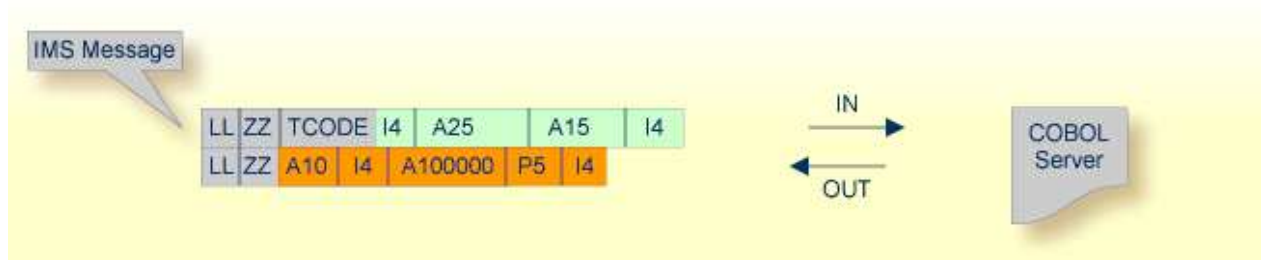


IMS MPP Message Interface (IMS Connect)

This chapter describes using the COBOL Mapping Editor to extract from a COBOL IMS MPP message interface (IMS Connect) server where the interface definition is within the working storage section and not given by the parameters provided in the PROCEDURE DIVISION header.



- Introduction
- Extracting from an IMS MPP Message Interface Program
- Mapping Editor User Interface
- Mapping Editor IDL Interface Mapping Functions
- Programming Techniques

Introduction

Depending on the programming style used in the IMS processing program (MPP) and the various techniques for accessing the IMS input and output messages, finding the relevant COBOL data structures can be a complex and time-consuming task that may require IMS programming knowledge.

IMS Message Processing Programs (MPPs) work as follows:

- IMS message processing programs (MPP) are invoked using an IMS transaction code. Transaction codes are linked to programs by the IMS system definition.
- An IMS message processing program (MPP) gets its parameters through an IMS message and returns the result by sending an output message to IMS. The structure of both messages is defined in the COBOL source program during the application design phase. Sender and receiver of the message must use the same data structure to interpret the message content.
- The server program accesses input and output messages using the IMS system call `CALL 'CBLTDLI' USING <function> IOPCB <message>`. The parameters are as follows:

Parameter	Description
GU	Flag indicating that an input message is to be read. In this case <message> describes the input message.
ISRT	Flag indicating that an output message is to be written. In this case <message> describes the output message.
IOPCB	The IO PCB pointer. An IMS-specific section defined in the linkage section of the program to access the IMS input and output message queue.
<message>	The layout of the message. For GU it is the structure of the input message, for ISRT it is the structure of the output message. The first two fields in every message (input as well as output), LL and ZZ, are technical fields, each two bytes long. LL contains the length of the message. The third field in an input message contains the transaction code and has a variable length (commonly 8 or 9 bytes). IMS can link one program to various different transaction codes. For each transaction, the program can apply a separate logic, or even accept a separate message layout.

Notes:

1. Instead of the IOPCB pointer, CALL 'CBLTDLI' statements are also used with database PCB pointers to access IMS databases.
2. IOPCB, GU and ISRT are defined in the COBOL source (often in a copybook) using COBOL data items. Names can differ in your program. The value of the COBOL VALUE clauses with 'GU' and 'ISRT' is fixed. In the example below, the IMS system call would be CALL 'CBLTDLI' USING FCT-GU IO-PCB <message> to read the input message:

```

WORKING-STORAGE SECTION.
. . .
* DLI Function Codes
77 FCT-GU                      PIC X(4) VALUE 'GU' .
77 FCT-ISRT                    PIC X(4) VALUE 'ISRT' .
. . .
LINKAGE SECTION.
. . .
1 IO-PCB.
3 LTERM-NAME                    PIC X(8) .
3 FILLER                        PIC X(2) .
3 IO-STATUS                     PIC X(2) .
. . .

```

If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

Extracting from an IMS MPP Message Interface Program

If you are extracting IDL from a COBOL source or extending the IDL file by extracting an additional COBOL source with interface type IMS MPP message interface (IMS Connect), the **Extractor Settings** dialog appears (see also *Step 4: Define the Extraction Settings and Start Extraction*).

Make sure the interface type is correct and specify how you want the transaction name to be determined.

COBOL Source

File Name:

Operating System:

Interface Type:

☐ Input Message same as Output Message

There are two ways of defining **Transaction Name**:

- **Fixed Value**

Check **Transaction Name** and specify a fixed value for the transaction name in extractor settings. Your IDL interface is free of this technical parameter, and RPC clients do not have to specify it at runtime.

IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: *

☒ Transaction Name: *

☐ Create IDL parameter for Transaction Name - specification at runtime

Specify the length of the transaction field, which is usually the third physical field starting from offset 5 (bytes) declared in the input message layout within the server program. Example:

```

1 INPUT-MESSAGE.
2 INPUT-IMS-META.
3 INPUT-LL                PIC S9(3) BINARY.
3 INPUT-ZZ                PIC S9(3) BINARY.
3 INPUT-TRANSACTION       PIC X(10).
2 INPUT-DATA.
3 OPERATION                PIC X(1).
3 OPERAND1                PIC S9(9) BINARY.
3 OPERAND2                PIC S9(9) BINARY.
```

In this example, the length to specify is "10".

- **Dynamically at Runtime**

Check **Create IDL parameter for Transaction Name....** Your *IDL Interface* will contain an IDL parameter for the transaction name. RPC clients are responsible for setting the correct transaction name dynamically at runtime.


IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: *

☐ Transaction Name: *


☒ Create IDL parameter for Transaction Name - specification at runtime

➤ To select the COBOL interface data items of your COBOL server

1. Define the IMS MPP (IMS Connect) input message. With toolbar icon **Find text in Source** , enter "CBLTDLI" to look for an IMS system call containing 'CBLTDLI', function GU and the IOPCB pointer, example:

```
CALL 'CBLTDLI' USING GU IOPCB input_message
```

Add the relevant COBOL data items of *input_message* to **Input Message** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. The relevant COBOL data items are contained in fields after the technical fields LL (length of message), ZZ and the COBOL data item containing the transaction code which is mostly the third physical field starting from offset 5 (bytes) in the *input_message*. Do not select the fields LL, ZZ and the transaction code. See Notes.

2. Similar to step 1, define the IMS MPP (IMS Connect) output message. Enter "CBLTDLI" in toolbar icon **Find text in Source**  to look for an IMS system call containing "CBLTDLI", function ISRT and the IOPCB pointer, example:

```
CALL 'CBLTDLI' USING ISRT IOPCB <output-message>
```

Select the corresponding *output_message* in **COBOL Interface**. See Notes.

Select the relevant COBOL data items of *output_message* to **Output Message** by using the context menu or toolbar. The relevant COBOL data items are the fields after the technical fields LL (length of message) and ZZ. Also, do not select LL and ZZ here.

3. Continue with *COBOL to IDL Mapping*.

Notes:

1. It is very important to select the right COBOL data items describing the interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
2. If your COBOL server contain REDEFINES, the first REDEFINE path is offered by default. Check manually whether this is the one you want. If not, correct it. You can select any other REDEFINE path.

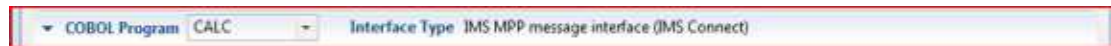
The user interface of the COBOL Mapping Editor is described below.

Mapping Editor User Interface

This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

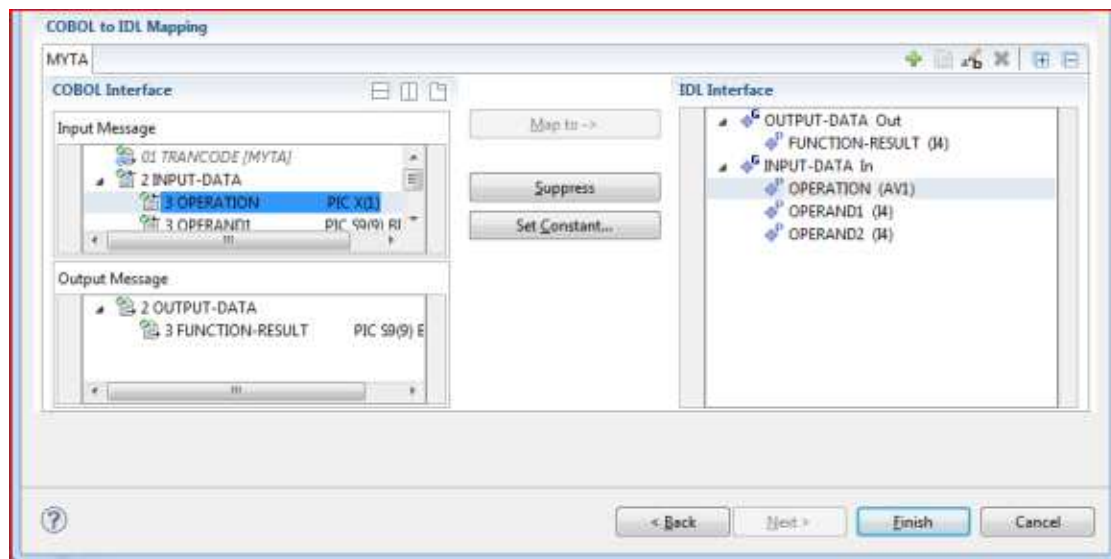
For COBOL server programs with IMS MPP message interface (IMS Connect), the user interface of the COBOL Mapping Editor looks like this:




