELETE. IDMS : ERASE, OBTAIN, GET, MODIFY, STORE. Fujitsu AIM/DB : ERASE, GET, MODIFY, STORE.

Metric	Description
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
LINK Statements	Number of CICS LINK statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions: AND, OR.
Loop Statements	Number of repetitively executing statements: PERFORM...TIMES, PERFORM...UNTIL, PERFORM... VARYING PERFORM...VARYING...AFTER (# of AFTER + 1).
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\text{sqrt}(2.46 * \text{CommentLines} / \text{SourceLines}))$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as COBOL XCTL statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Unique Operators.
Parameters	Number of COBOL Procedure Division USING...RETURNING parameters.
Pointers	Number of data elements declared as pointers. Data items with USAGE IS POINTER, PROCEDURE-POINTER.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.

Metric	Description
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

ICL Form

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source.

IDMS Schema File

Metric	Description
Areas	Number of areas.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Records	Number of records.
Sets	Number of sets.
Source Lines	Number of lines of source, including blank lines and comments.

IDMS Subschema File

Metric	Description
Areas	Number of areas.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Records	Number of logical records.

Metric	Description
Path Groups	Number of path groups.
Records	Number of records.
Sets	Number of sets.
Source Lines	Number of lines of source, including blank lines and comments.

Java Class

Metric	Description
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Catch Clauses	Average Catch Clauses in methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Loop Statements	Average Loop Statements in methods.
Average Maximum Switch Cases	Average Maximum Switch Cases in methods.
Average Nested Block Depth	Average Nested Block Depth of methods.
Average Parameters	Average Parameters in methods.
Average Switch Cases	Average Switch Cases in methods.
Average Switch Statements	Average Switch Statements in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Comments Ratio	Number of Comments divided by Lines of Code.
Constructors	Number of constructors.
Depth of Inheritance Hierarchy	Maximum nesting of the inheritance hierarchy.
Error Estimate	$B = E^{**}(2/3) / 3000$, where E is Programming Effort.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Include Statements	Number of include statements.

Metric	Description
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Interfaces	Number of interfaces.
Lack of Cohesion	$LOCM = (m - \sum(mA)/a1+a2) / (m-1)$, where mA is the number of methods that access class attributes.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Methods	Number of methods.
Number of Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Overridden Methods	Number of overridden methods.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.
Source Lines	Number of lines of source.
Specialization Index	Number of subclasses divided by number of superclasses.
Static Fields	Number of static fields.
Static Methods	Number of static methods.
Subtypes	Number of subtypes.

Metric	Description
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.
Weighted Methods	Number of weighted methods.

Java File

Metric	Description
Abstract Classes	Number of abstract classes and interfaces.
Abstract Methods	Number of abstract methods.
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Catch Clauses	Average Catch Clauses in methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Depth of Inheritance Hierarchy	Average Depth of Inheritance Hierarchy of classes and interfaces.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Lack of Cohesion Per Type	Average Lack of Cohesion of classes and interfaces.
Average Loop Statements	Average Loop Statements in methods.
Average Maximum Switch Cases	Average Maximum Switch Cases in methods.
Average Nested Block Depth	Average Nested Block Depth of methods.
Average Parameters	Average Parameters in methods.
Average Specialization Index Per Type	Average Specialization Index of classes and interfaces.
Average Switch Cases	Average Switch Cases in methods.
Average Switch Statements	Average Switch Statements in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Comments Ratio	Number of Comments divided by Lines of Code.
Constructors	Number of constructors.

Metric	Description
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Import Statements	Number of import statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Interfaces	Number of interfaces.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Methods	Number of methods.
Number of Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Overridden Methods	Number of overridden methods.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.

Metric	Description
Source Lines	Number of lines of source.
Static Fields	Number of static fields.
Static Methods	Number of static methods.
Subtypes	Number of subtypes.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.
Weighted Methods	Number of weighted methods.

Java Interface

Metric	Description
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Catch Clauses	Average Catch Clauses in methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Loop Statements	Average Loop Statements in methods.
Average Maximum Switch Cases	Average Maximum Switch Cases in methods.
Average Nested Block Depth	Average Nested Block Depth of methods.
Average Parameters	Average Parameters in methods.
Average Switch Cases	Average Switch Cases in methods.
Average Switch Statements	Average Switch Statements in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Comments Ratio	Number of Comments divided by Lines of Code.
Constructors	Number of constructors.
Depth of Inheritance Hierarchy	Maximum nesting of the inheritance hierarchy.
Error Estimate	$B = E^{**}(2/3) / 3000$, where E is Programming Effort.

Metric	Description
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Interfaces	Number of interfaces.
Lack of Cohesion	$LOCM = (m - \sum(mA)/a1+a2) / (m-1)$, where mA is the number of methods that access class attributes.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Methods	Number of methods.
Number of Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Overridden Methods	Number of overridden methods.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.

Metric	Description
Response for Class	Response for class.
Source Lines	Number of lines of source.
Specialization Index	Number of subclasses divided by number of superclasses.
Static Fields	Number of static fields.
Static Methods	Number of static methods.
Subtypes	Number of subtypes.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.
Weighted Methods	Number of weighted methods.

Java Method

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Catch Clauses	Number of catch clauses.
Comments Ratio	Number of Comments divided by Lines of Code.
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Constructors	Number of constructors.
Cyclomatic Complexity	$v(G) = e - n + 2$, where v(G) is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). v(G) for reduced graph without D-structured primes.

Metric	Description
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Switch Cases	Maximum Switch Cases.
Nested Block Depth	Maximum nesting of IF, CHOOSE, TRY, DO, SWITCH, or FOR constructs. 1 is added to the depth in the IF and ELSE parts of IF statements, the bodies of loops, and the code in each case of a Choose statement.
Number of Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.

Metric	Description
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Switch Cases	Number of SWITCH cases.
Switch/Choose Statements	Number of SWITCH in Java, CHOOSE statements. in PowerBuilders.
Unique Method Calls	Number of distinct method calls.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

Java Package

Metric	Description
Abstract Classes	Number of abstract classes and interfaces.
Abstract Methods	Number of abstract methods.
Abstractness	Ratio of Number of Abstract Classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.
Afferent Coupling	Number of classes outside the package that depend on classes inside the package. An indicator of the package's responsibility.
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Catch Clauses	Average Catch Clauses in methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Depth of Inheritance Hierarchy	Average Depth of Inheritance Hierarchy of classes and interfaces.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Lack of Cohesion Per Type	Average Lack of Cohesion of classes and interfaces.
Average Loop Statements	Average Loop Statements in methods.
Average Maximum Switch Cases	Average Maximum Switch Cases in methods.

Metric	Description
Average Nested Block Depth	Average Nested Block Depth of methods.
Average Parameters	Average Parameters in methods.
Average Specialization Index Per Type	Average Specialization Index of classes and interfaces.
Average Subtypes	Average Subtypes in methods.
Average Switch Cases	Average Switch Cases in methods.
Average Switch Statements	Average Switch Statements in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Comments Ratio	Number of Comments divided by Lines of Code.
Constructors	Number of constructors.
Efferent Coupling	Number of classes inside the package that depend on classes outside the package. An indicator of the package's independence.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Import Statements	Number of import statements.
Include Statements	Number of include statements.
Instability	Ratio of Efferent Coupling (Ce) to total coupling (Ce + Ca), such that $I = Ce / (Ce + Ca)$. An indicator of the package's resilience to change.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Interfaces	Number of interfaces.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.

Metric	Description
Methods	Number of methods.
Normalized Distance	$ A + I - 1 $, where A is the ratio of abstract types to all types defined in the package and I is Instability. An indicator of the package's balance between abstractness and stability.
Number of Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Overridden Methods	Number of overridden methods.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.
Source Lines	Number of lines of source.
Static Fields	Number of static fields.
Static Methods	Number of static methods.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.
Weighted Methods	Number of weighted methods.

JCL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).

Metric	Description
Control Cards Usages	Number of DD statements referencing control cards identified in Legacy.xml to generate program to control card relationships.
EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures.
EXEC In-stream Procedure Steps	Number of EXEC statements invoking instream procedures.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.
Steps	Number of top-level steps in the job (not including steps in invoked procedures).
Total EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures in the job and invoked procedures. If a procedure is invoked multiple times, it is counted each time.
Total Include Statements	Number of include statements in the job, invoked procedures, and any include files.

JCL Procedure

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Control Cards Usages	Number of DD statements referencing control cards identified in Legacy.xml to generate program to control card relationships.
EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures.
EXEC In-stream Procedure Steps	Number of EXEC statements invoking instream procedures.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.
Steps	Number of steps in the procedure.

Job

Metric	Description
Steps	Number of steps in the job and invoked procedures. If a procedure is invoked multiple times, it is counted each time.

Library Description File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Macro File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Map

Metric	Description
Hidden Fields	Number of hidden fields.
Input Fields	Number of input fields.
Input/Output Fields	Number of input/output fields.
Output Fields	Number of output fields.

MFS File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of include statements in the file and any used include files.

MFS Include File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Natural Data Area

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Natural DDM File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Fields of Adabas File Number	Number of fields of Adabas file number.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Natural File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Global Data Area Statements	Number of included global data area statements.
Include Local Data Area Statements	Number of included local data area statements.
Include Parameter Data Area Statements	Number of included parameter data area statements.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

Natural Include File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Natural Map

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Maps	Number of maps.
Source Lines	Number of lines of source.

Natural Program

Metric	Description
3-Maintainability	$3-MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, and LOC is Lines of Code.
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions.

Metric	Description
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields).
Dead Data Elements	Number of dead data elements in programs and used include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data Elements or Dead Statements.
Dead Lines from Includes	Number of dead lines in include files. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where $n1$ is Unique Operators, $N2$ is Operands, and $n2$ is Unique Operands.
Entry Points	Number of program entry points.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All statements, except DEFINE DATA and IGNORE.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Inner Call Statements	Number of statements that invoke Inner Procedures.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: Natural inline subroutines.

Metric	Description
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
LINK Statements	Number of CICS LINK statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as Cobol XCTL statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of Natural PARAMETER and PARAMETER USING parameters.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.
Returning Calls	Number of returning calls.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.

Metric	Description
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

.NET File

Metric	Description
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Catch Clauses	Average Catch Clauses in methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Conditional Statements	Average Conditional Statements of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Loop Statements	Average Loop Statements in methods.
Average Parameters	Average Parameters in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: COBOL=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOS.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.

Metric	Description
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.
Methods	Number of methods.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Source Lines	Number of lines of source.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

.NET Method

Metric	Description
Absolute Complexity	Absolute Complexity of methods.
Binary Decisions	Binary Decisions of methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Catch Clauses	Number of catch clauses.
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where $n1$ is Unique Operators, $N2$ is Operands, and $n2$ is Unique Operands.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines}/\text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Source Lines	Number of lines of source.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Method Calls	Number of distinct method calls.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

.NET Project

Metric	Description
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.

Metric	Description
Average Catch Clauses	Average Catch Clauses of methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Conditional Statements	Average Conditional Statements of methods.
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Depth of Inheritance Hierarchy	Average Depth of Inheritance Hierarchy of classes and interfaces.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Loop Statements	Average Loop Statements in methods.
Average Parameters	Average Parameters in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOS.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is

Metric	Description
	Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.
Methods	Number of methods.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Source Lines	Number of lines of source.
Static Fields	Number of static fields.
Subtypes	Number of subtypes.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

.NET Type

Metric	Description
Average Absolute Complexity	Average Absolute Complexity of methods.
Average Binary Decisions	Average Binary Decisions of methods.
Average Computational Statements	Average Computational Statements of methods.
Average Conditional Complexity	Average Conditional Complexity of methods.
Average Conditional Statements	Average Conditional Statements of methods.

Metric	Description
Average Cyclomatic Complexity	Average Cyclomatic Complexity of methods.
Average Essential Complexity	Average Essential Complexity of methods.
Average Extended Cyclomatic Complexity	Average Extended Cyclomatic Complexity of methods.
Average IF Statements	Average IF Statements in methods.
Average Loop Statements	Average Loop Statements in methods.
Average Parameters	Average Parameters in methods.
Average Unique Method Calls	Average Unique Method Calls in methods.
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Catch Clauses	Number of catch clauses.
Depth of Inheritance Hierarchy	Maximum nesting of the inheritance hierarchy.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Maximum Depth of Logic Nesting	Maximum nesting of logic.
Maximum Switch Cases	Maximum Switch Cases.
Member Function Calls	Number of member function calls.

