webMethods EntireX

Software AG IDL Extractor for COBOL

Version 10.8

October 2022

WEBMETHODS

This document applies to webMethods EntireX Version 10.8 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-EEXXCOBEXTRACTOR-108-20220601

Table of Contents

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Introduction to the IDL Extractor for COBOL

2 Introduction to the IDL Extractor for COBOL

Introduction

The Software AG IDL Extractor for COBOL inspects a COBOL source and its copybooks for COBOL data items to extract. It can also extract directly from copybooks. In a user-driven process supported by an *Extractor Wizard*, the interface of a COBOL server is extracted and - with various features offered by a *Mapping Editor* - modelled to a client interface.

1. Start the wizard, select your server program and make COBOL-specific settings.
2. Optional. This step is not always necessary: it is possible that parameters have already been selected, for example as a result of the COBOL `USING` clause.
3. Optional. If necessary, you can modify the parameter selection from the Mapping Editor.
4. Fine-tune the COBOL to IDL mapping.
5. Generate an IDL file and a server mapping file. These two related files map the client interface to the COBOL server program and are described below:

- **IDL File**
  The Software AG IDL file (interface definition language) contains the modelled interface of the COBOL server. In a follow-up step the IDL file is the starting point for the RPC client-side wrapping generation tools to generate client interface objects. See *EntireX Wrappers* in the Designer documentation.
- **Server Mapping File**
  A server mapping file to complete the mapping is generated only if it is required by the RPC server at runtime to call the COBOL server. See *Server Mapping Files for COBOL* in the Designer documentation.

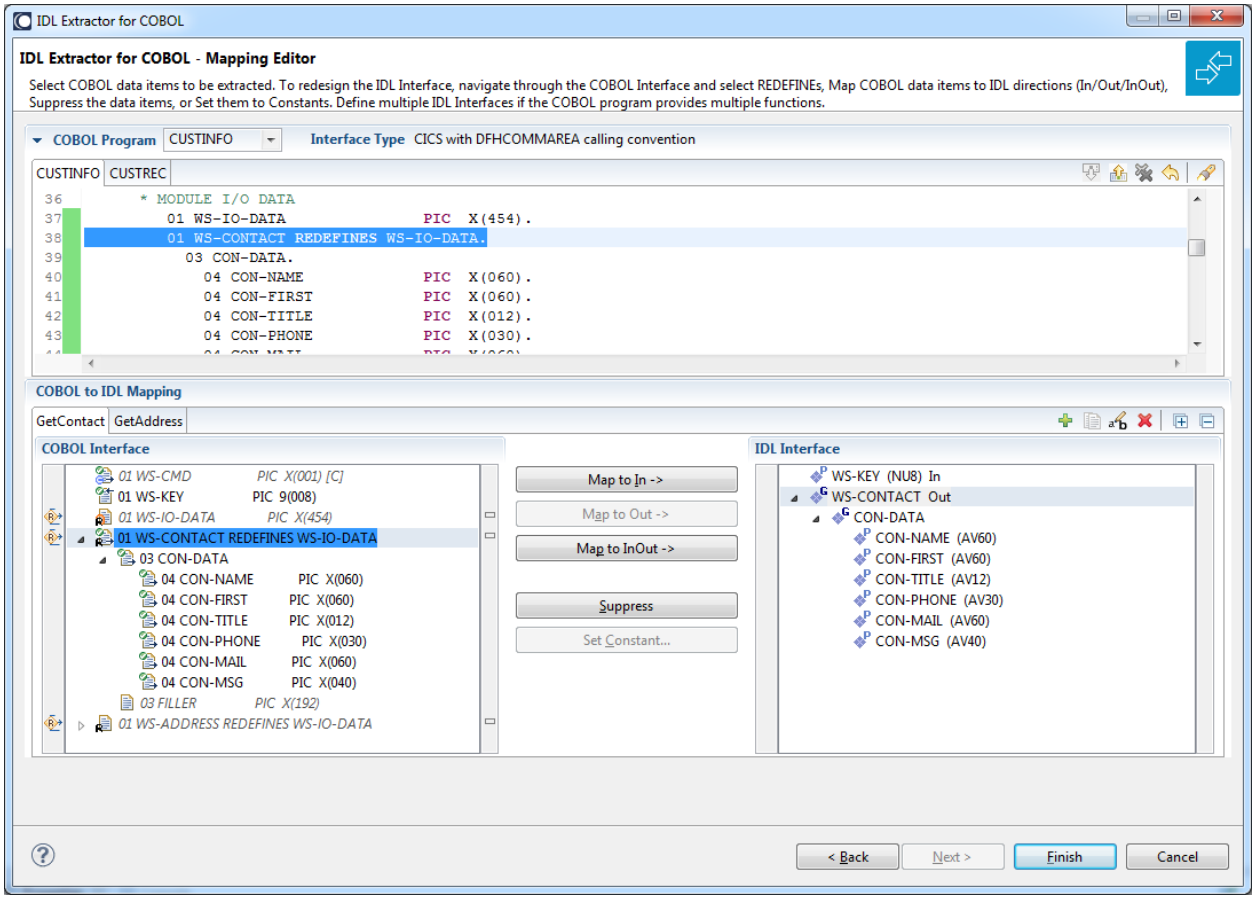
Extractor Wizard

The extractor wizard guides you through the extraction process. The wizard supports the following tasks:

- Accessing COBOL source files, either in the local file system where the Designer runs or remotely from the host computer with the RPC server extractor service. The wizard supports the following: z/OS partitioned data sets and CA Librarian data sets (including member archive levels) as well as BS2000 LMS libraries. See *Extractor Service* in the RPC Server documentation for Batch | IMS | BS2000. For this purpose, define a local or remote COBOL extractor environment. See ***COBOL Preferences***.
- Resolving of COBOL copybooks. If a relevant copybook from the COBOL `DATA DIVISON` is missing, a browse dialog is offered where you can locate the copybook - either a folder (local extractor environment) or data set (remote extractor environment) - interactively. Copybook folder or data sets can also be predefined in the COBOL extractor environment. See ***COBOL Preferences***.
- Resolving of COBOL copybooks with the `REPLACE` option.
- CA Librarian (`-INC`) and CA Panvalet (`++INCLUDE`) control statements are supported. They are handled in a similar way to copybooks.
- Various COBOL server interface types, such as standard CICS `DFHCOMMAREA`, CICS with different structures on input and output, CICS with a large buffer compatible to webMethods WMTLSRVR, standard batch, and IMS BMP server with PCB pointers. See ***Supported COBOL Interface Types***.
- Selecting the COBOL server interface manually within the ***COBOL Mapping Editor*** page. This allows you to extract from a COBOL server where the interface definition is not completely given by the parameters provided in the ***`PROCEDURE DIVISION` Mapping***, making it impossible to detect the parameters automatically.
- Defining the default COBOL to IDL mapping in the ***COBOL Preferences*** for the following fields:
  - COBOL pseudo-parameter `FILLER` fields. You can define whether they should be part of the RPC client interface or not. By default, they are not contained in the IDL.
  - The name prefix for `FILLER` and anonymous groups used for IDL parameters.
  - COBOL alphanumeric fields (`PICTURE X`, `A`, `G`, `N`). They can be mapped either to variable-length or fixed-length strings in the IDL. This option is provided for modern RPC clients that support variable-length strings, and also for legacy RPC clients that support fixed-length strings only.

The extractor wizard is described in a step-by-step tutorial; see ***Using the IDL Extractor for COBOL - Overview***.

Mapping Editor



The *COBOL Mapping Editor* is the tool to select and map the COBOL server interface to IDL. This section gives a short overview of the mapping features provided. These features are described in more detail in the documentation section for the respective interface type.

- Add and remove the parameters of the COBOL server in the top window of the COBOL Mapping Editor page. The current selection is shown in the bottom window for fine tuning.
- Provide IDL directions for parameters of the COBOL server. A COBOL server does not contain IDL direction information, so you can add this information manually in the Mapping Editor.
- Select `REDEFINE` paths used in the IDL. The Mapping Editor allows you to select a single `REDEFINE` path for every `REDEFINE` unit (all redefine paths addressing the same storage location).
- Suppress unneeded fields in the IDL. This keeps the IDL client interface lean and also minimizes the amount of data transferred during runtime.
- Define parameter constants as input for the COBOL server. Constant parameters are not contained in the IDL file, which means they are invisible for RPC clients. This makes the IDL client interface easier and safer to use, minimizing improper usage.

- For one COBOL server program, you can create and model multiple interfaces. If the IDL is processed further with a wrapper of the Designer, the business functions are provided as
  - Web service operations if exposed as a Web service instead of a Web service with a single operation
  - methods if wrapped with the Java Wrapper or .NET Wrapper instead of a Java class with a single method
  - etc.

See *COBOL Mapping Editor* for more information.

Supported COBOL Interface Types

The IDL Extractor for COBOL supports as input a COBOL server with various interface types. This section covers the following topics:

The interface type you are mostly working with can be set in the preferences. See *COBOL Preferences*.

Supported CICS COBOL Interface Types

Analyzing the technique used to access the interface with COBOL and CICS statements is the safest way to determine the interface type. The following CICS COBOL interface types are supported:

- *CICS with `DFHCOMMAREA` Calling Convention*
- *CICS with Channel Container Calling Convention*
- *CICS with `DFHCOMMAREA` Large Buffer Interface*

There is no clear and easy indication how to identify the interface type of a CICS COBOL server without COBOL and CICS knowledge. Below are some criteria that might help to determine the interface type. If you are unsure, consult a CICS COBOL specialist.

- The payload size of the CICS COBOL server is greater than 32 KB:

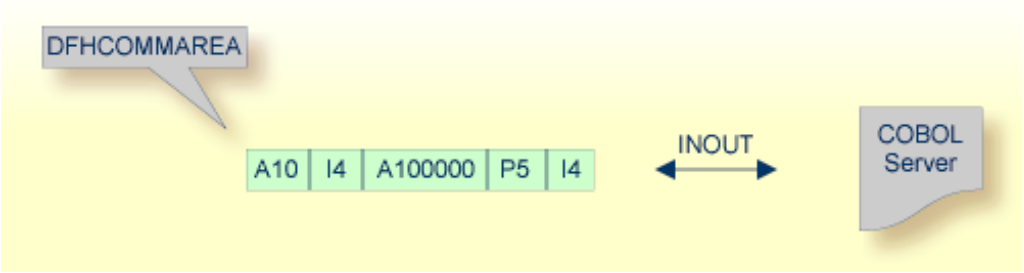
- In this case it is *not* a DFHCOMMAREA interface, because the DFHCOMMAREA is limited to 32 KB.
- It could be a large buffer or channel container interface, which are only limited by the storage (memory) available to them.

- The CICS COBOL server is located in a remote CICS region:
  - In this case it is *not* a large buffer interface, because large buffer programs must reside on the same CICS region as the caller, that is, the *RPC Server for CICS*.
  - It could be a DFHCOMMAREA or channel container interface, which can reside in a remote CICS region.

Note: The most used interface type is the DFHCOMMAREA interface. Large buffer and channel container interfaces are used much less frequently.

CICS with DFHCOMMAREA Calling Convention

The IDL Extractor for COBOL supports CICS programs using the standard `DFHCOMMAREA` calling convention.



The following illustrates roughly how you can determine whether a COBOL server follows the `DFHCOMMAREA` calling convention standard:

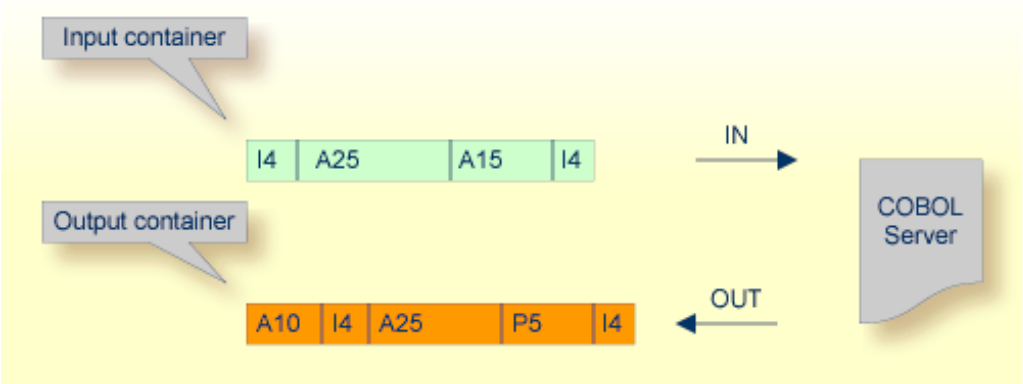
```
LINKAGE SECTION.
01 DFHCOMMAREA.
   02 OPERATION                    PIC X(1).
   02 OPERAND-1                    PIC S9(9) BINARY.
   02 OPERAND-2                    PIC S9(9) BINARY.
   02 FUNCTION-RESULT              PIC S9(9) BINARY.

PROCEDURE DIVISION USING DFHCOMMAREA.
   . . .
```

Most `DFHCOMMAREA` programs have a `DFHCOMMAREA` data item in their `LINKAGE SECTION` and may address this item in the `PROCEDURE DIVISION` header. If you find this in your COBOL source it's a clear indication it is a `DFHCOMMAREA` server program. But even if this is missing, it can be a `DFHCOMMAREA` program, because there are alternative programming styles. If you are unsure, consult a COBOL CICS specialist or see ***Supported CICS COBOL Interface Types*** for more information.

See *Step 4: Define the Extraction Settings and Start Extraction* and *CICS with `DFHCOMMAREA` Calling Convention - In same as Out* for more information on extracting COBOL servers with this interface type.

CICS with Channel Container Calling Convention

The IDL Extractor for COBOL supports CICS programs using the channel container calling convention.