COBOL Program Selection. Currently selected program with interface type  [More info](#)

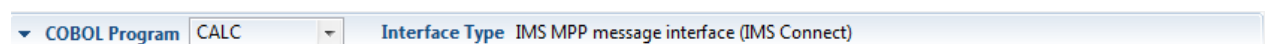


COBOL Source View. Contains all related sources for the currently selected COBOL program  [More info](#)



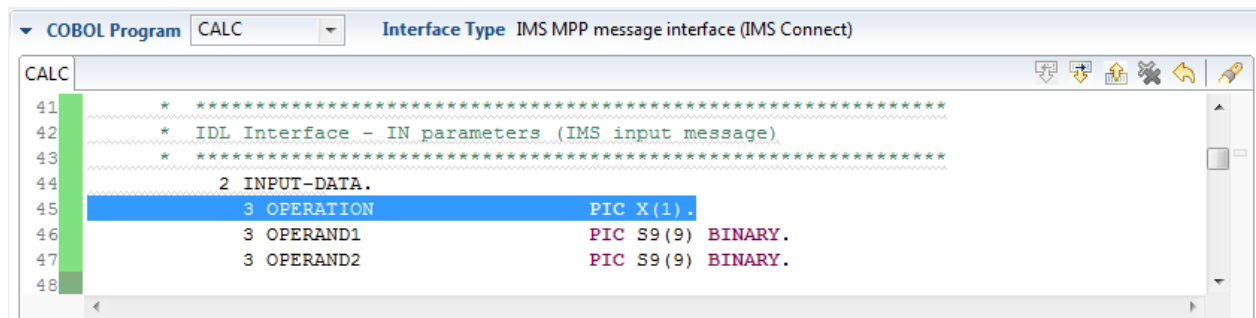
COBOL to IDL Mapping. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface  [More info](#)

COBOL Program Selection









The COBOL Program Selection displays the current selected COBOL program with its interface type. If you have extracted more than one COBOL program within associated IDL file, you can switch to another COBOL program with its mapping by selecting the name in the combo box.

COBOL Source View



All COBOL data items contained in the LINKAGE and WORKING-STORAGE SECTION are offered in a text view. The text view contains all related sources (including copybooks) for the currently selected COBOL program. It is used for selecting data items and retrieving information from the original COBOL sources. The light green bar indicates that the data item is already contained in the COBOL Interface; a dark green bar indicates the data item is selectable and can be added to the COBOL Interface. This section can be collapsed. If you open the Editor with **Modify Interface** it is collapsed by default. The toolbar provides the following actions:

-  Add selected COBOL data item to COBOL Interface as Input Message.
-  Add selected COBOL data item to COBOL Interface as Output Message.
-  Remove selected COBOL data item from COBOL Interface.
-  Remove all COBOL data items from COBOL Interface.
-  Reset COBOL Interface to initial state.
-  Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

- COBOL Interface
- Mapping Buttons
- IDL Interface

COBOL Interface

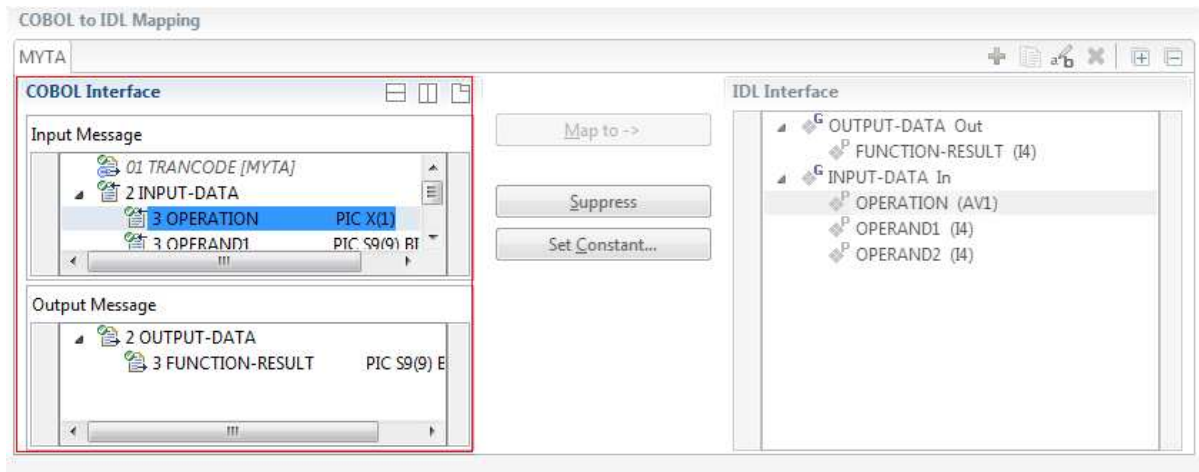
The **COBOL Interface** shows a tree view of your selected COBOL data items describing the interface of the COBOL server. A context menu is available for the COBOL data items, which provides mapping and other functions. On some COBOL data items, decision icons indicate where particular attention is needed, including mapping icons to visualize the COBOL data type and your current mapping.