Metric	Description
Methods	Number of methods.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Source Lines	Number of lines of source.
Static Fields	Number of static fields.
Subtypes	Number of subtypes.
Switch Cases	Number of SWITCH cases.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

PCT File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

PL/I File

Metric	Description
Blank Lines	Number of blank lines of source. Blank lines in a comment block are not included.
Comment Lines	Number of lines of source containing comments only and no code. Includes blank lines in a comment block.
Include Statements	Number of include statements: %INCLUDE, EXEC SQL INCLUDE.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Macro Assignments	Number of assignments to global macro variables outside any macro procedure.
Macro Declarations	Number of global macro variables. Macro variables within macro procedures are not counted.
Macro Lines	Number of lines for include statements, declarations, macro procedures, and other %-statements, excluding macro invocations. Blank lines and comments lines are ignored.
Macro Procedures	Number of macro procedures declared in the file.
Macro Statements	Number of %-statements and statements inside a macro procedure, excluding macro invocations.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of include statements in the file and any used include files.

PL/I Include File

Metric	Description
Blank Lines	Number of blank lines of source. Blank lines in a comment block are not included.
Comment Lines	Number of lines of source containing comments only and no code. Includes blank lines in a comment block.
Include Statements	Number of include statements: %INCLUDE, EXEC SQL INCLUDE.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Macro Assignments	Number of assignments to global macro variables outside any macro procedure.
Macro Declarations	Number of global macro variables. Macro variables within macro procedures are not counted.
Macro Lines	Number of lines for include statements, declarations, macro procedures, and other %-statements, excluding macro invocations. Blank lines and comments lines are ignored.

Metric	Description
Macro Procedures	Number of macro procedures declared in the file.
Macro Statements	Number of %-statements and statements inside a macro procedure, excluding macro invocations.
Source Lines	Number of lines of source, including blank lines and comments.

PL/I Program

Metric	Description
3-Maintainability	$3-MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, and LOC is Lines of Code.
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields). The implicit variable DFHEIBLK for CICS statements is counted as a data element.
Dead Data Elements	Number of declared data items in dead internal procedures.
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data Elements or Dead Statements.
Dead Lines from Includes	Number of dead lines in include files. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution. DECLARE statements are not calculated as dead statements.

Metric	Description
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All statements, except BEGIN, DECLARE, DO (block), END, ENTRY, PACKAGE, and PROCEDURE.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOS.
Inner Call Statements	Number of statements that invoke Inner Procedures.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: inner PL/I procedures.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
LINK Statements	Number of CICS LINK statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as Cobol XCTL statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.

Metric	Description
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of PL/I PROCEDURE parameters.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.
Returning Calls	Number of returning calls.
Sliceable Dead Lines	Number of Dead Lines that can be sliced using the Application Architect Dead Code Elimination method. Dead procedures containing either preprocessor statements or statements subject to macro preprocessor replacement are not counted as sliceable dead lines, since it is not always possible to determine whether they can be safely removed. Lines from include files are not counted.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

PSB Copybook File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

PSB File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Number of PCBs	Number of PCBs.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of include statements in the file and any used include files.

RPG Copybook File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

RPG File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

System Definition File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.

Metric	Description
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Unisys 2200 Program

Metric	Description
3-Maintainability	$3-MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, and LOC is Lines of Code.
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on): IF, EVALUATE (number of WHEN except WHEN OTHER), PERFORM...TIMES, PERFORM...UNTIL, PERFORM... VARYING, PERFORM...VARYING...AFTER (number of AFTER phrases + 1), statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID (one decision per statement), GOTO...DEPENDING ON (number of alternatives), SEARCH (number of WHEN, AT END). CDML: IF. IDMS: IF.
Computational Statements	Number of statements performing arithmetic calculations: ADD, SUBTRACT, DIVIDE, MULTIPLY, COMPUTE.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions, not including conditional GOTOs. IF, EVALUATE, SEARCH, PERFORM...UNTIL, PERFORM...VARYING...UNTIL, statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID. CDML: IF. IDMS: IF.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields).
Dead Data Elements	Number of dead data elements in programs and used include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.

Metric	Description
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data Elements or Dead Statements. Also, source lines containing dead constructs.
Dead Lines from Includes	Number of dead lines in include files. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes. Also, source lines containing dead constructs.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points: PROCEDURE DIVISION, ENTRY.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). $v(G)$ for reduced graph without D-structured primes.
Executable Statements	All Procedure Division statements, plus CONTINUE and NEXT STATEMENT.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Inner Call Statements	Number of statements that invoke Inner Procedures: PERFORM procedure-name.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: Cobol paragraphs (including nameless).
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations: OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START, SORT, MERGE, RETURN, RELEASE, ACCEPT, DISPLAY, STOP literal. SQL : INSERT, FETCH, SELECT, UPDATE, DELETE, EXECUTE. CICS : CONVERSE, SEND, SEND MAP, SEND TEXT, RECEIVE, RECEIVE MAP, RECEIVE TEXT, READQ, WRITEQ, DELETEQ, READ, READNEXT, READPREV, WRITE, REWRITE, DELETE. CDML : OPEN, CLOSE, FETCH, GET,

Metric	Description
	DELETE, MODIFY, STORE, INSERT, REMOVE. IDMS: ERASE, OBTAIN, GET, MODIFY, STORE.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
LINK Statements	Number of CICS LINK statements.
Logical Operators in Conditions	Number of binary logical operators used in conditions: AND, OR.
Loop Statements	Number of repetitively executing statements: PERFORM...TIMES, PERFORM...UNTIL, PERFORM... VARYING PERFORM...VARYING...AFTER (# of AFTER + 1).
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as Cobol XCTL statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Unique Operators.
Parameters	Number of Cobol Procedure Division USING...RETURNING parameters.
Pointers	Number of data elements declared as pointers. Data items with USAGE IS POINTER, PROCEDURE-POINTER.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Response for Class	Response for class.

Metric	Description
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

VALTAB File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Number of Entries	Number of entries.
Source Lines	Number of lines of source.

VB Class

Metric	Description
Destructors	Number of destructors.
Error Estimate	$B = E^{2/3} / 3000$, where E is Programming Effort.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Fields	Number of fields.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
Inherited Methods	Number of inherited methods.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Methods	Number of methods.

Metric	Description
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Public Fields	Number of public fields.
Public Methods	Number of public methods.
Static Fields	Number of static fields.
Static Methods	Number of static methods.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

VB File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

VB Function

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.

Metric	Description
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where $n1$ is Unique Operators, $N2$ is Operands, and $n2$ is Unique Operands.
Error Estimate	$B = E^{**}(2/3) / 3000$, where E is Programming Effort.
Exception Handling Statements	Number of exception handling statements.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Labels	Number of labels.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Local Data Items	Number of local data items.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\sqrt{2.46 * \text{CommentLines} / \text{SourceLines}})$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Method Calls	Number of method calls.

Metric	Description
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Static Data Items	Number of static data items.
SWITCH/SELECT Statements	Number of SWITCH statements.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

VB Library

Metric	Description
Classes	Number of classes.
Functions	Number of functions.

VB Method

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.

Metric	Description
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on).
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions, plus Unique Operands in Conditions.
Cyclomatic Complexity	$v(G) = e - n + 2$, where $v(G)$ is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). $v(G) = DE + 1$, where DE is the number of binary decisions made in the program.
Difficulty	$D = (n1 / 2) * (N2 / n2)$, where $n1$ is Unique Operators, $N2$ is Operands, and $n2$ is Unique Operands.
Error Estimate	$B = E^{**}(2/3) / 3000$, where E is Programming Effort.
Exception Handling Statements	Number of exception handling statements.
Executable Statements	All assignments, function calls (alone on a line), calls, returns, IF, DO, FOR, CHOOSE, EXIT, CONTINUE, and GOTO statements.
Extended Cyclomatic Complexity	Cyclomatic Complexity, plus Logical Operators in Conditions. Number of all paths in the program.
Function Calls	Number of function calls.
Function Points	Lines of Code divided by K, where K depends on the language: Cobol=77, Natural=52, PL/I=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
IF Statements	Number of IF statements.
Intelligent Content	$I = L * V$, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
Labels	Number of labels.
Lines of Code	Number of lines of code, plus the number of lines of code in included files and any files they include. Comments and blank lines are not counted.
Local Data Items	Number of local data items.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements.
Maintainability Index	$MI = 171 - 5.2 * \ln(\text{PgmVolume}) - 0.23 * \text{ExtCycComp} - 16.2 * \ln(\text{LOC}) + 50 * \sin(\text{sqrt}(2.46 * \text{CommentLines} / \text{SourceLines}))$, where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Method Calls	Number of method calls.

Metric	Description
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations. Compare Unique Operators.
Parameters	Number of parameters in method calls.
Pointers	Number of data elements declared as pointers.
Pointers	Number of data elements declared as pointers.
Program Length	$N = N1 + N2$, where N1 is Operators and N2 is Operands.
Program Level	$L = 1 / D$, where D is Difficulty.
Program Volume	$V = N * \log_2(n)$, where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	$E = V / L$, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	$T = E / 18$, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
SELECT (SWITCH) Statements	Number of SWITCH statements.
Static Data Items	Number of static data items.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	$n = n1 + n2$, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

VB Project File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

WFL File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.
Total Include Statements	Total Include Statements.

WFL Include File

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source.

Relationship Projections

Relationship Projections Overview

The tables in this section describe the relationships COBOL Analyzer generates for statements in each of the major supported languages. The tables are listed alphabetically by language.



Note: CA uses a generic "Program" object for programs written in different languages.

Assembler Statements

Program Relationship Projections

Explicitly defined calls (using the CALL macro or the commands described in "Using the External Name in a Call" below) are autoresolved at verification. Otherwise, decisions for external calls need to be resolved manually.

Getting the External Name

The following Assembler statements prepare the value of the register with an external name or memory address to which a subsequent call may be resolved.

Example	Type	External in
L reg, =A(external)	command	Reg

Example	Type	External in
L reg, =V(external)	command	reg
LOAD EP=external	system macro	register 0
LOAD EP=name, LOADPT=external	system macro	register 0
name DC =V(external) L reg, name	command	reg

Using the External Name in a Call

The following Assembler statements create calling relationships from the program object to a program entry point or program entry decision.

Example	Type	Relationship	Entry Point
CALL reg	system macro	Program Calls Program Entry Point Program Calls Program Entry Decision	reg
CALL name	system macro	Program Calls Program Entry Point Program Calls Program Entry Decision	name
BAR reg., reg.	command	Program Calls Program Entry Point Program Calls Program Entry Decision	reg.
BAR reg., reg.	command	Program Calls Program Entry Point Program Calls Program Entry Decision	reg.
BASS reg., reg.	command	Program Calls Program Entry Point Program Calls Program Entry Decision	reg.

Defining Program Entry Point

The following Assembler statements create Has Program Entry Point relationships for the program.

Example	Type	Relationship	Entry Point
name SECT.	system macro	Program Has Program Entry Point	name
ENTRY name	system macro	Program Has Program Entry Point	name

BMS Statements

BMS File Relationship Projections

Statement	Format	Relationship	Entities
COPY	COPY member	BMS File Includes BMS Copybook File	For resolved files: BmsCopy.Name = <resolved-name> For unresolved files: BmsCopy.Name = <member>
DFHMDI	mapset DFHMSD... map DFHMDI ...	BMS File Defines Screens	Map.Name= <mapset>.<map>
DFHMDI (no mapset)	map DFHMDI ...	BMS File Defines Screens	Map.Name = <source- filename- without- extension>.<map>

BMS Copybook File Relationship Projections

Statement	Format	Relationship	Entities
COPY	COPY member	BMS Copybook File Includes BMS Copybook File	For resolved files: BmsCopy.Name = <resolved-name> For unresolved files: BmsCopy.Name= <member>

C/C++ Statements

C File Relationship Projections

Statement	Format	Relationship	Entities
INCLUDE	#include "<header_name>"	C File Includes Header	Header.Name=<header_name>
DEFINE FUNCTION	<function_name>([list of parameters]);	C File Defines Function	Function.Name=<function_name>

C++ File Relationship Projections

Statement	Format	Relationship	Entities
INCLUDE	#include "<header_name>"	C++ File Includes Header	Header.Name=<header_file>
DEFINE FUNCTION	<function_name>([list of parameters]);	C++ File Defines Function	Function.Name=<function_name>
DEFINE MEMBER FUNCTION	<member_function_name>([list of parameters]);	C++ File Defines Member Function	MemberFunction.Name=<member_function_name>

Statement	Format	Relationship	Entities
DEFINE CLASS	<code>class <class_name> { };</code>	Class Definition	Class.Name=<class_name>
MEMBER FUNCTION	<code>[<result_type> <member_function_name> ([list of parameters]);</code>	Member Function	MemberFunction.Name=<member_function_name>

Header File Relationship Projections

Statement	Format	Relationship	Entities
INCLUDE	<code>#include "<header_name>"</code>	Header Includes Header	Header.Name=<header_name>

Statement	Format	Relationship	Entities
DEFINE FUNCTION	<code><function_name>([list of parameters]);</code>	Header Defines Function	Function.Name=<function_name>
DEFINE CLASS	<code>class <class_name> { };</code>	Header Defines Class	Class.Name=<class_name>
MEMBER FUNCTION	<code>[<result_type>] <member_function_name> ([list of parameters]);</code>	Header Defines Member Function	MemberFunction.Name=<member_function_name>

