Enterprise Analyzer 3.5 Update 2

Support Notes

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

Copyright © Micro Focus 2009-2015. All rights reserved.

MICRO FOCUS, the Micro Focus logo and Enterprise Analyzer are trademarks or registered trademarks of Micro Focus IP Development Limited or its subsidiaries or affiliated companies in the United States, United Kingdom and other countries.

All other marks are the property of their respective owners.

2015-01-29

Contents

Supported Features by Language 5
AS/400 CL Support 11
Assembler Support 12
Specifying Relationships for Assembler Customizations 12
HyperCode for Assembler 13
Restrictions 13
C/C++ Support 14
Registering C/C++ Files 14
Verifying C/C++ Files 14
CICS Support 15
Case Sensitive Transaction Name 15
Deprecated CICS Statements 15
Keyword Permutations 15
Statements Taken to Be the Same 16
COBOL Support 17
Object-Oriented Statements 17
Separators Must Be Followed by Blanks 17
Copybooks in Different Partitioned Datasets 17
Copybooks in a Library 17
How Enterprise Analyzer Calculates COBOL Dead Code Statistics 18
Dead Statements 18
Dead Data Elements 18
Dead Constructs 18
Dead Statements, Dead Data Elements, and Dead Lines from Copybooks 19
Interactive Analysis Usage 19
Special Handling of Cobol Program Complexity Metrics 19
Possible Padding in MOVE Statements 19
ECL Support 20
Hogan Framework Support 21
Troubleshooting Hogan Framework Support 23
IDMS Support 24
COPY IDMS Statements 24
NNCOPY Statements 24
Manipulation of Logical Records 24
IMS Support 25
Troubleshooting IMS Analysis 26
Mapping Root Programs to PSBs in JCL or System Definition Files 26
Verification Order for IMS Applications 27
Reverifying Files in IMS Applications 27
Restrictions on IMS Support 27
Generic IMS Restrictions 27
IMS Restrictions for Cobol EXEC DLI Support 29
IMS Restrictions for PL/I 30
Java Support 32
JSP Support 33
Resolving the Location of Java Types and Packages Referenced in JSP Files 33
Job Scheduler Support 34
Preparing CA-7 Job Schedule Information 34
Supplying Schedule IDs for a CA-7 Job Triggered by a Dataset 34

Preparing TWS (OPC) Job Schedule Information 34
Using an XML Job Schedule Format for Unsupported Job Schedules 35
JCL Support 39
Verification Output Notes 40
JCL Control Cards Support 41
Naming Requirements for External Control Cards 41
Detecting Programs Started by Driver Utilities 42
Natural Support 45
Verification Order for Natural Applications 45
Restoring Line Numbers in Natural Source 46
Restrictions on Natural Support 46
.NET Support 48
PL/I Support 49
Verification 49
Change Analyzer 49
How Macros Are Modeled in Interactive Analysis 49
Execution Path Labelled Variables and Branching 49
Global Data Element Flow 49
Common IMS, Domain Extraction, and Autoresolve Restrictions 50
Analysis 50
Variable Value Processing 50
Loop Analysis 51
Array Processing 51
External Call Processing 51
Unsupported Constructions 51
Unsupported Evaluations 52
PL/SQL Support 53
RPG Support 54
SQL Support 55
Renaming DCLGEN Include Files 55
Prefixes for SQL Names 55
VB Support 56
Generating Library Description Files 56
Restrictions on Visual Basic Support 57
WFL Support 58

Supported Features by Language

The table below shows Enterprise 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
Inventory Management											
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											

Functional Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
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 Analysis	**N**	**N**	**N**	**N**	Y	**N**	Y	Y	Y	**N**	**N**
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	Y	Y	Y	Y	Y	**N**	Y	**N**	Y	Y	**N**

Functional Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Anomalies											
- Prepackaged Code Anomalies	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Custom with Clipper 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
-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											

Functional Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
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
-Clipper 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
-Callie Pane (Call Diagram)	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Flowchart Pane	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Execution Path Pane	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Animator Pane	Y	**N**	**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 Analysis											
Global Data Element 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**

Functional Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
-Data Flow Pane	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
Impact Analysis	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Intraprogram	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
- Interprogram	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
-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**
Slicing Analysis											
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	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**

Functional Area/ Feature	COBOL	PL/I	Natural	RPG	Java	JSP	C/C++	.NET	VB	PL/SQL	ASM
Analysis											
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 Clipper Rules	Y	Y	Y	Y	Y	Y	Y	Y	Y	**N**	Y
-Autodetect Rules	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**
-Autodetect Parameters	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**	**N**

AS/400 CL Support

Enterprise Analyzer supports AS/400 CL, Version 5, Release 4.

Assembler Support

Enterprise Analyzer supports IBM 360/370/390 High Level Assembly Language. Follow the instructions in this section to specify relationships for Assembler customizations.

Embedded SQL and CICS statements are parsed and relationships are created.

Specifying Relationships for Assembler Customizations

Use the <Assembler> <APICalls> section of \<Enterprise Analyzer Home>\Data\Legacy.xml to specify relationships for Assembler code customizations.

Defining Program Entry Points

Use the <Defining> tag to specify macros that define program entry points:

```
<Defining>
  <Macro name="macro-name">
    <!-- Parameter is the label -->
    <Parameter number="Label" type="String" meaning="Name"/>
  </Macro>
</Defining>
```

where:

- The *name* attribute of the <Macro> tag is the name of the macro used to define an entry point.
- The <Parameter> tag describes the item in the source line that will be captured as the entry point name. The "Label" value of the *number* attribute indicates that the entry point name will be the label before the macro command.

Loading the External Name

Use the <Loading> tag to specify macros used to get external names prior to calls:

```
<Loading>
  <Macro name="macro-name">
    <!-- Parameters are all the parameters of the macro -->
    <Parameter number="*" type="String" meaning="Name"/>
  </Macro>
</Loading>
```

where:

- The *name* attribute of the <Macro> tag is the name of the macro used as the loading operator.
- The <Parameter> tag describes the item in the source line that will be used as the memory address or name that can be used to get the external name for a call.

Using an External Name in Calls

Use the <Calling> tag to specify macros that call external programs:

```
<Calling>
  <Macro name="macro-name">
    <!-- Parameter is the first parameter of the macro -->
    <Parameter number="1" type="String" meaning="Name"/>
  </Macro>
</Calling>
```

where:

- The *name* attribute of the <Macro> tag is the name of the macro used to call an external program.
- The <Parameter> tag describes the item in the source line that will be captured as the program to call. The "1" value of the *number* attribute indicates that the first parameter following the opcode will be used to determine the external name.

Usage Example

The following example illustrates how to specify relationships for Assembler code customizations:

```
<Assembler>
  <APICalls>

    <Defining>
      <Macro name="ENTER">
        <!-- Parameter is the label -->
        <Parameter number="Label" type="String" meaning="Name"/>
      </Macro>
    </Defining>

    <Loading>
      <Macro name="EPLOAD">
        <!-- Parameters are all the parameters of the macro -->
        <Parameter number="*" type="String" meaning="Name"/>
      </Macro>
    </Loading>

    <Calling>
      <Macro name="MODCALL">
        <!-- Parameter is the first parameter of the macro -->
        <Parameter number="1" type="String" meaning="Name"/>
      </Macro>
    </Calling>

  </APICalls>
</Assembler>
```

HyperCode for Assembler

The Hypercode metamodel represents the relationships between constructs in the model at the program level.

HyperCode for Assembler consists of line-by-line constructs and does not allow for navigation and analysis of an abstract syntax tree metamodel that is usually associated with HyperCode support. There are no explicit relationships at the program level between objects and constructs in the Assembler language. The call statements are identified separately so they can be searched in Clipper.

Note: For more information on the advanced searches included in the product, read the *Performing Advanced Searches* section of the documentation.

Restrictions

The following known limitations are found in this release:

Macro Generation	Macro expansion is not currently supported for Assembler. As a result, calculations of the Assembler program metrics will not be affected by macros and some decisions will be unresolved after verification.

C/C++ Support

Enterprise 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:

- **Preserve Folder Structure** on the Registration > Extensions tab of the Workspace Options. The folder structure for the application is preserved in the display names of registered files. You must select this option if your application uses the same program in multiple folders.
- **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 EA, if the include files were registered).

  Tip: You can also specify these folders in the verification options for a project, in which case EA 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.

CICS Support

Enterprise Analyzer supports the following CICS versions:

- CICS Transaction Server for OS/390, Version 1, Release 3.
- CICS Transaction Server for z/OS, Version 3, Release 1.

Case Sensitive Transaction Name

Enterprise Analyzer supports case sensitive transaction IDs for CSD files. You can choose the way Enterprise Analyzer treats transaction IDs. This option is needed when you have transactions defined with the same name but spelled with a different case. To activate the Case Sensitive Transaction Name support:

1. Open **Start > Programs > Micro Focus > Enterprise Analyzer > Enterprise Analyzer Administration**. The Enterprise Analyzer Administration opens.
2. Choose **Administer > Configure Micro Focus Enterprise Analyzer**. The Configuration Manager window opens.
3. Check **Case Sensitive Transaction Name** under **Legacy Configuration**, and then click **OK**.

Note: You must upgrade your workspace after the configuration change.

Deprecated CICS Statements

Deprecated CICS statements are supported. Programs containing these statements verify successfully.

