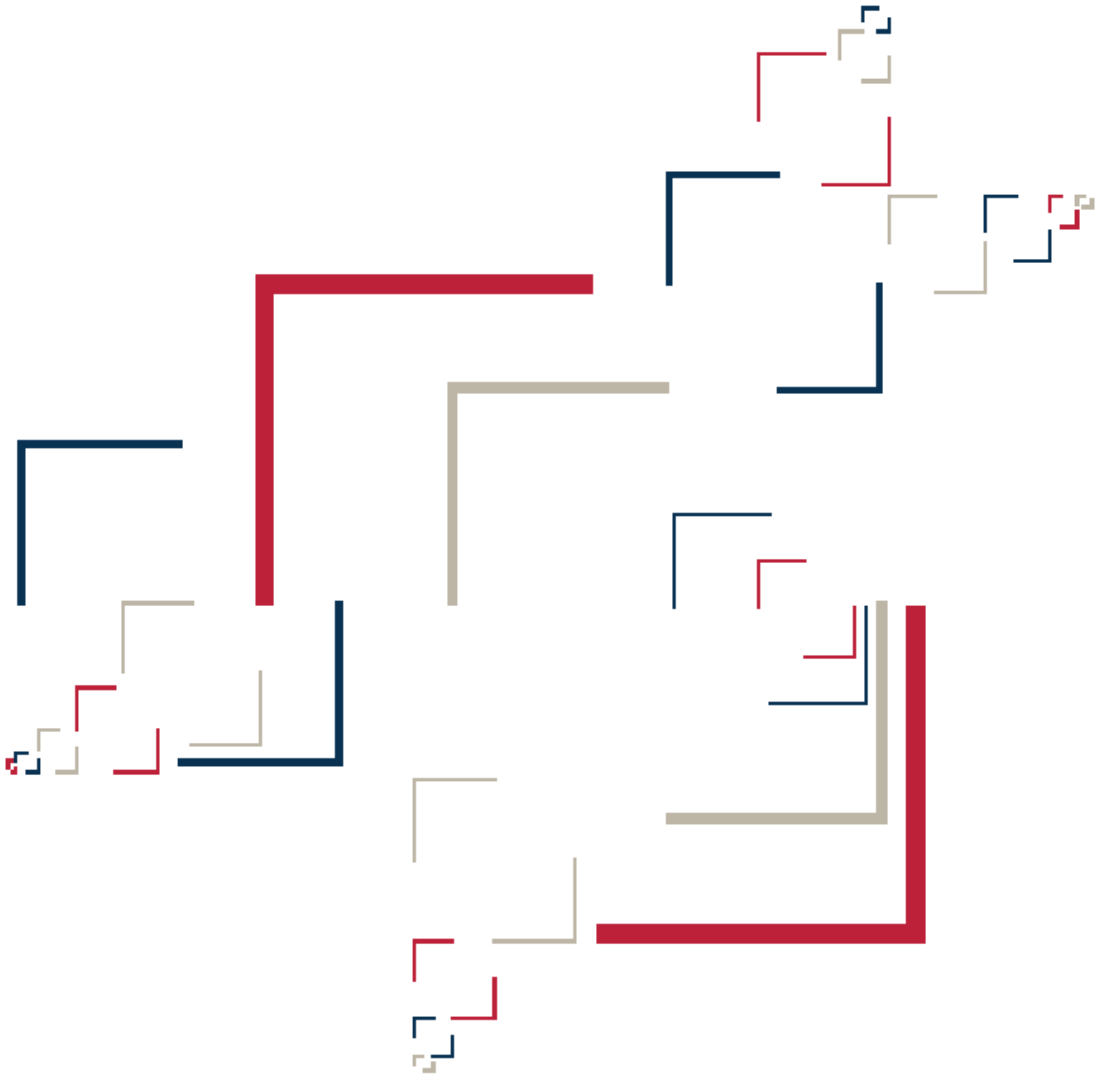

Micro Focus®

Modernization Workbench™

Cobol for OS/390 Support Guide

MICRO FOCUS®

Contents

1 Supported Platforms

This document describes Modernization Workbench (MW) support for Cobol and related platforms:

- COBOL for OS/390, Version 2 Release 2. See *COBOL Language Reference*, Publication No. SC26-9046-04, IBM, 2000.
- 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.
- IDMS, Release 15.0
  - For DML statements, see *CA-IDMS DML Reference – COBOL with FORTRAN and RPG II Supplements*, 15.0, Computer Associates International, 2001.
  - For database definition, see *CA-IDMS Database Administration*, 15.0, Computer Associates International, 2001.
- IMS, Version 7
  - For DL/I calls, see *Application Programming: Database Manager*, Publication No. SC26-9422-01, IBM, 2001.
  - For Exec DLI commands, see *IMS/ESA Version 4 Application Programming: EXEC DLI Commands*, Publication No. SC26-3063-01, IBM, 1994.
  - 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 Cobol Technical Reference

This section describes MW support for Cobol files and copybooks:

- "Support Notes" on page 3 describes MW limitations, caveats, and special usage for Cobol applications.
- "Complexity Metrics" on page 5 describes the supported complexity metrics for objects in the Cobol model.
- "Relationship Projections from Cobol Statements" on page 13 describes the relationships generated from Cobol statements in programs and support files.

Support Notes

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

Object-Oriented Statements

Object-oriented Cobol statements are not supported.

Program IDs

Program IDs must be unique.

Separators Must Be Followed by Blanks

The Cobol parser assumes that every separator must be followed by a blank. If you index a variable with a separator that is not followed by a blank, `MY-VARIABLE(1,1)`, the parser may treat `(1,1)` as a numeric literal, especially when the program was compiled with the DECIMAL POINT IS COMMA option. To index a variable, use the format `MY-VARIABLE(1, 1)`or `MY-VARIABLE(1 1)`.

Copybooks in a Library

If the copybooks used in a Cobol program are in a library, and the library is referenced in a COPY statement with the format `COPY text-name IN library-name` or `COPY text-name OF library-name`, the parser looks first for a copybook named library-name.text-name, and if it does not exist, for a copybook named text-name. If text-name does not exist, the parser reports library-name.text-name as an unresolved reference.

It is your responsibility to prefix library member names with library names or filepaths and dot (.) separators: dir1.dir2.member.cpy represents the copybook dir1/dir2/member, for example. When the parser encounters a reference to a member, it first searches for the longest possible name, dir1.dir2.member.cpy, and if not found, then the shorter versions, dir2.member.cpy and member.cpy.

NOTE: *Unresolved references to library members are always reported with the longest name. This means that if you subsequently register a missing copybook with a short name, the referencing source file will not be invalidated. It's up to you to remember that the referencing source needs to be reverified.*

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 Cobol model.

Cobol File Complexity Metrics

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

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

Copybook File Complexity Metrics

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

Metric	Description
Blank Lines	Number of blank lines of source (sequence number area content is ignored).
Data Elements	Number of declared data items (elementary structures and their fields).

Metric	Description
Include Statements	Number of include statements: COPY, ++INCLUDE, –INC, EXEC SQL INCLUDE.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
Source Lines	Number of lines of source, including blank lines and comments.

Program Complexity Metrics

The table below describes the supported complexity metrics for the Program object. For more information on dead code statistics, see "How MW Calculates Cobol Dead Code Statistics" on page 11.

Note the following:

- 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
```

Metric	Description
Absolute Complexity	Binary Decisions divided by the number of statements.
Asynchronous Calls	Number of asynchronous calls, such as Cobol INITIATE statements.

Metric	Description
Binary Decisions	Number of branching conditions in the flow graph with two possible outcomes. Includes statements with implicit condition evaluation (loops, AT END, and so on): IF, EVALUATE (number of WHEN except WHEN OTHER), PERFORM…TIMES, PERFORM…UNTIL, PERFORM…VARYING, PERFORM…VARYING…AFTER (number of AFTER phrases + 1), statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID (one decision per statement), GOTO…DEPENDING ON (number of alternatives), SEARCH (number of WHEN, AT END). **IDMS**: IF.
Computational Statements	Number of statements performing arithmetic calculations: ADD, SUBTRACT, DIVIDE, MULTIPLY, COMPUTE.
Conditional Complexity	Binary Decisions plus Unique Operands in Conditions.
Conditional Statements	Number of branching statements with nested statements executed under certain conditions, not including conditional GOTOs. IF, EVALUATE, SEARCH, PERFORM…UNTIL, PERFORM…VARYING…UNTIL, statements with ON/NOT ON, AT END/NOT AT END, INVALID/NOT INVALID. **IDMS**: IF.
Cyclomatic Complexity	v(G) = e - n + 2, where v(G) is the cyclomatic complexity of the flow graph (G) for the program in question, e is the number of edges in G, and n is the number of nodes. Quantity of decision logic. The number of linearly independent paths (minimum number of paths to be tested). v(G) = DE + 1, where DE is the number of binary decisions made in the program.
Data Elements	Number of declared data items (elementary structures and their fields).
Dead Data Elements	Number of dead data elements in programs and used include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.

Metric	Description
Dead Data Elements from Includes	Number of dead data elements in include files. Dead data elements are unused structures at any data level, all of whose parents and children are unused.
Dead Lines	Number of dead lines in programs and used include files. Dead lines are source lines containing Dead Data Elements or Dead Statements. Also, source lines containing dead constructs. 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. Dead lines are source lines containing Dead Data Elements from Includes or Dead Statements from Includes. Also, source lines containing dead constructs. 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.
Dead Statements from Includes	Number of dead statements in include files. A dead statement is a procedural statement that can never be reached during program execution.
Difficulty	D = (n1 / 2) * (N2 / n2), where n1 is Unique Operators, N2 is Operands, and n2 is Unique Operands.
Entry Points	Number of program entry points: PROCEDURE DIVISION, ENTRY.
Error Estimate	B = E**(2/3) / 3000, where E is Programming Effort.
Essential Complexity	Quantity of unstructured logic (a loop with an exiting GOTO statement, for example). v(G) for reduced graph without D-structured primes.
Executable Statements	All Procedure Division statements, plus CONTINUE and NEXT STATEMENT.
Extended Cyclomatic Complexity	Cyclomatic Complexity plus Logical Operators in Conditions. Number of all paths in the program.

Metric	Description
Function Points	Lines of Code/K, where K=77. Estimate of the number of end-user business functions implemented by the program.
GoTo Statements	Number of GOTO statements, including conditional GOTOs: GOTO, GOTO...DEPENDING ON.
Inner Call Statements	Number of statements that invoke Inner Procedures: PERFORM procedure-name.
Inner Procedures	Number of structured pieces of code that cannot be invoked from external programs: Cobol paragraphs (including nameless).
Intelligent Content	I = L * V, where L is Program Level and V is Program Volume. Complexity of a given algorithm independent of the language used to express the algorithm.
IO Statements	Number of statements performing input/output operations: OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START, SORT, MERGE, RETURN, RELEASE, ACCEPT, DISPLAY, STOP literal. **SQL**: INSERT, FETCH, SELECT, UPDATE, DELETE, EXECUTE. **CICS**: CONVERSE, SEND, SEND MAP, SEND TEXT, RECEIVE, RECEIVE MAP, RECEIVE TEXT, READQ, WRITEQ, DELETEQ, READ, READNEXT, READPREV, WRITE, REWRITE, DELETE. **IDMS**: ERASE, OBTAIN, GET, MODIFY, STORE.
Lines of Code	Number of lines of code, 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: AND, OR.
Loop Statements	Number of repetitively executing statements: PERFORM…TIMES, PERFORM…UNTIL, PERFORM…VARYING PERFORM…VARYING…AFTER (# of AFTER + 1).

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, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Unique Operators.
Parameters	Number of Cobol Procedure Division USING...RETURNING parameters.
Pointers	Number of data elements declared as pointers. Data items with USAGE IS POINTER, PROCEDURE-POINTER.
Program Length	N = N1 + N2, where N1 is Operators and N2 is Operands.
Program Level	L = 1 / D, where D is Difficulty.
Program Volume	V = N * 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.

Metric	Description
Returning Calls	Number of returning calls, such as CALL or LINK statements.
Unique Operands	Number of distinct operands (n2). Operands are variables and literals used in operators. Uniqueness of literals is determined by their notation. Compare Operands.
Unique Operands in Conditions	Number of distinct operands used in conditions.
Unique Operators	Number of distinct operators (n1). Operators are executable statements and unary and binary operations: +, -, *, /, **, NOT, AND, OR, <, <=, >, >=, =, IS, (subscript), (reference:modification), FUNCTION. Compare Operators.