Class Relationship Projections

Statements	Format	Relationships
INHERITS	<code>class <class_name1>: [public] <class_name2> { };</code>	Class 1.Name =<class_name1> > Class 2.Name =<class_name2> > Class
Any mention of a CLASS that is not a call of its member function	<code>class <class_name1> {</code>	Class 1.Name =<class_name1>

Statements	Format	Relationships
	<pre><class_name2> variable; }</pre>	<p>class_name1 D e P2.Name e=<cl n class_n ame2 s O n C l a s s</p>
<p>*Any mention of a function that is not a call*</p>	<pre>class <class_name> { Variable = <function_name>; }</pre>	<p>Class. l Name a=<clas ss_n ame> D Functi e on.Na p me=<f unctio n_n ame> s O n F u n c t i o n</p>
<p>*Any expression in a position of a class or a function that is not a call*</p>	<pre>class <class_name> { <expression> }</pre>	<p>Class. l Name a=<clas ss_n ame> D Decisi e on.Na p me=<i nterna n</p>

Statements	Format	Relationships
CALL	<pre>class <class_name> { <function_name>([list of parameters]); }</pre>	d_l_name O D e c i s i o n CClass. l Name a=<class_name> C Functi on.Name l me=<function_name> s n_name F u n c t i o n
CALL	<pre>class <class_name> { <member_function_name>([list of parameters]); }</pre>	CClass. l Name a=<class_name> C Functi on.Name l me=<function_name> s n_name M e m b e r F u


Statements	Format	Relationships
CALL	<pre>class <class_name> { <expression> }</pre>	<p>nc t i o n</p> <p>Class. Name a=<class_name></p> <p>Decision. Name Name=<internal_name></p> <p>Decision</p>
DEFINITION	<pre>class <class_name> { [result_type] <member_function_name>([list of parameters]){ } }</pre>	<p>Class. Name a=<class_name></p> <p>MemberFunction. Name Name=<member_function_name></p> <p>Function</p>

Function Relationship Projections


Statement	Format	Relationship	Entities
CALL	<code><function_name>([list of parameters]);</code>	Function Calls Function	Function.Name=<function_name>
CALL	<code><member_function_name>([list of parameters]);</code>	Function Calls Member Function	MemberFunction.Name=<member_function_name>
CALL	<code><expression>;</code>	Function Calls Decision	Decision.Name=<internal name>
any other statement	<code><class_name> <variable>;</code>	Function Depends On Class	Class.Name=<class_name>


CICS Statements

Program Relationship Projections


Statement	Format	Relationship	Entities
	<code>any EXEC CICS</code>		Program.EnvFlags = +CICS  Note: EnvFlags may contain other environment codes, so search as follows: Like '+CICS*'
DELETE	<code>DELETE FILE ('file-name') ...</code>	Program Deletes File	File attributes: <ul style="list-style-type: none"> Name = <program-name>.file-name DD Name = file-name File Type = FILE Online Flag = true
DELETE (dynamic)	<code>DELETE FILE (file-name) ...</code>	Program Deletes File	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@<internal-name> # Also Known As = <program-name>.DeletesDataPort.<file-name>


Statement	Format	Re lat io ns hi p	Entities
		e De cis ion	<ul style="list-style-type: none"> Decision Type = DATAPORT...
DOCUMENT	DOCUMENT CREATE TEMPLATE ('name') ... DOCUMENT INSERT TEMPLATE ('name') ...	Pr og ra m Us es Do cu me nt Te mp lat e	DocTemplate.Name = <name>
DOCUMENT (dynamic)	DOCUMENT CREATE TEMPLATE (name) ... DOCUMENT INSERT TEMPLATE (name) ...	Pr og ra m Us es Do cu me nt Te mp lat e De cis ion	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@<internal-name> # Also Known As = <program-name>.UsesDocTemplate<name> Decision Type = DOCTEMPLATE...
INVOKE	INVOKE WEBSERVICE ('name') OPERATION ('opname') ...	Pr og ra m Inv ok es Se rvi ce	Service.Name = <name>.<opname>

Statement	Format	Relationship	Entities
INVOKE (dynamic)	<pre>INVOKE WEBSERVICE (name) OPERATION ('opname') ...</pre> <pre>INVOKE WEBSERVICE ('name') OPERATION (opname) ...</pre> <pre>INVOKE WEBSERVICE (name) OPERATION (opname) ...</pre>	Program Invocation	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@<internal-name> # Also Known As = <program-name>.InvokesService.<name> Decision Type = SERVICE...
LINK	LINK PROGRAM ('pgm-name') ...	Program Links	ProgramEntry.Name = <program-name>  Note: If literal is long, only 8 leading characters are used as program name.
LINK (dynamic)	LINK PROGRAM (pgm-name) ...	Program Links	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@<internal-name> # Also Known As = <program-name>.Links.<program-name> Decision Type = PROGRAMENTRY...
READ READNEXT READPREV	<pre>READ FILE ('file-name') ...</pre> <pre>READNEXT FILE ('file-name') ...</pre> <pre>READPREV FILE ('file-name') ...</pre>	Program Reads	File attributes: <ul style="list-style-type: none"> Name = <program-name>.file-name DD Name = file-name File Type = FILE Online Flag = true

Statement	Format	Relationship	Entities
READ READNEXT READPREV (dynamic)	READ FILE (file-name) ... READNEXT FILE (file-name) ... READPREV FILE (file-name) ...	Program	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@<internal-name> # Also Known As =<program-name>.ReadsDataPort.<file-name> Decision Type = DATAPORT...
RECEIVE	RECEIVE ...		Program.OnlineFlag = true
RECEIVE MAP	RECEIVE MAP ('map-name') MAPSET ('mapset') ... RECEIVE MAP ('map-name')	Program	Screen.Name = <mapset>.<map-name> Program.OnlineFlag = true Screen.Name = <map-name>.<map-name> Program.OnlineFlag = true
RECEIVE MAP (dynamic)	RECEIVE MAP (map-name) MAPSET ('mapset') ... RECEIVE MAP (map-name) RECEIVE MAP ('map-name') MAPSET (mapset) ... RECEIVE MAP (map-name) MAPSET (mapset) ...	Program	Decision attributes: <ul style="list-style-type: none"> Name =<program-name>@ <internal-name> # Also Known As = <program-name>.Receives.<map-name> Decision Type = MAP ... Program.OnlineFlag = true
RETURN	RETURN TRANSID ('name') ...	Program	Transaction.Name = <name>  Note: If literal is long, only 4 leading characters are used as program name.
RETURN	RETURN TRANSID (name) ...	Program	Decision attributes:

Statement	Format	Re lat io ns hi p	Entities
(dynamic)		ra m St art s Tra ns act ion De cis ion	<ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>.Starts.<name> Decision Type = TRANSACTION
REWRITE	REWRITE FILE ('file-name') ...	Pr og ra m Up dat es Fil e	File attributes: <ul style="list-style-type: none"> Name = <program-name>. file-name DD Name = file-name File Type = FILE Online Flag = true
REWRITE (dynamic)	REWRITE FILE (file-name) ...	Pr og ra m Up dat es Fil e De cis ion	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>.UpdatesDataPort.<file-name> Decision Type = DATAPORT...
SEND MAP	SEND MAP ('map-name') MAPSET ('mapset') ... SEND MAP ('map-name')	Pr og ra m Se nd s Sc re en	Screen.Name = <mapset>. <map-name> Program.OnlineFlag = true Screen.Name = <map-name>.<map-name> Program.OnlineFlag = true
SEND MAP (dynamic)	SEND MAP (map-name) MAPSET ('mapset') ... SEND MAP (map-name) SEND MAP ('map-name') MAPSET (mapset) ... SEND MAP (map-name) MAPSET (mapset) ...	Pr og ra m Se nd s Sc re	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>.Sends.<mapset> Decision Type = MAP ...

Statement	Format	Re lat io ns hi p	Entities
		en De cis ion	<ul style="list-style-type: none"> Program.OnlineFlag = true
SEND	SEND		Program.OnlineFlag = true
START	START TRANSID ('name') ...	Pr og ra m St art s Tra ns act ion	Transaction.Name = <name>  Note: If literal is long, only 4 leading characters are used as program name.
START (dynamic)	START TRANSID (name) ...	Pr og ra m St art s Tra ns act ion De cis ion	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Starts.<name> Decision Type = TRANSACTION
WRITE	WRITE FILE ('file-name') ...	Pr og ra m Ins ert s Int o Fil e	File attributes: <ul style="list-style-type: none"> Name = <program-name>. file-name DD Name = file-name File Type = FILE Online Flag = true
WRITE (dynamic)	WRITE FILE (file-name) ...	Pr og ra m Ins ert s Int o Fil	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>. InsertsDataPort. <file-name>

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> Decision Type = DATAPORT...
XCTL	XCTL PROGRAM ('pgm-name') ...	Program Entry Point	ProgramEntry.Name = <pgm-name>  Note: If literal is long, only 8 leading characters are used as program name.
XCTL (dynamic)	XCTL PROGRAM (pgm-name) ...	Program Entry Point Decision	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> # Also Known As = <program-name>.Xctls.<pgm-name> Decision Type = PROGRAMENTRY...

Cobol Statements

Cobol File Relationship Projections

Statement	Format	Relationship	Entities
COPY	COPY member [OF library]	Cobol File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>


Statement	Format	Relationship	Entities
++INCLUDE (Panvalet)	++INCLUDE member	Cobol File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>
-INC (Librarian)	-INC member	Cobol File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>
PROGRAM- ID	PROGRAM-ID. name	Cobol File Defines Program	Program.Name = <name> ...

Cobol Copybook File Relationship Projections

Statement	Format	Relationship	Entities
COPY	COPY member [OF library]	Copybook File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>
++INCLUDE (Panvalet)	++INCLUDE member	Copybook File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>
-INC (Librarian)	-INC member	Copybook File Includes Copybook File	For resolved files: Copybook.Name = <resolved-name> For unresolved files: Copybook.Name = [<library>.]<member>


Program Relationship Projections

Statement	Format	Relationship	Entities
ACCEPT	ACCEPT varname [FROM mnemonic-name]		Program.OnlineFlag. = True
CALL	CALL 'name'	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL (dynamic)	CALL varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name>

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> • #Also Known As = <program-name>. Calls.<varname> • Decision Type = PROGRAMENTRY...
CALL PEM (Hogan)	CALL 'PEM' USING TRANSACTION-CONTROL-BLOCK.	Program Uses Hogan Activity Id	ActivityID.Name = <name>
ENTRY	ENTRY 'name'	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry. MainEntry = True
PROGRAM-ID	PROGRAM-ID.name	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry. MainEntry = True
File Description	<pre>SELECT file-name ASSIGN TO [label-][org-] <name1> [assignment- name2...]</pre> <pre>FD file-name.01 file- record-name..</pre>	See CRUD statements below.	<p>external-name = <name1></p> <p>File attributes:</p> <ul style="list-style-type: none"> • Name = <program-name>. external-file-name • DD Name = external-file-name • File Type = FILE <p> Note: A File object is generated only when the first CRUD statement for the file is encountered. File attributes do not depend on the CRUD statement itself.</p>
DELETE	DELETE file-name...	Program Deletes From File	See File Description for File attributes.
READ	READ file-name...	Program Reads File	See File Description for File attributes.
REWRITE	REWRITE file-record-name...	Program Updates File	See File Description for File attributes.
WRITE	WRITE file-record-name...	Program Inserts Into File	See File Description for File attributes.

CSD Statements

CSD File Relationship Projections

Statement	Format	Relationship	Entities
DEFINE DOCTEMPLATE	DEFINE DOCTEMPLATE (doc-name) GROUP (group-name)...	CSD File Defines Document Template	DocTemplate.Name = <doc-name>
DEFINE FILE	DEFINE FILE file-name) GROUP (group-name) DSNAME (dsn-name) LOAD(YES)...	CSD File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: <ul style="list-style-type: none"> Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>
DEFINE FILE (base file)	NSRGROUP (base-group) Base file: DEFINE FILE (base-group) DSNAME (base-dsn-name)	Data Store Based On Data Store	BaseDatastore.Name = <base-dsn-name>
DEFINE TRANSACTION	DEFINE TRANSACTION (tran-name) GROUP (group-name) PROGRAM (prg-name)...	CSD File Defines Transaction Transaction Initiates Program Entry Point	Transaction.Name = <tran-name> ProgramEntry.Name = <prg-name> Program.Root = True  Note: Root is empty if Program does not exist in the repository before CSD file verification.