The following illustrates roughly how you can determine whether a COBOL server follows the Channel Container standard.

```
WORKING-STORAGE SECTION.
01 WS-CONTAINER-IN-NAME                PIC X(16) VALUE "CALC-IN".
01 WS-CONTAINER-OUT-NAME               PIC X(16) VALUE "CALC-OUT".
. . .
LINKAGE SECTION.
01 LS-CONTAINER-IN-LAYOUT.
   02 OPERATION                        PIC X(1).
   02 OPERAND1                         PIC S9(9) BINARY.
   02 OPERAND2                         PIC S9(9) BINARY.
01 LS-CONTAINER-OUT-LAYOUT.
   02 FUNCTION-RESULT                  PIC S9(9) BINARY.

PROCEDURE DIVISION.
   . . .
   EXEC CICS GET CONTAINER (WS-CONTAINER-IN-NAME) SET (ADDRESS OF ↩
LS-CONTAINER-IN-LAYOUT) ...
   . . .
   EXEC CICS PUT CONTAINER (WS-CONTAINER-OUT-NAME) FROM (ADDRESS OF ↩
LS-CONTAINER-OUT-LAYOUT) ...
   . . .
```

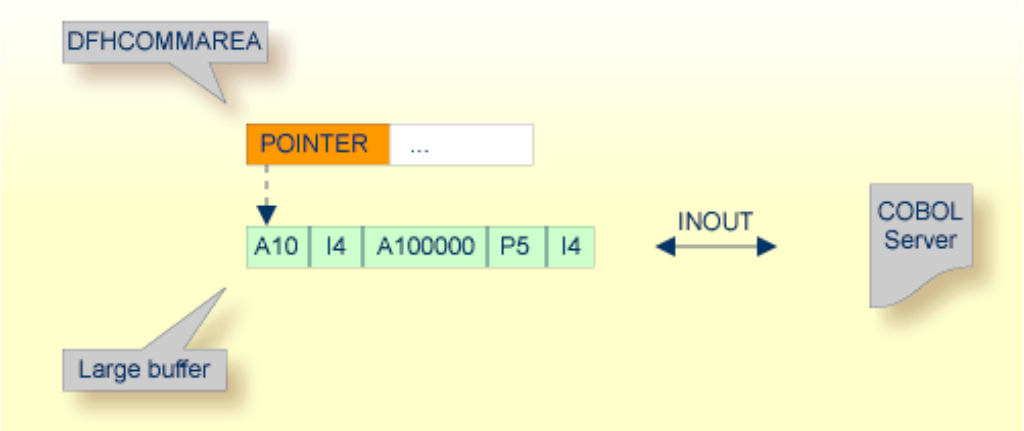
Channel Container programs use `EXEC CICS GET CONTAINER` in their program body (`PROCEDURE DIVISION`) to read their input parameters. Output parameters are written using `EXEC CICS PUT`

CONTAINER. There is no clear indication in the linkage or working storage section to identify a channel container program. If you are unsure, consult a COBOL CICS specialist for clarification.

See *Step 4: Define the Extraction Settings and Start Extraction* and *CICS with Channel Container Calling Convention* for more information on extracting COBOL servers with this interface type.

CICS with DFHCOMMAREA Large Buffer Interface

This type of program has a defined DFHCOMMAREA interface to access more than 31 KB of data in CICS. The interface is the same as the webMethods WMTLSRVR interface. This enables webMethods customers to migrate to EntireX.



Technically,

- the DFHCOMMAREA layout contains a structure with a *length* and a *pointer* to a large buffer. The following illustrates this:

```
LINKAGE SECTION.

01 DFHCOMMAREA.
   10 WM-LCB-MARKER                        PIC X(4).
   10 WM-LCB-INPUT-BUFFER                  POINTER.
   10 WM-LCB-INPUT-BUFFER-SIZE             PIC S9(8) BINARY.
   10 WM-LCB-OUTPUT-BUFFER                 POINTER.
   10 WM-LCB-OUTPUT-BUFFER-SIZE            PIC S9(8) BINARY.
   10 WM-LCB-FLAGS                         PIC X(1).
      88 WM-LCB-FREE-OUTPUT-BUFFER     VALUE 'F'.
   10 WM-LCB-RESERVED                      PIC X(3).
01 INOUT-BUFFER.
   02 OPERATION                            PIC X(1).
   02 OPERAND-1                            PIC S9(9) BINARY.
   02 OPERAND-2                            PIC S9(9) BINARY.
   02 FUNCTION-RESULT                      PIC S9(9) BINARY.
 . . .
PROCEDURE DIVISION USING DFHCOMMAREA.
```

```
  . . .
    SET ADDRESS OF INOUT-BUFFER TO WM-LCB-INPUT-BUFFER.
*  process the INOUT-BUFFER and provide result
    EXEC CICS RETURN.
```

The fields subordinated under `DFHCOMMAREA` prefixed with `WM-LCB` describe this structure. The field names themselves can be different, but the COBOL data types must match exactly.

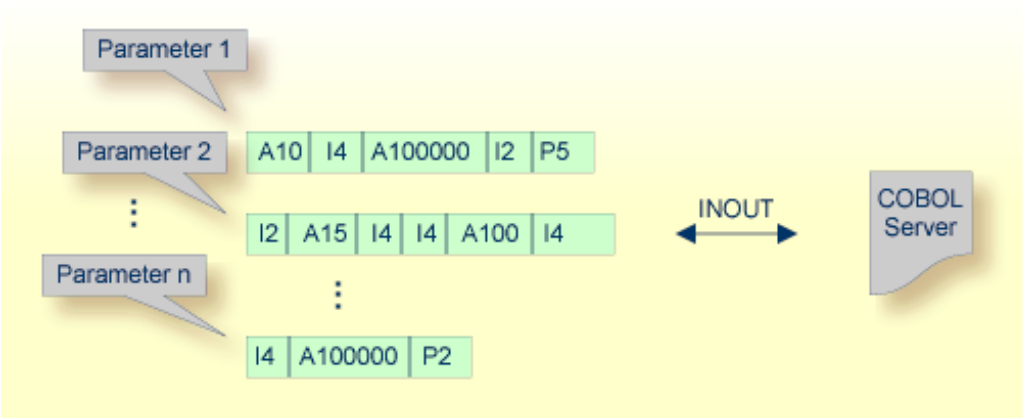
- data is described by separate structures, here `INOUT-BUFFER` in the linkage section.

  If you find this in your COBOL source, it's a clear indication it is a large buffer program. If you are unsure, consult a COBOL CICS specialist for clarification.

See ***Step 4: Define the Extraction Settings and Start Extraction*** and ***CICS with `DFHCOMMAREA` Large Buffer Interface - In same as Out*** for more information on extracting COBOL servers with this interface type.

Batch with Standard Linkage Calling Convention

Standard call interfaces with a given number of parameters are supported. Every parameter addresses a fixed COBOL structure.

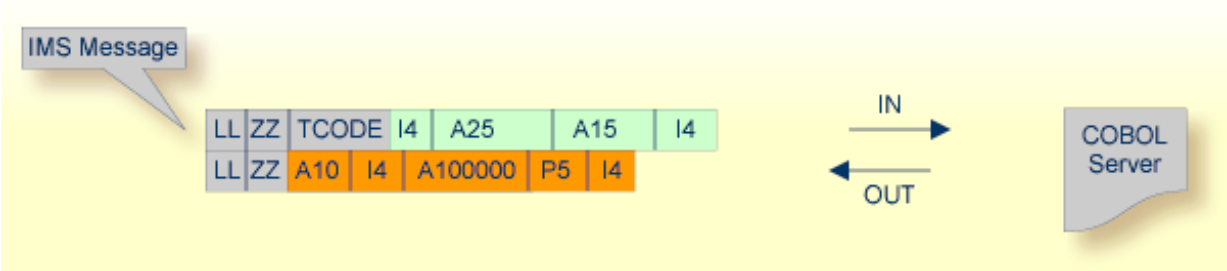


Technically, the COBOL server contains

- a parameter list: `PROCEDURE DIVISION USING PARM1 PARM2 ... PARM`*n*
- the parameters in the linkage section as COBOL data items on level 1

See *Step 4: Define the Extraction Settings and Start Extraction* and *Batch with Standard Linkage Calling Convention* for more information on extracting COBOL servers with this interface type.

IMS MPP Message Interface (IMS Connect)



IMS message processing programs (MPP) get their parameters through an IMS message and return the result by sending an output message to IMS. The IDL Extractor for COBOL enables extractions from such programs.

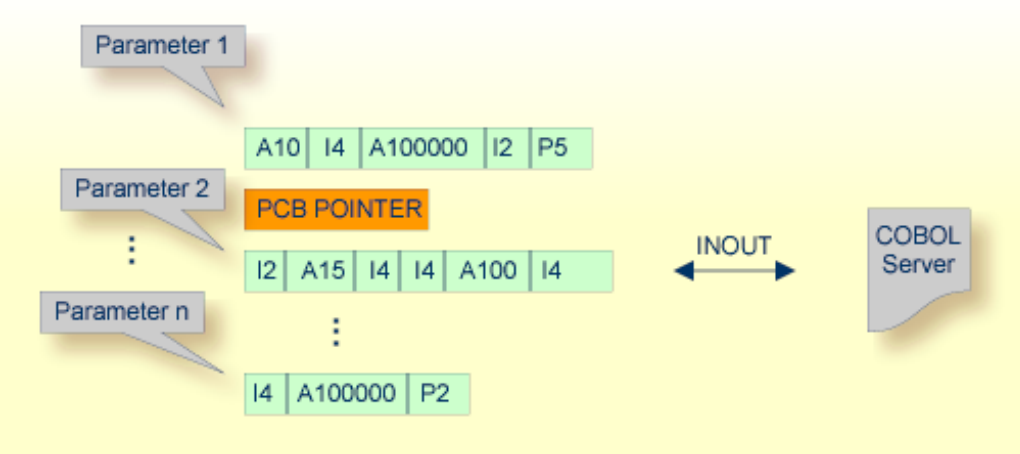
The COBOL server contains:

- a structure in the working storage section for the input and the output message.
- an IOPCB in the linkage section used to read input messages and write output messages using an IMS system call (i.e. `CALL` "CBLTDLI").
- The message contains also technical fields specific to IMS (see fields `LL`, `ZZ` and `TRANCODE` in the picture above).

See *Step 4: Define the Extraction Settings and Start Extraction* and *IMS MPP Message Interface (IMS Connect)* for more information on extracting COBOL servers with this interface type.

IMS BMP with Standard Linkage Calling Convention

IMS batch message processing programs (BMP) with PCB parameters are directly supported. You have the option to specify a PSB list as input to the extractor to locate PCB parameters.



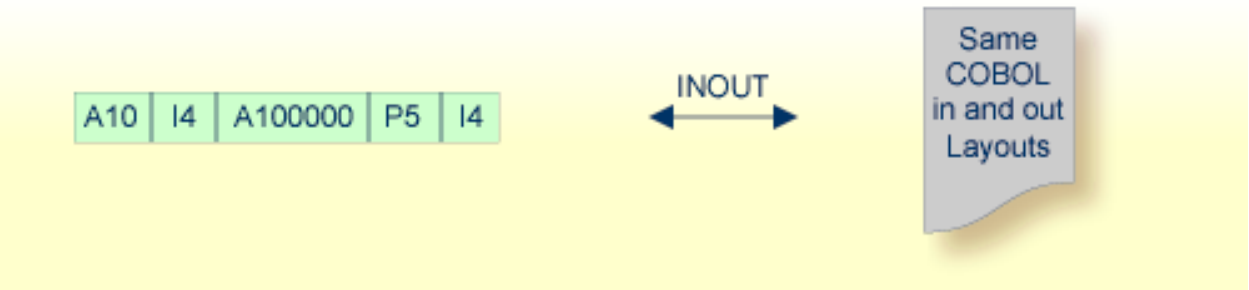
Technically, the COBOL server contains

- a parameter list: `PROCEDURE DIVISION USING PARM1 PCB PARM2 ... PARMn`
- IMS-specific *PCB pointers* within the parameter list
- the parameters in the linkage section as COBOL data items on level 1

See ***Step 4: Define the Extraction Settings and Start Extraction*** and ***IMS BMP with Standard Linkage Calling Convention*** for more information on extracting COBOL servers with this interface type.

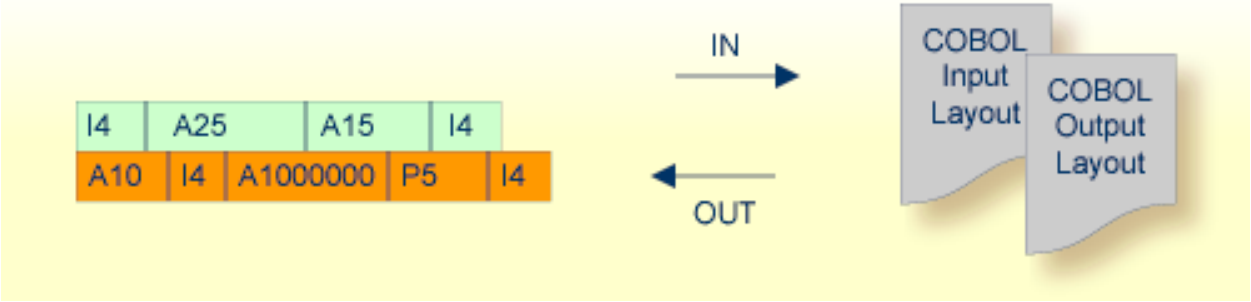
COBOL Converter

A file containing valid COBOL data items can be used to extract a COBOL Converter for the EntireX Adapter. In most cases the COBOL layout during input and output will be same:



See ***Step 4: Define the Extraction Settings and Start Extraction*** and ***COBOL Converter - In same as Out*** for more information on extracting COBOL with this interface type.

If required, the COBOL layout during input can be different as on output. For example input and output are described with different `REDEFINES` clauses.



See *COBOL Converter - In different to Out* for more information on extracting COBOL with this interface type.

What to do with other Interface Types?

Other interface types, for example CICS with non-DPL-enabled `DFHCOMMAREA`, can be supported by means of a custom wrapper. If you have to extract from such a COBOL server, proceed as follows:

1. Implement a custom wrapper, providing one of the supported interface types above.
2. Extract from this custom wrapper.

Compatibility between COBOL Interface Types and RPC Server

To call your server program successfully, the target RPC runtime component used must support the interface type. The table below gives an overview of possible combinations of an interface type and RPC server.

	z/OS			UNIX/Windows			IBM i	BS2000
Interface Type of your Server Program	**CICS**	**Batch**	**IMS**	**CICS ECI**	**CICS Socket Listener**	**IMS Connect**	**AS/400**	**Batch**
CICS with DFHCOMMAREA Calling Convention	x			x	x			
CICS with DFHCOMMAREA Large Buffer Interface	x				x			
CICS with Channel Container Calling Convention	x				x			
Batch with Standard Linkage Calling Convention		x	x				x	x
IMS BMP with Standard Linkage Calling Convention			x					
IMS MPP Message Interface (IMS Connect)						x		
COBOL Converter								

Compatibility between COBOL Interface Types and EntireX Adapter Connection Types

The table below gives an overview of COBOL interface types and EntireX Adapter connection types.

Interface Type of your Server Program	EntireX Adapter Connection Type	Note
CICS with DFHCOMMAREA Calling Convention	CICS ECI Connection or CICS Socket Listener Connection	
CICS with DFHCOMMAREA Large Buffer Interface	CICS Socket Listener Connection	
CICS with Channel Container Calling Convention	CICS Socket Listener Connection	
Batch with Standard Linkage Calling Convention	AS/400 Connection	To call your server program on a platform other than IBM i, use an RPC Connection or Direct RPC Connection to an appropriate RPC Server for Batch (z/OS | BS2000).
IMS BMP with Standard Linkage Calling Convention	RPC Connection or Direct RPC Connection	Use the *RPC Server for IMS* as RPC server.

