COBOL to IDL Mapping

This chapter describes how COBOL data items and related syntax are mapped to Software AG IDL by the IDL Extractor for COBOL using the *Extractor Wizard* and *Mapping Editor*.

This chapter covers the following topics:

- COBOL Data Type to Software AG IDL Mapping
- DATA DIVISION Mapping
- PROCEDURE DIVISION Mapping
- Copybooks

COBOL Data Type to Software AG IDL Mapping

The IDL Extractor for COBOL maps the following subset of COBOL data types to Software AG IDL data types.

The following metasymbols and informal terms are used for the IDL in the table below.

- The metasymbols "[" and "]" surround optional lexical entities.
- The informal terms *n* and *m* are sequences of numeric characters, for example 123.

COBOL Data Type			Software AG IDL Data Type		Notes
Alphabetic		PIC A(*n*)	A*n*, AV*n*	Alphanumeric	1,2
DBCS		PIC G(*n*)	K*n**2, KV*n***2*	Kanji	1,2,3
DBCS		PIC N(*n*) [USAGE] [IS] DISPLAY-1	K*n**2, KV*n**2	Kanji	1,2,3
Unicode or DBCS		PIC N(*n*)	U*n*, UV*n* or K*n**2, KV*n**2	Unicode or Kanji	1,2,3,10
Unicode		PIC N(*n*) [USAGE] [IS] NATIONAL	U*n*, UV*n*	Unicode	1,2
Alphanumeric		PIC X(*n*)	A*n*, AV*n*	Alphanumeric	1,2
Numeric	Zoned decimal	PIC 9(*n*)[V9(*m*)]	NU*n*[,*m*]	Unpacked decimal unsigned	2,4,8
	Zoned decimal	PIC S9(*n*)[V9(*m*)]	N*n*[,*m*]	Unpacked decimal	2,4,8
	Packed decimal	PIC 9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	PU*n*[,*m*]	Packed decimal unsigned	2,4,8
	Packed decimal	PIC S9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	P*n*[,*m*]	Packed decimal	2,4,8
	Packed decimal	PIC 9(*n*) [V9(*m*)] PACKED-DECIMAL	PU*n*[,*m*]	Packed decimal unsigned	2,4,8
	Packed decimal	PIC S9(*n*) [V9(*m*)] PACKED-DECIMAL	P*n*[,*m*]	Packed decimal	2,4,8
	Binary	PIC [S]9(*n*) BINARY (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6,7
	Binary	PIC [S]9(*n*) BINARY (5<=*n*<=9)	I4	Integer (large)	2,4,5,6,7
	Binary	PIC [S]9(*n*) COMP[UTATIONAL][-4] (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6,7
	Binary	PIC [S]9(*n*) COMP[UTATIONAL][-4] (5≤*n*<=9)	I4	Integer (large)	2,4,5,6,7
	Binary	PIC [S]9(*n*) COMP-5 (1<=*n*<=4)	I2	Integer (medium)	2,4,6,7
	Binary	PIC [S]9(*n*) COMP-5 (5<=*n*<=9)	I4	Integer (medium)	2,4,6,7
	Floating point	COMP[UTATIONAL]-1	F4	Floating point (small)	9
	Floating point	COMP[UTATIONAL]-2	F8	Floating point (large)	9
Alphanumeric-edited		Alphanumeric item containing "0" or "/"	A(*length of PIC*)	Alphanumeric	11
Numeric-edited		Numeric item containing "DB", "CR", "Z", "$", ".", ",", "+", "-", "*", "B", "0" or "/"	A(*length of PIC*)	Alphanumeric	11

Notes:

1. Mapping to fixed-length or variable-length Software AG IDL data type is controlled in the extraction settings of the extraction wizard, see *Step 4: Define the Extraction Settings and Start Extraction*.
2. Equivalent alternative forms of the PICTURE clause, e.g. XXX, AAA, NNN, GGG or 999 may also be used.
3. The length for IDL data type is given in bytes. For COBOL the length is in DBCS characters (2 bytes).