DBD Statements

DBD File Relationship Projections

Statement	Format	Relationship	Entities
DBD	DBD NAME=db-name ACCESS= (db-type, ...) ...	DBD File Defines Hierarchical Database	HiDatabase.Name = <db-name> HiDatabase.Type = <db-type>
DATASET (GSAM only)	DATASET DD1=dd-name1, DD2=dd-name2, ...	Hierarchical Database Has Hierarchical Database Segment	HiSegment.Name = <db-name>. <db-name> HiSegment.Segment Name = <db-name> HiSegment.DDName = <dd-name1>

Statement	Format	Relationship	Entities
			HiSegment. DDName2 = <dd-name2>
SEGM	SEGM NAME=seg-name, ... NAME= seg-name2, ... dbname2))	Hierarchical Database Has Hierarchical Database Segment Hierarchical Database Segment Has Logical Child Hierarchical Database Segment	HiSegment.Name = <db-name>. <seg-name> HiSegment.Segment Name = <seg-name> HiSegment.Name = <db-name>. <seg-name> HiSegmentChild. Name = <db-name2> .<seg-name2> HiSegmentChild. SegmentName = <seg-name2>
LCHILD	SEGM NAME= seg-name, ... LCHILD= NAME= (seg-name, dbname)	Hierarchical Database Segment Has Logical Child Hierarchical Database Segment	HiSegment.Name = <db-name>. <seg-name> HiSegmentChild. Name = <db-name> .<seg-name> HiSegmentChild. SegmentName = <seg-name>

EXEC SQL Statements


EXEC SQL Program Relationship Projections


Statement	Format	Relationship	Entities
ALTER TABLE	ALTER TABLE <table-name> ...	Program Manipulates Table	Table.Name = <table-name>
CALL	EXEC SQL CALL <procedure-name> ...	PROGRAM Calls Program Entry Point	Program Entry Point: <ul style="list-style-type: none"> Name = <procedure-name> MainEntry = false
CREATE INDEX	CREATE INDEX <index-name> ON <table-name> ...	Program Manipulates Table	Table.Name = <table-name>
CREATE TABLE	CREATE TABLE <table-name>... [IN [DATABASE] [<database>.] <tablespace>]	Program Manipulates Table	Table attributes: <ul style="list-style-type: none"> Database TableSpace Name = <table-name> Origin = <source-file-path>
CREATE PROCEDURE	CREATE [OR REPLACE] PROCEDURE <procedure-name> ...	Program Defines Stored Procedure	Program Entry Point: <ul style="list-style-type: none"> Name = <procedure-name>

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> MainEntry = false
DELETE	DELETE FROM <table-name>...	Program Deletes From Table	Table.Name = <table-name>
DROP TABLE	DROP TABLE <table-name>	Program Manipulates Table	Table.Name = <table-name>
INSERT	INSERT INTO <table-name> ...	Program Inserts Into Table	Table.Name = <table-name>
SELECT	SELECT ... FROM table-name	Program Reads Table	Table.Name = <table-name>
UPDATE	UPDATE <table-name> ...	Program Updates Table	Table.Name = <table-name>

FCT Statements

FCT File Relationship Projections

Statement	Format	Relationship	Entities
DFHFCT DATASET	DFHFCT TYPE = DATASET, DATASET = data-set, DSNAME = dsn-name...	FCT File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: <ul style="list-style-type: none"> Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>
DFHFCT DATASET (base file)	BASE = base-name Base file: DFHFCT TYPE = DATASET, DATASET = base-name, SNAME = base-dsn-name...  Note: Base file may be defined with any TYPE = FILE or TYPE = DATASET statement.	Data Store Based On Data Store	BaseDatastore.Name = <base-dsn-name>
DFHFCT FILE	DFHFCT TYPE = FILE, FILE = file-name, DSNAME = dsn-name...	FCT File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: <ul style="list-style-type: none"> Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>

Statement	Format	Relationship	Entities
DFHFCT FILE (base file)	<pre>BASE = base-name</pre> <pre>Base file: DFHFCT</pre> <pre>TYPE = FILE,</pre> <pre>FILE = base-name,</pre> <pre>DSNAME = base-dsn-</pre> <pre>name...</pre> <p> Note: Base file may be defined with any TYPE=FILE or TYPE=DATASET statement.</p>	Data Store Based On Datastore	BaseDatastore.Name = <base-dsn-name>

IDMS DML Statements

Cobol File Relationship Projections

Statement	Format	REntities
COPY IDMS	[level-number] COPY IDMS name	e l a t i o n s h i p CWhere member-name = o <name>: b o l f i e l n c l u d e s C o p y b o o k F i

Statement	Format	REntities e l a t i o n s h i p
		l e
COPY IDMS (file module)	COPY IDMS [FILE MODULE] name	CWhere member-name = o <name>: b o • For resolved files: l Copybook.Name = F <member-name> [.ext]... i • For unresolved files: l Copybook.Name = e <member-name> l n c l u d e s C o p y b o o k F i l e
COPY IDMS (record)	[level-number] COPY IDMS RECORD rec- name	CWhere member-name = o <rec-name>: b o • For resolved files: l Copybook.Name = F <member-name> [.ext]... i • For unresolved files: l Copybook.Name = e <member-name> l n c l u d e s

Statement	Format	R Entities e l a t i o n s h i p
		C o p y b o o k F i l e
COPY IDMS (subschema)	[level-number] COPY IDMS SUBSCHEMA- name	CWhere member-name = o <schema-name> b \$<subschema-name> o \$<name>: l F• For resolved files: i Copybook.Name = l <member-name> [.ext]... e• For unresolved files: l Copybook.Name = n <member-name> c l u d e s C o p y b o o k F i l e
COPY IDMS (subschema- ctrl)	[level-number] COPY IDMS SUBSCHEMA- CTRL [level-number] COPY IDMS SUBSCHEMA- LR- CTRL	CWhere member-name = o SUBSCHEMA-CTRL or b member-name = o SUBSCHEMA-LR- CTRL: l F i

Statement	Format	REntities e l a t i o n s h i p
		l e l n c l u d e s C o p y b o o k F i l e <ul style="list-style-type: none"> • For resolved files: Copybook.Name = <member-name> [.ext]... • For unresolved files: Copybook.Name = <member-name>
IDMS- CONTROL SECTION	IDMS- CONTROL SECTION... [IDMS -RECORDS WITHIN [WORKING- STORAGE LINKAGE]]	CWhere member-name = o SUBSCHEMA- CTRL or b member-name = <schema- o name> \$<subschema- l name> \$SUBSCHEMA- F RECORDS: i l e l n c l u d e s C o p y b o o k F <ul style="list-style-type: none"> • For resolved files: Copybook.Name = <member-name> [.ext]... • For unresolved files: Copybook.Name = <member-name>

Statement	Format	Relationships
		Relationship

Cobol Copybook File Relationship Projections

Statement	Format	Relationship	Entities
COPY IDMS	[level-number] COPY IDMS name	Copybook File Includes Copybook File	Where member-name = <name>: <ul style="list-style-type: none"> For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (file module)	COPY IDMS [FILE MODULE] name	Copybook File Includes Copybook File	Where member-name = <name>: <ul style="list-style-type: none"> For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (record)	[level-number] COPY IDMS RECORD rec-name	Copybook File Includes Copybook File	Where member-name = <rec-name>: <ul style="list-style-type: none"> For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (subschema)	[level-number] COPY IDMS SUBSCHEMA- name	Copybook File Includes Copybook File	Where member-name = <schema-name> \$<subschema-name> \$<name>: <ul style="list-style-type: none"> For resolved files: Copybook.Name = <member-name> [.ext]...

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> For unresolved files: Copybook.Name = <member-name>
COPY IDMS (subschema-ctrl)	[level-number] COPY IDMS SUBSCHEMA-CTRL [level-number] COPY IDMS SUBSCHEMA-LR-CTRL	Copybook File Includes Copybook File	Where member-name = SUBSCHEMA-CTRL or member-name = SUBSCHEMA-LR-CTRL: <ul style="list-style-type: none"> For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>

Program Relationship Projections

Statement	Format	Relationship	Entities
ERASE	ERASE record-name.	Program Deletes Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
GET	GET record-name.	Program Reads Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
MODIFY	MODIFY record-name.	Program Updates Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
OBTAIN	OBTAIN ... [CALC DUPLICATE] record-name. OBTAIN ... CURRENT record-name. OBTAIN ... record-name DB-KEY IS db-key. OBTAIN ... record-name WITHIN [set-name area-name].	Program Reads Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
SCHEMA SECTION	SCHEMA SECTION. DB subschema-name	Program Uses Network Database Schema	NetSchema.Name = <schema-name>

Statement	Format	Relationship	Entities
	WITHIN schema-name.		
STORE	STORE record-name.	Program Inserts Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>

IDMS Schema Statements

IDMS Schema File Relationship Projections

Statement	Format	Relationship	Entities
RECORD	ADD RECORD NAME IS recordName ...	Network Database Schema Has Network Database Record	NetRecord.Name = <schemaName>. <recordName> NetRecord.RecordName = <recordName>
SCHEMA	ADD SCHEMA NAME IS schemaName ...	IDMS Schema File Defines Network Database Schema	NetSchema.Name = <schemaName>

Java Statements

Java Annotation Relationship Projections

Statement	Format	Relationship	Entities
DEPENDS ON JAVAANNOTATION	@Annotation2 @interface Annotation1 { }	Java Annotation Depends On Java Annotation	Annotation1 depends on Annotation2
DEPENDS ON JAVACLASS	@interface Annotation1 { Class1 c(); }	Java Annotation Depends On Java Class	Annotation1 depends on Class1
DEPENDS ON JAVAENUMERATION	@interface Annotation1 { Enum1 e(); }	Java Annotation Depends On Java Enumeration	Annotation1 depends on Enum1
HAS JAVAANNOTATION	@interface Annotation1 { @interface Annotation2 { } }	Java Annotation Has Java Annotation	Annotation1 has Annotation2
HAS JAVACLASS	@interface Annotation1 { class Class1	Java Annotation Has Java Class	Annotation1 has Class1

Statement	Format	Relationship	Entities
	<pre>{ } }</pre>		
HAS JAVAENUMERATION	<pre>@interface Annotation1 { enum Enum1 { } }</pre>	Java Annotation Has Java Enumeration	Annotation1 has Enum1
HAS JAVAINTERFACE	<pre>@interface Annotation1 { interface Interface1 { } }</pre>	Java Annotation Has Java Interface	Annotation1 has Interface1

Java Class Relationship Projections

Statement	Format	Relationship	Entities
CONTAINS JAVACLASS	<pre>class Class1 { class Class2 { } }</pre>	Java Class Has Java Class	Class1 has Class2
CONTAINS JAVAINTERFACE	<pre>class Class1 { interface Interface1 { } }</pre>	Java Class Has Java Interface	Class1 has Interface1
DEPENDS ON JAVAANNOTATION	<pre>@Annotation1 class Class1 { }</pre>	Java Class Depends On Java Annotation	Class1 depends on Annotation1
DEPENDS ON JAVACLASS	<pre>class Class1 extends Class2 { Class3 c3; Class4 method1(Class5 c5) { Class6 c6; } }</pre>	Java Class Depends On Java Class	Class1 depends on Class2, Class3, Class4, Class5, Class6
DEPENDS ON JAVAENUMERATION	<pre>class Class1 extends Enum1 { Enum2 e2; Enum3 method1(Enum4 e4) { Enum5 e5; } }</pre>	Java Class Depends On Java Enumeration	Class1 depends on Enum1, Enum2, Enum3, Enum4, Enum5
DEPENDS ON JAVAINTERFACE	<pre>class Class1 implements Interface1 { Interface2 i2; Interface3 method1(Interface4 i4) { Interface5 i5; } }</pre>	Java Class Depends On Java Interface	Class1 depends on Interface1, Interface2, Interface3 Interface4, Interface5
EXTENDS JAVACLASS	<pre>class Class1 extends Class2 { }</pre>	Java Class Extends Java Class	Class1 extends Class2
EXTENDS JAVAENUMERATION	<pre>class Class1 extends Enum1 { }</pre>	Java Class Extends Java Enumeration	Class1 extends Enum1

Statement	Format	Relationship	Entities
HAS JAVAANNOTATION	<pre>class Class1 { @interface Annotation1 { } }</pre>	Java Class Has Java Annotation	Class1 has Annotation1
HAS JAVAENUMERATION	<pre>class Class1 { enum Enum1 { } }</pre>	Java Class Has Java Enumeration	Class1 has Enum1
HAS METHOD	<pre>class Class1 { void method1() { } }</pre>	Java Class Has Method	Class1 has method1
IMPLEMENTS JAVAINTERFACE	<pre>class Class1 implements Interface1 { }</pre>	Java Class Implements Java Interface	Class1 implements Interface1