Keyword Permutations

Keywords without parameters cannot be permuted if they start a statement. SEND TEXT NOEDIT, for example, must start with SEND TEXT NOEDIT. TEXT or NOEDIT should not be placed after other statement's keywords and parameters. The following statement is invalid, for example:

```
EXEC CICS SEND TEXT LENGTH (10) NOEDIT
```

Generally, you can permute statement keywords with parameters in any order, keeping in mind that the first keyword should not be permuted with the others. Below is a list of statements for which you cannot permute the second keyword. That is, the keywords must appear in the order shown:

- CHANGE PASSWORD
- CHANGE TASK
- CHECK ACQPROCESS
- CHECK ACTIVITY
- CHECK ACQACTIVITY
- CHECK TIMER
- DEFINE ACTIVITY
- DEFINE COMPOSITE
- DEFINE INPUT EVENT
- DEFINE PROCESS

- DEFINE TIMER
- DELETE CONTAINER
- DELETE COUNTER
- DELETE DCOUNTER
- EXTRACT CERTIFICATE
- GET CONTAINER
- GETNEXT ACTIVITY
- GETNEXT CONTAINER
- GETNEXT EVENT
- GETNEXT PROCESS
- INQUIRE ACTIVITYID
- INQUIRE CONTAINER
- INQUIRE EVENT
- INQUIRE TIMER
- LINK PROGRAM
- RETRIEVE SUBEVENT
- WAIT CONVID
- WAIT JOURNALNAME
- WAIT JOURNALNUM
- WRITE JOURNALNAME
- WRITE JOURNALNUM

Statements Taken to Be the Same

The statements in each set of statements below are recognized as the same statement and assumed to handle a united set of conditions:

- DOCUMENT CREATE, DOCUMENT INSERT, DOCUMENT RETRIEVE, DOCUMENT SET
- ENDBROWSE ACTIVITY, ENDBROWSE CONTAINER, ENDBROWSE EVENT, ENDBROWSE PROCESS
- START, START CHANNEL
- SYNCPOINT, SYNCPOINT ROLLBACK
- WEB ENDBROWSE HTTPHEADER, WEB ENDBROWSE FORMFIELD
- WEB READ FORMFIELD, WEB READ HTTPHEADER
- WEB READNEXT FORMFIELD, WEB READNEXT
- WEB STARTBROWSE FORMFIELD, WEB STARTBROWSE HTTPHEADER

BTS and CHANNEL versions of statements are not distinguished and assumed to handle a united set of conditions.

COBOL Support

Enterprise Analyzer supports the following COBOL versions:

- COBOL for OS/390, Version 2, Release 2.
- Enterprise COBOL for z/OS, Version 4.2.
- VS COBOL II, Release 4.
- Micro Focus COBOL, V3.2 (level 10).
- ACUCOBOL-GT®, Version 6.1.
- ILE COBOL for AS/400, Version 4, Release 4.
- HP3000, HP COBOL II/XL.
- Unisys COBOL
  - UCOB – Unisys 2200 UCS COBOL.
  - ACOB – Unisys 2200 ASCII COBOL.
  - UTS – UTS 4000 COBOL.
- Unisys MCP COBOL
  - Cobol-74.
  - Cobol-85.
- Fujitsu COBOL (for OS IV MSP and OS IV XSP systems).
- Siemens COBOL, Version 2.3 (BS2000/OSD).
- ICL C2 COBOL for OpenVME, Version 3.
- Tandem Screen COBOL, Pathway/TS D42+ (alpha offering).

Note: Tandem Server COBOL is not supported.

Object-Oriented Statements

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 Different Partitioned Datasets

Before registering copybooks in different Partitioned Data Sets (PDS) on the mainframe, you must select the option **Preserve Folder Structure** on the Registration > Extensions tab of the Workspace Options.

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 Enterprise Analyzer Calculates COBOL Dead Code Statistics

This section provides details on how EA 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 Clipper searches. In other words, Clipper 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.

ECL Support

Enterprise Analyzer supports ECL, ClearPath IX, Release 6.

Hogan Framework Support

Hogan Cobol applications use calls to Process Environment Manager (PEM) to identify program operations, or *activities*, mapped in Hogan Framework configuration files. These activities include LINK (invoke programs), SDB (read files), HDB (read hierarchical data segments), and DC (read screens). Other activities include WORK, DUMP, EXCEP, END, ABEND, CHECK, and special user-defined operations.

Each activity the program performs is identified in a variable moved to the program's Transaction-Control-Block (TCB). On finding a call to PEM:

```
CALL 'PEM' USING TRANSACTION-CONTROL-BLOCK.
```

Enterprise Analyzer matches the activity IDs in the TCB with the corresponding operations in Hogan Framework configuration files. EA then uses the configuration file information to create repository relationships between programs, activity IDs, and related application artifacts. For example:

- Program X64003 Uses Hogan Activity Id 64103_ SDB_TFDATEFL, where 64103 is the Activity ID, SDB is the Activity Type, and TFDATEFL is the DD name.
- Program X64003 Reads File TFDATEFL, where TFDATEFL is the DD name.

EA also displays the configuration file description of the activity in the Description tab of the Properties window for the activity ID: TF DATE FILE, for example.

Hogan batch applications invoke the system program DFSRRC00 in a JCL or a JCL procedure to execute programs mapped in the Hogan Framework configuration file CDMFTXNS.hogan. For every invocation of DFSRRC00 in a JCL file or JCL procedure, EA determines whether the second parameter resolves to BMPPEM or GFBMP. For example:

```
//PEM=BMPPEM, //* or GFBMP
...
//CKPTSTP  EXEC PGM=DFSRRC00,                          ** BATCHPEM **
//         COND=(1,NE,SETCODE),
//         PARM=(BMP,&PEM,&PSB,,,W00000,,,,,,&NBA,&OBA,IM&A&B,&AGN),
//         REGION=5M
```

Note: When the parameter resolves to GFBMP, the DD statement GFCNTRL must be present in the invoking JCL or JCL procedure, and the GFTCNTRL control card file must have BMPPEM as the program name coded in columns 1-8.

If PARM resolves to BMPPEM or GFBMP, EA locates each SYSIN DD statement that identifies a control card file. For example:

```
//CTLCRD2=CI010D2,
...
//SYSIN    DD DISP=SHR,DSN=&DINDEX..CI&STATUS.TO.DATALIB&VER(&CTLCRD2)
```

For each control card file, EA maps the application ID (columns 2-3) and function ID (columns 4-8) to the corresponding activity ID in the configuration file CDMFTXNS.hogan. EA then uses the configuration file information to create repository relationships between jobs and the programs they invoke: TOCIJN03 Runs Program Entry Point X64003, for example, where X64003 is the program identified by the activity ID 064003.

Follow the steps below to analyze Hogan Cobol applications in Enterprise Analyzer:

1. Register Hogan sources in Enterprise Analyzer. Depending on the application, the following file types may need to be registered:
   - Cobol source files (.cbl, .cob, .ccp)
   - Cobol copybook files (.cpy, .dcl)
   - JCL files (.jcl)

- JCL procedure files (.prc)
- Control card files (.crd, .srt)

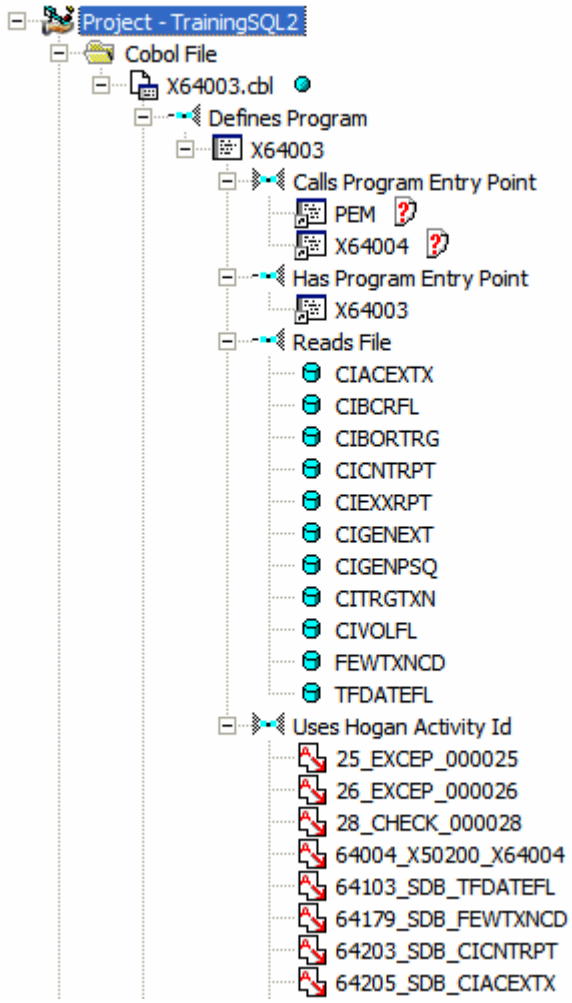
2. In Workspace Options > Verification > Legacy Dialects for the Cobol File type, select **Support Hogan Framework**, then specify the full path of the folder for Hogan Framework configuration files (.hogan) in the **Hogan Files Location** field.

   Note: If you are analyzing Hogan batch applications, the Hogan Framework configuration file CDMFTXNS.hogan must be in the specified folder.