Interface Type of your Server Program	EntireX Adapter Connection Type	Note
IMS MPP Message Interface (IMS Connect)	IMS Connect Connection	
COBOL Converter	COBOL Converter Connection	

II Using the IDL Extractor for COBOL - Overview

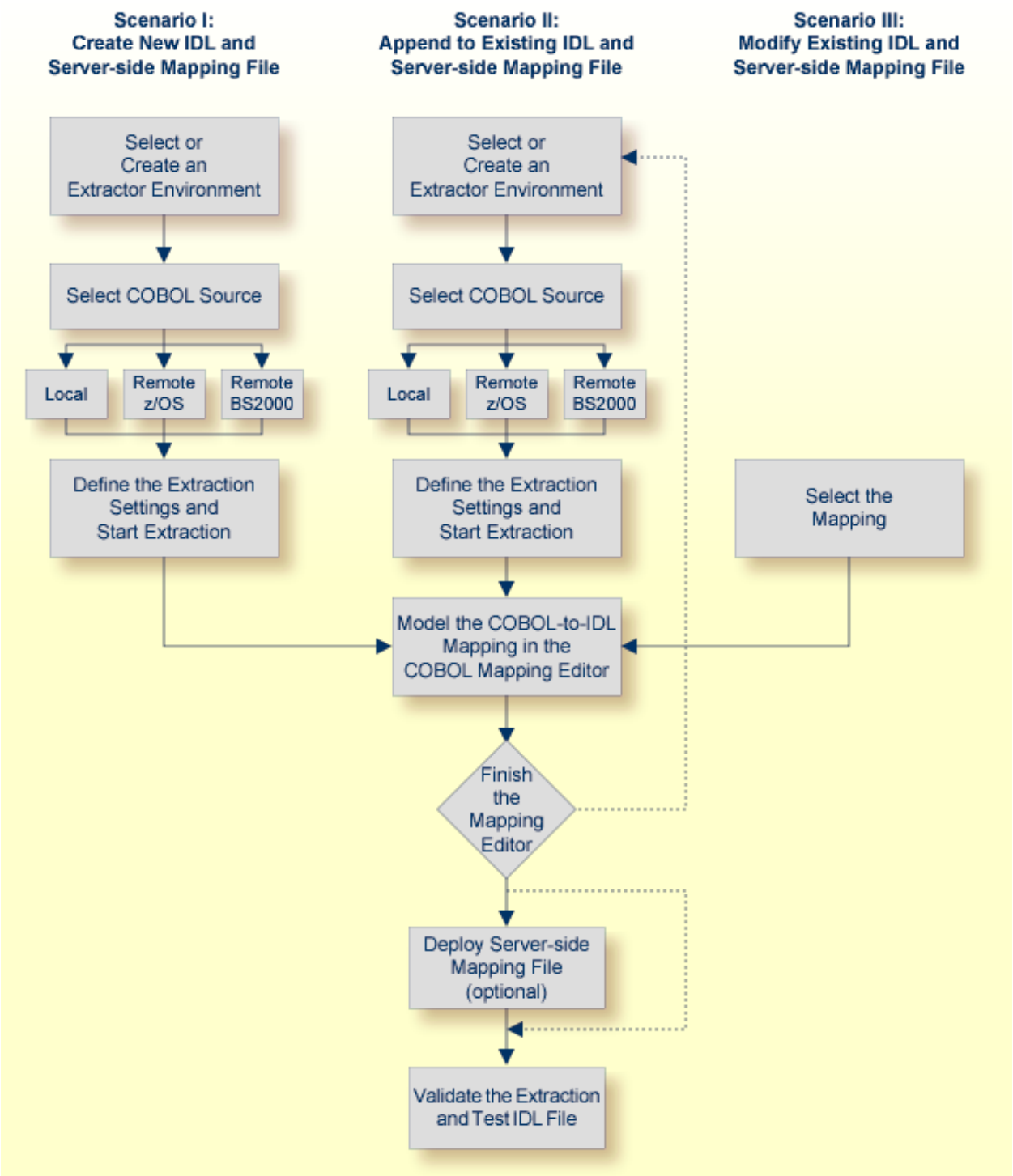
This chapter describes how to extract IDL from a COBOL source, using the IDL Extractor for COBOL, deploy, validate and test the extraction results. IDL extraction is supported by wizards, editors and generators.

Choosing a Scenario

The following scenarios are supported and are described in separate sections:

- *Scenario I: Create New IDL and Server Mapping Files*
- *Scenario II: Append to Existing IDL and Server Mapping Files*
- *Scenario III: Modify Existing IDL and Server Mapping Files*

See also *COBOL Mapping Editor*.


Scenario I:
Create New IDL and
Server-side Mapping File
Scenario II:
Append to Existing IDL and
Server-side Mapping File
Scenario III:
Modify Existing IDL and
Server-side Mapping File
Select or
Create an
Extractor Environment
Select or
Create an
Extractor Environment
Select COBOL Source
Select COBOL Source
Local
Remote
z/OS
Remote
BS2000
Local
Remote
z/OS
Remote
BS2000
Define the Extraction
Settings and
Start Extraction
Define the Extraction
Settings and
Start Extraction
Select the
Mapping
Model the COBOL-to-IDL
Mapping in the
COBOL Mapping Editor
Finish
the
Mapping
Editor
Deploy Server-side
Mapping File
(optional)
Validate the Extraction
and Test IDL File

Before Starting an Extraction

Before you start an extraction, we recommend you first clarify the following issues:

- The interface type of your COBOL program, see ***Supported COBOL Interface Types***.
- The input and output parameters of your COBOL server. Note the following:
  - COBOL REDEFINES are used in CICS as well as in batch servers. For all COBOL REDEFINES you have to clarify which redefine paths are the relevant ones for your extraction.
  - Particularly in CICS, the interface of a COBOL server is in most cases not described by the parameters given in the `PROCEDURE DIVISON` header. See ***`PROCEDURE DIVISION` Mapping*** and see `DFHCOMMAREA` examples under ***Programming Techniques***.
- We recommend you have a basic understanding of your COBOL server, especially if you can simplify your IDL with the following:
  - Map functions of the COBOL server to IDL programs.
  - Suppress unneeded fields.
  - Map COBOL data items to constants.

The COBOL sources can contain

- copybook references; see ***Copybooks*** under ***COBOL to IDL Mapping***
- CA Librarian (`-INC`) or CA Panvalet (`++INCLUDE`) control statements

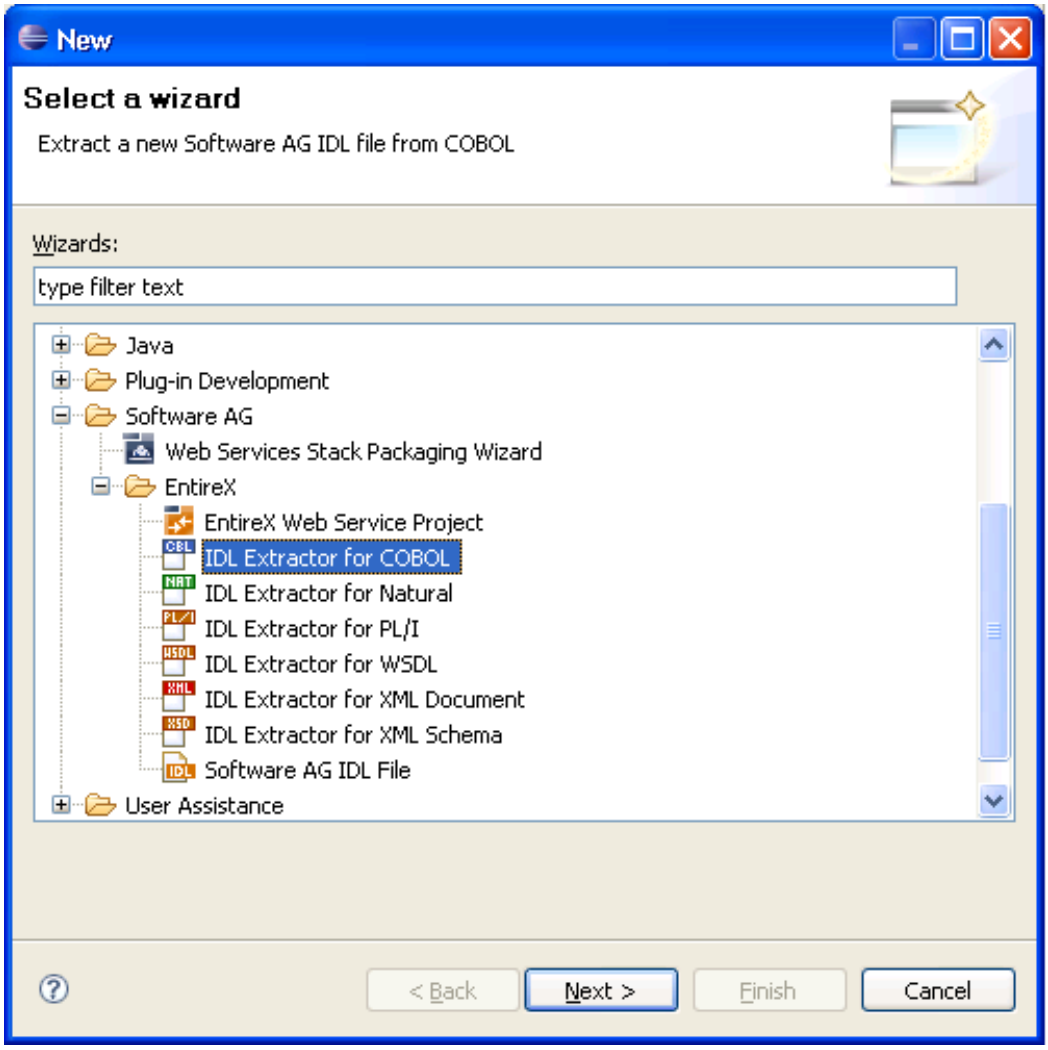
In section ***COBOL to IDL Mapping*** you will find information on how the COBOL syntax is mapped to Software AG IDL using this wizard and the Mapping Editor. We recommend you read this document because it describes possibilities and alternatives for handling COBOL syntax constructs.

Make sure the COBOL source

- can be compiled with no errors and no warning
- is written in COBOL fixed format, consisting of sequence number (column 1-6), indicator area (column7), area A, (column 8-11) and area B (column 12-72) for z/OS, BS2000, z/VSE and IBM i extractions

3 Scenario I: Create New IDL and Server Mapping Files

Step 1: Start the IDL Extractor for COBOL Wizard



To continue, press **Next** and continue with *Step 2: Select a COBOL Extractor Environment or Create a New One*.

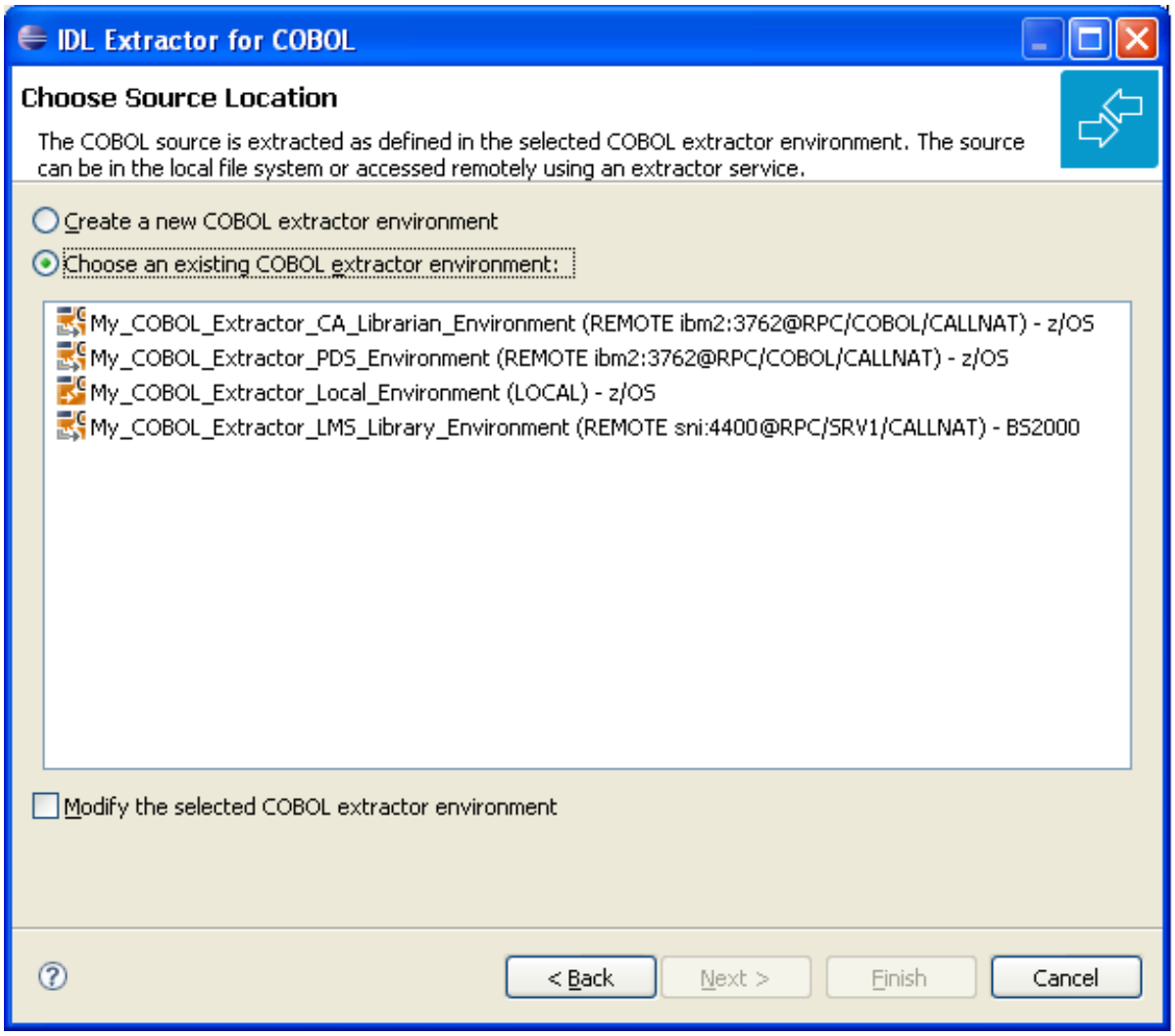
Step 2: Select a COBOL Extractor Environment or Create a New One

If no COBOL extractor environments are defined, you only have the option to create a new environment. An IDL Extractor for COBOL environment provides defaults for the extraction and refers to COBOL programs and copybooks that are

- stored locally on the same machine where the Designer is running: a *local* COBOL extractor environment

or

- stored remotely on a host computer: a *remote* COBOL extractor environment. The extractor service is required to access COBOL programs and copybooks remotely with a remote COBOL extractor environment. The extractor service is supported on platforms z/OS and BS2000. See *Extractor Service* in the RPC Server documentation for Batch | IMS | BS2000.