Java Enumeration Relationship Projections

Statement	Format	Relationship	Entities
DEPENDS ON JAVAANNOTATION	<pre>@Annotation1 enum Enum1 { }</pre>	Java Enumeration Depends On Java Annotation	Enum1 depends on Annotation1
DEPENDS ON JAVACLASS	<pre>enum Enum1 { Class1 c1; }</pre>	Java Enumeration Depends On Java Class	Enum1 depends on Class1
DEPENDS ON JAVAENUMERATION	<pre>enum Enum1 { Enum2 e2; }</pre>	Java Enumeration Depends On Java Enumeration	Enum1 depends on Enum2
DEPENDS ON JAVAINTERFACE	<pre>enum Enum1 { Interface1 i1; }</pre>	Java Enumeration Depends On Java Interface	Enum1 depends on Interface1
HAS JAVAANNOTATION	<pre>enum Enum1 { @interface Annotation1</pre>	Java Enumeration Has Java Annotation	Enum1 has Annotation1

Statement	Format	Relationship	Entities
	<pre>{ } }</pre>		
HAS JAVACLASS	<pre>enum Enum1 { class Class1 { } }</pre>	Java Enumeration Has Java Class	Enum1 has Class1
HAS JAVAENUMERATION	<pre>enum Enum1 { enum Enum2 { } }</pre>	Java Enumeration Has Java Enumeration	Enum1 has Enum2
HAS METHOD	<pre>enum Enum1 { void method 1(); }</pre>	Java Enumeration Has Method	Enum1 has method1
HAS JAVAINTERFACE	<pre>enum Enum1 { interf ace Interf ace1 { } }</pre>	Java Enumeration Has Java Interface	Enum1 has Interface1
IMPLEMENTS JAVAINTERFACE	<pre>enum Enum1 implem ents Interf ace1 { }</pre>	Java Enumeration Implements Java Interface	Enum1 implements Interface1

Java File Relationship Projections

Statement	Format	R Entities
DEFINES JAVAANNOTATION	<code>@interface Annotation1 { }</code>	J a v a F i l e D e f i n e s J a v a A n n o t a t i o n File1 defines Annotation1
DEFINES JAVACLASS	<code>class Class1 { }</code>	J a v a F i l e D e f i n e s J a v a C l a s s File1 defines Class1
DEFINES JAVAENUMERATION	<code>enum Enum1 { }</code>	J a v a F i l e D e f File1 defines Enum1

Statement	Format	Relationship	Entities
DEFINES JAVAINTERFACE	<code>interface Interface1 { }</code>	Java Interface Defines Java Annotation	File1 defines Interface1

Java Interface Relationship Projections


Statement	Format	Relationship	Entities
DEPENDS ON JAVAANNOTATION	<code>@Annotation1 interface Interface1 { }</code>	Java Interface Depends On Java Annotation	Interface1 depends on Annotation1
DEPENDS ON JAVACLASS	<code>interface Interface1 { Class1 method1(Class2 c2); }</code>	Java Interface Depends On Java Class	Interface1 depends on Class1, Class2

Statement	Format	Relationship	Entities
DEPENDS ON JAVAENUMERATION	<pre>interface Interface1 { Enum1 method1(Enum2 e2); }</pre>	Java Interface Depends On Java Enumeration	Interface1 depends on Enum1, Enum2
DEPENDS ON JAVAINTERFACE	<pre>interface Interface1 { Interface2 method1(Interface3 i3); }</pre>	Java Interface Depends On Java Interface	Interface1 depends on Interface2, Interface3
EXTENDS JAVAINTERFACE	<pre>interface Interface1 extends Interface2 { }</pre>	Java Interface Extends Java Interface	Interface 1 extends Interface2
HAS JAVAANNOTATION	<pre>interface Interface1 { @interface Annotation1 { } }</pre>	Java Interface Has Java Annotation	Interface1 has Annotation1
HAS JAVACLASS	<pre>interface Interface1 { class Class1 { } }</pre>	Java Interface Has Java Class	Interface1 has Class1
HAS JAVAENUMERATION	<pre>interface Interface1 { enum Enum1 { } }</pre>	Java Interface Has Java Enumeration	Interface1 has Enum1
HAS JAVAINTERFACE	<pre>interface Interface1 { interface Interface2 { } }</pre>	Java Interface Has Java Interface	Interface1 has Interface2
HAS METHOD	<pre>interface Interface1 { void method1(); }</pre>	Java Interface Has Method	Interface1 has method1

Java Package Relationship Projections

Statement	Format	Relationship	Entities
CONTAINS JAVAFILE	<pre>package package1;</pre>	Java Package Contains Java File	package1 contains File1

Method Relationship Projections

 **Note:** You use the Boundary Decisions tab of the Workspace Options to specify resource types your application uses to interface with databases, message queues, and other resources. When you specify the boundary decisions for an application, you tell the parser to create a decision object of a given resource type for each method call in the application.

The format pattern in the table below causes a DECISION relationship to be generated if the name of method2 matches a pattern specified in the Boundary Decisions tab. If, for example, the name of method2 is specified under the "Reads Table methods" type in Boundary Decisions, the parser generates a "Method Reads Table Decision" relationship, with entities "method1" and "Decision1".

Statement	Format	Relationship	Entities
INVOKES METHOD	<pre>void method1() { method2(); }</pre>	Method Invokes Method	method1 invokes method2

JCL Statements

JCL File Relationship Projections

Statement	Format	REntities
		e l a t i o n s h i p
DD (program)	<pre>//[execName] EXEC PGM = ProgName ...//ddName DD DSN = DSName ,...</pre>	JData Connector attributes: o b• Name = <Job.Name>. H <UniqueID> a• Program Entry Point = s <ProgName> D• DD Name = <ddName> a• Step Name = t <execName> a• Step Full Name = C <StepPath> o• Step Number = n <StepNumber> n eDatastore.Name = <dsn- cname> t o Datastore.DSN = <dsn- name> r _File.Name = <ProgName>. D <ddName> File.PortName = a <ddName> t a C o n n e c t o

Statement	Format	REntities e l a t i o n s h i p
		r R e f e r s T O D a t a S t o r e D a t a C o n n e c t o r R e f e r s T O F i l e
DD (system program)	//[execName] EXEC PGM = SysProgName ...//ddName DD DSN = DSName ,...	JData Connector attributes: o b • Name = <Job.Name>. H <UniqueID>

Statement	Format	REntities e l a t i o n s h i p
		<ul style="list-style-type: none"> a• Program Entry Point = s <ProgName> D• DD Name = <ddName> a• Step Name = t <execName> a• Step Full Name = C <StepPath> o• Step Number = n <StepNumber> <p>e Datastore.Name = <dsn- c name></p> <p>t Datastore.DSN = <dsn- r name></p> <p>D Sysprogram.Name = a <SysProgName></p> <p>t a C o n n e c t o r R e f e r s T o D a t a S t o r e C o</p>

Statement	Format	REntities e l a t i o n s h i p
		n n e c t o r l s R e a d l i n S y s t e m P r o g r a m C o n n e c t o r l s W r i t t e n l i n

Statement	Format	REntities e l a t i o n s h i p
		S y s t e m P r o g r a m
EXEC (program)	//EXEC PGM = ProgName	J ProgramEntry.Name = O <ProgName> b R u n s P r o g r a m E n t r y P o i n t
EXEC (system program)	//EXEC PGM = SysProgName	J Sysprogram.Name = O <SysProgName> b R u n s S y s

Statement	Format	REntities e l a t i o n s h i p
		t e m p r o g r a m
EXEC (procedure)	//[execName] EXEC [PROC =] ExternalProc Name	J For resolved files: C JclProc.Name = <resolved- Lname> F ; For unresolved files: I JclProc.Name = e <ExternalProcName> E x e c u t e s J C L P r o c e d u r e
INCLUDE	//INCLUDE MEMBER = member	J For resolved files: C JclProc.Name = <resolved- Lname> F ; For unresolved files: I Jclproc.Name = <member> e l n c l

Statement	Format	REntities e l a t i o n s h i p
		u d e s J C L P r o c e d u r e
-INC (Librarian)	-INC member	J For resolved files: CJclProc.Name = <resolved- Lname> F ; For unresolved files: Jclproc.Name = <member> e l n c l u d e s J C L P r o c e d u r e
JOB	//jobName JOB [parameters]	J Job.Name = <Jcl.Name>. C<jobName> L F Job.JobName = <jobName> i

Statement	Format	REntities e l a t i o n s h i p
		l Job.StepsNum = e <JobStepsNumber> D e f i n e s J o b

JCL Procedure Relationship Projections

Statement	Format	Rel ati on shi p	Entities
EXEC	//[execName] EXEC [PROC=]ExternalProcName	JC L File Ex ecu tes JC L Pro ced ure	For resolved files: JclProc.Name = <resolved- name> For unresolved files: JclProc.Name = <ExternalProcName>
INCLUDE	// INCLUDE MEMBER = member	JC L Pro ced ure Incl ude s JC L Pro ced ure	For resolved files: JclProc.Name = <resolved- name> For unresolved files: JclProc.Name = <member>

Statement	Format	Relationship	Entities
-INC (Librarian)	-INC member	JCL Procedure Includes	<p>For resolved files: JclProc.Name = <resolved-name></p> <p>For unresolved files: JclProc.Name = <member></p>

Natural Statements

Natural File Relationship Projections

Statement	Format	Relationship	Entities
INCLUDE	INCLUDE member	Natural File Includes Natural Include	<p>For resolved files: NatInclude.Name = <resolved-name></p> <p>For unresolved files: NatInclude.Name = <member></p>


Natural Include File Relationship Projections

Statement	Format	Relationship	Entities
INCLUDE	INCLUDE member	Natural Include Includes Natural Include	<p>For resolved files: NatInclude.Name = <resolved-name></p> <p>For unresolved files: NatInclude.Name = <member></p>

Program Relationship Projections

Statement	Format	Relationship	Entities
CALL	CALL 'name'	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL (dynamic)	CALL varname	Program Calls Program Entry Decision	<p>Decision attributes:</p> <ul style="list-style-type: none"> Name = <program-name>@ <internal-name>

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> • #Also Known As = <program-name>. Calls.<varname> • Decision Type = PROGRAMENTRY...
CALL FILE	CALL FILE `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL LOOP	CALL LOOP `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL LOOP (dynamic)	CALL LOOP varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> • Name = <program-name>@ <internal-name> • #Also Known As = <program-name>. Calls.<varname> • Decision Type = PROGRAMENTRY...
CALLNAT	CALLNAT `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALLNAT (dynamic)	CALLNAT varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> • Name = <program-name>@ <internal-name> • #Also Known As = <program-name>. Calls.<varname> • Decision Type = PROGRAMENTRY...
FETCH	FETCH `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>
FETCH (dynamic)	FETCH varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> • Name = <program-name>@ <internal-name> • #Also Known As = <program-name>. Calls.<varname> • Decision Type = PROGRAMENTRY...
PERFORM	PERFORM `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>

Statement	Format	Relationship	Entities
RUN	RUN `name`	Program Calls Program Entry Point	ProgramEntry.Name = <name>
RUN (dynamic)	RUN varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> #Also Known As = <program-name>. Calls.<varname> Decision Type = PROGRAMENTRY...
File Description	name FILE;	See CRUD statements below.	external-file-name = <name> external-file-name = <title> File attributes: <ul style="list-style-type: none"> Name = <program-name>. external-file-name DD Name = external-file-name File Type = ADABAS File  Note: An ADABAS File object is generated only when the first CRUD statement for the file is encountered. File attributes do not depend on the CRUD statement itself.
DELETE	DELETE RECORD IN STATEMENT	Program Deletes From ADABAS File	See File Description for ADABAS File attributes.
FIND	FIND RECORDS IN FILE (name)	Program Reads ADABAS File	See File Description for ADABAS File attributes.
GET	GET IN FILE (name) RECORD	Program Reads ADABAS File	See File Description for ADABAS File attributes.
HISTOGRAM	HISTOGRAM IN FILE (name)	Program Reads ADABAS File	See File Description for ADABAS File attributes.
READ	READ RECORDS IN FILE (name)	Program Reads ADABAS File	See File Description for ADABAS File attributes.
READ WORK FILE	READ WORK FILE (name)	Program Reads File	external-file-name = <name> external-file-name = <title> File attributes: <ul style="list-style-type: none"> Name = <program-name>. external-file-name

Statement	Format	Relationship	Entities
			<ul style="list-style-type: none"> • DD Name = external-file-name • File Type = FILE
STORE	STORE RECORD IN FILE (name)	Program Inserts Into ADABAS File	See File Description for ADABAS File attributes.
UPDATE	UPDATE RECORD IN STATEMENT	Program Updates ADABAS File	See File Description for ADABAS File attributes.
WRITE WORK FILE	WRITE . . .	Program Inserts Into File	external-file-name = <name> external-file-name = <title> File attributes: <ul style="list-style-type: none"> • Name = <program-name>. external-file-name • DD Name = external-file-name • File Type = FILE