Vocabulary	n = n1 + n2, where n1 is the number of Unique Operators and n2 is the number of Unique Operands.

How MW Calculates Cobol Dead Code Statistics

This section provides details on how MW 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.

HyperView Usage

In HyperView, 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 HyperView 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.*

Relationship Projections from Cobol 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 Cobol model objects from the statements in programs and support files.

Cobol File Relationship Projections

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

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

Statement	Format	Relationship	Entities
PROGRAM-ID	PROGRAM-ID. name	Cobol File Defines Program	Program.Name = <name> …

Copybook File Relationship Projections

The Copybook File object represents a Cobol copybook. The table below describes the relationships generated from Cobol statements in the copybook.

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

Program Relationship Projections

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

Statement	Format	Relationship	Entities
ACCEPT	ACCEPT varname [FROM mnemonic-name]		Program.OnlineFlag. = True
CALL	CALL 'name'	Program Calls Program Entry Point	ProgramEntry.Name = <name>
CALL (dynamic)	CALL varname	Program Calls Program Entry Decision	Decision attributes: Name = <program-name>@ <internal-name> #Also Known As = <program-name>. Calls.<varname> Decision Type = PROGRAMENTRY…
ENTRY	ENTRY 'name'	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry. MainEntry = True
PROGRAM-ID	PROGRAM-ID. name	Program Has Program Entry Point	ProgramEntry.Name = <name> ProgramEntry. MainEntry = True

Statement	Format	Relationship	Entities
File Description	SELECT file-name ASSIGN TO [label-][org-] <name1> [assignment-name2…] FD file-name.01 file-record-name …	See CRUD statements below.	external-name = <name1> 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.*
DELETE	DELETE file-name…	Program Deletes From File	See File Description for File attributes.
READ	READ file-name…	Program Reads File	See File Description for File attributes.
REWRITE	REWRITE file-record-name …	Program Updates File	See File Description for File attributes.
WRITE	WRITE file-record-name …	Program Inserts Into File	See File Description for File attributes.

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 17 describes MW limitations, caveats, and special usage for SQL.
- "Complexity Metrics" on page 18 describes the supported complexity metrics for objects in the SQL model.
- "Relationship Projections from EXEC SQL Statements" on page 19 describes the relationships generated from EXEC SQL statements in programs.
- "Relationship Projections from SQL DDL Statements" on page 20 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 23 describes MW limitations, caveats, and special usage for JCL applications.
- "Complexity Metrics" on page 25 describes the supported complexity metrics for objects in the JCL model.
- "Relationship Projections from JCL Statements" on page 27 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 31 describes MW limitations, caveats, and special usage for CICS applications.
- "Complexity Metrics" on page 34 describes the supported complexity metrics for objects in the CICS model.
- "Relationship Projections from BMS Statements" on page 37 describes the relationships generated from statements in BMS files and copybooks.
- "Relationship Projections from CSD, FCT, and PCT Statements" on page 38 describes the relationships generated from statements in CSD, FCT, and PCT files.
- "Relationship Projections from CICS Statements" on page 42 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 IDMS Technical Reference

This section describes MW support for IDMS schema and subschema files and IDMS DML statements in programs:

- "IDMS Support Notes" on page 49 describes MW limitations, caveats, and special usage for IDMS applications.
- "IDMS Complexity Metrics" on page 51 describes the supported complexity metrics for objects in the IDMS model.
- "Relationship Projections from IDMS Schema Statements" on page 53 describes the relationships generated from statements in IDMS schema files.
- "Relationship Projections from IDMS DML Statements" on page 54 describes the relationships generated from IDMS DML statements in programs.

IDMS Support Notes

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

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 corre-

sponding 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.

IDMS Complexity Metrics

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

IDMS Schema File Complexity Metrics

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

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

IDMS Subschema File Complexity Metrics

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

Metric	Description
Areas	Number of areas.

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.
Logical Records	Number of logical records.
Path Groups	Number of path groups.
Records	Number of records.
Sets	Number of sets.
Source Lines	Number of lines of source, including blank lines and comments.

Relationship Projections from IDMS Schema 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 IDMS model objects from the statements in IDMS schema files.

IDMS Schema File Relationship Projections

The IDMS Schema File object represents a schema in an IDMS application. The tables below describe the relationships generated from statements in the schema file.

Statement	Format	Relationship	Entities
