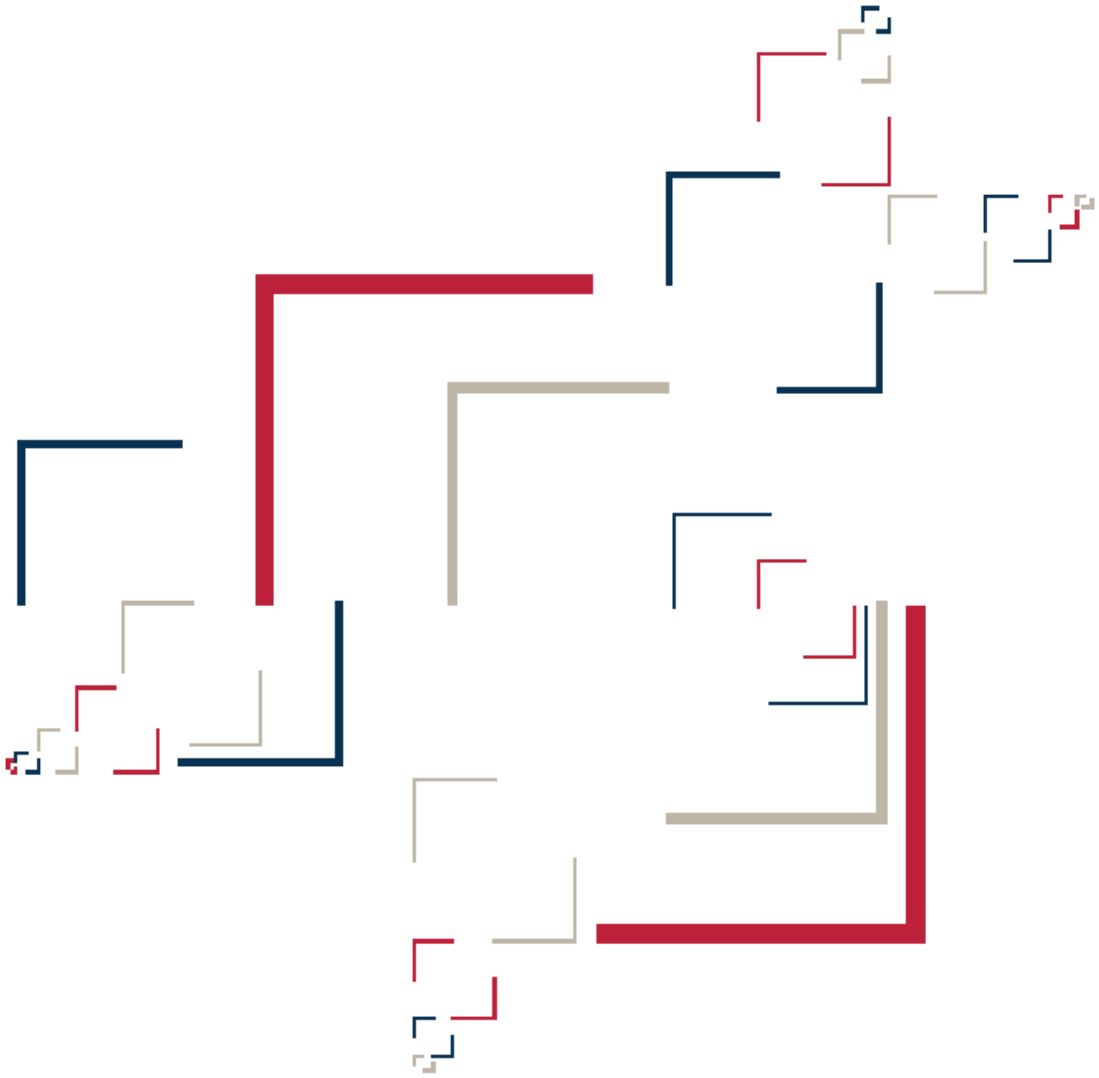

Micro Focus®

Modernization Workbench™

PL/I for OS/390 Support Guide

MICRO FOCUS®

Contents

1 Supported Platforms

This document describes Modernization Workbench (MW) support for PL/I and related platforms:

- VisualAge PL/I for OS/390, Version 2. See *VisualAge PL/I Language Reference,* Publication No. GC26-9178-00, IBM, 1996.
- CICS Transaction Server for OS/390, Version 1 Release 3
  - For CICS commands, see *CICS Application Programming Reference, CICS Transaction Server for OS/390, Release 3,* Publication No. SC33-1688-35, IBM, 2000.
  - For Basic Mapping Support (BMS), see *CICS Application Programming Reference, CICS Transaction Server for OS/390*, Release 3, Publication No. SC33-1688-35, IBM, 2000.
  - For resource definition, see *CICS Resource Definition Guide, CICS Transaction Server for OS/390, Release 3*, Publication No. SC33-1684-34, IBM, 2000.
- IMS, Version 7
  - For DL/I calls, see *Application Programming: Database Manager*, Publication No. SC26-9422-01, IBM, 2001.
  - For PSB and DBD files, see *Utilities Reference: System*, Publication No. SC26-9441-01, IBM, 2001.
  - For MFS files, see *Application Programming: Transaction Manager*, Publication No. SC26-9425-02, IBM, 2002.
- JCL, OS/390 Version 2 Release 10
  - For JCL files, see *OS/390 MVS JCL Reference*, Publication No. GC28-1757-09, IBM, 2000.
  - For sort cards, *DFSORT Application Programming Guide, Release 14,* Publication No. SC33-4035-21, IBM, 2002.
- SQL. DB2 Universal Database for z/OS, Version 8. See *DB2 UDB for z/OS SQL Reference*, Publication No. SC18-7426-02, IBM, 2005.