.NET Statements

.NET Assembly Relationship Projections

Statement	Format	Relationship	Entities
<i>class-declaration</i>	<i>attributes</i> _{opt} <i>class-modifiers</i> _{opt} <i>class identifier class-base</i> _{opt} <i>class-body</i> ; <i>opt</i>	.NET Assembly Contains .NET Type	DotNetType.Name=<names pace>.<identifier>
<i>struct-declaration</i>	<i>attributes</i> _{opt} <i>struct-modifiers</i> _{opt} <i>struct identifier</i> <i>struct-interfaces</i> _{opt} <i>struct-body</i> ; <i>opt</i>	.NET Assembly Contains .NET Type	DotNetType.Name=<names pace>.<identifier>
<i>interface-declaration</i>	<i>attributes</i> _{opt} <i>interface-modifiers</i> _{opt} <i>interface identifier interface-base</i> _{opt} <i>interface-body</i> ; <i>opt</i>	.NET Assembly Contains .NET Type	DotNetType.Name=<names pace>.<identifier>
<i>enum-declaration</i>	<i>attributes</i> _{opt} <i>enum-modifiers</i> _{opt} <i>enum identifier</i> <i>enum-base</i> _{opt} <i>enum-body</i> ; <i>opt</i>	.NET Assembly Contains .NET Type	DotNetType.Name=<names pace>.<identifier>
<i>delegate-declaration</i>	<i>attributes</i> _{opt} <i>delegate-modifiers</i> _{opt} <i>delegate return-type identifier</i> (<i>formal-parameter-list</i> _{opt})	.NET Assembly Contains .NET Type	DotNetType.Name=<names pace>.<identifier>

.NET Method Relationship Projections

Statement	Format	Relationship	Entities
Method call	/Project/PropertyGroup/ AssemblyName	.NET Method Calls .NET Method	DotNetMethod.Name=<type>.<Name>.<member-name>

.NET File Relationship Projections

Statement	Format	Relationship	Entities
<i>class-declaration</i>	<i>attributes</i> _{opt} <i>class-modifiers</i> _{opt} <i>class identifier class-base</i> _{opt} <i>class-body</i> ; <i>opt</i>	.NET File Contains .NET Type	DotNetType.Name=<namespace>.<identifier>
<i>struct-declaration</i>	<i>attributes</i> _{opt} <i>struct-modifiers</i> _{opt} <i>struct identifier</i> <i>struct-interfaces</i> _{opt} <i>struct-body</i> ; <i>opt</i>	.NET File Contains .NET Type	DotNetType.Name=<namespace>.<identifier>
<i>interface-declaration</i>	<i>attributes</i> _{opt} <i>interface-modifiers</i> _{opt} <i>interface identifier interface-base</i> _{opt} <i>interface-body</i> ; <i>opt</i>	.NET File Contains .NET Type	DotNetType.Name=<namespace>.<identifier>
<i>enum-declaration</i>	<i>attributes</i> _{opt} <i>enum-modifiers</i> _{opt} <i>enum identifier</i> <i>enum-base</i> _{opt} <i>enum-body</i> ; <i>opt</i>	.NET File Contains .NET Type	DotNetType.Name=<namespace>.<identifier>
<i>delegate-declaration</i>	<i>attributes</i> _{opt} <i>delegate-modifiers</i> _{opt} <i>delegate return-type identifier (formal-parameter-list</i> _{opt})	.NET File Contains .NET Type	DotNetType.Name=<namespace>.<identifier>

.NET Project Relationship Projections

Statement	Format	Relationship	Entities
/Project/PropertyGroup/ AssemblyName	/Project/PropertyGroup/ AssemblyName	.NET Project Defines .NET Assembly	DotNetAssembly.Name=<AssemblyName .Text>
/Project/ItemGroup/ Compile/@Include	/Project/ItemGroup/ Compile/@Include	.NET Project Includes .NET File	DotNetFile.Name=@Include


.NET Type Relationship Projections

Statement	Format	Relationship	Entities
<i>method-declaration</i>	<i>attributes</i> _{opt} <i>method-modifiers</i> _{opt} <i>return-type member-name (formal-parameter-list</i> _{opt})	.NET Type Has .NET Method	DotNetMethod.Name=<type>.<Name>.<member-name>
<i>constructor-declaration</i>	<i>attributes</i> _{opt} <i>constructor-modifiers</i> _{opt} <i>constructor-declarator constructor-body</i>	.NET Type Inherits .NET Type	DotNetType
Inheritance	: <i>class-type</i> : <i>interface-type-list</i> : <i>class-type</i> , <i>interface-type-list</i>	.NET Type Inherits .NET Type	DotNetType

Statement	Format	Relationship	Entities
<i>field-declaration:</i>	<i>attributes_{opt} field-modifiers_{opt} type variable-declarators</i>	.NET Type Uses .NET Type	DotNetType.Name=< type > .<Name>
<i>method-declaration</i>	<i>attributes_{opt} method-modifiers_{opt} return-type member-name (formal-parameter-list_{opt})</i>	.NET Type Uses .NET Type	DotNetType.Name=< return- t ype >.<Name>
<i>property-declaration</i>	<i>attributes_{opt} property-modifiers_{opt} type member-name { accessor-declarations }</i>	.NET Type Uses .NET Type	DotNetType.Name=< type > .<Name>
<i>indexer-declarator</i>	<i>type this [formal-parameter-list] type interface-type . this [formal-parameter-list]</i>	.NET Type Uses .NET Type	DotNetType.Name=< type > .<Name> Type.Name=< interface-type>.< Name>

PCT Statements

PCT File Relationship Projections

Statement	Format	R e l a t i o n s h i p	Entities
DFHPCT	DFHPCT TYPE = ENTRY, TRANSID = tran-name, PROGRAM = prg-name	P C T F i l e D e f i n e s T r a n s a c t i o n T r a n s a c t i o n	Transaction.Name = <tran- C name> ProgramEntry.Name = <prg- name> Program.Root = True  Note: Root is empty if Program does not exist in the repository before PCT file verification.

Statement	Format	R e l a t i o n s h i p
		I n i t i a t e s P r o g r a m E n t r y P o i n t

PL/I Statements

PL/I File Relationship Projections


Statement	Format	R e l a t i o n s h i p
%INCLUDE	%INCLUDE member	F o r r e s o l v e d f i l e s: L P I I n c l u d e .N a m e = / <resolved-name> F o r u n r e s o l v e d f i l e s: ; P I I n c l u d e .N a m e = <member> e l n c l u d e s P


Statement	Format	Entities
		Relationship
		Link Include File
PROCEDURE	name: PROC [(parms)] [options(...)] ... options(Main)	Program.Name = <name> /Program. MainProgram= True File Definitions Program

PL/I Include File Relationship Projections

Statement	Format	Relationship	Entities
%INCLUDE	%INCLUDE member	PL/I Include File Includes PL/I Include File	For resolved files: PliInclude.Name = <resolved-name> For unresolved files: PliInclude.Name = <member>

Program Relationship Projections

Statement	Format	Relationship	Entities
CALL	DCL name ENTRY; ... CALL name ...	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL (dynamic)	DCL varname ENTRY VARIABLE; ... CALL varname ...	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> Name = <program-name@ <internal-name> #Also Known As = <program-name>. Calls.<varname> Decision Type = PROGRAMENTRY...
ENTRY	name: ENTRY ...	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry.MainEntry = False
PROCEDURE	name: PROC [(parms)] [options(...)] ...	Program Has Program Entry Point	Program.Name = <name> ProgramEntry.MainEntry = True
File Description	DCL name FILE; OPEN FILE (name); DCL name FILE; OPEN FILE (name) TITLE ('title');	See CRU D statement below	external-file-name = <name> external-file-name = <title> File attributes: <ul style="list-style-type: none"> Name = <program-name>. external-file-name DD Name = external-file-name File Type = FILE <p> Note: A File object is generated only when the first CRUD statement for the file is encountered. File attributes do not depend on the CRUD statement itself.</p>


Statement	Format	Relationship	Entities
File Description (dynamic)	<pre>DCL varname FILE VARIABLE; OPEN FILE (varname); DCL name FILE; OPEN FILE (name) TITLE (vartitle);</pre>	See CRU D state ment s below	<p>decision-var = <varname> decision-var = <name></p> <p>Decision attributes:</p> <ul style="list-style-type: none"> Name = <program-name>@ <internal-name> #Also Known As =<program-name>.<decision-rel>.<decision-var> Decision Type = DATAPORT <p> Note: A Decision object is generated only when the first CRUD statement for the file is encountered. Only the Also Known As attribute depends on the CRUD statement itself.</p>
DELETE	DELETE FILE (name) ...	Program Deletes From File	See File Description for File attributes.
DELETE (dynamic)	DELETE FILE (varname) ...	Program Deletes From File Decision	See File Description (dynamic) for Decision attributes.
GET	GET FILE (name) ...	Program Reads File	See File Description for File attributes.
GET (dynamic)	GET FILE (varname) ...	Program Reads File Decision	See File Description (dynamic) for Decision attributes.
PUT	PUT FILE (name) ...	Program Inserts Into File	See File Description for File attributes.

Statement	Format	Relationship	Entities
PUT (dynamic)	PUT FILE (varname) ...	Program Inserts Into File Decision	See File Description (dynamic) for Decision attributes.
READ	READ FILE (name) ...	Program Reads File	See File Description for File attributes.
READ (dynamic)	READ FILE (varname) ...	Program Reads File Decision	See File Description (dynamic) for Decision attributes.
REWRITE	REWRITE FILE (name) ...	Program Updates File	See File Description for File attributes.
REWRITE (dynamic)	REWRITE FILE (varname) ...	Program Updates File Decision	See File Description (dynamic) for Decision attributes.
WRITE	WRITE FILE (name) ...	Program Inserts Into File	See File Description for File attributes.
WRITE (dynamic)	WRITE FILE (varname) ...	Program Inserts Into File Decision	See File Description (dynamic) for Decision attributes.

PSB Statements

PSB File Relationship Projections

State ment	Format	Relationship	Entities
PSB GEN	PSBGEN LANG=language, PSBNAME= psb-name, ...	P S B F i l e D e f i n e s P S B M o d u l e	PsbModule.Name = <psb-name> PsbModule.Language = <language>
PCB	PCB TYPE=DB, DBDNAME= dbd-name, ... PCB TYPE=GSAM, DBDNAME= dbd-name, ... PCB TYPE=DB, DBDNAME= ..., PROCSEQ= dbd-name, ... PCB TYPE=GSAM, DBDNAME= ..., PROCSEQ= dbd-name, ...	P S B M o d u l e R e f e r e n c e s T o H i e r a r c h i c a l D a t a b a s e	HiDatabase.Name = <dbd-name>
SENS EG	SENSEG NAME= ..., INDICES= (dbd-name1, ... dbd-nameN)	P S	HiDatabase.Name = <dbd-name1> ...

State ment	Format	R el at io n s h i p Entities
	Note: You can specify up to 32 DBD names of secondary indices.	B M o d u l e R e f e r e n c e s T o H i e r a r c h i c a l D a t a b a s e HiDatabase.Name = <dbd-nameN> ...
COP Y	member [OF library]	P S B F i l e I n c l u d e s P S B C o p y b o o k F i l e For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

State ment	Format	R el at io n s h i p Entities
+ +INC LUDE (Parv alet)	++INCLUDE member	P S B F i l e I n c l u d e s P S B C o p y b o o k F i l e For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
-INC (Libra rian)	-INC member	P S B F i l e I n c l u d e s P S B C o p y b o o k F i l e For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

PSB Copybook File Relationship Projections

Statement	Format	Relationship	Entities
COPY	member [OF library]	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
++INCLUDE (Parvalet)	++INCLUDE member	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
-INC (Librarian)	-INC member	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

RPG Statements

RPG File Relationship Projections

Statement	Format	Relationship	Entities
COPY INCLUDE E	/COPY member /COPY library/filename, member /INCLUDE [library/filename,]member	RP G Fil e Inc lud es RP G Co py bo ok Fil e	For resolved files: RPGCopy.Name = <resolved-name> For unresolved files: RPGCopy.Name = [<library>.<filename>.]<member>
Main procedur e		RP G Fil e De fin es	Program.Name=<source-name>

Statement	Format	Relationships	Entities
		Program	

Program Relationship Projections

Statement	Format	Relationships	Entities
Main procedure		Program Has Program Entry Point	ProgramEntry.Name=<source-name> ProgramEntry.MainEntry.Flag=true
Procedure specification	P name B EXPORT	Program Has Program Entry Point	ProgramEntry.Name=<name>
CALL	CALL 'name'	Program Calls Program Entry Point	ProgramEntry.Name=<name>
CALL	CALL varname	Program Calls Program Entry Decision	Decision attributes: <ul style="list-style-type: none"> Name=<program-name>. <internal-name> AKA=<program-name>. Calls.<varname> Type=PROGRAM