4. The character "P[(*n*)]" stands for a decimal scaling position, this character has no effect on the length of the generated data type. Only the data fraction will be mapped to the Software AG IDL:

   ```
   01 GROUP1.
    10 FIELD1 PIC PPP9999.
   ```

   will be mapped to IDL:

   ```
   1 GROUP1
    2 FIELD1 NU4
   ```

5. Behavior depends on the COBOL compiler settings:

   - With COBOL 85 standard, the value range depends on the number of digits in the PICTURE clause. This differs from the value range of the IDL data type using the binary field size instead. If the parameter is of direction "In" your RPC client application has to ensure the integer value sent is within the allowed range. See *Software AG IDL Grammar*.
   - With *no* COBOL 85 standard, the value range of the COBOL data type reflects the binary field size, thus matches the IDL data type exactly. In this case, there are no restrictions regarding value ranges. For example:
     - with operating system z/OS and IBM compiler, see option TRUNC(BIN) in your COBOL compiler documentation
     - with operating systems UNIX and Windows and a Micro Focus compiler, see option NOTRUNC in your Micro Focus COBOL documentation.

6. For unsigned COBOL data types (without "S" in the PICTURE clause) the value range of the IDL data type differs:

   - IDL allows negative values, COBOL does not.
   - For I2, the maximum is 32767 for IDL instead of 65535 for COBOL.
   - For I4, the maximum is 2147483647 for IDL instead of 4294967294 for COBOL.

7. Binaries with more than 9 digits in the PICTURE clause cannot be mapped to IDL. See the following table:

S9(10) thru S9(18)	Binary doubleword (8 bytes)	-9,223,372,036,854,775 thru +9.223,372,036,854,775
9(10) thru 9(18)	Binary doubleword (8 bytes)	0 thru 18,446,744,073,709,551

8. The value range of PACKED-DECIMAL and ZONED-DECIMAL is greater than the value range of the mapped IDL data type. COBOL supports 31 digits (IBM and Fujitsu Siemens), 38 digits (Micro Focus), and IDL 29 digits. If the COBOL program uses more than 29 digits for a PACKED-DECIMAL or ZONED-DECIMAL, it cannot be mapped to IDL.

   The precision (digits after decimal point) of PACKED-DECIMAL and ZONED-DECIMAL is greater than the value range of the mapped IDL data type, which is 7. If the COBOL program uses more than 7 digits after the decimal point for a PACKED-DECIMAL or ZONED-DECIMAL, it cannot be mapped

to IDL.

Only the IDL range 0=*n*=7 and 1=(*m*+*n*)=29 is supported.

9. `COMPUTATIONAL-1` (4-byte, single precision) and `COMPUTATIONAL-2` items (8-byte, double precision) items are an IBM-specific extension. When floating-point data types are used, rounding errors can occur, so the values of senders and receivers might differ slightly.

10. If this form is extracted from a COBOL program originally written for Micro Focus COBOL and operating system UNIX or Windows, the mapping to the IDL data type depends on the setting in the IDL Extractor for COBOL Preferences. See **Meaning of PIC N without USAGE clause** within pane **Compiler Directives** of *Step 2: Define the Default Settings*. For all other COBOL program extractions, the mapping is always to IDL data type `Un/Uvn`.

11. COBOL alphanumeric/numeric edited items will force the generation of IDL data type `A` with an inline comment containing the original `COBOL PICTURE` clause. The `CURRENCY SIGN` clause in the `SPECIAL-NAMES` and the `CURRENCY` compiler option is not considered.