3. Verify application source files. You can:

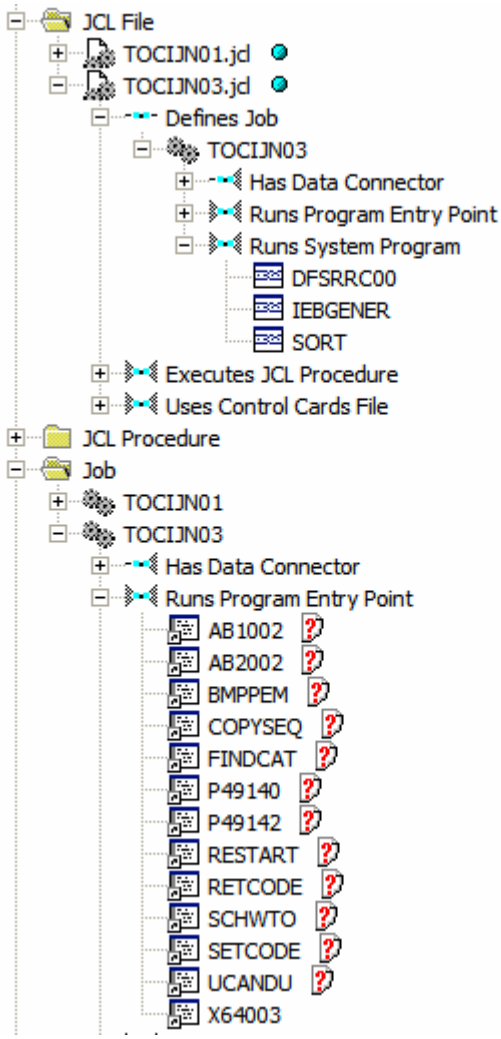
   - Verify the entire project, which ensures that files are parsed in the appropriate order, taking account of the dependencies between file types.
   - Verify files or file types individually, in which case you must verify JCL files last.

The figure below shows typical results of Hogan Cobol verification.



After Hogan Cobol verification, EA puts the file Hogancgf.flt in the folder for Hogan Framework configuration files. You can use this file to view how EA maps activity IDs to operations in Hogan Framework configuration files.

The figure below shows typical results of Hogan batch verification.



After Hogan batch verification, EA puts the file `Hogancgf.flt.TXN` in the folder for Hogan Framework configuration files. You can use this file to view how EA maps control card information to operations in CDMFTXNS.hogan.

Note: See *Troubleshooting Hogan Framework Support* for common problems you may encounter when analyzing Hogan Cobol or Hogan batch applications.

Troubleshooting Hogan Framework Support

If you receive the message "Number of paths exceeded the processing limit of *nn*," set the processing limit to a higher value in the **Maximum Number of Variable's Values** field in Project Options > Verification > Advanced for the Cobol File type.

IDMS Support

Enterprise Analyzer supports CA-IDMS, Release 15.0.

COPY IDMS Statements

COPY IDMS statements are the source manipulation statements for IDMS DML:

```
– [level-number] COPY IDMS [RECORD] copybook-name [REDEFINES data-item-name]
```

You must register a a separate copybook <*copybook-name*> for each COPY IDMS statement in the application. These copybooks should describe corresponding IDMS database records. They can be extracted manually from IDMS-preprocessed sources.

If the COPY IDMS statement depends on a schema or subschema:

```
– COPY IDMS SUBSCHEMA-< copybook>
```

the copybook name must be <schema_name>$<subschema_name>$SUBSCHEMA-<copybook>. SUBSCHEMA-CTRL and SUBSCHEMA-LR-CTRL are considered to be independent of schema/ subschema and should not be prefixed.

NNCOPY Statements

The NNCOPY statement is an extension of the common COPY statement:

```
– [level-number] NNCOPY copybook-name [ ([struct, ] substruct)] [suffix]
```

Select Handle NNCOPY syntax in the project verification options for Cobol files if you want the parser to recognize NNCOPY statements.

Manipulation of Logical Records

Manipulation of logical records in Cobol programs is not supported.

IMS Support

Cobol and PL/I applications that use IMS/DB and/or IMS/DC interact with IMS via CBLTDLI calls and PLITDLI calls, respectively. CICS and batch applications may use EXEC DLI commands to interface with IMS. The Enterprise Analyzer IMS Analysis feature examines these call- and command-level interfaces to determine the interactions between programs and the IMS database. It analyzes:

- The DLI-FUNC-CODE parameter to determine the type of operation requested from IMS.
- The PCB-MASK parameter to determine the target segment or target MFS screen for the operation.

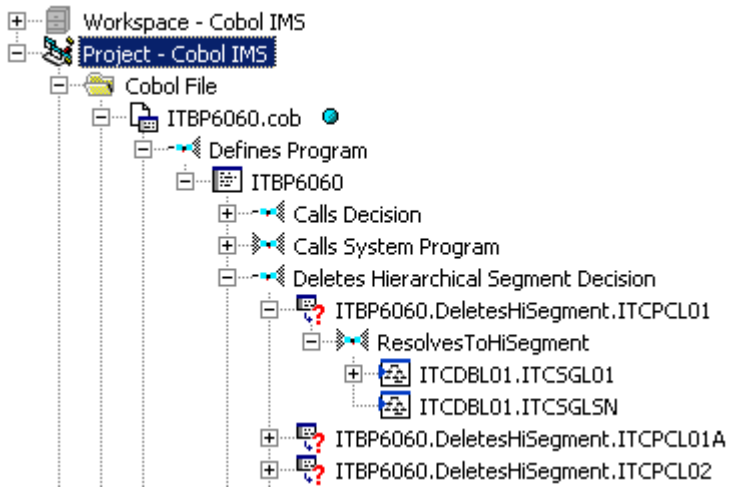
IMS Analysis then follows the call chain from the programs performing these calls up to the programs triggered by the operating system, known as *root programs*. Since the root programs have a PSB associated with them, the analysis can establish which PCB is used in the call to IMS. With this information, Enterprise Analyzer can create repository relationships that reflect the interactions between programs and the IMS database. Before IMS analysis, in other words, you will see only calls to CBLTDLI, PLITDLI, or EXEC DLI. After IMS analysis, you will see CRUD information and MFS screen interactions.

Follow the steps below to perform IMS Analysis:

1. In addition to the basic source files for the Cobol or PL/I application, register the source files that describe the IMS environment for the application. Enterprise Analyzer supports the following IMS-related file types:
   - DBD files (.dbd). These files contain definitions for IMS segments and are the input to the macro DBDGEN in IMS.
   - MFS files (.mfs). These files contain the screen definitions for an IMS system and are the input for MFSGEN.
   - MFS Include files (.mfi).
   - PSB files (.psb). These files contain the list of PCBs accessible by the program and are the input for PSBGEN.
   - PSB Copybook File (.psbcpy).
   - System Definition Files (.ims).
2. Ensure verification options are set correctly:
   - For Cobol files, ensure the default **Perform Program Analysis** and **Enable Data Element Flow** options are set in Project Options > Verification > Advanced Program Analysis.
   - For PL/I files, ensure the default **Enable Data Element Flow** option is set in Project Options > Verification > Advanced Program Analysis.
3. Verify application source files. You can:
   - Verify the entire project, which ensures that files are parsed in the appropriate order, taking account of the dependencies between file types.
   - Verify files or file types individually, in which case you should follow the order specified in "Verification Order for IMS Applications."
4. When all source files have been verified successfully, choose **Prepare > IMS Analysis**. Enterprise Analyzer analyzes the calls to IMS to determine the type of IMS operation performed (Insert, Read, Change, Delete) and the target segment or target MFS screen for the operation.

Important: See "Restrictions on IMS Support" for limitations in Enterprise Analyzer support that may affect your project.

The figure below shows typical results of IMS Analysis. The objects marked with the icon are abstract decision objects, indicating that the database operation, in this case, Deletes, has been resolved to multiple segments.



Note: See "Troubleshooting IMS Analysis" for common problems you may encounter when performing IMS Analysis.

Troubleshooting IMS Analysis

The following are common problems you may encounter when performing IMS Analysis:

- Missing root programs. Root programs are called from the operating system and are passed a PSB that is passed in turn to any called programs. The relationship with the root program is established in the JCL for batch programs or in a system definition file for online programs. If you are missing the JCL or system definition files, you can create them manually, as described in "Mapping Root Programs to PBS in JCL or System Definition Files."
- All content is present but no relationships are created from programs to IMS segments. See "Restrictions on IMS Support" for coding patterns that are not supported by IMS Analysis.

Mapping Root Programs to PSBs in JCL or System Definition Files

Root programs are called from the operating system and passed a PSB that is passed, in turn, to any called programs. The relationship with the root program is established in the JCL for batch programs or in a system definition file for online programs.

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

```
Sample JCL file:
//imsbatch JOB
//S1 EXEC PGM=DFSRRC00,REGION=2048K,
//
PARM=(DLI,progname,psbname,7,0000,,0,,N,0,T,0,,N,,,N)
//
```

```
Sample System Definition file:
APPLCTN PSB=progname
TRANSACT CODE=trnname
```

Verification Order for IMS Applications

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

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

  Note: For Cobol files, make sure the default **Perform Program Analysis** and **Enable Data Element Flow** project verification options are set. For PL/I files, make sure the default **Enable Data Element Flow** project verification option is set.
- JCL or System Definition files

Reverifying Files in IMS Applications

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

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

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

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

Restrictions on IMS Support

Enterprise Analyzer supports IMS/MFS, Version 7. Refer to the topics below for restrictions on Enterprise Analyzer support for IMS.