The COBOL data item names are derived from the COBOL source from which they were extracted. If your COBOL interface contains parameters without a name, that is, the keyword FILLER is used, those COBOL data items are shown as [FILLER]. See *FILLER Pseudo-Parameter*.

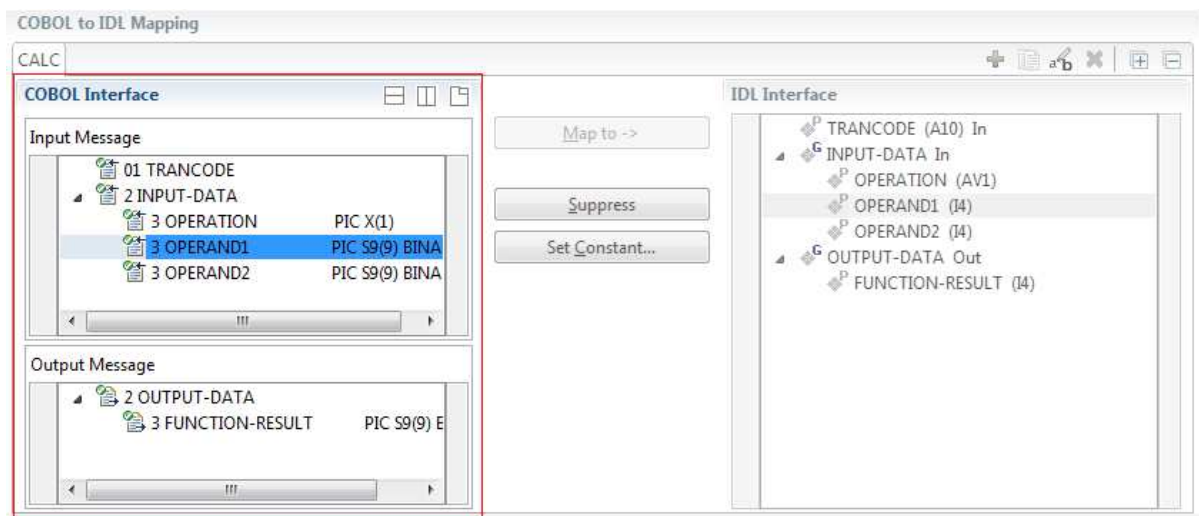
The appearance of the **COBOL Interface** depends on how the transaction name is specified in the **Extractor Settings**:

-

If **Transaction Name** is checked, a hidden parameter with this fixed value appears:



- If **Create IDL parameter for Transaction Name...** is checked, the IDL parameter "TRANCODE" sets the transaction name dynamically at runtime.



You can modify the COBOL interface using context menu or toolbar; decision and mapping icons provide additional information.

Context Menu







The context menu on COBOL data items provides the following mapping and other functions, depending on the data item type, the COBOL level and the current mapping.

These functions are described in more detail under *Mapping Editor IDL Interface Mapping Functions*.

Map to	A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another REDEFINE path.
Suppress	Suppress unneeded COBOL data items.
Set Constant	Set COBOL data items to constant.
Set Array Mapping	Map an array to a fixed sized or unbounded array.
<p>Note: This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping could result in runtime errors.</p>	
Remove from COBOL Interface	Remove the data item from the COBOL interface. This also removes the mapped IDL parameter from all IDL interfaces for the current COBOL program. See <i>COBOL Program Selection</i> .

Toolbar


The toolbar offers the following actions:

-  Create IDL Interface. Creates a new IDL interface based on the current COBOL interface: all IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL REDEFINE, the first REDEFINE path is mapped to IDL; FILLERS are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
-  Copy current IDL Interface. Creates a duplicate of the current IDL interface: all modifications such as IDL directions, suppress, selection of REDEFINE paths etc. are kept.
-  Remove current IDL Interface.
-  Rename current IDL Interface.
-  Expand the full tree.
-  Collapse the full tree.

See also *Map to Multiple IDL Interfaces*






Decision Icons

The decision icons in the first column are set on COBOL data items where particular attention is needed:

-  This icon visualizes a COBOL REDEFINE. It is essential that you map the correct redefine path for your mapping to In, Out or InOut using the context menu. If you map a REDEFINE path, all other sibling REDEFINE paths are automatically set to "Suppress".