This page offers the following options:

To select an existing local COBOL extractor environment

1 Check radio button **Choose an existing COBOL extractor environment** and select a local COBOL extractor environment.

2 Continue with *Step 3: Select the COBOL Source* below.

To select an existing remote COBOL extractor environment

1 Check radio button **Choose an existing COBOL extractor environment** and select a remote COBOL extractor environment.

2 Continue with *Step 3: Select the COBOL Source* below.

To create a new local COBOL extractor environment

1 Check radio button **Create a new COBOL extractor environment**.

2 Follow the instructions in the Preferences section under *Create New Local Extractor Environment* (z/OS, BS2000, z/VSE and IBM i).

3 Continue with *Step 3: Select the COBOL Source* below.

To create a new remote COBOL extractor environment

1 Check radio button **Create a new COBOL extractor environment**.

2 Follow the instructions in the Preferences section under *Create New Remote Extractor Environment* z/OS | BS2000.

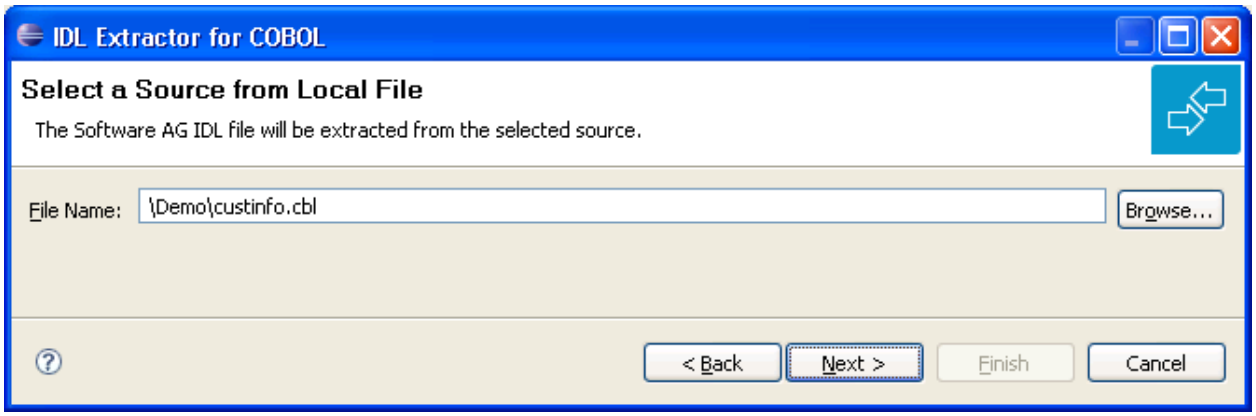
3 Continue with *Step 3: Select the COBOL Source* below.

Step 3: Select the COBOL Source

Selecting the COBOL source is different depending on whether the COBOL source is stored locally on the same machine where the Software AG Designer is running, or on a remote host computer.

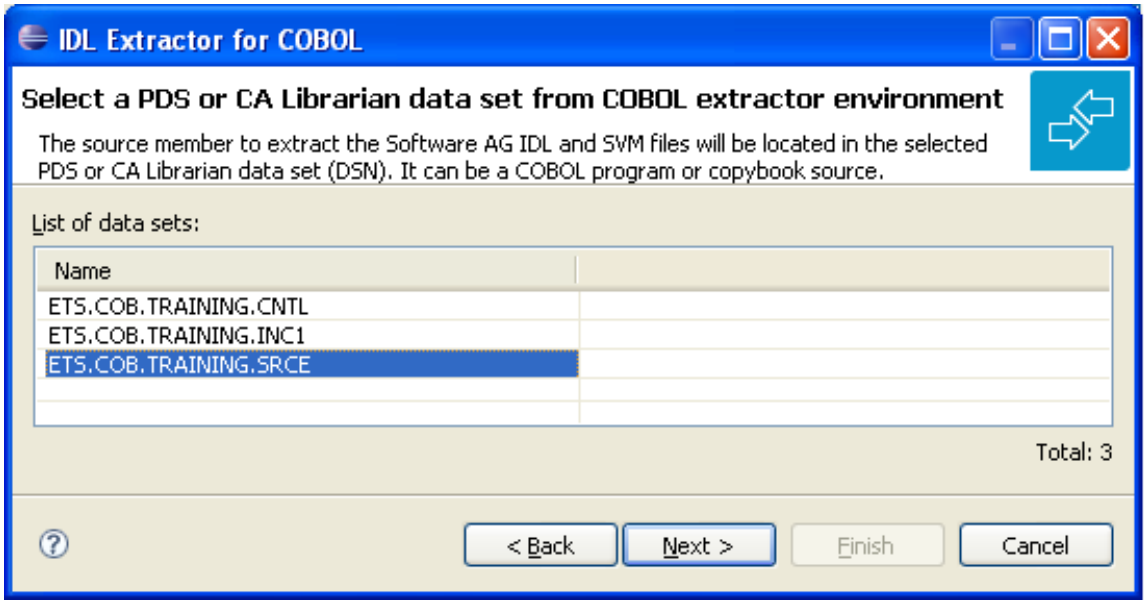
Selecting a COBOL Source Stored Locally

In step 2 above you selected or created a local extractor environment for z/OS. If you select a local COBOL extractor environment, you can browse for the COBOL program in the local file system. If you selected the COBOL source file before you started the wizard, and do not have a directory defined in the preferences of your Local Extractor Environment, the file location is already present. See *Create New Local Extractor Environment* (z/OS, BS2000, z/VSE and IBM i). To browse for the COBOL source file, choose **Browse**.



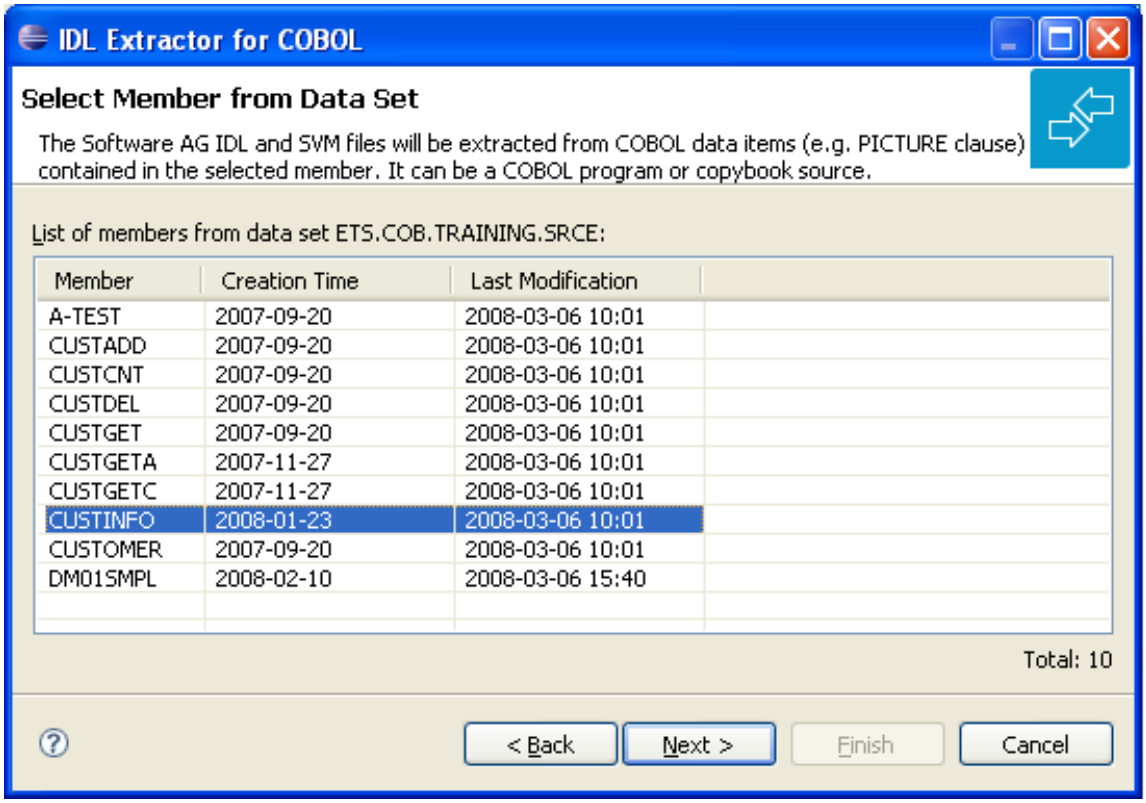
Selecting a Member from a Partitioned Data Set on Remote Host (z/OS)

In step 2 above you selected or created a remote extractor environment. The following page offers all data sets starting with the high-level qualifier defined in the **Filter Settings** of the remote extractor environment. See ***Creating a New Remote Extractor Environment (z/OS)*** under ***COBOL Preferences***.



Select the partitioned data set from which you want to extract and choose **Next**. Proceed depending on the selected data set type. See *Selecting a Member from a Partitioned Data Set on Remote Host (z/OS)*.

The following page offers all members contained in the partitioned data set selected in the previous step, starting with the member name prefix defined in the **Filter Settings** of the remote extractor environment. See *Define the remote extractor environment* under *COBOL Preferences*.

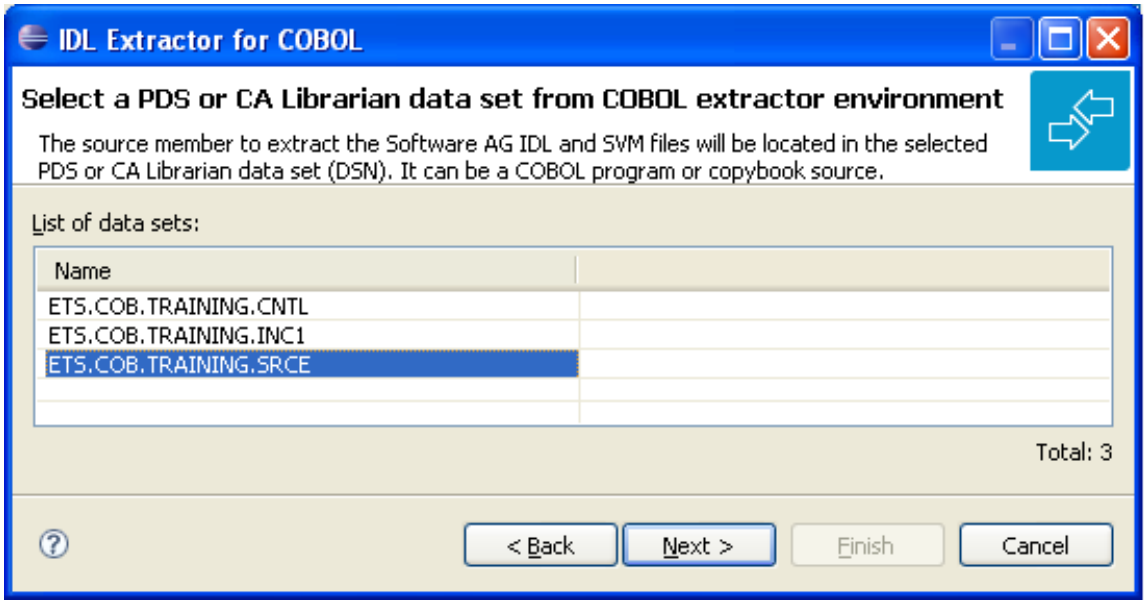


Select the member from which you want to extract. You can select only one COBOL source. The source can be a COBOL program or a COBOL copybook.

Choose **Next** and continue with *Step 4: Define the Extraction Settings and Start Extraction* below.

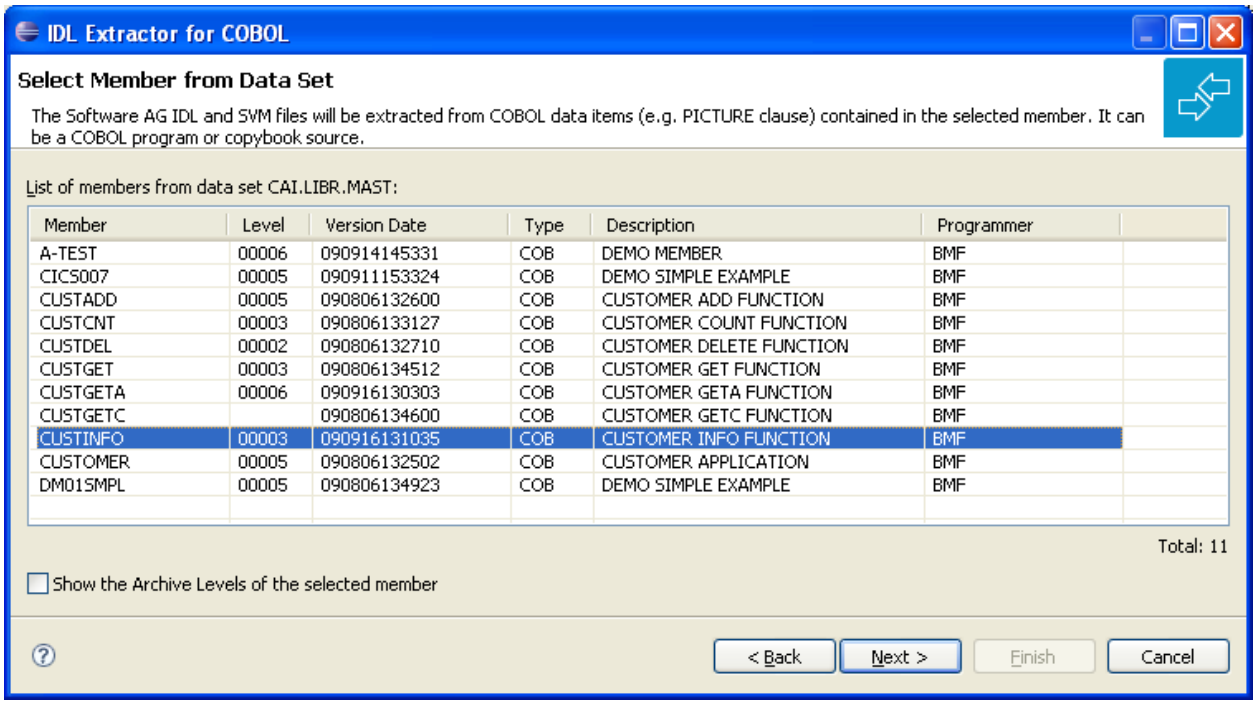
Selecting a Member from a CA Librarian Data Set on Remote Host (z/OS)

In step 2 above you selected or created a remote extractor environment. The following page offers all data sets starting with the high-level qualifier defined in the **Filter Settings** of the remote extractor environment. See *Creating a New Remote Extractor Environment (z/OS)* under *COBOL Preferences*.