State ment	Format	Rela tion ship s	Entities
			MENTR Y ...
CALLB	CALLB `name` ...	Program Calls Bound Proce dure Program Entry Point	ProgramEn try.Name=< name>
CALLB	CALLB procptr	Program Calls Bound Proce dure Deci sion	Decision attributes: <ul style="list-style-type: none"> Name=<progra m- name>. <interna l-name> AKA=<progra m- name>. CallsPr c.<proc ptr> Type=P ROGRA MENTR Y ...
CALLP	D name PR EXTPGM(`ename`) C CALLP name	Program Calls Program Entry Point	ProgramEn try.Name=< ename>
CALLP	D name PR EXTPGM (varname) C CALLP name	Program Calls Program Entry Deci sion	Decision attributes: <ul style="list-style-type: none"> Name=<progra m- name>. <interna l-name> AKA=<progra

State ment	Format	Rela tion ship s	Entities
			m- name>. Calls.<v arname > <ul style="list-style-type: none"> Type=P ROGRA MENTR Y ...
CALLP	D name PR EXTPROC('ename') C CALLP name 1)	Prog ram Calls Boun d Proc edur e Prog ram Entr y Point	ProgramEn try.Name=< ename>
CALLP	D name PR EXTPROC(procptr) C CALLP name	Prog ram Calls Boun d Proc edur e Deci sion	Decision attributes: <ul style="list-style-type: none"> Name= <progra m- name>. <interna l-name> AKA=< progra m- name>. CallsPr c.<proc ptr> Type=P ROGRA MENTR Y ...
CALLP	D name PR C CALLP name 2)	Prog ram Calls Boun d Proc edur e Prog ram Entr	ProgramEn try.Name=< name>

State ment	Format	Rela tion ship s	Entities
		y Point	
File descrip tion (static)	F file-name		external- file- name=<file -name>
File descrip tion (static)	F file-name EXTFILE('ExtFileName')		external- file- name=<Ext FileName>
File descrip tion (static)	F file-name EXTMBR('ExtMbrName')		external- file-name= <ExtMbrNa me>
File descrip tion (static)	F file-name EXTFILE('ExtFileName') EXTMBR('ExtMbrName')		external- file-name= <ExtFileNa me>.<ExtM brName>
File descrip tion (dyna mic)	F file-name EXTFILE(varname)		decision- var=<varna me>
File descrip tion (dyna mic)	F file-name EXTMBR(varname)		decision- var=<varna me>
File descrip tion (dyna mic)	F file-name EXTFILE(varname) EXTMBR(varname2)		decision- var=<varna me>
File descrip tion (dyna mic)	F file-name EXTFILE(varname) EXTMBR('ExtMbrName')		decision- var=<varna me>
CHAIN	F file-name (static-file-description) 3) C CHAIN record-name	Prog ram Rea ds File	File.Name= <program- name>.<file -name> File.File Type=FILE File.DD Name=<file -name>
CHAIN	F file-name (static-file-description) 3) C CHAIN record-name	File Assi	DataStore. Name=<ext

State ment	Format	Relation ship s	Entities
		igned To Data Store	ernal-file- name> DataStore. DSNAME= <external- file-name>
CHAIN	F file-name (dynamic-file-description) 3) C CHAIN record-name	Program Reads File Decision	Decision attributes: <ul style="list-style-type: none">Name= <progra m- name>. <interna l-name>AKA=< progra m- name>. ReadsD ataPort. <decisi on-var>Type=D ATAPO RT ...
DELET E	F file-name (static-file-description) 3) C DELETE record-name	Program Deletes From File	File.Name= <program- name>.<file -name> File.File Type=FILE File.DD Name=<file -name>
DELET E	F file-name (static-file-description) 3 C DELETE record-name	File Assigned To Data Store	DataStore. Name=<ext ernal-file- name>Data Store.DSN AME=<exte rnal-file- name>
DELET E	F file-name (dynamic-file-description) 3) C DELETE record-name	Program Deletes From File Decision	Decision attributes: <ul style="list-style-type: none">Name= <progra m- name>. <interna l-name>AKA=< progra

State ment	Format	Rela tion ship s	Entities
			m- name>. Deletes DataPor t.<decis ion-var> <ul style="list-style-type: none"> • Type=D ATAPO RT ...
EXFM T	F file-name WORKSTN (static-file-description) 3) C EXFMT record-name	Prog ram Send Rece ives Scre en	Screen.Na me= <external- file- name>.<re cord- name>
EXFM T	F file-name WORKSTN (dynamic-file-description) 3 C EXFMT record-name	Prog ram Send s/ Rece ives Scre en Deci sion	Decision attributes: <ul style="list-style-type: none"> • Name= <progra m- name>. <interna l-name> • AKA=< progra m- name>. SendsR eceivees. <decisi on-var> • Type=M AP ...
READ, READ E, READ P, READ PE,	F file-name (static-file-description) 3) C READ record-name	Prog ram Rea ds File	File.Name= <program- name>.<file -name> File.File Type=FILE File.DD Name=<file -name>
READ, READ E, READ P, READ PE,	F file-name (static-file-description) 3) C READ record-name	File Assi gned To Data Stor e	DataStore. Name=<ext ernal-file- name> DataStore. DSNAME=

State ment	Format	Rela tion ship s	Entities
			<external- file-name>
READ, READ E, READ P, READ PE,	F file-name (dynamic-file-description) 3) C READ record-name	Prog ram Rea ds File Deci sion	Decision attributes: <ul style="list-style-type: none"> Name= <progra m- name>. <interna l-name> AKA=< progra m- name>. ReadsD ataPort. <decisi on-var> Type=D ATAPO RT ...
READ, READ E, READ P, READ PE,	F file-name WORKSTN (static-file-description) 3) C READ record-name	Prog ram Rece ives Scre en	Screen.Na me= <external- file- name>.<re cord- name>
READ, READ E, READ P, READ PE,	F file-name WORKSTN (dynamic-file-description) 3) C READ record-name	Prog ram Rece ives Scre en Deci sion	Decision attributes: <ul style="list-style-type: none"> Name= <progra m- name>. <interna l-name> AKA=< progra m- name>. Receive s.<dec ision- var> Type=M AP ...
WRITE	F file-name (static-file-description) 3) C WRITE record-name	Prog ram Inser ts Into File	File.Name= <program- name>.<file -name>

State ment	Format	Relation ship s	Entities
			File.File Type=FILE File.DD Name=<file -name>
WRITE	F file-name (static-file-description) 3) C WRITE record-name	File Assi gned To Data Stor e	DataStore. Name=<ext ernal-file- name>Data Store.DSN AME=<exte rnal-file- name>
WRITE	F file-name (dynamic-file-description) 3) C WRITE record-name	Prog ram Inser ts Into File Deci sion	Decision attributes: <ul style="list-style-type: none"> Name= <progra m- name>. <interna l-name> AKA=< progra m- name>.I nsertsD ataPort. <decisi on-var> Type=D ATAPO RT ...
WRITE	F file-name WORKSTN (static-file-description) 3) C WRITE record-name	Prog ram Send s Scre en	Screen.Na me= <external- file- name>.<re cord-nam
WRITE	F file-name WRKSTN (dynamic-file-description) 3) C WRITE record-name	Prog ram Send s Scre en Deci sion	Decision attributes: <ul style="list-style-type: none"> Name= <progra m- name>. <interna l-name> AKA=< progra m- name>. Sends.

State ment	Format	Relation ship s	Entities
			<ul style="list-style-type: none"> • <decision-var> • Type=M AP ...
UPDAT E	F file-name (static-file-description) 3) C UPDATE record-name	Program Updates File	File.Name= <program- name>.<file- name> File.File Type=FILE File.DD Name=<file- name>
UPDAT E	F file-name (static-file-description) 3) C UPDATE record-name	File Assigned To Data Store	DataStore. Name=<ext ernal-file- name>Data Store.DSN AME=<exte rnal-file- name>
UPDAT E	F file-name (dynamic-file-description) 3) C UPDATE record-name	Program Updates File Decision	Decision attributes: <ul style="list-style-type: none"> • Name= <progra m- name>. <interna l-name> • AKA=< progra m- name>. UpdatesDataP ort.<de cision- var> • Type=D ATAPO RT ...
UPDAT E	F file-name WORKSTN (static-file-description) 3) C UPDATE record-name	Program Send Screen	Screen.Na me= <external- file- name>.<re cord- name>
UPDAT E	F file-name WORKSTN (dynamic-file-description) 3 C UPDATE record-name	Program Send	Decision attributes:

Statement	Format	Relationships	Entities
		Screen Decision	<ul style="list-style-type: none"> Name= <program-name>. <internal-name> AKA=<program-name>. Sends. <decision-var> Type=MAP ...

SQL DDL Statements


DDL File Relationship Projections

Statement	Format	Relationship	Entities
ALTER TABLE	ALTER TABLE <table-name> ...	DDL File Refers To Table	Table attributes: <ul style="list-style-type: none"> Name = <table-name> Is View = False
COMMENT	COMMENT ON [TABLE] <table-name> ...	DDL File Refers To Table	Table.Name = <table-name>
CREATE ALIAS	CREATE ALIAS <alias-name> ON <table-name> ...	DDL File Defines Table Table Represents Table	Table attributes: <ul style="list-style-type: none"> Name = <alias-name> Is View = True IsAlias = True Source = "DBSchema.mdb" Origin = <source-file-path> Table.Name = <table-name>
CREATE INDEX	CREATE INDEX <index-name> ON <table-name> ...	DDL File Refers To Table	Table.Name = <table-name>
CREATE SYNONYM	CREATE SYNONYM <ViewName> FOR <TableName>	DDL File Defines Table Table Represents Table	Table attributes: <ul style="list-style-type: none"> Name = <synonym> Is View = True IsAlias = True Source = "DBSchema.mdb" Origin = <source-file-path> Table.Name = <table-name>

Statement	Format	Relationship	Entities
CREATE TABLE	CREATE TABLE <table-name>... [IN [DATABASE] [<database>.] <tablespace>]	DDL File Defines Table	Table attributes: <ul style="list-style-type: none"> Database TableSpace Name = <table-name> Is View = False Source = "DBSchema.mdb" Origin = <source-file-path>
CREATE PROCEDURE	CREATE (OR REPLACE)? PROCEDURE <procedure-name> ...	DDL File Defines Stored Procedure	Program Entry Point: <ul style="list-style-type: none"> Name = procedure-name MainEntry = false
CREATE VIEW	CREATE VIEW <view-name> ...AS SELECT ... FROM <table-name>	DDL File Defines Table Table Represents Table	Table attributes: <ul style="list-style-type: none"> Name = <view-name> Is View = True Source = "DBSchema.mdb" Origin = <source-file-path> Table.Name = <table-name>
Referential constraint	FOREIGN KEY key REFERENCES <base-table>...	DDL File Refers To Table	Table.Name = <base-table>
SET CURRENT SQLID	SET CURRENT SQLID = 'user-name'		sqlid = <user-name>

System Definition Statements

System Definition File Relationship Projections

Statement	Format	Relationship	Entities
APPLCTN	APPLCTN PSB= psb-name ... TRANSACTION CODE= (trancode [rtran-code], ...)...		Transaction.Name = <trancode> ProgramEntry.Name = <psb-name> Program.Root = True Program.Ims Completed = False PsbModule.Name = <psb-name>
	 Note: Relationships are created for every TRANSACTION macro that follows an APPLCTN macro, and for every transaction code in the TRANSACTION macro.		