Generic IMS Restrictions

Before analyzing IMS applications in Enterprise Analyzer, you need to be aware of the following generic restrictions on IMS support.

AIB Interface

The AIB interface is not supported.

Manual Decision Resolutions

Manual resolution of IMS-related decisions (PSB module decisions and the like) do not affect the results of IMS call analysis.

SYSSERVE Parameter of SCHD Call

The SYSSERVE parameter of the SCHD call does not affect analysis. The number of PCB blocks in the PSB is determined statically during PSB verification. IO PCBs are added automatically as needed. That might affect PCB numbering.

CALL Without Parameters

The active PSB name is not traced between programs if CALL without parameters is used.

Active PSB Name Calculated in Called Subroutine

The active PSB name is not detected if it is calculated in a called subroutine. Only “forward” passing of parameters is supported from calling to called module.

Impact Analysis and Interprogram Data Flows

Impact analysis and interprogram data flows are not supported.

Extra Dependencies Between Variables

There may be extra dependencies between variables in IMS calls (in intraprogram analysis, computational components, and so forth) because all CALL arguments except function-code are considered as being used in INOUT mode while in fact some CALLs have input-only and output-only parameters. This also may cause decisions to appear not to have been resolved, when in fact they have been.

When PCB Content Is Considered to Be Altered

PCB content is considered to be altered only by CBLTDLI(PLITDLI) calls or via a group MOVE of the whole PCB structure to another structure. Presence of MOVEs to subfields of a PCB may lead to incorrect analysis results. GU call is not considered as nullifying alternate PCBs.

Port Analysis for IMS Database Calls

For unqualified IMS database calls (without SSAs), port analysis uses only PCB information and does not analyze preceding IMS calls (for example, GNP after GU). Similarly, dependencies between any other IMS calls are not traced except the CHNG – ISRT pair.

For qualified database calls, only the unqualified portion of SSA is analyzed. Command codes are not supported (for example, a path call will be interpreted as a call reading only the last segment in a path).

PARMCOUNT Parameter

The PARMCOUNT parameter is accepted but not analyzed. All CALL parameters are considered as valid.

CHNG Calls

All CHNG calls are treated as setting transaction code destinations because, with no indicators of destination type, it is impossible to distinguish between transaction and terminal names. System tables with transactions and terminal names are needed to check type.

ISRT Calls

ISRT calls to IO-PCB without MOD name are ignored. Most likely they represent the construction of multi-segment messages.

Parsing of Macro Statements

PSB/DBD parsers do not perform full semantic checks of corresponding macro statements. Moreover, the PSB parser does not check that all referenced segments and fields are defined in corresponding DBDs.

Online CICS Applications Using IMS

Online CICS applications using IMS do no need System Definition files. They need only native CICS PCT files.

SET ADDRESS OF and PSB Scheduling

Limited support is available for SET ADDRESS OF <variable> to <PCB> and scheduling of PSBs (calls to PCB functions in CICS programs).

IMS Restrictions for Cobol EXEC DLI Support

Both batch and online CICS programs with EXEC DLI are supported. In addition to the generic restrictions described earlier in this section, the following restrictions apply to IMS support in Cobol EXEC DLI applications.

Subsequent Runs of IMS Call Analysis for Online CICS Applications

Subsequent runs of IMS call analysis for online CICS applications may produce incorrect results. Make sure that all root programs and PCT files are reverified before you repeat IMS call analysis.

Order of Command Options

The order of command options should correspond to the order of options as they are specified in the EXEC DLI reference manual (there is no free format there). Exceptions are the various options for SEGMENT, which can be coded in any order.

Quoted Literals

Quoted literals, where not defined by command syntax, are treated exactly as non-quoted IMS names. The only exception is the LOCKCLASS option.

Host Variables

Host variables must have the form: simple identifier, qualified identifier (a OF b), LENGTH OF special register or a subscripted table element reference. Arithmetic expressions, reference modifications, and other expressions are not supported.

Operators in WHERE Clauses

In WHERE clauses, only relational operators =, <, <=, >, >= and logical operators AND, OR, and NOT are supported.

Comma-Separated Lists

In comma-separated lists, such as for the FIELDLENGTH option, only uniform elements are supported (all literals or all identifiers). Option is verification only.

PCB Option

The PCB option must be explicitly specified on EXEC DLI calls if applicable.

CBLTDLI Calls in CICS Call-Level Programs

For CICS call-level programs, CBLTDLI calls are not recognized as modifying DLIUIB block content. Statements are not treated as dependent:

```
CALL ‘CBLTDLI’ USING GU-FUNC, PCB1, IOAREA1
IF UIBRCODE = SPACES THEN …
```

IMS Restrictions for PL/I

In addition to the generic restrictions described earlier in this section, the following restrictions apply to IMS support in PL/I applications.

Call-Level and Command-Level Programming

Only call-level programming (CALL ‘CBLTDLI’) is supported. Command-level programming (EXEC DLI) is not supported.

More than One Value for DBPCB Parameter at IMS CALL

Only one port per operator is allowed in the HyperCode model. If more than one value is found for DBPCB at IMS CALL, then ImsUnknown is generated instead of a list of ports.

Code Sample	Ports Found	Actual Ports
A:PROC; ... ... IF J>0 THEN DBPCB = EHL1_1; ELSE DBPCB = EUDV_1; ... ... CALLPLITDLI (K4,GNP,DBPCB,ALT_UKARUK, SSA_HORUK_U); END A;	ImsUnknown	EHL1.HORUK,EUDV.HORUK

Non-Const Values of DBPCB or OPCODE Parameters

Non-const values of DBPCB or OPCODE parameters are not taken into account when generating IMS ports.

Code Sample	Ports Found	Actual Ports

A:PROC;
...
DCL P POINTER;
DCL OP CHAR(10);
IF E>0 THEN
BEGIN;
P = EPUF_1;
OP = 'P8CCLA';
END;
... CALLPLITDLI
	EPUF.P8CCLA	EPUF.P8CCLA,ImsUnknown

Code Sample	Ports Found	Actual Ports
(K4,ISRT,P,OP,SEG_SSA(10)); END A;		

Multiple Unsupported Ports

Multiple unsupported ports are represented as a single port in the Interactive Analysis model.

Code Sample	Ports Found	Actual Ports
A:PROC; ... DCL P POINTER; DCL OP CHAR(10); ISRT = 'ISRT'; IF E>0 THEN ISRT = 'REPL'; ... CALLPLITDLI (K4,ISRT,P,OP,SEG_SSA(10)); ... END A;	ImsREPL	ImsISRT, ImsREPL

Called Procedure Rewrites Parameters

During interprogram analysis processing, some values can be lost if a called procedure rewrites parameters either directly or by locator reference.

Code Sample	Ports Found	Actual Ports
A:PROC; DBPCB = EHL1_1; CALL B(DBPCB); CALL PLITDLI (K4,GNP,DBPCB,ALT_UKARUK, SSA_HORUK_U); … END A; /* FILE2.PLI */ B:PROC(DBPCB); …DBPCB = EUDV_1; … END B;	EHL1.HORUK	EUDV.HORUK

Multiple Ports at an IMS CALL inside a Loop Body

DO <VAR>=E1,…,E2 can produce ImsUnknown if there are multiple ports at an IMS CALL inside a loop body.

Code Sample	Ports Found	Actual Ports
A:PROC(...); … DCL SEG CHAR(10); DBPCB = EPUF_1; DO SEG ='PAMSGS','PPPROF'; CALL PLITDLI (K4,ISRT,DBPCB,P0ROOTX,SEG); END; … END A;	ImsUnknown	EPUF.PAMSGS,EPUF.PPPROF

Java Support

Enterprise Analyzer supports Java SE 1.3-1.6. You must install the Java add-on to enable Java. For installation instructions, see the installation manual for your product.

Follow the steps below to analyze Java applications in Enterprise Analyzer:

1. Specify the path of the folder for the Java Runtime Environment (JRE) on your machine in the **Path to JRE Folder** field in User Preferences > Environment. 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.
2. If you want to register Java files with names derived from their locations in the folder structure, rather than with names based on the package declaration, set **Preserve Folder Structure** in Workspace Options > Registration > Extensions, then drag-and-drop the folders (or a parent folder) onto the Repository Browser when you register the files. Choose this option when Java files with the same name and package declaration are used in different applications. So Id.java in the app1 folder is registered as "app1\com\company\Id.java", for example, while Id.java in the app2 folder is registered as "app2\com \company\Id.java".

   Note: The full repository name is not reflected in the display name. Check Properties > General for the full name.
3. Register Java sources (.java) in Enterprise Analyzer. You do not need to register "external" Java files. These are files you reference in the application but do not maintain yourself. You can point to their locations, and to the locations of Java class libraries, when you verify the application.

   Note: If **Preserve Folder Structure** is not set, a Java file in the root folder is considered to be in the default package, and registered with the package name "(default package)". So "com \File1.java" is registered as "(default package).File1.java".
4. In the **Enter classpath to JAR Files and/or path to external Java file root directories** field in Workspace Options > Verification > Settings, right-click in the field and choose:

   - **Add** in the drop-down menu to specify JAR files containing class libraries referenced in the application, or Zip files containing external Java files referenced in the application. Alternatively, choose **Add folder** in the drop-down menu to specify the path of the folder for the JAR or Zip files, then select **Include Jar/Zip Files From Directories**.
   - **Add folder** in the drop-down menu to specify the path of the root folder for external Java files referenced in the application.