Select the CA Librarian data set from which you want to extract and choose **Next**. Proceed depending on the selected data set type. See *Selecting a Member from a CA Librarian Data Set on Remote Host (z/OS)*.

The following page offers all members contained in the CA Librarian data set selected in the previous step, starting with the member name prefix defined in the **Filter Settings** of the remote extractor environment. See *Define the remote extractor environment* under *COBOL Preferences*.

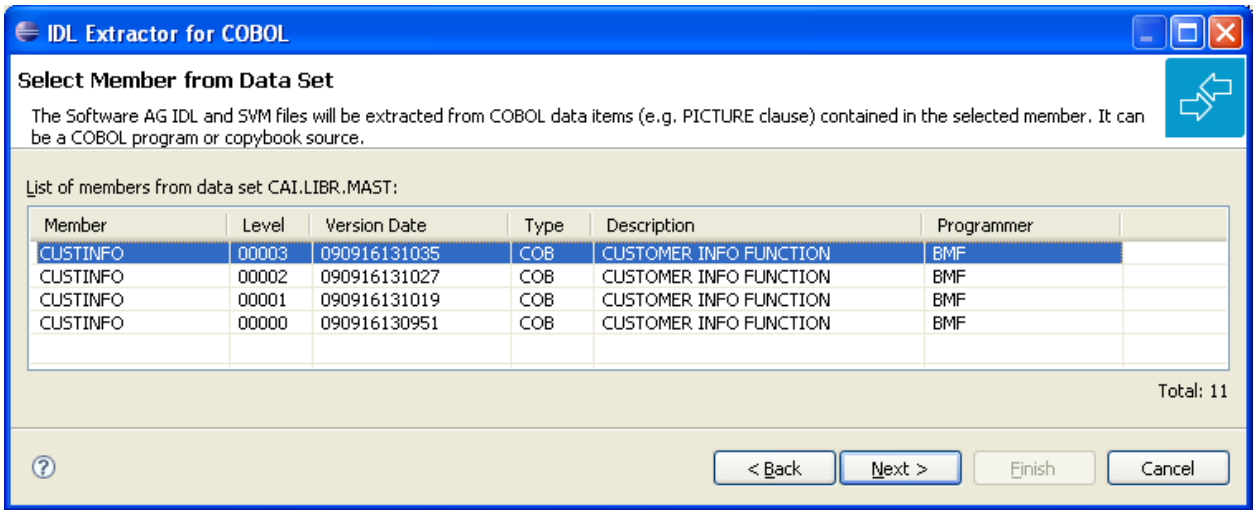


You can select only one COBOL source. The source can be a COBOL program or a COBOL copybook. If you want to extract from

- the latest (current) version of the member, select the member, choose **Next** and continue with *Step 4: Define the Extraction Settings and Start Extraction* below.
- a previous (archived) version of the member, check the box **Show the Archive Levels of the selected member**, select the member, choose **Next** and continue with *Selecting a Member Archive Level from a CA Librarian Data Set on Remote Host (z/OS)*.

Selecting a Member Archive Level from a CA Librarian Data Set on Remote Host (z/OS)

The following page offers all archive levels of the previously selected member.

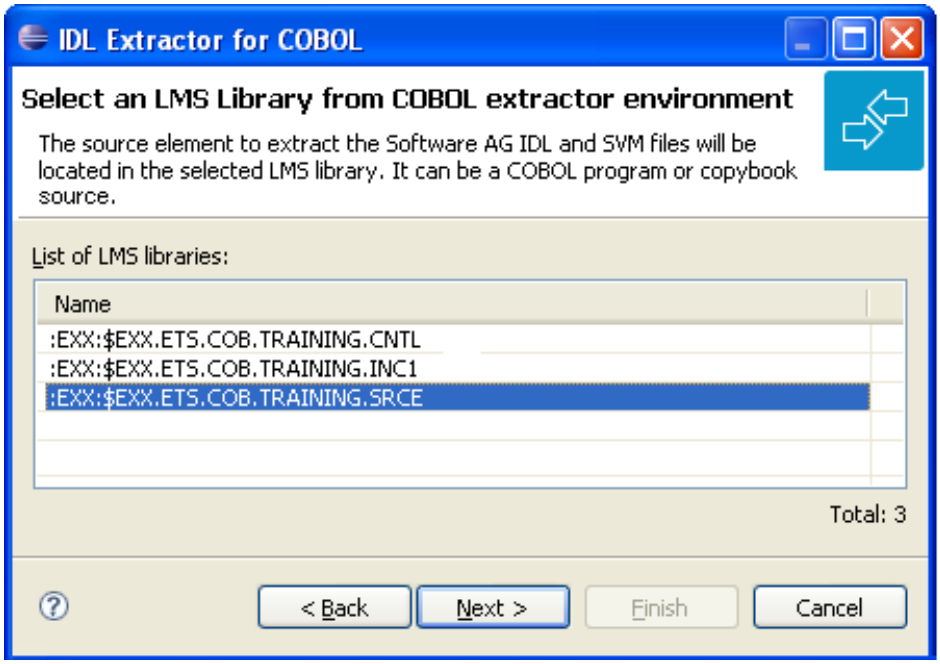


Select the member from which you want to extract. You can select only one archive level. Choose **Next** and continue with *Step 4: Define the Extraction Settings and Start Extraction* below.

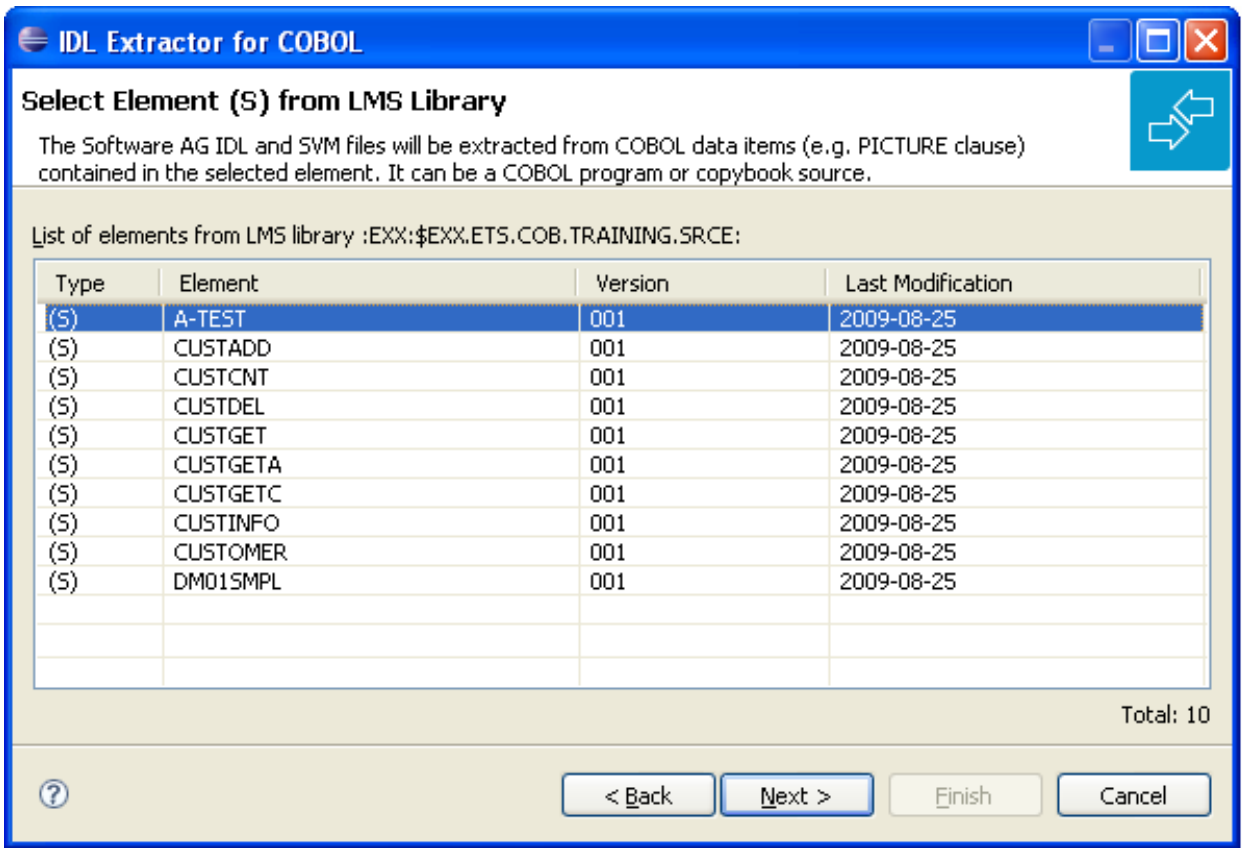
Selecting an Element (S) from an LMS Library on Remote Host (BS2000)

In step 2 above you selected or created a remote extractor environment.

The following page offers all data sets starting with the high-level qualifier defined in the **Filter Settings** of the remote extractor environment. See *Creating a New Remote Extractor Environment (BS2000)* under *COBOL Preferences* .



The following page offers all elements contained in the LMS library selected in the previous step, starting with the member name prefix defined in the Filter Settings of the remote extractor environment. See *Define the remote extractor environment* under *COBOL Preferences*.



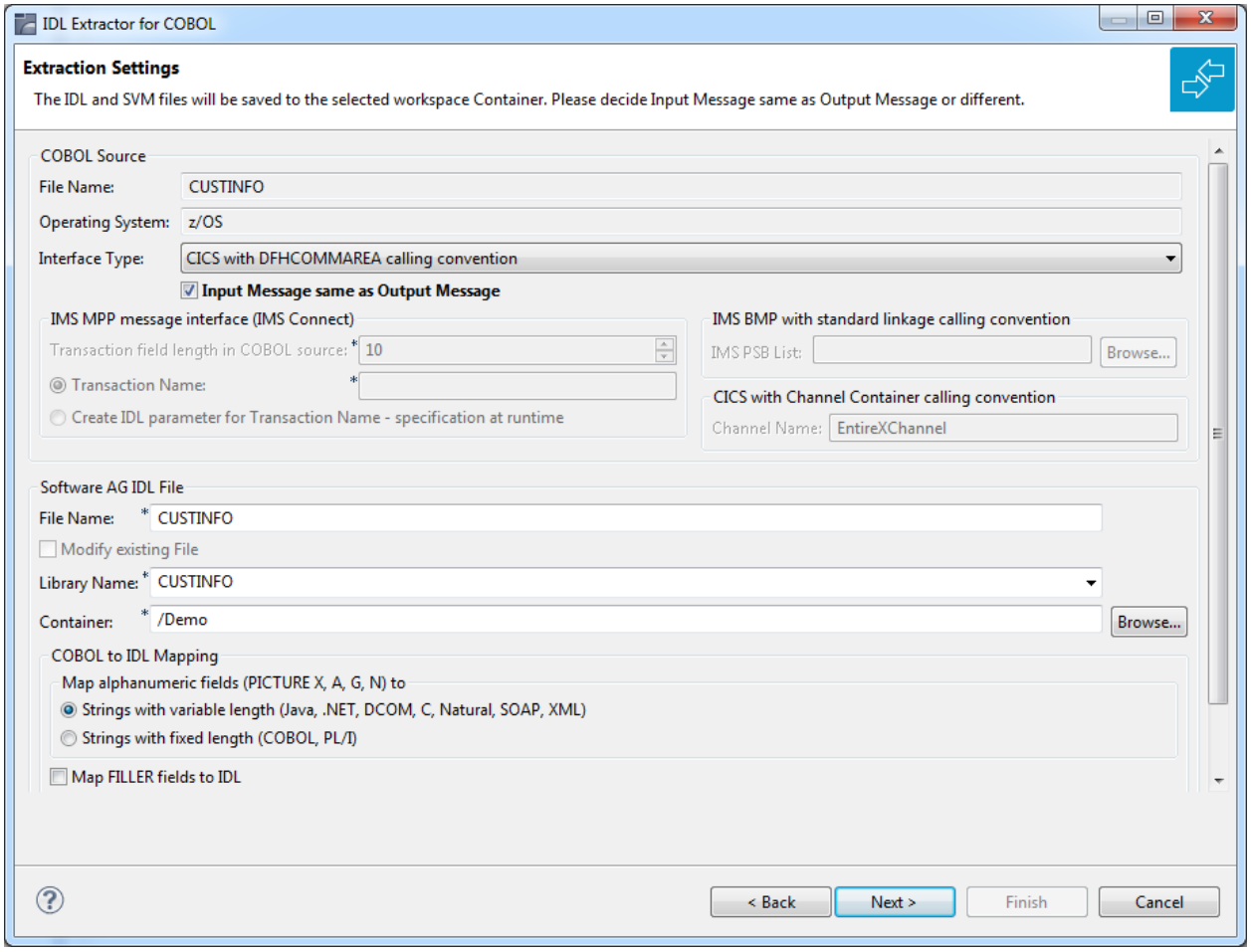
Select the element from which you want to extract. You can select only one COBOL source. The source can be a COBOL program or a COBOL copybook.

Choose **Next** and continue with *Step 4: Define the Extraction Settings and Start Extraction* below.

Step 4: Define the Extraction Settings and Start Extraction

In this page you specify the COBOL source and Software AG IDL target options used for IDL extraction.

- Operating System
- Interface Type
- Software AG IDL File
- COBOL to IDL Mapping



Operating System

The operating system is already defined in the extractor environment in the IDL Extractor for COBOL preferences, see *COBOL Preferences*.

Interface Type

The interface type must match the type of your COBOL server program. It is used by the RPC server and the EntireX Adapter at runtime to correctly call the COBOL server and must be a supported interface type of the EntireX runtime component used. See *Compatibility between COBOL Interface Types and RPC Server*.

Additional information may be required depending on the interface type:

- **CICS with DFHCOMMAREA Calling Convention**
  Specify **Input Message same as Output Message**. If the COBOL server program uses a different COBOL output data structure compared to its input data structure, that is, the input message layout is overlaid with another layout on output, you need to *uncheck* **Input Message same as Output Message**. See the following COBOL server examples:
  - *Example 1: Redefines*
  - *Example 2: Buffer Technique*
  - *Example 3: COBOL `SET ADDRESS` Statements*

  If the COBOL server program uses the same COBOL data structure on input as well as on output, you need to *check* **Input Message same as Output Message**. See the following COBOL server examples:
  - *Example 1: Redefines*
  - *Example 2: Buffer Technique*
  - *Example 3: COBOL `SET ADDRESS` Statements*
- **CICS with Channel Container Calling Convention**
  Optionally, specify a channel name. See *Extracting from a CICS Channel Container Program*.
- **CICS with DFHCOMMAREA Large Buffer Calling Convention**
  Specify **Input Message same as Output Message**. If the COBOL server program uses a different COBOL large output buffer data structure compared to its large input buffer data structure, you need to *uncheck* **Input Message same as Output Message**. See *CICS with `DFHCOMMAREA` Large Buffer Interface* (In same as Out, In different to Out).
- **COBOL Converter**
  Specify **Input Message same as Output Message**. If a different COBOL output data structure compared to its input data structure is used (that is, the input message layout is overlaid with another layout on output) you need to *uncheck* **Input Message same as Output Message**. See *COBOL Converter* (In same as Out, In different to Out).
- **IMS MPP Message Interface (IMS Connect)**
  Specify how you want the transaction name to be determined. See *Extracting from an IMS MPP Message Interface Program*.