2 PL/I Technical Reference

This section describes MW support for PL/I applications:

- "Support Notes" on page 3 describes MW limitations, caveats, and special usage for PL/I applications.
- "Supported Complexity Metrics" on page 9 describes the supported complexity metrics for objects in the PL/I model.
- "Relationship Projections from PL/I Statements" on page 16 describes the relationships generated from PL/I statements in programs and support files.

Support Notes

These notes describe MW limitations, caveats, and special usage for PL/I applications. Make sure to check the *Release Notes* on the installation CD for any late-breaking support information.

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 HyperView

Although macro statements are not captured as HyperCode in the HyperView Source pane, constructs resulting from macro expansion are correctly modeled in the HyperView 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 applicable to:

- IMS Analysis
- Domain-Based Extraction
- 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.

Supported Complexity Metrics

The complexity of an object is an estimate of how difficult it is to maintain, analyze, transform, and so forth. This section describes the supported metrics for objects in the PL/I model.

PL/I File Complexity Metrics

The table below describes the supported complexity metrics for the PL/I File object.

Metric	Description
Blank Lines	Number of blank lines of source. Blank lines in a comment block are not included.
Comment Lines	Number of lines of source containing comments only and no code. Includes blank lines in a comment block.
Include Statements	Number of include statements: %INCLUDE, EXEC SQL INCLUDE.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Macro Assignments	Number of assignments to global macro variables outside any macro procedure.
Macro Declarations	Number of global macro variables. Macro variables within macro procedures are not counted.
Macro Lines	Number of lines for include statements, declarations, macro procedures, and other %-statements, excluding macro invocations. Blank lines and comments lines are ignored.
Macro Procedures	Number of macro procedures declared in the file.
Macro Statements	Number of %-statements and statements inside a macro procedure, excluding macro invocations.

Metric	Description
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of include statements in the file and any used include files.

PL/I Include File Complexity Metrics

The table below describes the supported complexity metrics for the PL/I Include File object.

Metric	Description
Blank Lines	Number of blank lines of source. Blank lines in a comment block are not included.
Comment Lines	Number of lines of source containing comments only and no code. Includes blank lines in a comment block.
Include Statements	Number of include statements: %INCLUDE, EXEC SQL INCLUDE.
Lines with Both Comments and Code	Number of lines of source containing both comments and code.
Macro Assignments	Number of assignments to global macro variables outside any macro procedure.
Macro Declarations	Number of global macro variables. Macro variables within macro procedures are not counted.
Macro Lines	Number of lines for include statements, declarations, macro procedures, and other %-statements, excluding macro invocations. Blank lines and comments lines are ignored.
Macro Procedures	Number of macro procedures declared in the file.
Macro Statements	Number of %-statements and statements inside a macro procedure, excluding macro invocations.

Metric	Description
Source Lines	Number of lines of source, including blank lines and comments.

PL/I Program Complexity Metrics

The table below describes the supported complexity metrics for the PL/I program object.

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on): IF, DO, SELECT.
Computational Statements	Number of statements performing arithmetic calculations.
Conditional Complexity	Binary Decisions plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions: IF, DO, SELECT.
Cyclomatic Complexity	v(G) = e - n + 2, where v(G) is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). v(G) = DE + 1, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields). The implicit variable DFHEIBLK for CICS statements is counted as a data element.

Metric	Description
Dead Data Elements	Number of declared data items in dead internal procedures.
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Lines	Number of dead lines in programs and used include files before macro preprocessing. Dead lines are source lines containing Dead Data Elements or Dead Statements. If an include file is included multiple times, it is counted each time.
Dead Lines from Includes	Number of dead lines in include files and used include files before macro preprocessing. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes. If an include file is included multiple times, it is counted each time.
Dead Statements	Number of dead statements in programs and used include files. A dead statement is a procedural statement that can never be reached during program execution. DECLARE statements are not calculated as dead statements.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	D = (n1 / 2) * (N2 / n2), where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points: PROCEDURE (top-level), ENTRY.
Error Estimate	B = E**(2/3) / 3000, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). v(G) for reduced graph without D-structured primes.

Metric	Description
Executable Statements	All statements, except BEGIN, DECLARE, DO (block), END, ENTRY, PACKAGE, and PROCEDURE.
Extended Cyclomatic Complexity	Cyclomatic Complexity plus Logical Operators in Conditions. Number of all paths in the program.
Function Points	Lines of Code/K, where K=67. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs.
Inner Call Statements	Number of statements that invoke Inner Procedures.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: inner PL/I procedures.
Intelligent Content	I = L * V, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations: DISPLAY, PUT, GET, READ, REWRITE, WRITE, DELETE, OPEN, CLOSE, LOCATE, UNLOCK.
Lines of Code	Number of lines of code before macro preprocessing, including include files, but not including comments and blank lines. If an include file is included multiple times, it is counted each time.
Logical Operators in Conditions	Number of binary logical operators used in conditions.
Loop Statements	Number of repetitively executing statements: DO.

Metric	Description
Maintainability Index	MI = 171 - 5.2 * ln (PgmVolume) - 0.23 * ExtCycComp - 16.2 * ln (LOC) + 50 * sin (sqrt (2.46 * CommentLines/SourceLines)), where PgmVolume is Program Volume, ExtCycComp is Extended Cyclomatic Complexity, LOC is Lines of Code, CommentLines is Comment Lines, and SourceLines is Source Lines.
Nesting Level	Maximum nesting of conditional statements within conditional statements (0 if no conditional statements, 1 if no nesting).
Non-returning Calls	Number of non-returning calls, such as CICS XCTL statements.
Operands	Number of operand occurrences (N2). Operands are variables and literals used in operators. Compare Unique Operands.
Operators	Number of operator occurrences (N1). Operators are executable statements and unary and binary operations: +, -, *, /, **, ||, ^, &, |, NOT, AND, OR, <, <=, >, >=, =, (subscript). Compare Unique Operators.
Parameters	Number of PL/I PROCEDURE parameters.
Pointers	Number of data elements declared as pointers.
Program Length	N = N1 + N2, where N1 is Operators and N2 is Operands.
Program Level	L = 1 / D, where D is Difficulty.
Program Volume	V = N * log2(n), where N is Program Length and n is Vocabulary. Minimum number of bits required to code the program.
Programming Effort	E = V / L, where V is Program Volume and L is Program Level. Estimated mental effort required to develop the program.
Programming Time	T = E / 18, where E is the Programming Effort and 18 is Stroud's Number. Estimated amount of time required to implement the algorithm, in seconds.
Returning Calls	Number of returning calls.