   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.
5. 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.
6. Specify the resource types for method calls your application uses to interface with databases, message queues, or other resources in Workspace Options > Boundary Decisions.
7. Verify the application source files.

JSP Support

Enterprise 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 Enterprise Analyzer:

1. Register JSP (**.jsp**), Tag Library Descriptor (`.tld`), and Web Config (`web.xml`) files in Enterprise Analyzer.
2. If your JSP files reference Java types or packages defined in JAR files, external Java files, or Java files registered with the **Preserve Folder Structure** option, 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

Resolving the location of Java types and packages referenced in JSP files is needed when Java files are registered using the **Preserve Folder Structure** setting.

By default Java files are registered in a workspace with names based on their package declaration. For example, `com\company\id.java`. When files with the same name and package declaration are used in different applications, the files must be registered with names derived from their locations in the folder structure, under the **Preserve Folder Structure** setting for the Java file type in **Workspace Options > Registration > Extensions**.

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.

Job Scheduler Support

Enterprise applications use job scheduling software to define dependencies between jobs. Enterprise Analyzer supports CA-7 and TWS (OPC) job scheduling formats. It also provides an XML job schedule format that you can use to define unsupported job schedules.

Preparing CA-7 Job Schedule Information

You make CA-7 job schedule information available to Enterprise Analyzer in LJOB reports. Use the LJOB utility with the LIST=NODD parameter to generate the reports on the mainframe.

The reports must have a .ca7 extension. Each report should have ANSI carriage control characters in column 1. You can create a report with carriage control characters in column 1 by specifying DCB=RECFM=FBA (or FA) in the JCL used to run the LJOB utility.

Supplying Schedule IDs for a CA-7 Job Triggered by a Dataset

When a CA-7 job is triggered by the creation of a dataset, rather than by a job or manual operation, the parser cannot always determine which schedule IDs are used to run the triggered job. You can supply this information in a file.

The file should have the same name as the .CA7 file, and be in the same directory, but have an .SID extension. Specify the location of the file in the **Job to Schedule Ids File** field for a CA-7 Schedule on the Project Verification options tab.

Each line of the file should contain a comma-separated list with the job name and schedule IDs used for the job. For example, if job JOB0001 has the trigger:

```
--- TRIGGERED BY JOBS/DATASETS/NETWORKS ---
DSN=DS00061219(POSTED.BY.XXXX.YYY.ZZZZ)
SCHID=000 QTM=0030 LEADTM=0010 SUBMTM=0000
```

you would enter the following line in the .SID file:

```
JOB0001,10,30
```

That line identifies schedule IDs 10 and 30 for JOB0001.

Preparing TWS (OPC) Job Schedule Information

You make TWS job schedule information available to Enterprise Analyzer in three reports:

- The *application description report* contains most of the information about jobs and how they are scheduled. Use the EQQADPRT utility to generate application description reports on the mainframe. The reports must have a .adr extension. Only these reports are registered in the workspace.
- The *workstation description report* identifies components and the tasks they perform. Use the EQQWSPRT utility to generate workstation description reports on the mainframe. The reports must have a .wdr extension. These reports are not registered in the workspace, but must be identified in the **Workstation Report** field for a TWS Schedule on the Verification > Settings tab of the Workspace Options.
- The optional *cross reference of applications and external dependencies report* resolves job names. Use the EQQADDEP utility to generate cross-reference reports on the mainframe. The reports must have

a .xrf extension. These reports are not registered in the workspace, but must be identified in the **Cross-reference Report** field for a TWS Schedule on the Verification > Settings tab of the Workspace Options.

Each report should have ANSI carriage control characters in column 1. You can create a report with carriage control characters in column 1 by specifying DCB=RECFM=FBA (or FA) in the JCL used to run the utility.

Using an XML Job Schedule Format for Unsupported Job Schedules

Use the Enterprise Analyzer XML job schedule format to define unsupported job schedules. The XML file must have a `.jsx` extension. An annotated sample appears below.

```
 <?xml version="1.0" ?>
<!--
    This is an annotated sample job schedule definition.
    The main purpose of this file is to document the format.
    It is not suitable for testing, as no attempt was made to keep
    it internally consistent.

    The document-level element is the job-schedule. This element is
    required, though the domain reference and name are ignored by  EA.

    domain          (required) - name of domain.
                                 Ignored by EA.
    name            (required) - name of the job schedule.
                                 Ignored by EA.
    source-ref-type (optional) - default is internal.
                                 Specify "external" if this document
                                 contains references to source
                                 locations in external source files.
                                 Otherwise the schedule
loader
                                 generates source reference to this
                                 document.
-->
<job-schedule domain="SP1" name="mainJobSchedule" source-ref-type="external">
    <!--
        source-lib elements define libraries or directories that
        contain source files.

        Source files are used here for two purposes.

        - if this file refers to external source files, e.g., reports
          from a job scheduling system, the source files must be
          identified by source-lib and source-file elements.

        - if jcl-lib elements are used to specify JCL source
          libraries, the libraries must be identified by
          source-lib elements.

        For use in EA, the form using path is used.

        Either path must be specified or both domain and name must
        be specified.

        id     (required)  - unique identifier for the source-lib.
                             The id attribute must be unique among
                             all id attributes in this document.
        path   (see above) - path to a directory containing source
```

```
                          files.

-->
<source-lib id="sl1" domain="SP1" name="SYS1.PROCLIB"/>
<source-lib id="sl2" domain="SP1" name="APPL1.JCLLIB"/>
<source-lib id="sl3" path="c:\source\jcl\sys1.proclib"/>

<!--
     Source files are identified by name and refer to the
     containing library.

     Source file elements must appear before any source-ref
     elements that reference them.

     id     (required) - unique identifier for the source file.
                         The id attribute must be unique within
                         all id attributes in this document.
     libref (required) - id of the associated source-lib element.
                         The source-lib element must appear before
                         any source-file elements that
                         reference it.
     name   (required) - name of the source file.
-->
<source-file id="sf1" libref="sl2" name="report.ctm"/>

<!--
    jcl-libs (and the nested jcl-lib elements identify an order
    list of source libraries that are to be searched to resolve
    references to JCL.

    jcl-lib element are not used in this release of EA. We plan
    to add this support in an upcoming release. These elements
    are included here for completeness.
-->
<jcl-libs>
    <jcl-lib libref="sl1"/>
    <jcl-lib libref="sl2"/>
</jcl-libs>

<!--
    job-stream element starts a job stream definition.

    name - name of the job stream. It must
           be unique within all job stream names within the
           schedule.
    desc - description.
-->
<job-stream name="DAILYOP" desc="Daily order processing">
    <!--
        The documentation nested element can be used with
        job-schedule, job-stream, job-exec, predecessor,
        event, and manual-operation elements to provied more
        extensive documentation than is provided by the desc
        attribute.
    -->
    <documentation>
        This element can be nested under other elements and is
        used to provide more extensive information than is
        included in the description field.
    </documentation>

    <!--
        source-ref (source reference) elements can appear as
        nested elements for job-schedule, job-stream, job-exec,
```

```
        predecessor, manual-operation, and external-predecessor
        elements.

        refid     - (required) id of a source-file element.
        line      - (required) starting line number in source
                    file (first line is 1).
        col       - (optional) starting column number within the
                    line (first column is 1).
                    If omitted, column 1 is assumed.
        endline   - (optional) end line. if omitted, starting
                    line is assumed.
        endcolumn - (optional) end column. If omitted, last
                    character of endline is assumed.
    -->
    <source-ref refid="sf1" line="100" col="1" endline="102" endcol="70"/>

    <!--
        job-exec (job execution) element. defines the execution of a job
        within a job stream.

        name    - (required) name of the job execution. Must
                  be unique among all job executions,
                  manual operations, and events within the job
                  stream, within the schedule.
        jobname - (required) name of the JCL member to be
                  executed.
        desc    - (optional) description of the job execution.
    -->
    <job-exec name="job1" jobname="job1" desc="Description">

        <!--
            The predecessor nested element specifies a
            dependency on another job stream operation
            (another job-exec, external-predecessor, manual
            operation, or event.
        -->
        <predecessor name="mo1">

            <!--
                attr elements can be nested within most other
                elements. Specifically, attr elements can be
                contained by predecessor, external-predecessor,
                job-exec, manual-operation, and event. attr
                elements are never required but can be used to
                provide values that are, for example, specific
                to a particular job scheduling system.

                The exact processing of these values varies with
                the specific implementation. The XML schedule
                loader keeps these values as named attributes of
                the parent object.

                The dep.type nested element for predecessor can
                have one of the following values: GENERAL,
                EXCLUSION, TRIGGER.
            -->
            <attr name="dep.type" value="GENERAL"/>
        </predecessor>

        <!--
            External predecessor elements can be nested under
            job-exec and manual-operation elements to indicate
            a predecessor that is not in the current job stream.
```

```
                All attributes are required, though some may be
                ignored by EA.

                name     - name of this external-predecessor. Must
                           be unique among all job-exec manual-
                           operation, event, and external-predecessor
                           elements in the job stream.
                domain   - containing domain. Ignored by EA.
                sched    - containing schedule. Ignored by EA.
                stream   - containing job stream.
                opername - name of the operation (job-exec,
                           manual-operation, or event) in the external
                           job stream.
            -->
            <external-predecessor name="ep1" domain="dom1" sched="sch1"
             stream="strname" opername="ref1"/>
        </job-exec>
        <!--
            The manual-operation element defines a manual operation.

            It can have predecessor and external-predecessor
            nested elements.

            name - (required). The name of the operation must be
                   unique within manual-operation, job-exec, and
                   even elements within the job stream.
        -->
        <manual-operation name="JobPrep" desc="Description">
        </manual-operation>
        <event name="Event1" desc="description" type="evtype">
        </event>
    </job-stream>
</job-schedule>
```

JCL Support

Enterprise Analyzer supports JCL, OS/390 Version 2, Release 10:

- *Batch Application Viewer* performs low-level analysis of batch processes: whether batch jobs are dependent on one another, the programs that use a data store, and the flow of data into or out of a data store.
- *System calls analysis* analyzes system program input to determine the application program started in a job step.
- *Advanced data flow analysis* traces the flow of data into or out of a dataset through the entire calling chain: not only the program referenced in the control language file (whether or not that program writes to or reads from the dataset), but also any programs that program calls, and any programs they call in turn.
- *Sort card analysis* traces the flow of data through control cards supplied for SORT utility invocations (sort cards). Both inline cards (DD *) and external cards (DSN=) are supported.
- *Driver utility analysis* lets you model batch programs started by a driver utility.