Statement	Format	Relationship	Entities
			Transaction Initiates Program Entry Point Program Uses PSB Module

Visual Basic Statements

VB Project File Relationship Projections

Statement	Format	Relationship	Entities
ExeName32	Name="name"	VB Project File Defines	Library.LibName=<name>
Name	ExeName32="dllname"	Library	Library.Name=<dllname>
Title			
Reference	Reference=guid#ver#path dllname#descr	VB Project File Uses Library	Library.Name=<dllname>
Object	Object=guid#ver#;dllname	VB Project File Uses Library	Library.Name=<dllname>
Module	Module=name; filename	VB Project File Includes VB File	VbFile.Name=<filename> VbFile.Type=Module
Class	Class=name; filename	VB Project File Includes VB File	VbFile.Name=<filename> VbFile.Type=Class
Form	Form=name; filename	VB Project File Includes VB File	VbFile.Name=<filename> VbFile.Type=Form
UserControl	UserControl=name; filename	VB Project File Includes VB File	VbFile.Name=<filename> VbFile.Type=UserControl

Statement	Format	Relationship	Entities
HelpFile	N/A	N/A	N/A
ResFile32	N/A	N/A	N/A

Library Description File Relationship Projections

Statement	Format	Relationship	Entities
Name	Name="name"	Library Description File	Library.LibName=<name>
ExeName	File = "dllname"	Defines Library	Library.Name=<dllname>
Interface	<interface name="InterfaceName" > <dispinterface name=" InterfaceName">	Library HasClass VB Class	VbClass.Name=<Library.Na me>.<InterfaceName> VbClass.ClassName=<Inter faceName>
Coclass	<class name="ClassName">	Library HasClass VB Class	VbClass.Name=<Library.Na me>.< Class Name > VbClass.ClassName=<Clas sName>
Method	<interface> <function name="Name" invoke="MethodType" >	VB Class Has VB Method	VbMethod.Name=<VbClass .Name>.<Name> VbMethod.MethodName=< Name> VbMethod.MethodType=Me thodType VbMethod. AccessModifier=Public
Function	<module name="ModuleName"> <function name="Name">	Library HasFunction VB Function	VbFunction.Name= <Library.Name>.<ModuleN ame>.<Name> VbFunction.FunctionName= <Name> VbFunction. AccessModifier=Public

VB Class, Form, and User Control Relationship Projections

Statement	Format	Relationship	Entities
Name	Attribute VB_Name = "name"	Library HasClass VB Class VB File Defines VB Class For Forms: VB Class Defines Map	VbClass.Name=<Library.Na me>.<name> VbClass.ClassName=<nam e> Map.Name=<name>
Inheritance	Begin <ParentName> <ClassName>	VB Class Inherits VB Class	
Property Get	<AccessModifier> Property Get <name> () As <ReturnType>	VB Class Has VB Method	VbMethod.Name=Get.<Vb Class.Name>.<name> VbMethod.MethodName=< name>

Statement	Format	Relationship	Entities
Property Set	<AccessModifier> Property Set <name> (args)	VB Class Has VB Method	VbMethod.MethodType=PropertyGet VbMethod.AccessModifier=<AccessModifier> VbMethod.Name=Set.<VbClassName>.<name> VbMethod.MethodName=<name> VbMethod.MethodType=PropertySet VbMethod.AccessModifier=<AccessModifier>
Function	<AccessModifier> Function <name> (args) As <ReturnType>	VB Class Has VB Method	VbMethod.Name=<VbClassName>.<name> VbMethod.MethodName=<name> VbMethod.MethodType=Method VbMethod.AccessModifier=<AccessModifier>
Sub	<AccessModifier> Sub <name> (args)	VB Class Has VB Method	VbMethod.Name=<VbClassName>.<name> VbMethod.MethodName=<name> VbMethod.MethodType=Method VbMethod.AccessModifier=<AccessModifier>
Method call	<Object>.<MethodName> (args)	VB Class Calls VB Method	VbMethod.MethodName=<MethodName>
Object var declaration	Dim <VarName> As <ClassName>	VB Class DependsOn VB Class	VbClass.ClassName=<ClassName>

VB Module Relationship Projections

Statement	Format	Relationship	Entities
Function	<AccessModifier> Function <name> (args) As <ReturnType>	Library HasFunction VB Function	VbFunction.Name=<LibraryName>.<name> If public, VbFunction.Name=<LibraryName>.<ModuleName>.<name> Otherwise, VbFunction.FunctionName=<name>

Statement	Format	Relationship	Entities
Sub	<AccessModifier> Sub <name> (args)	Library HasFunction VB Function	VbMethod. AccessModifier= <AccessModifier> VbFunction.Name=<Library. Name>.<name> If public, VbFunction.Name=<Library. Name>.<ModuleName>.<n ame> Otherwise, VbFunction.FunctionName= <name> VbMethod. AccessModifier= <AccessModifier>
Method call	<Object>.<MethodName> (args)	VB Function Calls VB Method	VbMethod.MethodName=< MethodName>
Object var declaration	Dim <VarName> As <ClassName>	VB Function DependsOn VB Class	VbClass.ClassName=<Clas sName>

Index

.NET support 356

A

activity log 73
Activity Log 67
AddNew.bj 277
Advanced Search facility 170, 175, 177, 178
AffectedCodeReport.bj 278
AnalyzeProgram.bj 278
Animator 200
Application Analyzer 42
Application Architect 228–236, 238–257, 259, 260
application-level metamodel 318
ApplyPCF.bj 279
Architecture Modeler

- adding a source file 322
- creating metamodel 322
- defining entity types 324
- defining relationship types 325
- exporting metamodel 334
- loading metamodel 322
- opening 317
- overview 317–321
- reconfiguring MW 334
- saving extended metamodel 322
- specifying formatting options 323
- specifying search patterns for entities and relationships 327–329
- tasks 321
- troubleshooting metamodel 334

Architecture Modeller

- gallery 335

archivers 52, 72, 73
auto-extracting business names 205
autodetecting environment 95

B

Batch Duplicate Finder (BDF) 225–228
Batch Refresh Process 261
Batch Refresh Process configuration

- queue processor 264

Batch Refresh Process execution 265, 267, 268
Batch Refresh Process utilities 268–271, 273
Birds Eye pane 169
black-boxing objects 135
boundary decisions 97
Brave.exe 274
Browser 61
Browser Fetch Buffer Size 71
Browser tab 61, 62
business function 208, 210
business names

- assigning manually 147, 163, 204
- deleting 206
- generating in batch mode 204

importing and exporting 205
modifying in batch mode 204
propagating 206
business rule 207
Business Rule Manager 43
Business Rule Manager (BRM) 207–218, 220–225
business rules

- generating in a specified rule set 182

BusinessRulesReport.bj 280
BusinessRulesValidation.bj 280

C

C support 352
C++ support 352
Callie pane 196
Change Analyzer 151–156
change magnitude 151
ChangeTraffic.bj 281
CheckQueue.bj 282
client hardware requirements 12
client installation 18
client software requirements 12
Clipper pane 178, 179, 183
ClipperDetails.bj 283
ClipperMetrics.bj 283
ClipperMultiSearch.bj 284
ClipperSearch.bj 276, 285
Cobol support 352
Code Search pane 178–184
code segment 207
CodeSearchReport.bj 286
collapsing subtrees 62
compiler constant directives 96
complexity 148–150
ComplexityReport.bj 286
Component Maker 228–236, 238–257, 259, 260
conditional compiler constants 96
Configuration Manager 18
connecting to a workspace 50
construct model 43, 160
Context pane 166
control conditions 209
control flows 194
CPP support 352
CreatePCF.bj 287
CreateWS.bj 287
creating a workspace 47
creating metamodel 322
Cross-reference Report 101
CRUD report 100
CRUDReport.bj 288

D

data flow analysis 145, 146, 148
data flow relationships 146, 187
database indexes 47
database server 45

- database setup 9
- dataport 145, 146, 148
- DBA.Cobol.bj 289
- DCE.bj 290
- Decision Resolution 101, 105–108, 115
- deployment scenario 8
- depth 152
- Details Report 184
- DiagramCallie.bj 290
- DiagramFlowchart.bj 291
- Diagrammer 128–143, 145
- DiagramProject.bj 292
- DiagramTS.bj 275, 293
- dropping repository indexes 47

E

- EA client 45
- EA server 45
- Editor 64
- Editor User Preferences 65
- effort 148–150
- EffortReport.bj 294
- encoding 79
- Enterprise Analyzer 41
- Enterprise View 42
- Enterprise View web client 45
- Enterprise View web server 45
- entities 318
- entity flags 319
- entity type attributes 320
- entity type properties 318, 319
- Environment User Preferences 72
- Execution Path pane 197
- ExecutiveReport.bj 294
- expanding subtrees 62
- Export Options 72
- ExportDescriptions.bj 295
- exporting metamodel 334
- ExportRules.bj 295
- ExportScreens.bj 296
- External Reference Report 101
- extracting business names automatically 205
- extracting business names manually 168

F

- file extensions 78
- file server 46
- filtering subtrees 62
- finding and replacing text 63
- Flowchart pane 198, 199
- Folder Browser 61
- FortifyMBSExport.bj 296

G

- General Options 71
- Generic API Analysis 108–115, 117–120
- GenScreens.bj 297
- Global Data Flow 145, 146, 148
- Glossary pane
 - generating reports 207

- guides 73

I

- I/O data elements 209
- Impact pane
 - generating trace 186
- impact trace relationships 146, 187
- ImpactReport.bj 298
- ImpactReportFromList.bj 299
- ImportRules.bj 300
- installation overview 7
- installation tasks 7
- Interactive Analysis 160–162
- Invalidate.bj 301
- invalidating objects 85
- Inventory Report 66
- InventoryReport.bj 302

J

- Japanese-language support 79, 82
- Java support 355
- JSP support 355, 356

L

- Legacy Search tab 63
- Library Description files 357
- licensing 19, 20
- loading metamodel 322
- loading source files 52, 73
- Logic Analyzer 183, 228–230, 232–236, 238–257, 259, 260

M

- master user
 - adding new 49
 - privileges 46
- menus
 - overview 58
- Metrics Report 184
- MFCobolCLink.bj 302
- Model Reference pane 169
- multiuser environment
 - overview 44

O

- Object Information pane 66
- object model
 - overview 43
- object properties 66
- Objects pane 166
- opening a workspace 51
- option sets 69
- options 69
- Options Manager
 - assigning an option set 71
 - copying an option set 71
 - creating an option set 70

- deleting an option set 71
- editing an option set 70
- importing and exporting an option set 71, 141
- making an option set the default 71
- overview 69
- renaming an option set 70
- sharing an option set 71

Orphan Analysis 101, 103–105

P

- parallel verification 85, 93, 266
- parse tree 160, 166
- PL/SQL support 356
- Populate.bj 303
- PortabilityAssessment .bj 303
- private resources 46
- privileges 45
- ProcessChangeUnit.bj 304
- Program Control Flow pane 194–197
- program control flows 194
- project
 - creating 82
 - deleting 84
 - emptying 84
 - including referenced or referencing objects 84
 - moving or copying files 83
 - overview 44
 - protecting 46, 83
 - removing unused support objects 84
 - sharing 83
- Project Options 69
- properties 66
- Properties window 162, 163
- public resources 46
- purge activity log 73

Q

- QualityAssessment.bj 304
- Query Repository 67
- queue processor 56

R

- rar files 52, 72, 73
- reconfiguring EA 334
- Reference Reports 101, 102
- ReferenceReport.bj 274, 305
- Refresh.bj 305
- refreshing the browser 62
- refreshing the workspace path 50
- Register.bj 307
- registering files 52, 73
- Registration Options 78, 79
- regular expression generator 327–329
- Related.bj 307
- relationship flags 321
- relationship type properties 320, 321
- relationships 43, 318
- relaxed parsing 93
- replacing text 63

- reports 98
- Repository Browser 61, 62
- Repository Exchange Protocol Syntax 156, 158
- repository indexes 47
- repository model
 - overview 43
- Repository pane 61–63
- repository server hardware requirements 10
- repository server software requirements 10
- repository, overview 44
- resizing panes 68
- ResolveDecisions.bj 309
- RestoreDecisions.bj 310
- rule set 208, 210
- Rules pane 207–218, 220–225
- RXP 156, 158
- RXP.bj 275, 310

S

- SaveDecisions.bj 311
- saving extended metamodel 322
- Scope Editor 139–143, 145
- scopes 129
- Screen pane 168
- search
 - Advanced Search facility 170, 175, 177, 178
 - simple search facility 164
- searching for objects 63
- selecting all 61
- selecting multiple 61
- selecting objects 61
- server hardware requirements 11
- server installation 18
- server software requirements 11
- SetChangeDate.bj 311
- SetProject.bj 312
- shared folder setup 40
- shared resources 46
- Simple Security Policy 46, 49
- single-user environment, overview 44
- SME, designating 49
- source files
 - creating 80
 - exporting from a workspace 82
 - registering 52, 73
- Source Name property 319
- Source pane 164–166
- Source Type property 319
- SQL Server database setup 9
- SQL support 356
- staged parsing 91, 92
- subexpression editing 328
- subject matter expert, designating 49
- subtrees
 - collapsing 62
 - expanding 62
 - filtering 62
- supported features by language 346
- system programs 96

T

- tablespace 47
- Tag Manager
 - assigning tags 124
 - cleaning up window 128
 - creating relationships between tags 126
 - creating tag queries 127
 - creating tags 124
 - deleting relationships between tags 126
 - deleting tags 127
 - generating reports 128
 - overview 121, 122
 - refreshing 128
 - removing tags 125
 - specifying access to tags 124
- tag queries 124, 127
- TagCmd.bj 312
- text files 327, 328
- text search 63
- Three-Group Security Policy 46, 49
- To Do List 68
- toolbars 58
- triggers 209
- troubleshooting metamodel 334
- type-ahead select 61

U

- uninstallation 18
- Unreferenced Report 101
- Unregister.bj 313
- Unresolved Report 101
- UpdateOnly.bj 314
- UpdateTrendingSnapshot.bj 315
- Upgrade.bj 315
- upgrading workspaces 40
- User Preferences 69

V

- VB Project files 357
- verification 57, 84, 85, 266
- verification options 86, 88, 91–93, 95, 96
- Verification Report 98, 99
- verification results 58
- Verify.bj 315
- Viewer 64
- Visual Basic support 357, 358

W

- Watch pane 167
- wildcard patterns 63
- workspace
 - connecting to 50
 - creating 47
 - deleting 82
 - deleting objects 82
 - opening 51
 - overview 44
 - registering files 52, 73
- Workspace Options 69
- workspace path
 - refreshing 50

X

- XDL files 357
- XML files 329

Z

- zip files 52, 72, 73
- zooming 132