Metric	Description
Sliceable Dead Lines	Number of Dead Lines that can be sliced using the Application Architect Dead Code Elimination method. Dead procedures containing either preprocessor statements or statements subject to macro preprocessor replacement are not counted as sliceable dead lines, since it is not always possible to determine whether they can be safely removed. Lines from include files are not counted.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations. Compare Operators.
Vocabulary	n = n1 + n2, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

Relationship Projections from PL/I Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for PL/I model objects from the statements in programs and support files.

PL/I File Relationship Projections

The PL/I File object represents the source file for a PL/I program. The table below describes the relationships generated from PL/I statements in the source file.

Statement	Format	Relationship	Entities
%INCLUDE	%INCLUDE member	PL/I File Includes PL/I Include File	For resolved files: PliInclude.Name = <resolved-name> For unresolved files: PliInclude.Name = <member>
PROCEDURE	name: PROC [(parms)] [options(...)] ... options(Main)	PL/I File Defines Program	Program.Name = <name> Program. MainProgram= True

Pl/I Include File Relationship Projections

The PL/I Include File object represents a PL/I include file. The table below describes the relationships generated from PL/I statements in the include file.

Statement	Format	Relationship	Entities
%INCLUDE	%INCLUDE member	PL/I Include File Includes PL/I Include File	For resolved files: PliInclude.Name = <resolved-name> For unresolved files: PliInclude.Name = <member>

PL/I Program Relationship Projections

The Program object represents a PL/I program. The table below describes the relationships generated from PL/I statements in the program.

Statement	Format	Relationship	Entities
CALL	DCL name ENTRY; ... CALL name ...	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL (dynamic)	DCL varname ENTRY VARIABLE; ... CALL varname ...	Program Calls Program Entry Decision	Decision attributes: Name = <program-name>@<internal-name> #Also Known As = <program-name>.Calls.<varname> Decision Type = PROGRAMENTRY…
ENTRY	name: ENTRY ...	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry.MainEntry = False
PROCEDURE	name: PROC [(parms)] [options(...)] ...	Program Has Program Entry Point	Program.Name = <name> ProgramEntry.MainEntry = True

Statement	Format	Relationship	Entities
File Description	DCL name FILE; OPEN FILE (name); DCL name FILE; OPEN FILE (name) TITLE ('title');	See CRUD statements below.	external-file-name = <name> external-file-name = <title> File attributes: Name = <program-name>.external-file-name DD Name = external-file-name File Type = FILE ***NOTE:*** *A File object is generated only when the first CRUD statement for the file is encountered. File attributes do not depend on the CRUD statement itself.*
File Description (dynamic)	DCL varname FILE VARIABLE; OPEN FILE (varname); DCL name FILE; OPEN FILE (name) TITLE (vartitle);	See CRUD statements below.	decision-var = <varname> decision-var = <name> Decision attributes: Name = <program-name>@<internal-name> #Also Known As =<program-name>.<decision-rel>.<decision-var> Decision Type = DATAPORT ***NOTE:*** *A Decision object is generated only when the first CRUD statement for the file is encountered. Only the Also Known As attribute depends on the CRUD statement itself.*
DELETE	DELETE FILE (name) …	Program Deletes From File	See File Description for File attributes.
DELETE (dynamic)	DELETE FILE (varname) …	Program Deletes From File Decision	See File Description (dynamic) for Decision attributes.
GET	GET FILE (name) …	Program Reads File	See File Description for File attributes.

Statement	Format	Relationship	Entities
GET (dynamic)	GET FILE (varname) …	Program Reads File Decision	See File Description (dynamic) for Decision attributes.
PUT	PUT FILE (name) …	Program Inserts Into File	See File Description for File attributes.
PUT (dynamic)	PUT FILE (varname) …	Program Inserts Into File Decision	See File Description (dynamic) for Decision attributes.
READ	READ FILE (name) …	Program Reads File	See File Description for File attributes.
READ (dynamic)	READ FILE (varname) …	Program Reads File Decision	See File Description (dynamic) for Decision attributes.
REWRITE	REWRITE FILE (name) …	Program Updates File	See File Description for File attributes.
REWRITE (dynamic)	REWRITE FILE (varname) …	Program Updates File Decision	See File Description (dynamic) for Decision attributes.
WRITE	WRITE FILE (name) …	Program Inserts Into File	See File Description for File attributes.
WRITE (dynamic)	WRITE FILE (varname) …	Program Inserts Into File Decision	See File Description (dynamic) for Decision attributes.

3 SQL Technical Reference

This section describes MW support for EXEC SQL statements in programs and SQL DDL statements in DDL files:

- "Support Notes" on page 21 describes MW limitations, caveats, and special usage for SQL.
- "Complexity Metrics" on page 22 describes the supported complexity metrics for objects in the SQL model.
- "Relationship Projections from EXEC SQL Statements" on page 23 describes the relationships generated from EXEC SQL statements in programs.
- "Relationship Projections from SQL DDL Statements" on page 24 describes the relationships generated from SQL DDL statements in DDL files.