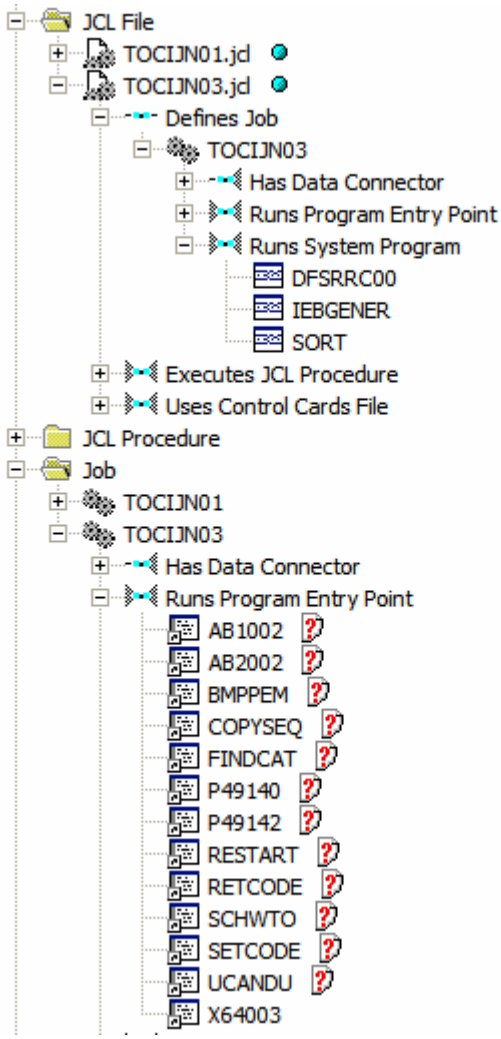
Follow the steps below to analyze JCL applications in Enterprise Analyzer:

1. Depending on the application, the following file types may need to be registered:
   - JCL files (.jcl)
   - JCL procedure files (.prc)
   - External control card files (.crd, .srt). For more information, see "Naming Requirements for External Control Cards."
2. In Workspace Options > Verification > Settings for the JCL file type:
   - Select **Perform System Calls Analysis** to enable system calls analysis.
   - Select **Perform DSN Calling Chains Analysis** to enable advanced data flow analysis.
   - Choose the names of the SORT utilities the application uses in **Sort Program Aliases**. Add names as necessary.
   - Choose the names of the system procedures the application uses in **System Procedures**. Add names as necessary.
   - Select **Allow Implicit Instream Data** to specify that a DD * statement be inserted before implicit instream data if the statement was omitted from JCL.
3. If necessary, use the <JCL> <BatchProgs> section of \<Enterprise Analyzer Home>\Data\Legacy.xml to identify batch programs started by a driver utility. For more information, see "Detecting Programs Started by Driver Utilities."

   Note: For altered `Legacy.xml` files: The `Legacy.xml` file that resides in the `\<EA 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 `\<EA 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.
4. Verify application source files. You can:
   - Verify the entire project, which ensures that files are parsed in the appropriate order, taking account of the dependencies between file types.
   - Verify files or file types individually, in which case you must verify JCL files last.

The figure below shows typical results of JCL verification. For more information, see "Verification Output Notes."



Verification Output Notes

- Job steps are enumerated from the beginning of the job, after all procedures are expanded. The EXEC PROC= command is counted first, as a separate step. Thereafter, all steps inside the invoked procedure are enumerated. The number of job steps, then, is the number of all EXEC commands processed during job execution.
- No relationships are generated for EXECs to internal procedures.
- For steps placed directly in a job, the Step Full Name attribute of the Data Connector object generated from DD statements is <*execName*> if specified, or “Ln <*line-number*>” if <*execName*> is empty. For steps within procedures, the following is the path to the step from the EXEC PROC= command placed inside the job through all intermediate procedures:

  ```
  <jobExecName>[/<ProcName>.<procExecName>]…
  ```
- The parser creates an artificial program entity that defines the inputs and outputs for each sort utility invocation. The program has a name of the form *JCLFileName*.*JobName*.*StepName*.*SequenceNumber*, where *SequenceNumber* identifies the order of the step in the job. For every sort invocation in the program, you can view data structures for sort input and output records and the data movements between them in the HyperCode for the JCL file.

- An invoked program is known as a system program if it is defined with a sort utility alias in **Sort Program Aliases** in Workspace Options > Verification > Settings for the JCL file type, or specified in the \<Enterprise Analyzer Home>\Data\Legacy.xml.

JCL Control Cards Support

JCL EXECs can reference input data defined in-stream or by using control cards. By default, EA processes the control cards that are passed as parameters to the SORT program and generates relationships to them. For example:

```
//STEP1   EXEC PGM=SORT
...
//SYSIN    DD DSN=DD1.PROD.CNTL(SORTCOPY),DISP=SHR
...
```

EA creates a relationship between the JOB and the control card `DD1.PROD.CNTL(SORTCOPY)` which defines parameters for the `SORT` program.

If you want EA to process relationships to other control cards, you must specify them in the `Legacy.xml` file. The XML section you must edit is used to define which DSNs refer to control card files. It is located under the JCL section, in a subsection called ProgUsesCard:

```
<JCL>
  <!-- to generate relationship Program Uses ControlCard -->
  <ProgUsesCard>
    <item> IDCAMS,SYSIN,.* </item>
  </ProgUsesCard>
</JCL>
```

Each item element contains a specification with three parts separated by comma (in the following order):

- The PGM name.
- The DD name.
- The DSN name.

These are regular expressions. In the `Legacy.xml` template, the DSN name part is set to ".* ", which is a match against any string. This instructs EA to do control card processing for the `SYSIN DD` on any call to `IDCAMS`, regardless of the DSN name.

Naming Requirements for External Control Cards

Source files for external cards are registered in the repository as Control Cards files, and must be named as follows, where .srt is the default file extension:

- For an ordinary dataset, if the SYSIN DD statement is:

  ```
  //SYSIN DD DSN=MY.SORTCARDS.LIB.FILE1
  ```

  the source file name must be MY.SORTCARDS.LIB.FILE1.srt.
- For a PDS member, if the SYSIN DD statement is:

  ```
  //SYSIN DD DSN=MY.SORTCARDS.LIB(FILE2)
  ```

  the source file name must be MY.SORTCARDS.LIB(FILE2).srt, or if the member name is unique, FILE2.srt.
- For a generation dataset, if the SYSIN DD statement is:

  ```
  //SYSIN DD DSN=MY.SORTCARDS.LIB.FILE3(+1)
  ```

the source file name must be MY.SORTCARDS.LIB.FILE3.srt, without the generation number.

Detecting Programs Started by Driver Utilities

Batch applications often use jobs to launch driver utilities that start other programs. If you want to model the relationship between jobs and these kinds of programs, you need to define entries that identify the programs in the <JCL> <BatchProgs> section of \<Enterprise Analyzer Home>\Data\Legacy.xml.

<BatchProgs> entries have the form:

```
<item> <driver_utility>,PARM|DD*|DSN,<search_value>
```

where:

- *driver_utility* is the name of the driver utility.
- PARM, DD*, and DSN are the available search options:
  - PARM searches the PARM= parameter of the EXEC statement.
  - DD* searches the instream data of a DD statement.
  - DD* searches the DSN name of a DD statement.
- *search_value* is the value of the specified search option.

Tip: The <BatchProgsNames> section of Legacy.xml contains patterns you can use to filter program names from other input data. Use <Accept> to specify patterns for valid program names, <Except> for invalid program names. For example: `<Accept> ^([A-Z]\w{4,7})$ </Accept>`.

Using the PARM Option

The PARM option searches for the name of the program of interest in the PARM= parameter of the EXEC statement. You can also use it to find the PSB name for an IMS application. The syntax is:

```
PARM,<prg_parm_num>[,%P,<psb_parm_num>,%B]
```

where:

- <*prg_parm_num*> is the number of the parameter that specifies the program name.
- <*psb_parm_num*> is the number of the parameter that specifies the PSB name.

The value is either a string literal, with parameters separated by blanks or by commas, or a list of comma-separated parameters enclosed in brackets.