RECORD	ADD RECORD NAME IS recordName …	Network Database Schema Has Network Database Record	NETRECORD.Name = <schemaName>.<recordName> NETRECORD.RecordName = <recordName>
SCHEMA	ADD SCHEMA NAME IS schemaName …	IDMS Schema File Defines Network Database Schema	NETSCHEMA.Name = <schemaName>

Relationship Projections from IDMS DML 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 IDMS model objects from the IDMS DML statements in programs.

Cobol File Relationship Projections

The Cobol File object represents the source file for a Cobol program. The table below describes the relationships generated from IDMS DML statements in the source file.

Statement	Format	Relationship	Entities
COPY IDMS	[level-number] COPY IDMS name	Cobol File Includes Copybook File	Where member-name = <name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (file|module)	COPY IDMS [FILE | MODULE] name	Cobol File Includes Copybook File	Where member-name = <name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>

Statement	Format	Relationship	Entities
COPY IDMS (record)	[level-number] COPY IDMS RECORD rec-name	Cobol File Includes Copybook File	Where member-name = <rec-name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (subschema)	[level-number] COPY IDMS SUBSCHEMA-name	Cobol File Includes Copybook File	Where member-name = <schema-name> $<subschema-name> $<name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (subschema-ctrl)	[level-number] COPY IDMS SUBSCHEMA-CTRL [level-number] COPY IDMS SUBSCHEMA-LR-CTRL	Cobol File Includes Copybook File	Where member-name = SUBSCHEMA-CTRL or member-name = SUBSCHEMA-LR-CTRL: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
IDMS-CONTROL SECTION	IDMS-CONTROL SECTION… [IDMS -RECORDS WITHIN [WORKING-STORAGE | LINKAGE]]	Cobol File Includes Copybook File	Where member-name = SUBSCHEMA-CTRL or member-name = <schema-name> $<subschema-name> $SUBSCHEMA-RECORDS: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>

Copybook File Relationship Projections

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

Statement	Format	Relationship	Entities
COPY IDMS	[level-number] COPY IDMS name	Copybook File Includes Copybook File	Where member-name = <name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (file|module)	COPY IDMS [FILE | MODULE] name	Copybook File Includes Copybook File	Where member-name = <name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (record)	[level-number] COPY IDMS RECORD rec-name	Copybook File Includes Copybook File	Where member-name = <rec-name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>
COPY IDMS (subschema)	[level-number] COPY IDMS SUBSCHEMA-name	Copybook File Includes Copybook File	Where member-name = <schema-name> $<subschema-name> $<name>: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>

Statement	Format	Relationship	Entities
COPY IDMS (subschema-ctrl)	[level-number] COPY IDMS SUBSCHEMA-CTRL [level-number] COPY IDMS SUBSCHEMA-LR-CTRL	Copybook File Includes Copybook File	Where member-name = SUBSCHEMA-CTRL or member-name = SUBSCHEMA-LR-CTRL: For resolved files: Copybook.Name = <member-name> [.ext]... For unresolved files: Copybook.Name = <member-name>

Program Relationship Projections

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

Statement	Format	Relationship	Entities
ERASE	ERASE record-name.	Program Deletes Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
GET	GET record-name.	ProgramReads Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
MODIFY	MODIFY record-name.	ProgramUpdates Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>

Statement	Format	Relationship	Entities
OBTAIN	OBTAIN … [CALC | DUPLICATE] record-name. OBTAIN … CURRENT record-name. OBTAIN … record-name DB-KEY IS db-key. OBTAIN … record-name WITHIN [set-name | area-name].	ProgramReads Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>
SCHEMA SECTION	SCHEMA SECTION. DB subschema-name WITHIN schema-name.	ProgramUses Network Database Schema	NetSchema.Name = <schema-name>
STORE	STORE record-name.	ProgramInserts Network Database Record	NetRecord.Name = <schema-name>. <record-name> NetRecord.Record Name = <record-name>

7 IMS Technical Reference

This section describes MW 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 59 describes MW limitations, caveats, and special usage for IMS applications.
- "IMS Complexity Metrics" on page 63 describes the supported complexity metrics for objects in the IMS model.
- "Relationship Projections from DBD and PSB Statements" on page 67 describes the relationships generated from statements in DBD and PSB files.
- "Relationship Projections from System Definition Statements" on page 71 describes the relationships generated from statements in System Definition files.

IMS Support Notes

These notes describe MW 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.

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).

Online CICS Applications Using IMS

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

EXEC DLI Support

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

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.

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.

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 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).

Metric	Description
Include Statements	Number of include statements.
Lines with Comments	Number of lines of source containing comments, including inline comments placed on lines with statements.
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.
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.

Metric	Description
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.

Metric	Description
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.

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.

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>