Support Notes

These notes describe MW limitations, caveats, and special usage for SQL. Make sure to check the *Release Notes* on the installation CD for any late-breaking support information.

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.*

Complexity Metrics

The complexity of an object is an estimate of how difficult it is to maintain, analyze, transform, and so forth. This section describes the supported metrics for objects in the SQL model.

DDL File Complexity Metrics

The table below describes the supported complexity metrics for the DDL File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Columns	Number of columns.
Foreign Keys	Number of foreign keys.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Primary Keys	Number of primary keys.
Source Lines	Number of lines of source, including blank lines and comments.
Tables	Number of tables.

Relationship Projections from EXEC SQL Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for SQL model objects from the EXEC SQL statements in programs.

Program Relationship Projections

The Program object represents a Cobol program. The table below describes the relationships generated from EXEC SQL statements in the program.

Statement	Format	Relationship	Entities
ALTER TABLE	ALTER TABLE <table-name> …	Program Manipulates Table	Table.Name = <table-name>
CREATE INDEX	CREATE INDEX <index-name> ON <table-name> …	Program Manipulates Table	Table.Name = <table-name>
CREATE TABLE	CREATE TABLE <table-name>...	Program Manipulates Table	Table attributes: Name = <table-name> Origin = <source-file-path>
DELETE	DELETE FROM <table-name>…	Program Deletes From Table	Table.Name = <table-name>
DROP TABLE	DROP TABLE <table-name>	Program Manipulates Table	Table.Name = <table-name>
INSERT	INSERT INTO <table-name> …	Program Inserts Into Table	Table.Name = <table-name>
SELECT	SELECT … FROM table-name	Program Reads Table	Table.Name = <table-name>
UPDATE	UPDATE <table-name> …	Program Updates Table	Table.Name = <table-name>

Relationship Projections from SQL DDL Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for SQL model objects from the SQL DDL statements in DDL files.

DDL File Relationship Projections

The DDL File object represents a Data Definition Language file. The table below describes the relationships generated from SQL DDL statements in the file. Note the following:

- To maintain uniqueness of ERD entity names, MW 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.

Statement	Format	Relationship	Entities
ALTER TABLE	ALTER TABLE table-name …	DDL File Refers To Table	Table attributes: Name = <table-name> Is View = False
COMMENT	COMMENT ON [TABLE] table-name …	DDL File Refers To Table	Table.Name = <table-name>
CREATE ALIAS	CREATE ALIAS alias-name ON table-name …	DDL File Defines Table Table Represents Table	Table attributes: Name = <alias-name> Is View = True Source = "DBSchema.mdb' Origin = <source-file-path> Table.Name = <table-name>
CREATE INDEX	CREATE INDEX index-name ON table-name …	DDL File Refers To Table	Table.Name = <table-name>

Statement	Format	Relationship	Entities
CREATE SYNONYM	CREATE SYNONYM synonym FOR authorization-name.table-name…	DDL File Defines Table Table Represents Table	Table attributes: Name = <synonym> Is View = True Source = "DBSchema.mdb" Origin = <source-file-path> Table.Name = <table-name>
CREATE TABLE	CREATE TABLE table-name …	DDL File Defines Table	Table attributes: Name = <table-name> Is View = False Source = "DBSchema.mdb" Origin = <source-file-path>
CREATE VIEW	CREATE VIEW view-name …AS SELECT … FROM table-name	DDL File Defines Table Table Represents Table	Table attributes: Name = <view-name> Is View = True Source = "DBSchema.mdb" Origin = <source-file-path> Table.Name = <table-name>
Referential constraint	FOREIGN KEY key REFERENCES base-table…	DDL File Refers To Table	Table.Name = <base-table>
SET CURRENT SQLID	SET CURRENT SQLID = 'user-name'		sqlid = <user-name>

4 JCL Technical Reference

This section describes MW support for JCL files, JCL procedures, and control card files:

- "Support Notes" on page 27 describes MW limitations, caveats, and special usage for JCL applications.
- "Complexity Metrics" on page 29 describes the supported complexity metrics for objects in the JCL model.
- "Relationship Projections from JCL Statements" on page 31 describes the relationships generated from statements in JCL files and procedures.

Support Notes

These notes describe MW limitations, caveats, and special usage for JCL applications. Make sure to check the *Release Notes* on the installation CD for any late-breaking support information.

External Control Cards Registration Requirements

Both inline cards (DD *) and external cards (DSN=) are supported. 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:

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

the source file name must be MY.SORTCARDS.LIB.FILE1.srt.

For a PDS member:

```
//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:

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

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

Sort Cards Verification Requirements

Before verification, specify the names of the sort utilities you use in the Sort Program Aliases workspace verification option for JCL files. The defaults are SORT, DFSORT, and SYNCSORT.

Sort Cards Parser Output

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.

Detecting Programs Started by Driver Utilities

Use the Driver Utility Analysis feature to model programs started by a driver utility. For more information, see the *Parser Reference Manual* in the workbench documentation set.

Complexity Metrics

The complexity of an object is an estimate of how difficult it is to maintain, analyze, transform, and so forth. This section describes the supported metrics for objects in the JCL model.

JCL File Complexity Metrics

The table below describes the supported complexity metrics for the JCL File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Control Cards Usages	Number of DD statements referencing control cards identified in Legacy.xml to generate program to control card relationships.
EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures.
EXEC In-stream Procedure Steps	Number of EXEC statements invoking instream procedures.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.
Steps	Number of top-level steps in the job (not including steps in invoked procedures).
Total EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures in the job and invoked procedures. If a procedure is invoked multiple times, it is counted each time.
Total Include Statements	Number of include statements in the job, invoked procedures, and any include files.

JCL Procedure Complexity Metrics

The table below describes the supported complexity metrics for the JCL Procedure File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Control Cards Usages	Number of DD statements referencing control cards identified in Legacy.xml to generate program to control card relationships.
EXEC Cataloged Procedure Steps	Number of EXEC statements invoking cataloged procedures.
EXEC In-stream Procedure Steps	Number of EXEC statements invoking instream procedures.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.
Steps	Number of steps in the procedure.

Job Complexity Metrics

The table below describes the supported complexity metrics for the Job object.

Metric	Description
Steps	Number of steps in the job and invoked procedures. If a procedure is invoked multiple times, it is counted each time.

Control Cards File Complexity Metrics

The table below describes the supported complexity metrics for the Control Cards File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Relationship Projections from JCL Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for JCL model objects from the statements in JCL files and procedures.

JCL File Relationship Projections

The JCL File object represents a Job Control Language file. The tables below describe the relationships generated from JCL statements in the file.

Note the following:

- 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 speci-

fied, 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>]…
```