For example, the entry:

```
<item> IKJEFT01,PARM,3 </item>
```

identifies the programs APPPRG1 and APPPRG in the following code fragment:

```
//step1    EXEC PGM=IKJEFT01,
//              PARM='APP DB2P APPPRG1 APPPRG1L'
//SYSTSIN  DD DUMMY,DCB=BLKSIZE=80

//step2    EXEC PGM=IKJEFT01,REGION=2048K,DYNAMNBR=20,
//              PARM=('APP',DB2P,APPPRG2,APPPRG2L)
//SYSTSIN  DD DUMMY,DCB=BLKSIZE=80
```

For IMS:

```
<item> DFSRRC00,PARM,2,%P,3,%B </item>
```

identifies the program IMSPRG1 and the PSB PSBM1 in the following code fragment:

```
//           EXEC PGM=DFSRRC00,REGION=2048K,
//           PARM=(DLI,IMSPRG1,PSBM1,7,0000,,0,,N,0,T,0,,N,,,N)
```

Using the DD* Option

The DD* option searches for the name of the program of interest in the instream data of a DD statement. You can also use it to find Natural library names. The syntax is:

```
DD*,<dd_name>,<search_string>[%L] %P[+] [,%?]
```

where:

- <*dd_name*> is the name of the DD statement.
- <*search_string*> is the string to match. Matched code can appear on different lines in the source.
- %L specifies the position of the Natural library name relative to the matched string.
- %P specifies the position of the program name relative to the matched string. %P+ indicates that multiple program names are at the specified position.
- %? specifies the position relative to the matched string of the program name to be ignored by the parser.

  Note: So `<item> IKJEFT01,DD*,SYSTSIN,PARMS('%?,%?,%P) </item>` matches *only* BF4P422 in the code fragment `PARMS ('BF422DA1,STEP1,BF4P422 /')`.

For example, the entry:

```
<item> IKJEFT01,DD*,SYSTSIN,RUN PROGRAM(%P) </item>
```

identifies the programs MYPRG1 and MYPRG3 in the following code fragment:

```
//step4   EXEC PGM=IKJEFT01
//SYSTSIN  DD
*
RUN PROGRAM(MYPRG1) qqq
/*

//step5    EXEC PGM=IKJEFT01
//SYSTSIN  DD
*
PROFILE    PREFIX(C34)
DSN SYSTEM(DSN)  RETRY(0)
TEST(0)
RUN
PROGRAM(MYPRG3) PLAN(MYPLAN) -
  LIB('CO737.PDS.MCPBR2.LOAD') -
```

For Natural:

```
<item> ADANATV3,DD*,CMSYNIN,LOGON %L %P+ </item>
```

identifies the library MYLIB and the programs MYNAT1 and MYNAT2 in the following code fragment:

```
//S040   EXEC PROG=ADANATV3
//CMSYNIN  DD
*
LOGON
MYLIB
MYNAT1
MYNAT2
```

Using the DSN Option

The DSN option searches for the name of the program of interest in the DSN name of a DD statement. The syntax is:

```
DSN,<dd_name>,<search_string>%P
```

where:

- <*dd_name*> is the name of the DD statement.

- <*search_string*> is the string to match.
- %P specifies the position of the program name relative to the matched string.

For example, the entry:

```
<item> IKJEFT01,DSN,SYSTSIN,RUNCARDS(%P) </item>
```

identifies the programs PROG1 and PROG2 in the following code fragment:

```
///step8   EXEC PGM=IKJEFT01
//SYSTSIN  DD DSN=SYS.DP.TE.RUNCARDS(PROG1),DISP=SHR

//step9  EXEC PGM=IKJEFT01
//SYSTSIN  DD DSN=RUNCARDS(PROG2),DISP=SHR
```

Natural Support

Enterprise Analyzer supports the following versions of ADABAS/Natural:

- Natural 2.2.3 for mainframes.
- Natural 2.3.4 for mainframes, 3.1.1. for Windows NT, Unix, Open VMS.
- Natural 3.1.2 for mainframes, 4.1.1. for Windows NT.
- Natural 3.1.3 for mainframes, 4.1.2. for Windows NT, Unix, Open VMS.

Follow the steps below to analyze Natural applications in Enterprise Analyzer:

1. Register Natural sources in Enterprise Analyzer. The following objects must be registered:
   - Natural Data Areas (.nda, .nsl)
   - Natural DDM files (.nsd, .ddm)
   - Natural files (.nat, .nsn, .nsp, ..nsh)
   - Natural Include files (.ncp, .inc)
   - Natural Maps (.nsm, .map)
   - Natural Subroutine files (.nss)

   Note: It's almost always useful to qualify Natural programs with their library names in call diagrams and the like. To enable library name qualification, register Natural files with names of the form *library.program.extension*. For assistance renaming files, contact support services.
2. For Natural files registered with library names:
   - Ensure the default **Libraries support** option is set in Workspace Options > Verification > Settings for the Natural File type.
   - In the **Libraries** field in Project Options > Verification for the Natural File type, specify the order in which the parser searches for library names when it resolves program calls. Right-click in the field and choose **Add** in the pop-up menu to enter a library name. Place a check mark next to a library to include it in the search. The parser always searches for the library of the object being verified, the *current library*, first. You can specify a different library search order for each project in your workspace.

     Tip: For example, if Lib2 appears in the list before Lib1, a call to ProgA is resolved to Lib2.ProgA rather than Lib1.ProgA (assuming the current library is not Lib1).
3. In the **Helproutines** field in Project Options > Verification for the Natural Map file type, specify how you want the parser to treat helproutines, as programs (the default) or helpmaps.
4. Verify application source files. You can:
   - Verify the entire project, which ensures that files are parsed in the appropriate order, taking account of the dependencies between file types.
   - Verify files or file types individually, in which case you should follow the order specified in "Verification Order for Natural Applications."

Important: Before registering Natural sources, see "Restrictions on Natural Support" for limitations in Enterprise Analyzer support that may affect your project.

Verification Order for Natural Applications

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

- Natural DDM files (.nsd, .ddm)
- Natural Data Areas (.nda, .nsl)
- Natural Maps
- Natural Subroutine files (.nss)
- Natural files

Restoring Line Numbers in Natural Source

To restore stripped line numbers in Natural source, choose **Edit > Restore Line Numbers in Natural Source**.

Restrictions on Natural Support

Before analyzing Natural applications in Enterprise Analyzer, you need to be aware of the following restrictions on Natural support.

Supported CALLDBPROC Syntax

The following syntax of the CALLDBPROC statement is supported:

```
CALLDBPROC dbproc ddm-name [USING] [ parameter [ AD = {M | O | A}] ]…
[ RESULT SETS result-set…]
[ GIVING sqlcode ]
[ CALLMODE = { NONE | NATURAL }]
```

Supported READ RESULT SET Syntax

The following syntax of the READ RESULT SET statement in structure mode is supported:

```
READ [(limit)] { RESULT-SET | RESULT SET } result-set
INTO { VIEW view-name | parameter,… }
FROM ddm-name
[ GIVING sqlcode]
statement…
END-RESULT
```

The following syntax of the READ RESULT SET statement in reporting mode is supported:

```
READ [(limit)] { RESULT-SET | RESULT SET } result-set
INTO { VIEW view-name | parameter,… }
FROM ddm-name
[ GIVING sqlcode]
statement…
[ [ CLOSE ] LOOP [(r)] ]
```

Program Analysis

The following notes apply to tools related to control flow and/or data flow analysis for programs and to component extraction.

- It is assumed that IF, DECIDE ON, and DECIDE FOR statements always have a FALL THRU, even if all the control flows of the branches do not reach the statement after these statements.
- Definition and usage of array slices are treated as whole array variables.
- Control flow of cyclic constructs, such as FOR loops, have no backward edge. This rough bit of the analysis may lead to insufficiency in data dependencies between definitions inside a loop body and usages at the top of a loop body.
- Control flow analysis between subroutines is call site-independent. It is assumed that there are paths from any call site to any return point of a subroutine. This may lead to superfluous data dependencies between definitions inside a subroutine and their usages beyond the return points.

Component Extraction

Both Computation-Based Extraction and Dead Code Elimination are supported. For Computation-Based Extraction:

- A PERFORM statement without parameters cannot be selected as a slice point.
- An ON ERROR statement and all statements inside an ON ERROR block cannot be selected as a slice point.

For Dead Code Elimination, statements and declarations that are unused INDIRECTLY are not detected. This means that constructions included in unused constructions are retained.

Impact Analysis

Group MOVEs are split into field MOVEs and data flow relationships are calculated for these split MOVEs. This means there are no relationships between whole structures for group MOVEs.

.NET Support

Enterprise 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/I Support

Enterprise Analyzer supports the following PL/I versions:

- VisualAge PL/I for OS/390, Version 2.
- Enterprise PL/I for z/OS, Version 3, Release 6.

Verification

- Preprocessor %PUSH and %POP statements are ignored.
- The in-place initialization of an array of labels is not supported.
- Built-in subroutines and built-in functions are partially supported. The parser does not match the argument list of calls to built-in subroutines with their actual signatures. It may successfully verify source code containing incorrect usage of built-in subroutines or built-in functions.
- The DATE type is not supported.
- The GENERIC attribute is partially supported.

Change Analyzer

- Built-in function calls are not analyzed for synonyms.
- Same memory location synonyms (DEFINED, BASED, UNION) are not supported.

How Macros Are Modeled in Interactive Analysis

Although macro statements are not captured as HyperCode in the Interactive Analysis Source pane, constructs resulting from macro expansion are correctly modeled in the Interactive Analysis Context pane. Clicking on these constructs in the Context pane moves the focus in the Source pane to the location where the macro is used.