DATA DIVISION Mapping

This section discusses the syntax relevant for extracting the `DATA DIVISION`:

- BLANK WHEN ZERO Clause
- Condition Names - Level-88 Data Items
- Continuation Lines
- DATE FORMAT Clause
- FILLER Pseudo-Parameter
- GLOBAL and EXTERNAL Clause
- JUSTIFIED Clause
- OBJECT REFERENCE Phrase
- Parameter Names
- POINTER Phrase
- PROCEDURE-POINTER Phrase
- REDEFINE Clause
- RENAMES Clause - LEVEL 66 Data Items
- SIGN LEADING and TRAILING SEPARATE Clause
- SYNCHRONIZED Clause
- Tables with Fixed Size
- Tables with Variable Size - DEPENDING ON Clause
- Unstructured Data Types - LEVEL 77 Data Items
- USAGE Clause on Group Level
- USAGE IS INDEX Clause
- VALUE Clause

BLANK WHEN ZERO Clause

The `BLANK WHEN ZERO` clause specifies that an item contains nothing but spaces when its value is zero. The `BLANK WHEN ZERO` clause is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `BLANK WHEN ZERO` clause. Because the `BLANK WHEN ZERO` clause only has an impact if the item is displayed, such a program can be mapped to IDL. The workaround for RPC clients is to imitate the `BLANK WHEN ZERO` clause.

Condition Names - Level-88 Data Items

See the following COBOL syntax:

```
88 condition_name VALUE [IS] 'literal_1'
88 condition_name VALUE [IS] 'literal_1' [THRU | THROUGH] 'literal_2'
88 condition_name VALUES [ARE] 'literal_1' [THRU | THROUGH] 'literal_2'
```

Semantically, level-88 condition names can be

- **Enumeration Type Values**
  If your COBOL server requires the level-88 value to be provided on a call-by-call basis, that is, the value may change with every call, map the level-88 base variable to a simple IDL parameter with the desired direction In, Out or InOut. RPC clients have to pass correct values, same as defined by the level-88 condition names.

- **Single Constant Values**
  If your COBOL server interface expects for your purpose always a constant value, map the level-88 condition names to a constant.

- **Function or Operation Codes**
  If the level-88 values are function or operation codes, map the level-88 condition names to an operation.

Continuation Lines

Continuation lines, starting with a hyphen in the indicator area, are supported.

DATE FORMAT Clause

The `DATE FORMAT` clause is an IBM-specific extension. The `DATE FORMAT` clause specifies that a data item is a windowed or expanded date field.

The `DATE FORMAT` clause is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `BLANK WHEN ZERO` clause. The semantic given by the `DATE FORMAT` clause has to be considered by RPC clients.

FILLER Pseudo-Parameter

In the check box **Map FILLER fields to IDL** of the COBOL to IDL in the extraction settings of the wizard (see *Step 4: Define the Extraction Settings and Start Extraction*) you can define whether COBOL `FILLER` pseudo-parameters should be part of the RPC client interface by default or not. By default they are not mapped to IDL. In the *COBOL Mapping Editor* you can change the mapping for a `FILLER` field individually, e.g. mapping required ones to IDL. If `FILLER` fields are mapped to IDL, they are made unique by appending a sequence number. You can set the prefix to be used in the *IDL Extractor for COBOL Preferences*.

If the resulting names are not suitable, you can rename IDL field names in the Mapping Editor with the **Rename** function of the context menu. See the following example:

```
01 GROUP1.
 10  FIELD1 PIC XX.
 10  FILLER PIC XX.
 10  FIELD2 PIC S99.
 10  FILLER PIC XX.
```

This will be mapped to Software AG IDL:

```
1 GROUP1
 2 FIELD1   (A2)
 2 FILLER_1 (A2)
 2 FIELD2   (N2.0)
 2 FILLER_2 (A2)
```

If a group is named `FILLER` and the group has scalar fields, the group is always mapped to IDL, independent of the check box **Map FILLER fields to IDL**. For example:

```
01 GROUP1.
 10 FIELD1 PIC XX.
 10        PIC XX.
 10 FIELD2 PIC S99.
 10 FILLER PIC XX.
 10 .
  20 FIELD3 PIC S9(4) BINARY.
  20 FIELD4 PIC S9(4) BINARY.
```

This will be mapped to Software AG IDL:

```
1 GROUP1
 2 FIELD1   (A2)
 2 FILLER_1 (A2)
 2 FIELD2   (N2.0)
 2 FILLER_2 (A2)
 2 FILLER_3
   3 FIELD3 (I2)
   3 FIELD4 (I2)
```

GLOBAL and EXTERNAL Clause

The `GLOBAL` clause

- specifies that a data-name is available to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name.
- is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `GLOBAL` clause.