- An invoked program is known as a system program if it is defined with a sort utility alias in the workspace verification options for a JCL file, or specified in the <SystemPrograms> section of the Legacy.xml file.

Statement	Format	Relationship	Entities
DD (program)	//[execName] EXEC PGM = ProgName …//ddName DD DSN = DSName,…	Job Has Data Connector Data Connector Refers To Data Store Data Connector Refers To File	Data Connector attributes: Name = <Job.Name>.<UniqueID> Program Entry Point = <ProgName> DD Name = <ddName> Step Name = <execName> Step Full Name = <StepPath> Step Number = <StepNumber> Datastore.Name = <dsn-name> Datastore.DSN = <dsn-name> File.Name = <ProgName>.<ddName> File.PortName = <ddName>

Statement	Format	Relationship	Entities
DD (system program)	//[execName] EXEC PGM = SysProgName …//ddName DD DSN = DSName,…	Job Has Data Connector Data Connector Refers To Data Store Connector Is Read In System Program Connector Is Written In System Program	Data Connector attributes: Name = <Job.Name>. <UniqueID> Program Entry Point = <ProgName> DD Name = <ddName> Step Name = <execName> Step Full Name = <StepPath> Step Number = <StepNumber> Datastore.Name = <dsn-name> Datastore.DSN = <dsn-name> Sysprogram.Name = <SysProgName>
EXEC (program)	//EXEC PGM = ProgName	Job Runs Program Entry Point	ProgramEntry.Name = <ProgName>
EXEC (system program)	//EXEC PGM = SysProgName	Job Runs System Program	Sysprogram.Name = <SysProgName>
EXEC (procedure)	//[execName] EXEC [PROC =] ExternalProc Name	JCL File Executes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: JclProc.Name = <ExternalProcName>
INCLUDE	//INCLUDE MEMBER = member	JCL File Includes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: Jclproc.Name = <member>
-INC (Librarian)	-INC member	JCL File Includes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: Jclproc.Name = <member>

Statement	Format	Relationship	Entities
JOB	//jobName JOB [parameters]	JCL File Defines Job	Job.Name = <Jcl.Name>.<jobName> Job.JobName = <jobName> Job.StepsNum = <JobStepsNumber>

JCL Procedure Relationship Projections

The JCL Procedure File object represents a Job Control Language Procedure file. The tables below describe the relationships generated from JCL statements in the file.

Statement	Format	Relationship	Entities
EXEC	//[execName] EXEC [PROC=]External ProcName	JCL File Executes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: JclProc.Name = <ExternalProcName>
INCLUDE	// INCLUDE MEMBER = member	JCL Procedure Includes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: JclProc.Name = <member>
-INC (Librarian)	-INC member	JCL Procedure Includes JCL Procedure	For resolved files: JclProc.Name = <resolved-name> For unresolved files: JclProc.Name = <member>

5 CICS Technical Reference

This section describes MW support for BMS files and copybooks, CSD, FCT, and PCT files, and CICS statements in programs:

- "Support Notes" on page 35 describes MW limitations, caveats, and special usage for CICS applications.
- "Complexity Metrics" on page 38 describes the supported complexity metrics for objects in the CICS model.
- "Relationship Projections from BMS Statements" on page 41 describes the relationships generated from statements in BMS files and copybooks.
- "Relationship Projections from CSD, FCT, and PCT Statements" on page 42 describes the relationships generated from statements in CSD, FCT, and PCT files.
- "Relationship Projections from CICS Statements" on page 46 describes the relationships generated from CICS statements in programs.

Support Notes

These notes describe MW limitations, caveats, and special usage for CICS applications. Make sure to check the *Release Notes* on the installation CD for any late-breaking support information.

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.

Complexity Metrics

The complexity of an object is an estimate of how difficult it is to maintain, analyze, transform, and so forth. This section describes the supported metrics for objects in the CICS model.

BMS File Complexity Metrics

The table below describes the supported complexity metrics for the BMS File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of include statements in the file and any used include files.

BMS Copybook File Complexity Metrics

The table below describes the supported complexity metrics for the BMS Copybook File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.

Metric	Description
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Screen Complexity Metrics

The table below describes the supported complexity metrics for the Screen object.

Metric	Description
Hidden Fields	Number of hidden fields.
Input Fields	Number of input fields.
Input/Output Fields	Number of input/output fields.
Output Fields	Number of output fields.

CSD File Complexity Metrics

The table below describes the supported complexity metrics for the CSD File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Source Lines	Number of lines of source, including blank lines and comments.

FCT File Complexity Metrics

The table below describes the supported complexity metrics for the FCT File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

PCT File Complexity Metrics

The table below describes the supported complexity metrics for the PCT File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.

Metric	Description
Source Lines	Number of lines of source, including blank lines and comments.