Execution Path Labelled Variables and Branching

Labelled variables and branching connected with exceptions (ON and SIGNAL statements) are not supported.

Global Data Element Flow

- Memory release is not taken into account (for example, the FREE statement).
- Entities which can be allocated to several offsets are doubled to convert dynamic memory allocation into a static allocation. For example:

```
    DCL 1 DATA1,
                              2 B FIXED;
    DCL 1 DATA2 LIKE DATA1;
    DCL 1 A1 BASED(P),
                              2 B FIXED;
    DCL 1 A2 BASED(P),
                              2 B FLOAT;
```

```
    DCL 1 A3 BASED(Q),
                            2 B FIXED;
    P = ADDR(DATA1);
    Q = ADDR(DATA2);
    A1.B = 1;           /* 1 */
    Q->A1.B = 1;        /* 2 */
    P = Q;              /* 3 */
```

- Entities for which memory is not allocated are still included in the Data View.
- Fetch and Pointer Add are not currently supported. DataView will be as follows:

DATA1	B	A1	B	A2	B		
DATA2	B	A1	B	A2	B	A3	B

Common IMS, Domain Extraction, and Autoresolve Restrictions

Restrictions described in this section are common to IMS analysis, domain-based extraction, and decision autoresolution.

Analysis

- Analysis is call-site independent, that is, all calls of a procedure are considered simultaneously. First, all the actual parameter values from the call sites are evaluated and collected at the formal parameters and then the procedure constant propagation is performed.
- Analysis is performed for a given file without processing other PL/I files that could be called from the file being analyzed. '`NonConst`' means that the variable obtains no value and so will not be substituted. Moreover, since it has no value, other variables may not be resolved as well.

Variable Value Processing

- When an assignment to a structure field or an array element is interpreted, the entire structure and array value is calculated and stored to the value set. Calculation of values of structure fields of array elements avoids this redundancy, but a '`Nonconst`' value may appear, or some values may not be evaluated because of value set overflow.
- Entire alias values are calculated and stored in variables during processing, with the result that more than one alias value may be stored in a variable even if the variable has only one proper value. Calculation of proper values of the variable avoids this redundancy, but a '`Nonconst`' value may appear, or some values may not be evaluated because of value set overflow.
- Maximum size of the value set of a variable is 32. This means that analysis will not be fully complete when the amount of possible values exceeds this number. A '`NonConst`' value indicates that there are values that are not calculated. Warnings about overflow of value sets of variables are not generated.
- For performance reasons, the maximum size of a memory block is limited to 32KB. Every access beyond this limit is ignored. (Memory blocks are used for structures, arrays, and variables accessed through a DEFINED attribute or pointers). This means that strings, structures and arrays that do not fit in 32KB are not processed correctly.
- Read variables of a statement are enumerated independently from each other in the sense that the statement is interpreted for all combinations (no more than 16) of the values of the variables used in the statement. It may lead to superfluous write values when the statement is evaluated for some combinations that cannot occur during real program execution.
- A '`NonConst`' value is added to a value set if some values cannot be evaluated or the number of the values exceeds the value set limit.
- During interpretation of an operator, a '`NonConst`' value is added to write variables if the number of combinations of values of read variables exceeds the limit.

Loop Analysis

- Loops are executed until value sets of variables inside the loop body stop expanding. Because lower and upper bounds and WHILE conditions are ignored, the resulting sets of variable values may be superfluous.
- DO statements with a cyclic header are always treated as a loop control flow construction, though the body of this statement can be executed only once when the program is executed (for example, `DO I=1;`).

Array Processing

- If all values of an index in a slice expression cannot be evaluated, then the slice is interpreted for all index values from the appropriate array bound interval. The maximum size of the implicit set of index values is 32.
- All arithmetic operations (including built-ins) are performed with DOUBLE values (there is no artificial increase/decrease of digit number). Excess precision may be rounded away during arithmetic type casting.
- Assignment to an array is supported only if no two asterisks in the array reference are separated with an ordinary array index or a structure member selection. The system would try to minimize the amount of memory erased by an unsupported array write, but it can destroy the whole array.
- Complex DEFINED is not supported (the DEFINED declaration is mapped to the base field by field and dimension by dimension). DEFINED(X) is interpreted strictly as if it was BASED(ADDR(X)).
- POSITION for variables defined on bit strings is not supported. Also, no alignment other than on a one-byte boundary is supported.
- DEFINED on sparse areas is not supported. In a real system, one can write `DEFINED(A(*,4))` and if structures match it would map the newly defined array on top of sparse array `A(*,4)`. Structure matching, however, is unsupported. This means that although this type of definition is valid in a real system, it is not supported.

  Note: With one possible exception: `A(4,*)` might work as expected if structures are not only matching but strictly equivalent
- Array expressions are not supported, so `A(*) = B(*) + 5` will not work.
- Out-of-bounds access is always ignored and a warning is issued.
- Dynamic-sized arrays are not supported and are interpreted as fixed-sized arrays with extent 10.

External Call Processing

- When external calls and unresolved dynamic internal calls are interpreted, values of whole aliases are passed, so that such calls are treated as rewriting their parameters, with the result that the values of the whole aliases are assumed to be rewritten and are set to '`NonConst`'.
- It is assumed that interprocedure dynamic calls do not rewrite any context variables (except parameters).

Unsupported Constructions

- Dynamic control flow is unsupported. The following cases are classified as dynamic control flow:
  - Jumps to expression of label type, such as label-variable, element of array of labels, field of structure, and so forth (treated as program stop).
  - Jumps by GOTO outside procedure (treated as program stop).
- Condition handling:
  - Control flow is constructed as if condition-handling operators are replaced by empty operators. Nested operators are also skipped.

- Alias analysis processing:
  - Aliases induced by RETURN(<addr expression>) are not supported
- Non-null pointers in structures are lost when the structure is passed as a parameter to a top-level procedure (except PSB pointers).

Unsupported Evaluations

- Calculation of values of CONTROLLED or AREA variables.
- Calculation of initial values of variables declared in packages is unlike the declaration of the variables in contained procedures.
- SUBSTR as a pseudovariable will not assign a value to a string with a previously undetermined value even if the substring covers the entire string.
- Assignment to the first argument of SUBSTR when this argument is an array.
- VARYINGZ strings are handled as simple VARYING, that is, as using a length field instead of a zero terminator.
- ALLOCATE is only supported if it is used on a single BASED variable and each statement produces only one value of the pointer, no matter how many times it may be called.
- Calculation with multibyte character set values for variables may or may not work correctly.

PL/SQL Support

Enterprise Analyzer supports Oracle PL/SQL for Oracle Database 11g, Release 1.

RPG Support

Enterprise Analyzer supports RPG IV ILE, Version 5, Release 3.

Note: RPG programs often use copy statements that reference Database Description or Device Description files rather than copybooks. If your application uses copy statements to reference these types of files, you need to verify the files and generate copybooks for the application before you verify program files. For instructions on generating copybooks, see *Preparing Projects* in the product documentation set.

Preparing for RPG Source Files Registration

If the source files are not in RPGIV format, then the appropriate IBM RPG conversion tool must be run to convert them to the RPGIV format before the files are downloaded:

- RPGII: Convert to RPGIII using a manual process or direct to RPGIV using a third party tool such as Target/400 or RPG Tool Box
- RPGIII: Convert to RPGIV using CVTRPGSRC (IBM)
- RPG400: Convert to RPGIV using CVTRPGSRC (IBM)

Remember: The iSeries is an EBCDIC platform, so ensure the correct conversion is performed. This might require changes to the code page used for unprintable characters.

SQL Support

Enterprise Analyzer supports DB2 UDB for z/OS, Version 8 (DCLGENs, SQL, DDL).

Renaming DCLGEN Include Files

Installations that use DCLGEN include files with the same names as ordinary include files should rename the DCLGEN includes with a DCLGEN prefix and dot (.) separator, so that both types of file can be registered: ATTR.<valid extension>, for example, and DCLGEN.ATTR.<valid extension>. When the parser encounters `EXEC SQL INCLUDE <name>`, it first searches for DCLGEN.<name>.<valid extension>, and if not found, then <name>.<valid extension>.

Note: Unresolved references to library members are always reported with the longest name. This means that if you subsequently register a missing include file 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.

Prefixes for SQL Names

To maintain uniqueness of ERD entity names, EA 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

Enterprise Analyzer supports Microsoft Visual Basic 6.0. Follow the steps below to analyze VB applications in Enterprise Analyzer.

1. Use the XDL.exe utility shipped with EA 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: Enterprise Analyzer provides Library Description files for ActiveX system libraries in the EA \Templates\VB directory.
2. Register the VB application in Enterprise Analyzer. The following objects must be registered:
   - VB project files (.vbp)
   - System Library Description files (.xdl) in the EA \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, EA parses the source files in appropriate order, taking account of the dependencies between file types.

   Bear in mind that dependencies between EA 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 Enterprise 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 EA 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)

Enterprise Analyzer provides Library Description files for ActiveX system libraries in the EA \Templates\VB folder. Use the XDL.exe utility shipped with EA 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 *EA Home*\Bin\XDL.exe *component_name.file_extension* "C:\XDLS*component_name*.XDL"
   - In the EA \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 #

WFL Support

Enterprise Analyzer supports Unisys WFL, ClearPath MCP, Release 7.