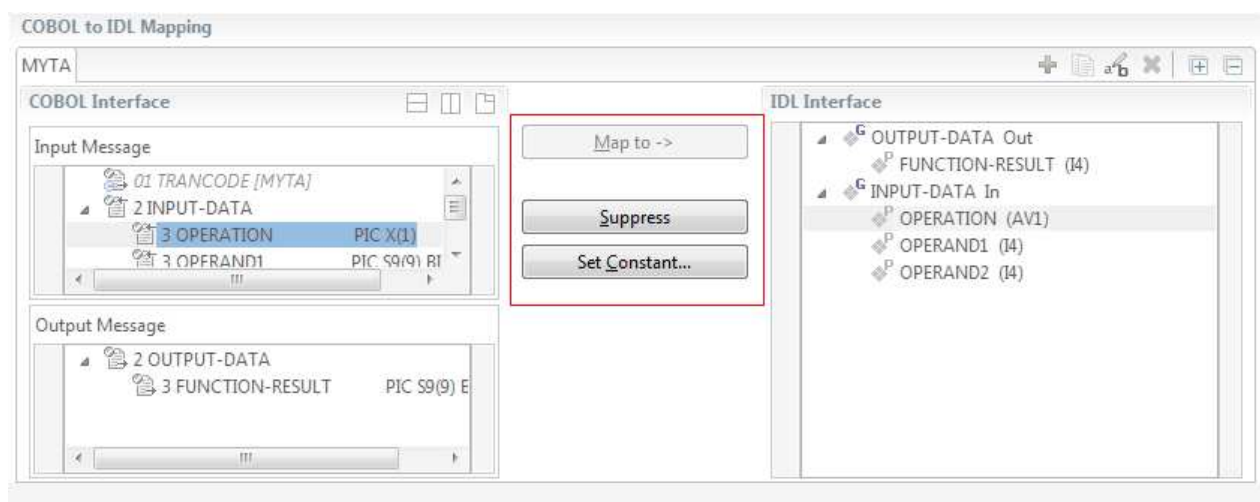
Mapping Icons

The following mapping icons on the COBOL data items indicate your current IDL mapping:

-  Scalar parameter, mapped to In.
-  Scalar parameter, mapped to Out.
-  Group parameter, here mapped to In.
-  REDEFINE parameter, here mapped to Out.
-  Parameter set to Constant.

Mapping Buttons

The following buttons are available:



Note:

In this example, a fixed value for transaction name was specified in the **Extractor Settings**.

Map to ->

A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another REDEFINE path.

Suppress

See *Suppress Unneeded COBOL Data Items*.

Set Constant...

See *Set COBOL Data Items to Constants*.

IDL Interface

If you have mapped the COBOL interface to multiple IDL interfaces, select the IDL interface by choosing the tabs. In the **IDL Interface** tree view, a context menu is also available with the following possibilities:

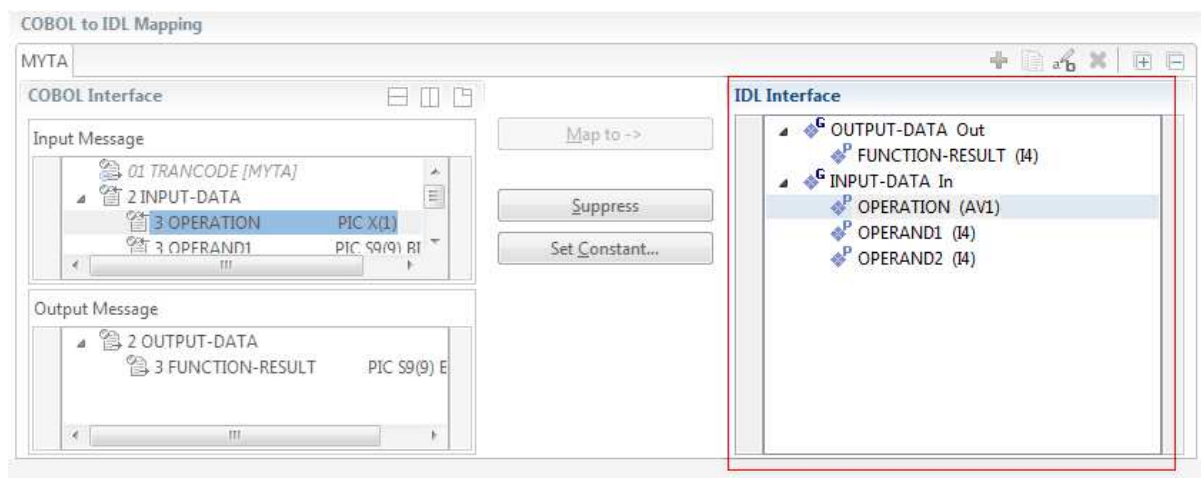
- Rename

- **Remove from COBOL Interface.** This also removes the mapped IDL parameter from all IDL interfaces for the current COBOL program. See *COBOL Program Selection* above.