Relationship Projections from BMS Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for CICS model objects from the statements in BMS files and copybooks.

BMS File Relationship Projections

The BMS File object represents a BMS file in a CICS application. The table below describes the relationships generated from statements in the BMS file.

Statement	Format	Relationship	Entities
COPY	COPY member	BMS File Includes BMS Copybook File	For resolved files: BmsCopy.Name = <resolved-name> For unresolved files: BmsCopy.Name = <member>
DFHMDI	mapset DFHMSD... map DFHMDI ...	BMS File Defines Screen	Map.Name= <mapset>.<map>
DFHMDI (no mapset)	map DFHMDI ...	BMS File Defines Screen	Map.Name = <source-filename-without-extension>. <map>

BMS Copybook File Relationship Projections

The BMS Copybook File object represents a BMS copybook included in a BMS file or in another BMS copybook. The table below describes the relationships generated from statements in the BMS copybook file.

Statement	Format	Relationship	Entities
COPY	COPY member	BMS Copybook File Includes BMS Copybook File	For resolved files: BmsCopy.Name = <resolved-name> For unresolved files: BmsCopy.Name= <member>

Relationship Projections from CSD, FCT, and PCT Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for CICS model objects from statements in CSD, FCT, and PCT files.

CSD File Relationship Projections

The CSD File object represents a CICS System Definition dataset. The table below describes the relationships generated from statements in the CSD file.

Statement	Format	Relationship	Entities
DEFINE DOCTEMPLATE	DEFINE DOCTEMPLATE (doc-name) GROUP (group-name)...	CSD File Defines Document Template	DocTemplate.Name = <doc-name>

Statement	Format	Relationship	Entities
DEFINE FILE	DEFINE FILE (file-name) GROUP (group-name) DSNAME (dsn-name) LOAD(YES)… ***NOTE:*** *Nothing is generated if LOAD (NO) is specified.*	CSD File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>
DEFINE FILE (base file)	NSRGROUP (base-group) Base file: DEFINE FILE (base-group) DSNAME (base-dsn-name)	Data Store Based On Data Store	BaseDatastore.Name = <base-dsn-name>
DEFINE TRANSACTION	DEFINE TRANSACTION (tran-name) GROUP (group-name) PROGRAM (prg-name)...	CSD File Defines Transaction Transaction Initiates Program Entry Point	Transaction.Name = <tran-name> ProgramEntry.Name = <prg-name> Program.Root = True ***NOTE:*** *Root is empty if Program does not exist in the repository before CSD file verification.*

FCT File Relationship Projections

The FCT File object represents a CICS File Control Table. The table below describes the relationships generated from statements in the FCT file.

Statement	Format	Relationship	Entities
DFHFCT DATASET	DFHFCT TYPE = DATASET, DATASET = data-set, DSNAME = dsn-name...	FCT File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>
DFHFCT DATASET (base file)	BASE = base-name Base file: DFHFCT TYPE = DATASET, DATASET = base-name, DSNAME = base-dsn-name… ***NOTE:*** *Base file may be defined with any TYPE = FILE or TYPE = DATASET statement.*	Data Store Based On Data Store	BaseDatastore.Name = <base-dsn-name>
DFHFCT FILE	DFHFCT TYPE = FILE, FILE = file-name, DSNAME = dsn-name...	FCT File Has Data Connector Data Connector Refers To Data Store	Data Connector attributes: Name = CICS.<file-name> DD Name = <file-name> Program Entry Point = * Datastore.Name = <dsn-name>

Statement	Format	Relationship	Entities
DFHFCT FILE (base file)	BASE = base-name Base file: DFHFCT TYPE = FILE, FILE = base-name, DSNAME = base-dsn-name... ***NOTE:*** *Base file may be defined with any TYPE=FILE or TYPE=DATASET statement.*	Data Store Based On Datastore	BaseDatastore.Name = <base-dsn-name>

PCT File Relationship Projections

The PCT File object represents a CICS Program Control Table. The table below describes the relationships generated from statements in the PCT file.

Statement	Format	Relationship	Entities
DFHPCT	DFHPCT TYPE = ENTRY, TRANSID = tran-name, PROGRAM = prg-name	PCT File Defines Transaction Transaction Initiates Program Entry Point	Transaction.Name = <tran-name> ProgramEntry.Name = <prg-name> Program.Root = True ***NOTE:*** *Root is empty if Program does not exist in the repository before PCT file verification.*

Relationship Projections from CICS Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for CICS model objects from CICS statements in programs.

Program Relationship Projections

The Program object represents a Cobol program. The tables below describe the relationships generated from CICS statements in the program.

Statement	Format	Relationship	Entities
	any EXEC CICS		Program.EnvFlags = +CICS ***NOTE:*** *EnvFlags may contain other environment codes, so search as follows: Like '*+CICS*'*
DELETE	DELETE FILE ('file-name') …	Program Deletes From File	File attributes: Name = <program-name>.file-name DD Name = file-name File Type = FILE Online Flag = true
DELETE (dynamic)	DELETE FILE (file-name) …	Program Deletes From File Decision	Decision attributes: Name = <program-name>@<internal-name> # Also Known As = <program-name>.DeletesDataPort.<file-name> Decision Type = DATAPORT…

Statement	Format	Relationship	Entities
DOCUMENT	DOCUMENT CREATE TEMPLATE ('name') … DOCUMENT INSERT TEMPLATE ('name') …	Program Uses Document Template	DocTemplate.Name = <name>
DOCUMENT (dynamic)	DOCUMENT CREATE TEMPLATE (name) … DOCUMENT INSERT TEMPLATE (name) …	Program Uses Document Template Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program- name>. UsesDocTemplate <name> Decision Type = DOCTEMPLATE…
INVOKE	INVOKE WEBSERVICE ('name') OPERATION ('opname') …	Program Invokes Service	Service.Name = <name>.<opname>
INVOKE (dynamic)	INVOKE WEBSERVICE (name) OPERATION ('opname') … INVOKE WEBSERVICE ('name') OPERATION (opname) … INVOKE WEBSERVICE (name) OPERATION (opname) …	Program Invokes Service Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program- name>. InvokesService.<name> Decision Type = SERVICE…
LINK	LINK PROGRAM ('pgm-name') …	Program Links Program Entry Point	ProgramEntry.Name = <program-name> ***NOTE:*** *If literal is long, only 8 leading characters are used as program name.*