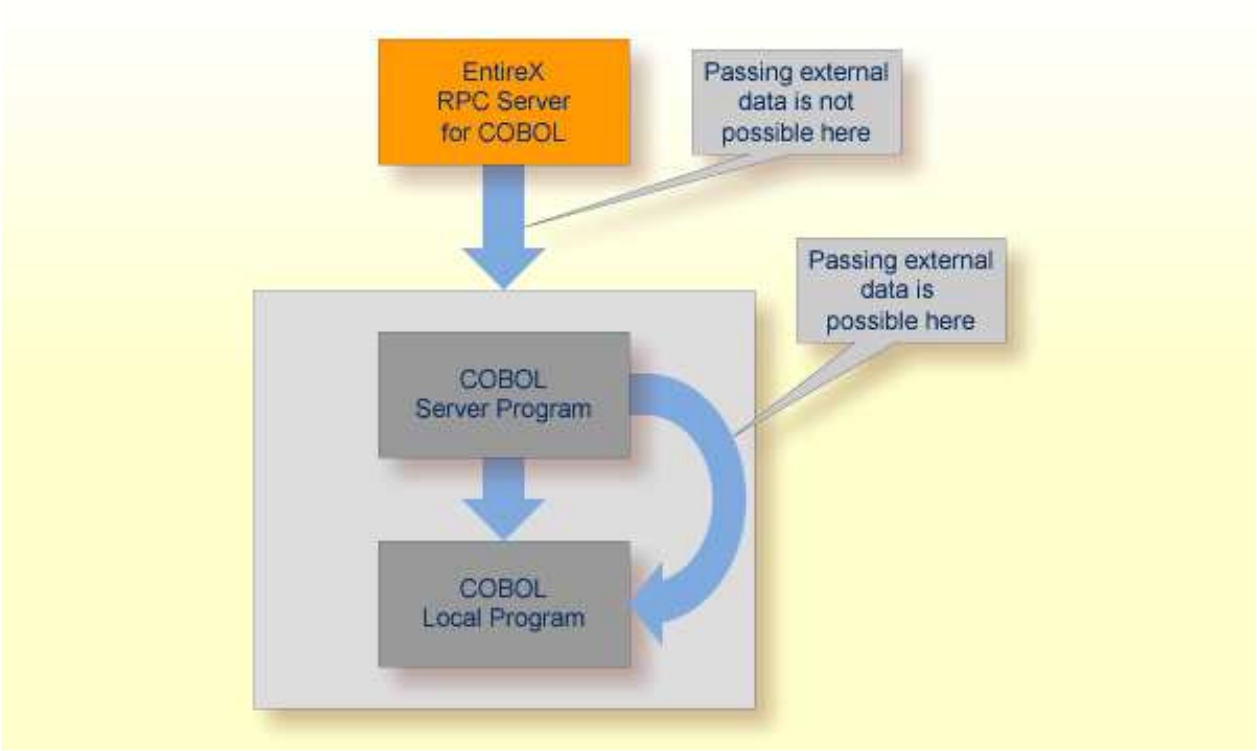
However, program parameters containing the `GLOBAL` clause can be mapped to IDL, which can make sense as long as the `EXTERNAL DATA` clause is used to pass parameters from the called COBOL server to further subprograms called.

The `EXTERNAL` clause

- can only be specified on data description entries that are in the Working-Storage section of a program.

- is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `EXTERNAL` clause.

  Note:
  EntireX RPC technology cannot pass data using `EXTERNAL` linkage from the RPC server to the COBOL server. However, program parameters containing the `EXTERNAL` clause can be mapped to IDL, which can make sense as long as the `EXTERNAL DATA` clause is used to pass parameters from the called COBOL server to further subprograms called.



JUSTIFIED Clause

The IDL Extractor for COBOL ignores the `JUSTIFIED` clause. The `DATA DIVISION` is parsed as without the `JUSTIFIED` clause. The workaround for RPC clients is to imitate the `JUSTIFIED` clause.