The appearance of the **IDL Interface** depends on how the transaction name is specified in the **Extractor Settings**. See *Extracting from an IMS MPP Message Interface Program*.

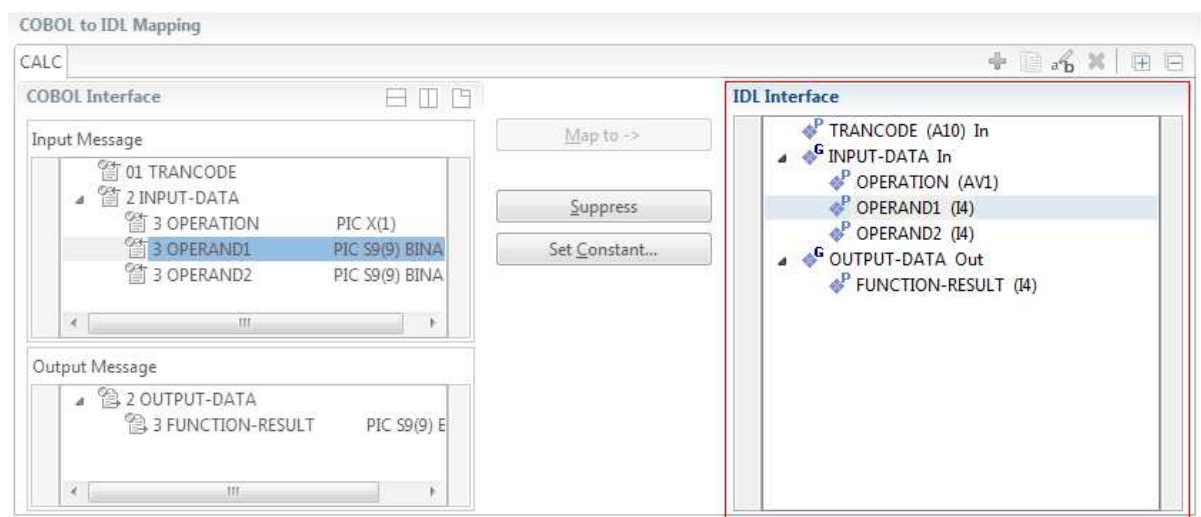
- **Fixed Value**

In the **COBOL Interface** pane the first parameter shows the value for your transaction name in square brackets. There is no IDL parameter contained in the *IDL Interface* for it. Your IDL interface is free of this technical parameter, and RPC clients do not have to specify it at runtime.



- **Dynamically at Runtime**

Your *IDL Interface* contains an IDL parameter for the transaction name ("TRANCODE"). RPC clients set the name dynamically at runtime.



Mapping Editor IDL Interface Mapping Functions

This section covers the following topics:

- Map to
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths
- Set Arrays (Fixed <-> Unbounded)

Map to

With the **Map to** functions you make a COBOL data item visible as an IDL parameter in the IDL interface, that is, you design the IDL interface by defining input and output parameters.

➤ To map a COBOL data item to IDL interface

1. Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to** function available in the context menu and as mapping button to make a COBOL data item visible as an IDL parameter in the input message of the IDL interface.
2. Do the same for the output message of the IDL interface.

Notes:

1. If a COBOL group is mapped, all subsequent child COBOL data items are also made visible in the IDL interface.
2. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu and as mapping button, a COBOL data item can be removed from the IDL interface.

Suppress Unneeded COBOL Data Items

COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified – it becomes shorter and tidier. This is useful, for example

- for FILLER data items
- if the RPC client or Adapter Service does not need an Out parameter
- if the RPC server or Adapter Service does not need an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is also reduced.

➤ To suppress unneeded COBOL data items

- Use the **Suppress** function available in the context menu and as mapping button to make the COBOL data item invisible in the IDL interface.

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to** (see above) available in the context menu and as mapping button, a COBOL data item can be made visible in the IDL interface again.

Set COBOL Data Items to Constants

COBOL data items that always require fixed constant values on input to the COBOL server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as RECORD-TYPES. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

➤ **To map COBOL data items to constants**

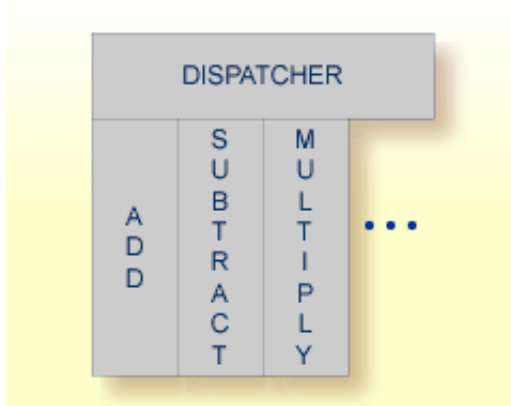
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the defined constant in the COBOL data item to your COBOL server.
3. With the function **Map to** (see above) available in the context menu and as mapping button, a COBOL data item can be made visible in the IDL interface again.