- **IMS BMP with Standard Linkage Calling Convention**
  You can optionally set the **IMS PSB List**. See *Extracting from an IMS BMP Standard Call Interface*.
- **Batch with Standard Linkage Convention**
  No further information is required.

For an introduction to interface types, see *Supported COBOL Interface Types*.

Software AG IDL File

With the Software AG IDL file target options you specify the IDL file and IDL library names used:

- **File name** specifies the file name used by the operating system.
- **Modify existing file** is enabled only when the IDL file already exists. If enabled, check this option to continue the extraction.
- **Library name** defines the IDL library name used in the IDL file. The dialog box cannot be edited when you modify an existing IDL file. If there are multiple libraries, you can select one of these; if there is only one library, the box is disabled. When you extract the IDL the first time or you specify the name of an existing IDL file, the box can be edited (like a text widget). If you specified an existing IDL file, the currently existing library names are available in the box.
- **Container** specifies the eclipse container used for the IDL file

COBOL to IDL Mapping

With these target options you specify how COBOL data items are mapped to IDL. You can turn fixed length COBOL data types into variable length data types. This is useful if connecting COBOL to endpoints with a concept of string types - such as Java, .NET, C, XML, Web services etc. Real strings without trailing blanks are received. It also reduces the messages size of RPC requests.

- With a mapping to **Strings with variable length** (1), the transfer of data in the RPC data stream depends on the actual length of the data and not the field size, as seen in COBOL. For the COBOL side, the actual content length of such fields is determined using a trim mechanism.

  For `PIC X`, `A` and `G`, all trailing `SPACE`s are ignored before send. After receive, the content is padded with trailing `SPACE`s up to the COBOL field size.

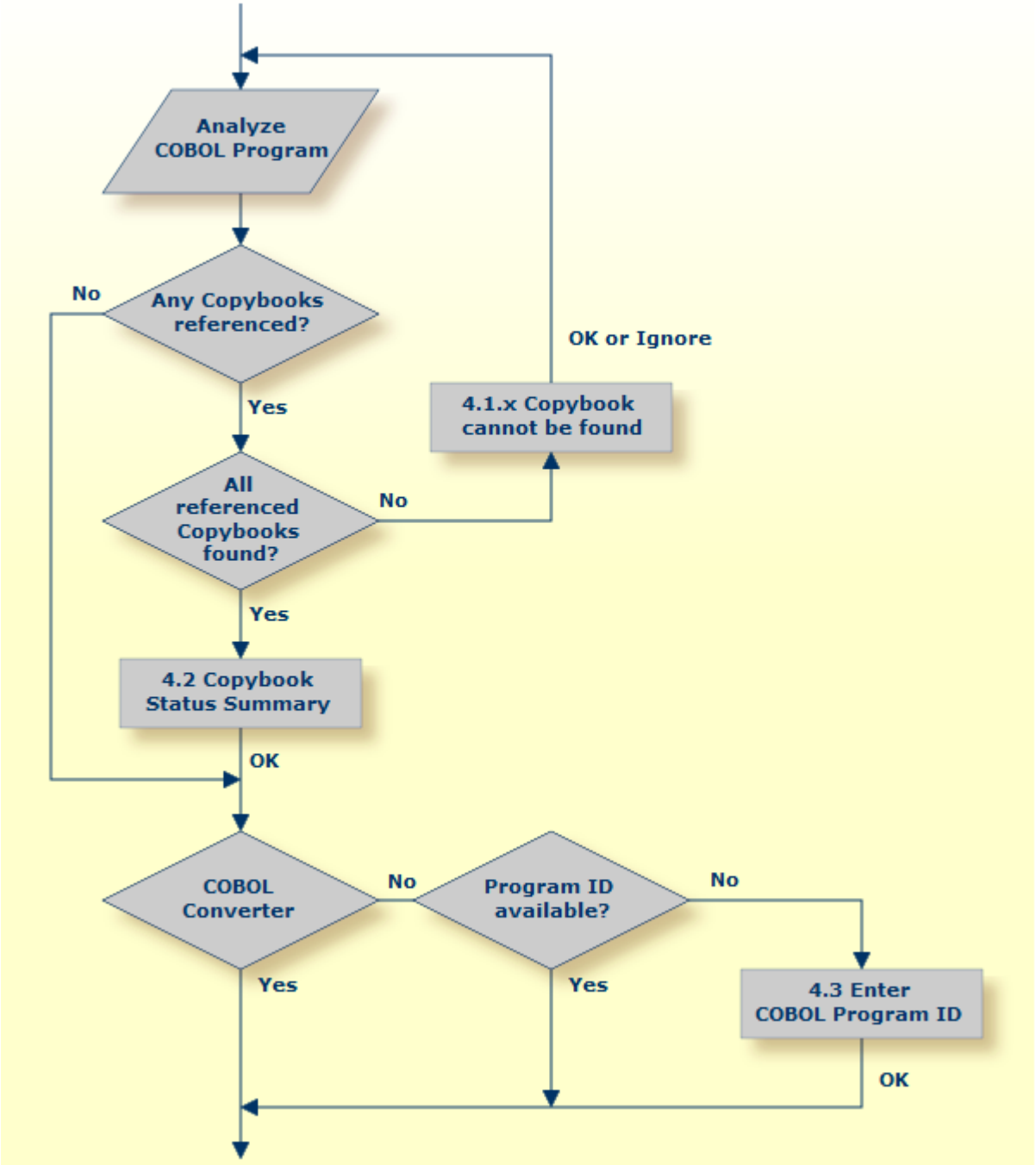
  For `PIC N` (3), the Unicode code point U+0020 is used for trimming and padding.

  If your application relies on trailing `SPACE`s or Unicode code points U+0020, you cannot use a mapping to strings with variable length (1). Use strings with fixed length (2) instead.
- With a mapping to **Strings with fixed length** (2), no trimming takes place. If the mapping in the calling endpoint calling COBOL is a variable length string data type, in most cases you will receive trailing `SPACE`s or trailing Unicode code points U+0020 respectively.
- Check the box **Map FILLER fields to IDL** if COBOL `FILLER` pseudo-parameters are to be part of the RPC client interface. By default they are not mapped to IDL.

Notes:

1. Technically, a mapping to **Strings with variable length** forces IDL types `AV`*n*, `KV`*n* or `UV`*n* to be extracted. See also the notes under *IDL Data Types* in the IDL Editor documentation.
2. Technically, a mapping to **Strings with fixed length** the IDL types `A`*n*, `K`*n* or `U`*n*. See also the notes under *IDL Data Types* in the IDL Editor documentation.

Choose **Next** and start the extraction. The wizard now analyzes the COBOL program. During this process the following situations are possible:

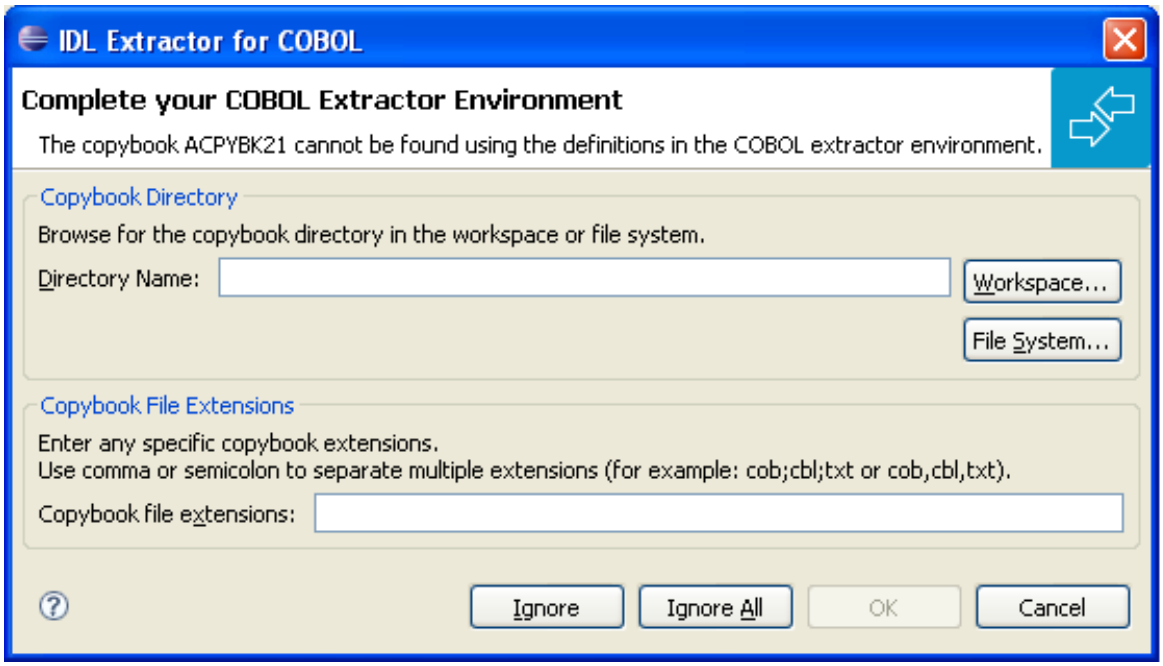


- Referenced copybooks cannot be found. In this case the wizard prompts you for every missing copybook. Continue with optional step *Step 4.1x: Copybook Cannot be Found* - Local Extraction | Remote Extraction (z/OS) | Remote Extraction (BS2000) depending on your situation.
- If referenced copybooks are not available, you can choose **Ignore** or **Ignore All**, a copybook status summary page is displayed, see *Step 4.2: Copybook Status Summary (Optional)*.

- No COBOL program ID can be located if you extract, for example, from a copybook that contains COBOL data items only. In this case, the wizard prompts you to enter the COBOL program ID. Continue with *Step 4.3: Enter COBOL Program ID (Optional)*.
- There is no copybook reference in your COBOL source or all referenced copybooks are found. Also the COBOL program ID can be located or is not needed as for interface type *COBOL Converter*. In this case continue with *Step 5: Select the COBOL Interface and Map to IDL Interface* under *Scenario I: Create New IDL and Server Mapping Files*.

Step 4.1a: Copybook Cannot be Found - Local Extraction

This dialog enables you to browse directories where missing copybooks might be found. If there are any specific copybook file extensions, you can define them here.



The copybook that cannot be found is given in the window, here its name is "ACPYBK21". In the extractor Preferences, the copybook directory that contains the copybook or the copybook file extension is not defined.

Continue with one of the following actions:

To ignore this copybook only

1 Choose **Ignore** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To ignore this and all further copybooks

1 Choose **Ignore All** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To complete the extractor environment

1 Choose **Workspace** or **File System** to browse for the copybook directory.

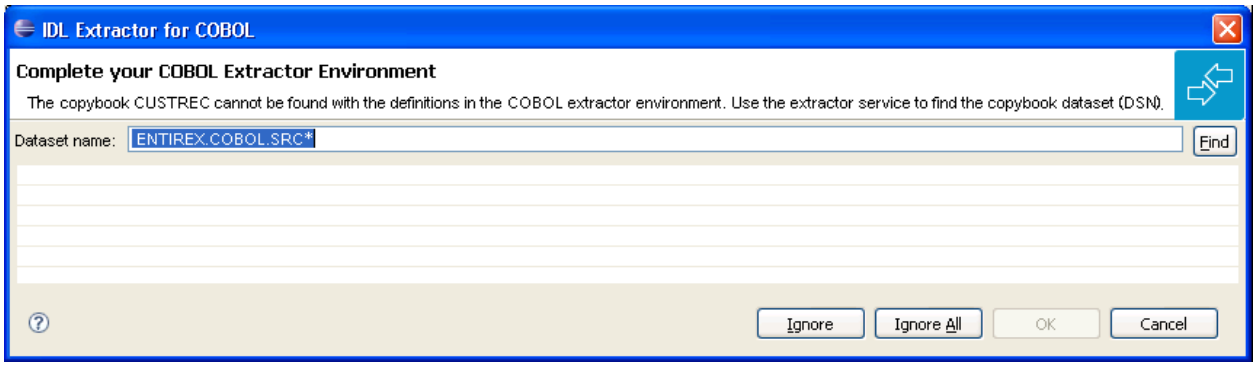
2 Check the copybook file extensions. Both will be saved in the COBOL extractor preferences and reused in further extractions.

3 Choose **OK** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

4 Choose **Next** to start extraction again.

Step 4.1b: Copybook Cannot be Found - z/OS Remote Extraction

This dialog enables you to browse remote locations (partitioned or CA Librarian data sets) where missing copybooks might be found.



The copybook that cannot be found is given in the window; here its name is "CUSTREC". In the extractor preferences, the copybook data set that contains the copybook is not defined.

Continue with one of the following choices:

To ignore this copybook only

1 Choose **Ignore** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To ignore this and all further copybooks

1 Choose **Ignore All** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To complete the extractor environment

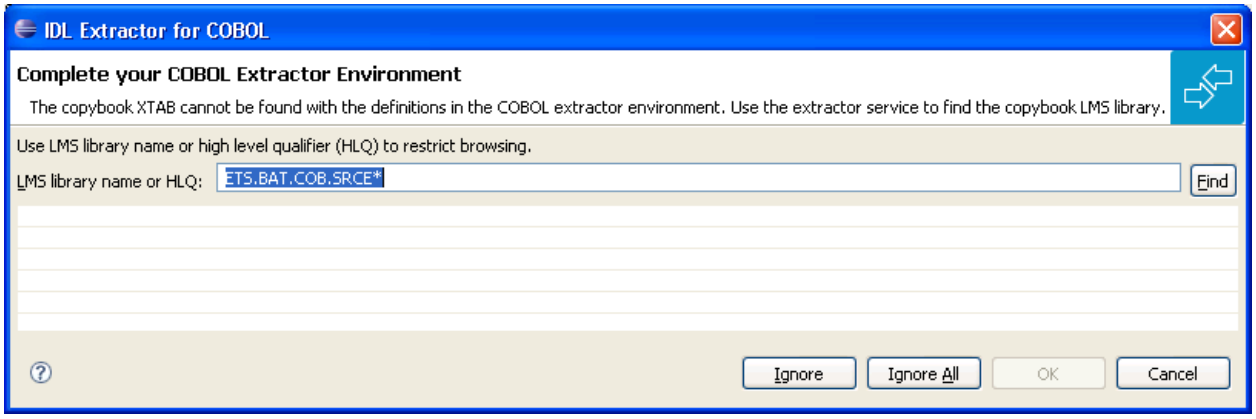
1 Choose **Find** to browse for the copybook data set. It will be saved in the COBOL extractor preferences and reused in further extractions.