Statement	Format	Relationship	Entities
LINK (dynamic)	LINK PROGRAM (pgm-name) …	Program Links Program Entry Point Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program- name>. Links. <program-name> Decision Type = PROGRAMENTRY…
READ READNEXT READPREV	READ FILE ('file-name') … READNEXT FILE ('file-name') … READPREV FILE ('file-name') …	Program Reads File	File attributes: Name = <program-name>. file-name DD Name = file-name File Type = FILE Online Flag = true
READ READNEXT READPREV (dynamic)	READ FILE (file-name) … READNEXT FILE (file-name) … READPREV FILE (file-name) …	Program Reads File Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. ReadsDataPort. <file-name> Decision Type = DATAPORT…
RECEIVE	RECEIVE …		Program.OnlineFlag = true
RECEIVE MAP	RECEIVE MAP ('map-name') MAPSET ('mapset') … RECEIVE MAP ('map-name')	Program Receives Screen	Screen.Name = <mapset>. <map-name> Program.OnlineFlag = true Screen.Name = <map-name>.<map-name> Program.OnlineFlag = true

Statement	Format	Relationship	Entities
RECEIVE MAP (dynamic)	RECEIVE MAP (map-name) MAPSET ('mapset') … RECEIVE MAP (map-name) RECEIVE MAP ('map-name') MAPSET (mapset) … RECEIVE MAP (map-name) MAPSET (mapset) …	Program Receives Screen Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Receives. <map-name> Decision Type = MAP … Program.OnlineFlag = true
RETURN	RETURN TRANSID ('name') …	Program Starts Transaction	Transaction.Name = <name> ***NOTE:*** *If literal is long, only 4 leading characters are used as program name.*
RETURN (dynamic)	RETURN TRANSID (name) …	Program Starts Transaction Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Starts.<name> Decision Type = TRANSACTION
REWRITE	REWRITE FILE ('file-name') …	Program Updates File	File attributes: Name = <program-name>. file-name DD Name = file-name File Type = FILE Online Flag = true

Statement	Format	Relationship	Entities
REWRITE (dynamic)	REWRITE FILE (file-name) …	Program Updates File Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. UpdatesDataPort. <file-name> Decision Type = DATAPORT…
SEND MAP	SEND MAP ('map-name') MAPSET ('mapset') … SEND MAP ('map-name')	Program Sends Screen	Screen.Name = <mapset>. <map-name> Program.OnlineFlag = true Screen.Name = <map-name>.<map-name> Program.OnlineFlag = true
SEND MAP (dynamic)	SEND MAP (map-name) MAPSET ('mapset') … SEND MAP (map-name) SEND MAP ('map-name') MAPSET (mapset) … SEND MAP (map-name) MAPSET (mapset) …	Program Sends Screen Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Sends.<mapset> Decision Type = MAP … Program.OnlineFlag = true
SEND	SEND		Program.OnlineFlag = true
START	START TRANSID ('name') …	Program Starts Transaction	Transaction.Name = <name> ***NOTE:*** *If literal is long, only 4 leading characters are used as program name.*

Statement	Format	Relationship	Entities
START (dynamic)	START TRANSID (name) …	Program Starts Transaction Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Starts.<name> Decision Type = TRANSACTION
WRITE	WRITE FILE ('file-name') …	Program Inserts Into File	File attributes: Name = <program-name>. file-name DD Name = file-name File Type = FILE Online Flag = true
WRITE (dynamic)	WRITE FILE (file-name) …	Program Inserts Into File Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. InsertsDataPort. <file-name> Decision Type = DATAPORT…
XCTL	XCTL PROGRAM ('pgm-name') …	Program Xctls To Program Entry Point	ProgramEntry.Name = <pgm-name> ***NOTE:*** *If literal is long, only 8 leading characters are used as program name.*
XCTL (dynamic)	XCTL PROGRAM (pgm-name) …	Program Xctls To Program Entry Point Decision	Decision attributes: Name = <program-name>@ <internal-name> # Also Known As = <program-name>. Xctls.<pgm-name> Decision Type = PROGRAMENTRY…

6 IMS Technical Reference

This section describes RTW support for MFS files and MFS include files, DBD, PSB, and PSB copybook files, System Definition files, and call-level (CALL 'CBLTDLI') and command-level (EXEC DLI) statements in programs:

- "IMS Support Notes" on page 53 describes RTW limitations, caveats, and special usage for IMS applications.
- "IMS Complexity Metrics" on page 58 describes the supported complexity metrics for objects in the IMS model.
- "Relationship Projections from DBD and PSB Statements" on page 62 describes the relationships generated from statements in DBD and PSB files.
- "Relationship Projections from System Definition Statements" on page 66 describes the relationships generated from statements in System Definition files.

IMS Support Notes

These notes describe RTW limitations, caveats, and special usage for IMS applications. Make sure to check the Release Notes on the installation CD for any late-breaking support information.

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.

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
(K4,ISRT,P,OP,SEG_SSA(10));
END A;
	EPUF.P8CCLA	EPUF.P8CCLA,ImsUnknown

Multiple Unsupported Ports

Multiple unsupported ports are represented as a single port in the HyperView 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

IMS Complexity Metrics

The complexity of an object is an estimate of how difficult it is to maintain, analyze, transform, and so forth. This section describes the supported metrics for objects in the IMS model.