Map to Multiple IDL Interfaces

Assume the COBOL server program provides multiple functions or operations, in the following example ADD, SUBTRACT, MULTIPLY. Some dispatcher front-end code executes the correct function, for example, depending on a *function-code* or *operation-code* parameter:






This example is described in more detail under *Example 1: COBOL Server with Multiple Functions*.

If you have such a situation, a good approach is to expose each COBOL server program function separately as an IDL program. This gives advantages in further processing of the IDL and COBOL mapping files (SVM and CVM). For example:

- If your target endpoint is a web service: instead having a Web service with a single operation, you get a web service with multiple operation, one operation for each COBOL function.
- If your target endpoint is Java or .NET: instead having a class with a single method, you get a class with multiple methods, one method for each COBOL function.

➤ **To map a COBOL interface to multiple IDL interfaces**





1. Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions  or .
2. Give the IDL interfaces meaningful names with the toolbar function .
3. Define the required constant values to the *function-code* or *operation-code* parameter, see *Set COBOL Data Items to Constants* above.

For the delivered Example 1: COBOL Server with Multiple Functions:

- First, for step 1 above: Extract and define 3 separate IDL programs ADD, SUBTRACT, MULTIPLY.
- Second, for step 2 above: Rename them to suitable names, e.g. 'ADD', 'SUBTRACT', MULTIPLY'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter OPERATION respectively.

Notes:

1. The following functions are offered to create further mappings from the COBOL interface, resulting in multiple IDL interfaces (IDL programs).

Icon	Function	Description
	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL REDEFINE, the first REDEFINE path is mapped to IDL; FILLERS are suppressed according to your selection, see <i>Step 4: Define the Extraction Settings and Start Extraction</i> .
	Copy current IDL Interface	Creates a duplicate of current IDL interface. All modifications such as IDL directions, suppress, selection of REDEFINE paths etc. are kept.
	Rename current IDL Interface	The default name for the IDL interface is based on the COBOL program name plus appended number. With this function you can give the IDL interface a suitable name.
	Remove current IDL Interface	Deletes the current IDL interface.

- With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

For COBOL server programs containing COBOL REDEFINES, the correct REDEFINE path needs to be chosen for the IDL interface.

To select redefine paths

- Use the **Map to** function available in the context menu and as mapping button to make the COBOL REDEFINE path available in the IDL interface.

Begin with the COBOL REDEFINE defined at the highest level first. Work through all inner COBOL REDEFINE data items, going from higher levels to lower levels.

Notes:

- Only one REDEFINE path of a COBOL REDEFINE can be mapped to the IDL interface. All COBOL REDEFINE siblings are suppressed.
- If a REDEFINE path is actively mapped to the IDL interface, all COBOL REDEFINE siblings are suppressed.
- You can suppress all REDEFINE paths of a COBOL REDEFINE. Simply suppress the active REDEFINE path, see *Suppress Unneeded COBOL Data Items above*.

Set Arrays (Fixed <-> Unbounded)

For COBOL server programs using the message length to transfer a variable number of elements in a COBOL table with a fixed size (see *Tables with Fixed Size*) in a variable manner (see *Tables with Variable Size - DEPENDING ON Clause*) you need to set the mapping to unbounded array.

For details of such a COBOL server program see *Example 2: COBOL Server Using Data Length to Process a Variable Number of Array Elements*.

➤ To set arrays from fixed to unbounded or vice versa

- Select the COBOL table and use the function **Set Arrays (Fixed<->Unbounded)** available in the context menu. A modal window is displayed. Select **Unbounded array**. The IDL array parameter will be changed from fixed array */number* to an unbounded array */Nnumber*, see *array-definition* under *Software AG IDL Grammar* in the IDL Editor documentation.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL Table with a fixed size (see *Tables with Fixed Size*) used in this manner must be the last parameter of the COBOL interface; it must not be a subparameter of any other COBOL table and must not contain any *DEPENDING ON* clause (see *Tables with Variable Size - DEPENDING ON Clause*).