2 Choose **OK** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

3 Choose **Next** to start extraction again.

Step 4.1c: Copybook Cannot be Found - BS2000 Remote Extraction

This dialog enables you to browse remote locations (LMS libraries) where missing copybooks might be found.



The copybook that cannot be found is given in the window; here its name is "XTAB". In the extractor preferences, the copybook LMS library that contains the copybook is not defined.

Continue with one of the following choices:

To ignore this copybook only

1 Choose **Ignore** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To ignore this and all further copybooks

1 Choose **Ignore All** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

2 Choose **Next** to start extraction again.

To complete the extractor environment

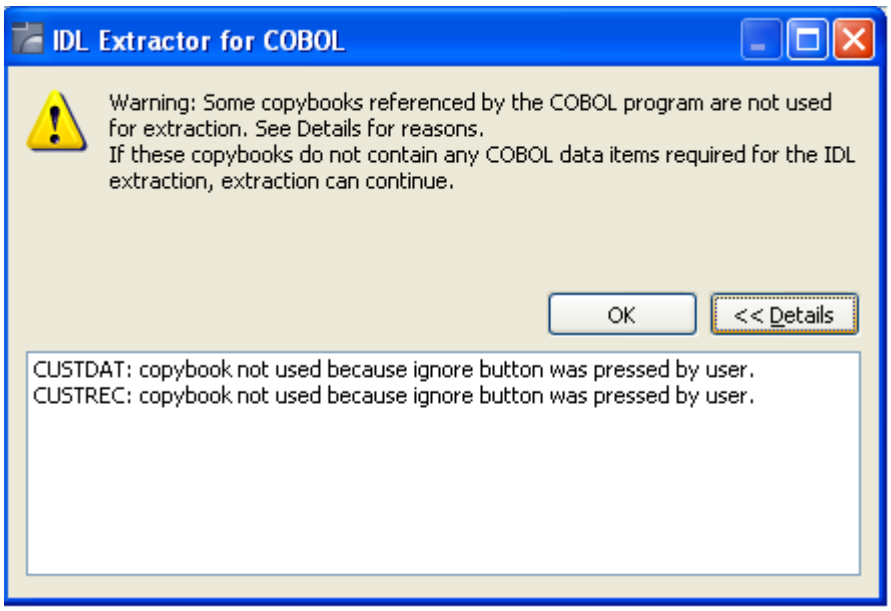
1 Choose **Find** to browse for the copybook LMS library. It will be saved in the COBOL extractor preferences and reused in further extractions.

2 Choose **OK** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.

3 Choose **Next** to start extraction again.

Step 4.2: Copybook Status Summary (Optional)

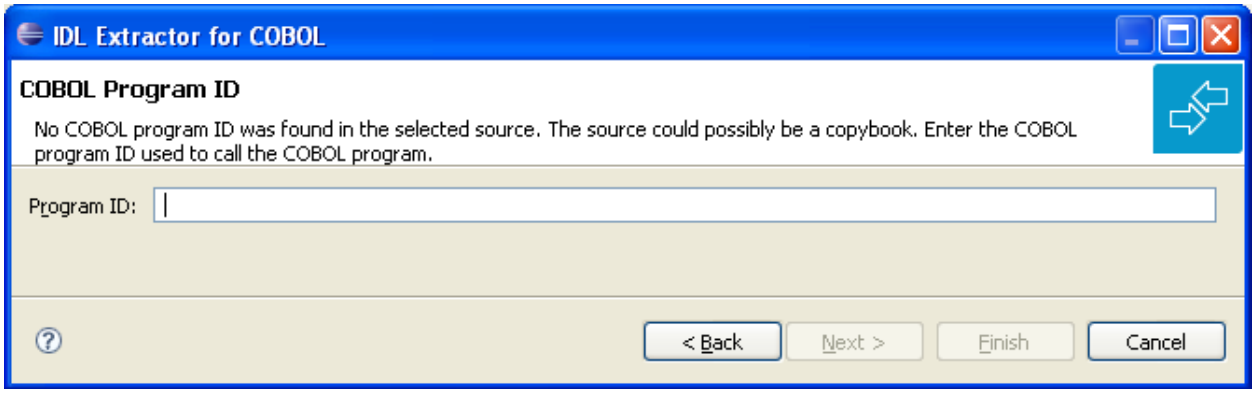
This summary page lists all COBOL copybooks which were not available during extraction.



- If any relevant COBOL data item describing the server interface is contained in one of the listed copybooks, you cannot continue. Terminate the extraction and try to get the missing copybooks.
- If no relevant COBOL data item describing the server interface is contained in the copybooks, you can continue. Choose **OK**.

Step 4.3: Enter COBOL Program ID (Optional)

This page is shown whenever the program ID of the COBOL source is missing. Entering a COBOL program name is compulsory.



No COBOL program ID can be located if you extract, for example, from a copybook that contains COBOL data items only. The COBOL program ID

- is the COBOL program name
- is often the name of the executable or load module
- can be found in the `IDENTIFICATION DIVISION` (abbreviated to`"ID"`). Example

```
ID DIVISION.
PROGRAM-ID.   CUSTINFO.
AUTHOR.       BMF.
DATE-WRITTEN. 26-11-1996
```

To complete the extraction

1 Enter the COBOL program ID.

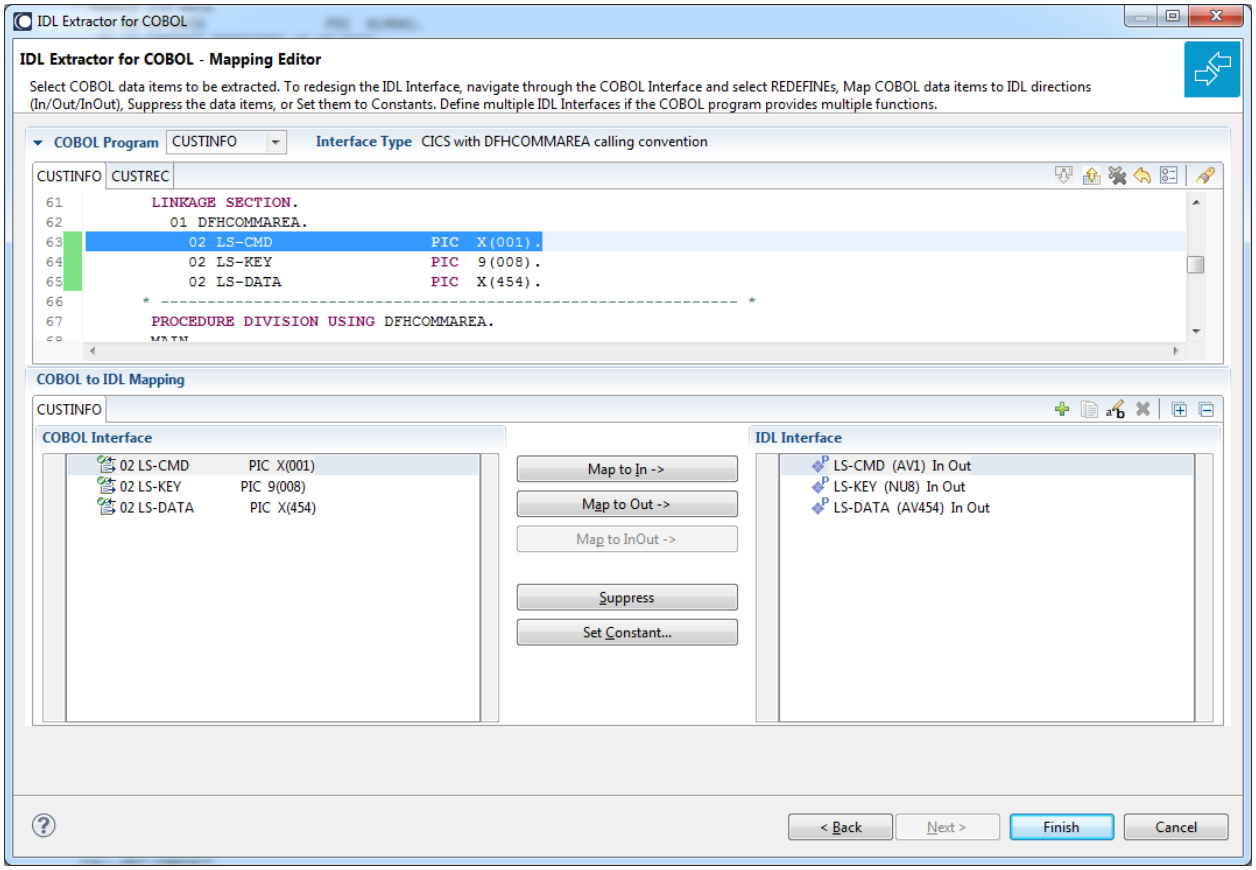
2 Choose **OK** to continue with *Step 5: Select the COBOL Interface and Map to IDL Interface*.

Step 5: Select the COBOL Interface and Map to IDL Interface

A COBOL source program mostly does not contain all the information needed for IDL mapping. With the Mapping Editor you enter this missing information. In general, mapping the COBOL data items to IDL with the Mapping Editor is a two-step process:

1. First, select the COBOL data items of the COBOL interface.
2. Then map the COBOL interface to the IDL interface. Define
   - which COBOL data items are mapped to IDL (**Select REDEFINE paths**, **Suppress Unneeded COBOL Data Items**)
   - the direction of the COBOL data items (**Map to [In, Out, InOut]**)
   - field values for COBOL data items that are not sent by clients to the COBOL server (**Set COBOL Data Items to Constant**)
   - COBOL server with multiple functions (**Map to Multiple IDL Interfaces**)
   - COBOL server output depends on COBOL input (**Map to Multiple IDL Interfaces**)
   - COBOL server with conditional output (**Set Multiple Possible Output (MPO) Structures**)
   - COBOL table usage (**Set Array Mapping (fixed <-> unbounded)**)
   - COBOL data items mapped to binary (**Map to Binary, Revert Binary Mapping**)
   - etc.

See the guidelines on *IDL Extraction per Interface Type* for the *COBOL Mapping Editor* or by COBOL syntax in *User-defined Mapping* under *COBOL to IDL Mapping* for further information on this important extraction step.



The outcome of the Mapping Editor is the IDL file and a server mapping file (optional). See *When is a Server Mapping File Required?* under *Server Mapping Files for COBOL* in the Designer documentation. Both files are written with the file name entered for the IDL file in ***Step 4: Define the Extraction Settings and Start Extraction***.

Step 6: Finish the Mapping Editor

When you choose **Save** in the Mapping Editor, the IDL file is generated. If required, a server mapping file (.cvm) is generated, too.

Step 7: Validate the Extraction and Test the IDL File

The IDL file is used to build RPC clients using an EntireX Wrapper of your choice, or an IS adapter service using the *Integration Server Wrapper*.

If a server mapping file (.cvm) is extracted:

- You need to rebuild all existing RPC clients communicating with this RPC server program and re-generate the client interface objects.
- For existing IS adapters generated with the *EntireX Adapter* need to be updated. See *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

Caution: Do not edit the IDL file manually or with the IDL Editor, except for changing parameter names. Otherwise, consistency between the IDL file and the server mapping file will be lost, resulting in unexpected behavior. For this purpose use the COBOL Mapping Editor instead and choose ***Scenario III: Modify Existing IDL and Server Mapping Files***.

Caution: A server mapping file extracted this way must not be re-created by the COBOL Wrapper. Server mapping specifications of such a file would not be powerful enough to adequately describe your COBOL server program extracted here.

If you are using the RPC Server for CICS, before calling your extracted RPC server, check if you need to alter

- CICS settings, for example `TWASIZE`. See *CICS Settings*.
- For z/OS additionally *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDs etc.

For a quick validation of your extraction (all interface types except ***COBOL Converter***) you can

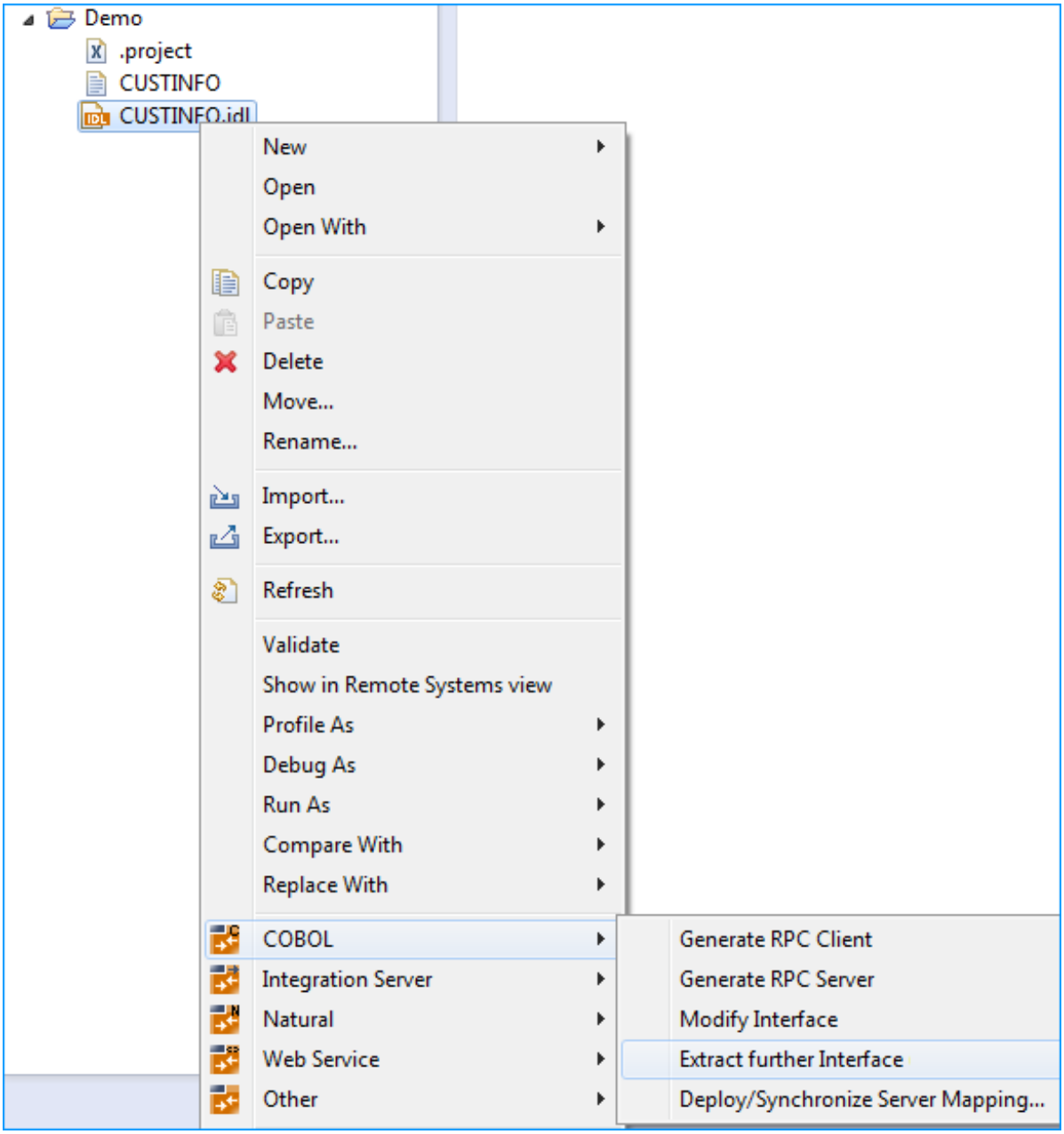
- use the IDL Tester to validate the extraction, see *EntireX IDL Tester* in the Designer documentation.
- generate an XML mapping file (XMM) and use the EntireX XML Tester for verification. See *EntireX XML Tester* in the XML/SOAP Wrapper documentation.