MFS File Complexity Metrics

The table below describes the supported complexity metrics for the MFS File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Screens	Number of screens.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of Include Statements in the file and any used include files.

MFS Include File Complexity Metrics

The table below describes the supported complexity metrics for the MFS Include File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.

Metric	Description
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Screen Complexity Metrics

The table below describes the supported complexity metrics for the Screen object.

Metric	Description
Hidden Fields	Number of hidden fields.
Input Fields	Number of input fields.
Input/Output Fields	Number of input/output fields.
Output Fields	Number of output fields.

DBD File Complexity Metrics

The table below describes the supported complexity metrics for the DBD File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Segments	Number of segments.
Source Lines	Number of lines of source, including blank lines and comments.

PSB File Complexity Metrics

The table below describes the supported complexity metrics for the PSB File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Number of PCBs	Number of PCBs.
Source Lines	Number of lines of source, including blank lines and comments.
Total Include Statements	Number of Include Statements in the file and any used include files.

PSB Copybook File Complexity Metrics

The table below describes the supported complexity metrics for the PSB Copybook File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

System Definition File Complexity Metrics

The table below describes the supported complexity metrics for the System Definition File object.

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Entries	Number of entries.
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Relationship Projections from DBD and PSB Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for IMS model objects from statements in DBD and PSB files.

DBD File Relationship Projections

The DBD File object represents an IMS Database Description file. The table below describes the relationships generated from statements in the DBD file.

Statement	Format	Relationship	Entities
DBD	DBD NAME=db-name ACCESS= (db-type, ...) ...	DBD File Defines Hierarchical Database	HiDatabase.Name = <db-name> HiDatabase.Type = <db-type>
DATASET (GSAM only)	DATASET DD1=dd-name1, DD2=dd-name2, ...	Hierarchical Database Has Hierarchical Database Segment	HiSegment.Name = <db-name>. <db-name> HiSegment.Segment Name = <db-name> HiSegment.DDName = <dd-name1> HiSegment. DDName2 = <dd-name2>

Statement	Format	Relationship	Entities
SEGM	SEGM NAME=seg-name, ... NAME= seg-name2, ... dbname2))	Hierarchical Database Has Hierarchical Database Segment Hierarchical Database Segment Has Logical Child Hierarchical Database Segment	HiSegment.Name = <db-name>.<seg-name> HiSegment.Segment Name = <seg-name> HiSegment.Name = <db-name>.<seg-name> HiSegmentChild.Name = <db-name2>.<seg-name2> HiSegmentChild.SegmentName = <seg-name2>
LCHILD	SEGM NAME= seg-name, ... LCHILD= NAME= (seg-name, dbname)	Hierarchical Database Segment Has Logical Child Hierarchical Database Segment	HiSegment.Name = <db-name>.<seg-name> HiSegmentChild.Name = <db-name>.<seg-name> HiSegmentChild.SegmentName = <seg-name>

PSB File Relationship Projections

The PSB File object represents an IMS Program Specification Block file. The table below describes the relationships generated from statements in the PSB file.

Statement	Format	Relationship	Entities
PSBGEN	PSBGEN LANG=language, PSBNAME= psb-name, ...	PSB File Defines PSB Module	PsbModule.Name = <psb-name> PsbModule.Language = <language>

Statement	Format	Relationship	Entities
PCB	PCB TYPE=DB, DBDNAME= dbd-name, ... PCB TYPE=GSAM, DBDNAME= dbd-name, ... PCB TYPE=DB, DBDNAME= ..., PROCSEQ= dbd-name, ... PCB TYPE=GSAM, DBDNAME= ..., PROCSEQ= dbd-name, ...	PSB Module Refers To Hierarchical Database	HiDatabase.Name = <dbd-name>
SENSEG	SENSEG NAME= ..., INDICES= (dbd-name1, ... dbd-nameN) ***NOTE:*** *You can specify up to 32 DBD names of secondary indices.*	PSB Module Refers To Hierarchical Database	HiDatabase.Name = <dbd-name1> ... HiDatabase.Name = <dbd-nameN> ...
COPY	member [OF library]	PSB File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
++INCLUDE (Panvalet)	++INCLUDE member	PSB File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

Statement	Format	Relationship	Entities
-INC (Librarian)	-INC member	PSB File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

PSB Copybook File Relationship Projections

The PSB Copybook File object represents an IMS Program Specification Block copybook file. The table below describes the relationships generated from statements in the PSB copybook file.

Statement	Format	Relationship	Entities
COPY	member [OF library]	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
++INCLUDE (Panvalet)	++INCLUDE member	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>
-INC (Librarian)	-INC member	PSB Copybook File Includes PSB Copybook File	For resolved files: PSBCopy.Name = <resolved-name> For unresolved files: PSBCopy.Name = [<library>.] <member>

Relationship Projections from System Definition Statements

When you verify application source files, the parser generates a model of the application that represents the objects it uses and how they interact. This section describes the relationships generated for IMS model objects from statements in System Definition files.

System Definition File Relationship Projections

The System Definition File object represents an IMS System Definition file. The table below describes the relationships generated from statements in the System Definition file.

Statement	Format	Relationship	Entities
APPLCTN	APPLCTN PSB= psb-name ... TRANSACT CODE= (trancode [rtran-code], ...)... ***NOTE:*** *Relationships are created for every TRANSACT macro that follows an APPLCTN macro, and for every transaction code in the TRANSACT macro.*	System Definition File Defines Transaction Transaction Initiates Program Entry Point Program Uses PSB Module	Transaction.Name = <tran-code> ProgramEntry.Name = <psb-name> Program.Root = True Program.Ims Completed = False PsbModule.Name = <psb-name>