Programming Techniques

This section covers the following topics:

- Example 1: COBOL Server with Multiple Functions
- Example 2: COBOL Server Using Data Length to Process a Variable Number of Array Elements

Example 1: COBOL Server with Multiple Functions

Assume a COBOL server program has a *FUNCTION* or *OPERATION* code COBOL data item in its COBOL interface. The COBOL server program behaves differently depending on field values of this data item. See the following example where a COBOL programs implements a calculator with the functions *ADD*, *SUBTRACT*, *MULTIPLY*, etc. The execution of the different functions is controlled by the COBOL data item *OPERATION*:

. . .

```

01 OPERATION                      PIC X(1) .
01 OPERAND1                       PIC S9(9) BINARY.
01 OPERAND2                       PIC S9(9) BINARY.
01 FUNCTION-RESULT                PIC S9(9) BINARY.
. . .
MOVE 0 TO FUNCTION-RESULT.
EVALUATE OPERATION
  WHEN "+"
    ADD OPERAND1 OPERAND2
    GIVING FUNCTION-RESULT
  WHEN "-"

```

```

        SUBTRACT OPERAND2 FROM OPERAND1
        GIVING FUNCTION-RESULT
    WHEN " * "
        MULTIPLY OPERAND1 BY OPERAND2
        GIVING FUNCTION-RESULT
    WHEN . . .

END-EVALUATE.
. . .

```

You can expose each COBOL server program function separately. The advantages or reasons for wanting this depend on the target endpoint. For example:

- **Web Service**

Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.

- **Java or .NET**

Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.

- etc.

To do this you need to extract the COBOL server program as described under *Map to Multiple IDL Interfaces*.

Example 2: COBOL Server Using Data Length to Process a Variable Number of Array Elements

Assume a COBOL IMS MPP (IMS Connect) server program has a fixed-sized COBOL table as its last parameter, similar to COBOL data item COBOL-TABLE-FIX in the example below; each table element is 100 bytes; the length of COBOL-FIELD1 + COBOL-FIELD2 + COBOL-FIELD3; the length of the data preceding the COBOL table is described by COBOL-GROUP1; its length is 1000 bytes.

```

WORKING-STORAGE SECTION.
01 NUMBER-OF-INCOMING-ELEMENTS      PIC S9(8) BINARY.
01 NUMBER-OF-OUTGOING-ELEMENTS      PIC S9(8) BINARY.
. . .

01 INPUT-MESSAGE.
    05 INPUT-IMS-META.
        10 INPUT-LL                  PIC S9(3) BINARY.
        10 INPUT-ZZ                  PIC S9(3) BINARY.
        10 INPUT-TRANSACTION         PIC X(10).
    05 INPUT-DATA.
        10 COBOL-GROUP1.
        20 COBOL-TABLE-PREFIX        PIC X(1000).
        10 COBOL-TABLE-FIX          OCCURS 20.
            20 COBOL-GROUP2.
                25 COBOL-FIELD1      PIC X(4).
                25 COBOL-FIELD2      PIC X(3).
                25 COBOL-FIELD3      PIC X(50).
01 OUTPUT-MESSAGE.
    05 OUTPUT-IMS-META.
        10 OUTPUT-LL                  PIC S9(3) BINARY.
        10 OUTPUT-ZZ                  PIC S9(3) BINARY.
    05 OUTPUT-DATA.
        10 COBOL-GROUP1.

```



```

      20 COBOL-TABLE-PREFIX          PIC X(1000).
    10 COBOL-TABLE-FIX              OCCURS 20.
      20 COBOL-GROUP2.
        25 COBOL-FIELD1            PIC X(30).
        25 COBOL-FIELD2            PIC X(20).
        25 COBOL-FIELD3            PIC X(50).
LINKAGE SECTION.
. . .
PROCEDURE DIVISION USING IO-PCB.
  CALL "CBLTDLI" USING GU, IO-PCB, INPUT-MESSAGE.
. . .
  COMPUTE NUMBER-OF-INCOMING-ELEMENTS = (INPUT-LL
    - LENGTH OF COBOL-GROUP1 IN AREA INPUT-MESSAGE)
    / LENGTH OF COBOL-GROUP2 IN AREA INPUT-MESSAGE.
. . .
  COMPUTE OUTPUT-LL = LENGTH OF COBOL-GROUP2 IN AREA OUTPUT-MESSAGE
    + (NUMBER-OF-OUTGOING-ELEMENTS
    * LENGTH OF COBOL-GROUP2 IN AREA OUTPUT-MESSAGE).

  CALL "CBLTDLI" USING ISRT, IO-PCB, OUTPUT-MESSAGE.
. . .

GOBACK.

```

During input the COBOL IMS MPP (IMS Connect) server program uses the IMS input message length INPUT-LL to evaluate the NUMBER-OF-INCOMING-ELEMENTS. During output the IMS output message length is determined accordingly to the NUMBER-OF-OUTGOING-ELEMENTS and set in OUTPUT-LL.

Although the COBOL table is defined as a table with a fixed size (see *Tables with Fixed Size*) it is used in a variable manner, similar to tables with variable size (see *Tables with Variable Size - DEPENDING ON Clause.*) In this case you need to map the COBOL table to an IDL unbounded array. See *Set Arrays (Fixed <-> Unbounded)*.