OBJECT REFERENCE Phrase

The `OBJECT REFERENCE` phrase is an IBM-specific extension. A program containing an `OBJECT REFERENCE` phrase cannot be mapped to IDL.

Parameter Names

Numbers in the first position of the parameter name are not allowed in Software AG IDL syntax (see *Software AG IDL Grammar*). Thus COBOL names starting with a number are prefixed with the character "#" by default. You can rename all IDL parameters in the *COBOL Mapping Editor*. For example,

```
01 1BSP  PIC XXX.
```

by default will be mapped to Software AG IDL:

```
01 #1BSP A(3).
```

If a parameter name is not specified, e.g.

```
01 GROUP1.
 10 FIELD1 PIC XX.
 10        PIC XX.
 10 FIELD2 PIC S99.
 10 FILLER PIC XX.
 10 .
  20 FIELD3 PIC S9(4) BINARY.
  20 FIELD4 PIC S9(4) BINARY.
```

see *FILLER Pseudo-Parameter* above.

POINTER Phrase

The POINTER phrase is an IBM-specific extension.

COBOL Syntax	Software AG IDL Syntax
1 *name* USAGE IS POINTER	none
1 *name* POINTER	none

All pointers are mapped to "suppressed" in the Mapping Editor because the Software AG IDL does not support pointers. Offsets to following parameters are maintained by the *Usage of Server Mapping Files*. At runtime, the RPC server passes a null pointer to the COBOL server.

PROCEDURE-POINTER Phrase

The PROCEDURE-POINTER phrase is an IBM-specific extension. A program containing a procedure-reference phrase cannot be mapped to IDL.

REDEFINE Clause

A redefinition is a second parameter layout of the same memory portion. In most modern programming languages, and also the Software AG IDL, this is not supported. With the wizard you can select a single redefine path for IDL usage. You can do this in the

- *COBOL Mapping Editor*
  - Select the single REDEFINE path for a level 1 REDEFINE unit (all redefine paths addressing the same storage location) in the parameter selection window. See *Step 5: Select the COBOL Interface and Map to IDL Interface* in *Using the IDL Extractor for COBOL*. This is the simplest and most straightforward approach for a COBOL server with a single interface, because All REDEFINE siblings are no longer considered. Further processing is like a single parameter for the level 1 REDEFINE path.
  - Select the complete REDEFINE unit on level 1 with all paths for a COBOL server with multiple interfaces that have to be mapped to operation, and where each operation interface is described by a level 1 REDEFINE path in the parameter selection window. See *Step 5: Select the COBOL*

Interface and Map to IDL Interface in *Using the IDL Extractor for COBOL*. Then model the operation interfaces in the Mapping Editor to IDL programs.

- Only `REDEFINE` units on level 1 can be selected in the parameter selection. `REDEFINE` units on level greater than 1 have to be selected in the Mapping Editor, see below.

- *COBOL Mapping Editor*
  - For all `REDEFINE` units on level greater than 1, the `REDEFINE` path used by your interface has to be selected in the Mapping Editor.
  - For `REDEFINE` units on level 1 that are not selected uniquely in the parameter selection window above, map the required `REDEFINE` path to IDL.

If a `REDEFINE` path is selected, the mapping is as follows:

COBOL Syntax	Software AG IDL Syntax
`1 [ name_1 ] REDEFINES name_2`	`1 name_1`
`1 FILLER REDEFINES name_2`	`1 FILLER_n`

RENAMES Clause - LEVEL 66 Data Items

Level-66 entries are ignored and cannot be used for mapping to IDL. The `DATA DIVISION` is parsed as without the level-66 entry.

SIGN LEADING and TRAILING SEPARATE Clause

The `SIGN LEADING` and `TRAILING SEPARATE` clause is supported. The mapping is internal within the *Usage of Server Mapping Files*.

SYNCHRONIZED Clause

The synchronized clause aligns COBOL data items at word boundaries. The clause does not have any relevance for RPC clients and is not written into the IDL file but into the server mapping file. At runtime, the RPC server aligns the data items accordingly.

Tables with Fixed Size

Fixed-size COBOL tables are converted to fixed-size IDL arrays. See the following syntax.

COBOL Syntax	Software AG IDL Syntax
`1 name OCCURS n [TIMES]`	`1 name (/n)`
`1 name OCCURS n [TIMES] [ ASCENDING | DESCENDING [KEY] [IS] key_name ]`	`1 name (/n)`
`1 name OCCURS n [TIMES] [ [ INDEXED [BY] index_name]`	`1 name (/n)`

Rules

- The combination of the `ASCENDING` and `INDEXED BY` phrase as well as `DESCENDING` and `INDEXED BY` phrase is also supported.

Tables with Variable Size - DEPENDING ON Clause

Variable size COBOL tables are converted to unbounded groups with a maximum upper bound set. The lower-bound is always set to 1. The index is not part of the IDL, but it is in the server mapping file. See the following example:

```
01 COUNTER-1 PIC 99.
01 TABLE OCCURS FROM 1 TO 10 DEPENDING ON COUNTER-1
 02 FIELD1 PIC XX.
 02 FIELD2 PIC 99.
```

A variable length group (with maximum) will be defined. A presence of the index in the IDL would be wrong, because the number of elements is implicitly available with the unbounded group. Therefore the index is not part of the IDL, but the mapping is within the *Usage of Server Mapping Files*.

```
01 TABLES (/V10)
 02 FIELD1 (A2)
 02 FIELD2 (NU2.0)
```

COBOL Syntax	Software AG IDL Syntax
`1 name OCCURS n TO m [TIMES] DEPENDING [ON] index`	`1 name (/m)`
`1 name OCCURS n TO m [TIMES] DEPENDING [ON] index [ ASCENDING | DESCENDING [KEY] [IS] key_name ]`	`1 name (/m)`
`1 name OCCURS n TO m [TIMES] DEPENDING [ON] index [ INDEXED [BY] index_name]`	`1 name (/m)`

Rules

- The data item referenced by the `OCCURS DEPENDING ON` clause has to be part of the COBOL server interface as well - in the same COBOL data item direction. This means that if the variable-size table is selected as a
  - COBOL InOut Parameter (see *Step 5: Select the COBOL Interface and Map to IDL Interface*), the *index* data item (ODO subject) must be selected as a COBOL InOut parameter as well.
  - COBOL In Parameter, the *index* data item (ODO subject) must be selected as a COBOL In parameter as well.
  - COBOL Out Parameter, the *index* data item (ODO subject) must be selected as a COBOL Out parameter as well.
- If the *index* data item (ODO subject) is not selected correctly with the variable-size table, unexpected behavior occurs.

- The COBOL from value, *n* above, is semantically different from the IDL lower bound and means a lower-bound of elements which must not be crossed. It is the duty of the calling RPC client to take care of this and set the corresponding number of elements correctly. Do not send less than the COBOL lower bound.
- The combination of the `ASCENDING` and `INDEXED BY` phrase as well as `DESCENDING` and `INDEXED BY` phrase is also supported.

Unstructured Data Types - LEVEL 77 Data Items

COBOL level-77 data items are handled as COBOL data items on level 1. They are always mapped to IDL level 1.

USAGE Clause on Group Level

A `USAGE` clause can be specified on group level, which defines the data type of subsequent groups or parameters. The `USAGE` clause on subsequent groups or parameters may not contradict a higher level definition. Therefore IDL data types may depend on `USAGE` clauses of parent groups if the COBOL data structure is defined as explained.

USAGE IS INDEX Clause

COBOL data items defined with `USAGE IS INDEX` are parsed as without `USAGE IS INDEX`. The `USAGE IS INDEX` clause is ignored.

VALUE Clause

The `VALUE` clause specifies the initial contents of a data item or the value(s) associated with a condition name. For condition names, see *Condition Names - Level-88 Data Items* above.

COBOL Syntax
`1 name <COBOL data type> VALUE [IS] 'literal'`

Initial values can be specified on data items in the Working-Storage Section. As an IBM extension, in the File and Linkage Sections, the VALUE clause is treated as a comment.

The IDL Extractor for COBOL ignores initial values of data items. The `DATA DIVISION` is parsed as without the `VALUE` clause.

PROCEDURE DIVISION Mapping

This section discusses the syntax relevant for extraction of the `PROCEDURE DIVISION`:

- PROCEDURE DIVISION Header
- BY VALUE Phrase
- RETURNING Phrase

PROCEDURE DIVISION Header

For batch and IMS programs, the `PROCEDURE DIVISION` header is relevant for the COBOL InOut parameters. The parameters of the header are suggested as default COBOL InOut parameters.

For CICS, the `PROCEDURE DIVISION` header is of no interest, because the `DFHCOMMAREA` is the relevant information to get the COBOL InOut parameters from. If the `DFHCOMMAREA` is defined in the linkage section all parameters of the `DFHCOMMAREA` are suggested as default COBOL InOut parameters. If there is no `DFHCOMMAREA` there is no suggestion.

However, you can always add, change and correct the suggested parameters if they are not the correct ones in the extraction wizard. See also *Step 5: Select the COBOL Interface and Map to IDL Interface* in *Using the IDL Extractor for COBOL*.

BY VALUE Phrase

The `BY VALUE` clause is an IBM-specific extension for COBOL batch programs. It is ignored by the IDL Extractor for COBOL. Directions are added in the Mapping Editor manually.

RETURNING Phrase

The `RETURNING` phrase is an IBM-specific extension for COBOL batch programs. It is ignored by the IDL Extractor for COBOL. Handling is as without the phrase. No return value is transferred during execution time. If the `RETURNING` phrase is relevant for the interface, the COBOL program cannot be mapped to IDL.

Copybooks

Copybook Support

COPY statements are supported if nested copy statements do not recursively call the same source file.

If copybooks cannot be located, the following rules apply:

- In the case of a remote extraction, the copybook location (data set) is unknown.
- In the case of a local extraction, either the copybook location (directory) or the copybook extension is unknown.
- In both cases, the extraction wizard will appear with a dialog to browse for the copybook location (local directory or remote data set) and allows you to append your copybook extensions. Both will be saved in the preferences.

You can also predefine the following in the preferences:

- the copybook locations, see *Step 4: Define the Remote Copybook Locations* or *Step 4: Define the Local Copybook Locations* in *IDL Extractor for COBOL Preferences*.
- the copybook extensions for local extractions, see *Step 4: Define the Local Copybook Locations* in *IDL Extractor for COBOL Preferences*.

Copybooks with REPLACE Option

COPY statements with the REPLACE option are supported. Beneath the REPLACE option, those copybooks are worked off like all other copybooks above. Example:

- a copybook ACPYBK with REPLACE option

```
01 WS-ZEUGNIS.
           :F: WS-AKTIONEN          PIC  9(01).
             :L: :C:-NEU                     VALUE 'N'.
             :L: :C:-MOD                     VALUE 'M'.
             :L: :C:-INS                     VALUE 'I'.
             :L: :C:-WEG                     VALUE 'W'.
             :L: :C:-SIG                     VALUE 'S'.
           :F: WS-NOTEN             PIC  X(03).
             :L: SEHR-GUT                    VALUE 100.
             :L: GUT                         VALUE  95 THROUGH 99.
             :L: BEFRIEDIGEND                VALUE  80 THROUGH 94.
             :L: AUSREICHEND                 VALUE  50 THROUGH 79.
             :L: MANGELHAFT                  VALUE  01 THROUGH 49.
             :L: UNGENUEGEND                 VALUE   0.
```

- referencing the copybook above

```
COPY ACPYBK
     REPLACING
       ==:F:==   BY ==10==,
       ==:L:==   BY ==88==,
       ==:C:==   BY ==CMD==,
       95        BY 90,
       94        BY 89,
       WS-NOTEN  BY WS-PROZENT,
       ==X(03)== BY ==9(03)==,
       ==9(01)== BY ==X(01)==.
```