4 Scenario II: Append to Existing IDL and Server Mapping Files

The IDL Extractor for COBOL can be started from an existing pair of IDL and server mapping files. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation.

To start the IDL Extractor for COBOL

- Open the context menu of an IDL file and choose **COBOL > Extract further Interface**.



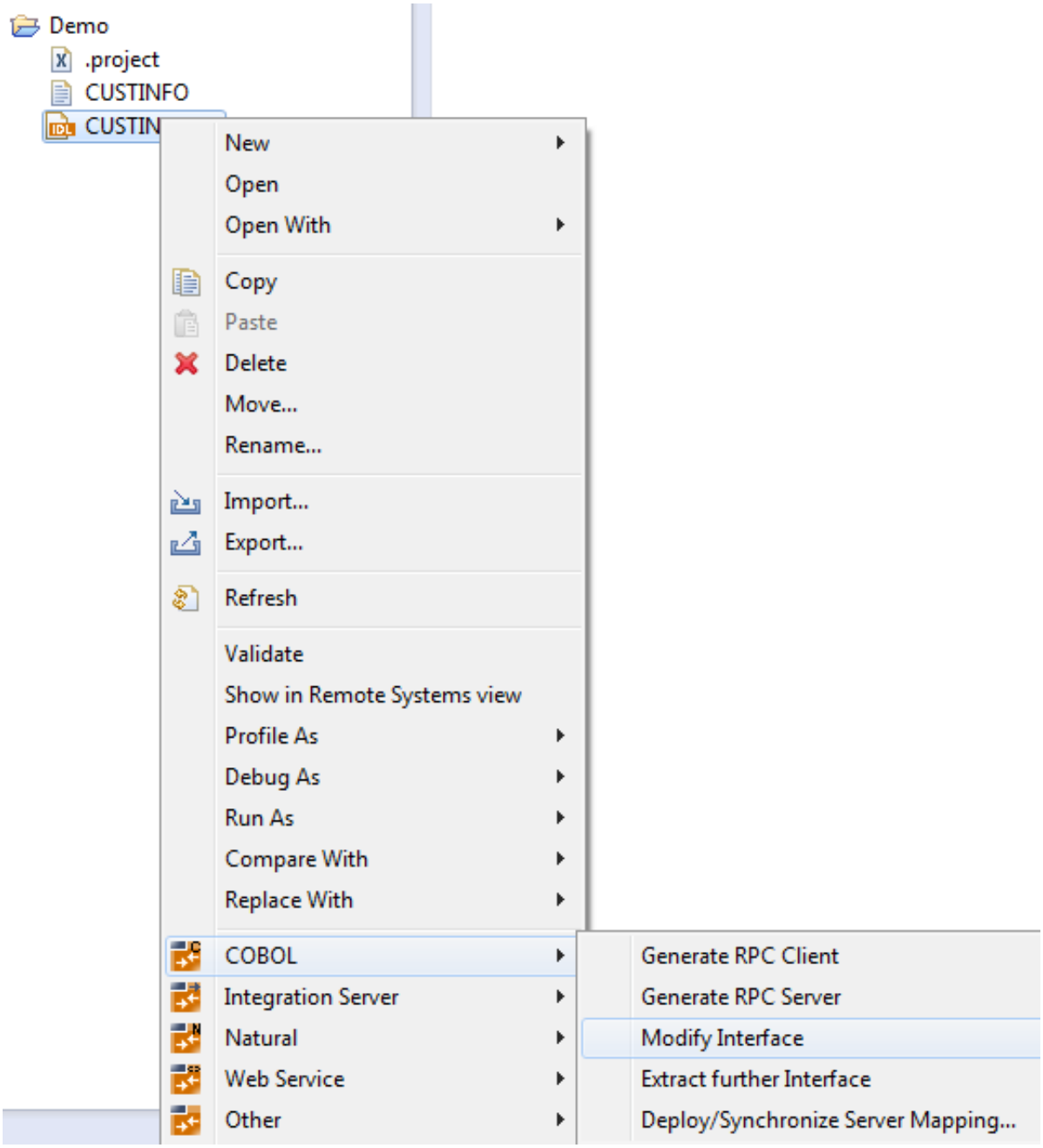
Continue with *Step 2: Select a COBOL Extractor Environment or Create a New One* as described under *Scenario I: Create New IDL and Server Mapping Files*.

5 Scenario III: Modify Existing IDL and Server Mapping Files

The IDL Extractor for COBOL can be started from an existing pair of IDL and server mapping files. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation.

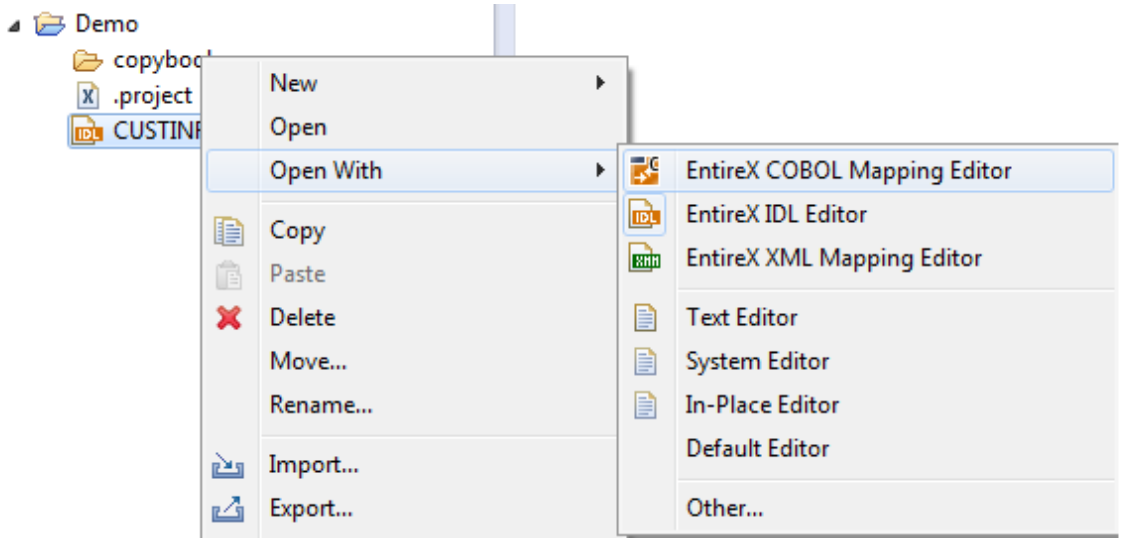
To start the COBOL Mapping Editor

- Open the context menu of an IDL file and choose **COBOL > Modify Interface**.



Or:

Choose **Open With > EntireX COBOL Mapping Editor**.



Continue with *Step 5: Select the COBOL Interface and Map to IDL Interface* as described under *Scenario I: Create New IDL and Server Mapping Files*.

III COBOL Mapping Editor

See also *User-defined Mapping* under *COBOL to IDL Mapping* for guidelines on IDL extraction by COBOL syntax.

Introduction

A COBOL source program mostly does not contain all the information needed for IDL mapping. With the Mapping Editor you enter this missing information. In general, mapping the COBOL data items to IDL with the Mapping Editor is a two-step process:

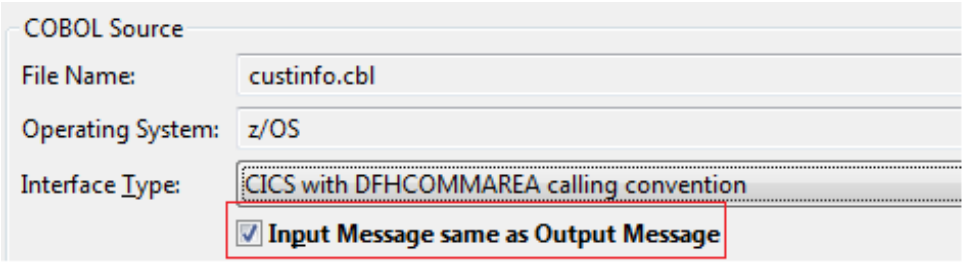
1. First, select the COBOL data items of the COBOL interface.
2. Then map the COBOL interface to the IDL interface. Define
   - which COBOL data items are mapped to IDL (**Select REDEFINE paths**, **Suppress Unneeded COBOL Data Items**)
   - the direction of the COBOL data items (**Map to [In, Out, InOut]**)
   - field values for COBOL data items that are not sent by clients to the COBOL server (**Set COBOL Data Items to Constant**)
   - COBOL server with multiple functions (**Map to Multiple IDL Interfaces**)
   - COBOL server output depends on COBOL input (**Map to Multiple IDL Interfaces**)
   - COBOL server with conditional output (**Set Multiple Possible Output (MPO) Structures**)
   - COBOL table usage (**Set Array Mapping (fixed <-> unbounded)**)
   - COBOL data items mapped to binary (**Map to Binary, Revert Binary Mapping**)
   - etc.

IDL Extraction per Interface Type

The following table provides guidelines on IDL extraction per interface type. See *Supported COBOL Interface Types*. For the CICS interface types DFHCOMMAREA and DFHCOMMAREA Large Buffer, the guidelines distinguish further between

- COBOL server programs overlaying the input data structure with a different output data structure, and
- COBOL server programs using same structures on input and output.

You already selected this in the checkbox **Input Message same as Output Message** in *Step 4: Define the Extraction Settings and Start Extraction*:



Environment	Interface Type	CICS Message on Input and Output
CICS	DFHCOMMAREA (3)	same (1,4)
		different (2,5)
	Large Buffer	same (1)
		different (2)
	Channel Container	
webMethods Integration Server	COBOL Converter (for use by EntireX Adapter)	same (1)
		different (2)
Batch	Standard Linkage	
IMS	BMP with Standard Linkage	
	MPP Message Interface (IMS Connect)	

Notes:

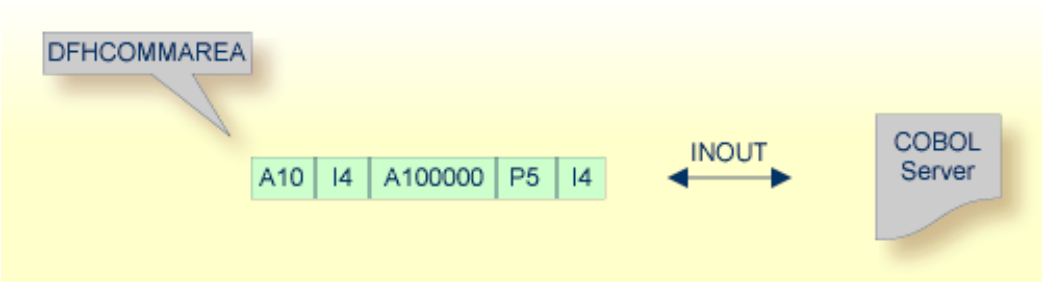
1. Checkbox **Input Message same as Output Message** in *Step 4: Define the Extraction Settings and Start Extraction* is checked. The COBOL data structure of the input message is the same as the structure of the output message (applies to CICS or COBOL Converter).
2. Checkbox **Input Message same as Output Message** in *Step 4: Define the Extraction Settings and Start Extraction* is cleared. The COBOL data structure of the input message is different to

the structure of the output message (that is, the output overlays the input; applies to CICS or COBOL Converter).

3. Your DFHCOMMAREA COBOL server must be DPL-enabled to be directly supported by EntireX. The distributed program (DPL) link function enables a CICS client program to call another CICS program (the server program) in a remote CICS region. Technically, a COBOL server is DPL-enabled if
   - CICS is able to call the COBOL server remotely
   - the DFHCOMMAREA layout does *not* contain pointers.

   If your program is not DPL-enabled, see ***What to do with other Interface Types?*** in ***Introduction to the IDL Extractor for COBOL***.
4. See the following COBOL server examples for CICS input message *the same as* CICS output message:
   - ***Example 1: Redefines***
   - ***Example 2: Buffer Technique***
   - ***Example 3: COBOL SET ADDRESS Statements***
5. See the following COBOL server examples for CICS input message *different to* CICS output message:
   - ***Example 1: Redefines***
   - ***Example 2: Buffer Technique***
   - ***Example 3: COBOL SET ADDRESS Statements***

6 CICS with DFHCOMMAREA Calling Convention - In same as Out



Introduction

Depending on the programming style used in the CICS program and the various different techniques for accessing the CICS `DFHCOMMAREA` interface, finding the relevant COBOL data structures can be a complex and time-consuming task that may require CICS COBOL programming knowledge. Note the following:

- A CICS program does not require a `PROCEDURE DIVISION` header, where parameters are normally defined. See *`PROCEDURE DIVISION` Mapping*.
- The `DFHCOMMAEA` can be omitted in the linkage section.
- If there is no `DFHCOMMAREA` in the linkage section or no `PROCEDURE DIVISION` header present in the `PROCEDURE DIVISION`, the CICS preprocessor completes the interface of the COBOL server and adds a `DFHCOMMAREA` and a `PROCEDURE DIVISON` header to the CICS program before compilation.

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 a CICS DFHCOMMAREA Program

This section assumes **Input Message same as Output Message** is checked. COBOL output and COBOL input parameters are the same, that is, the DFHCOMMAREA on output is not overlaid with a data structure different to the data structure on input.

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

Make sure the interface type is correct and checkbox **Input Message same as Output Message** is not cleared.

Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items of the CICS message to **COBOL Interface** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. See **Notes**.

2 Continue with *COBOL to IDL Mapping*.

Notes:

1. If a `DFHCOMMAREA` is present, the `DFHCOMMAREA` COBOL data item itself cannot be selected. In this case, select the COBOL data items directly subordinated to `DFHCOMMAREA` and map to IDL. See *Map to In, Out, InOut*.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL interface contains `REDEFINE`s, 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.

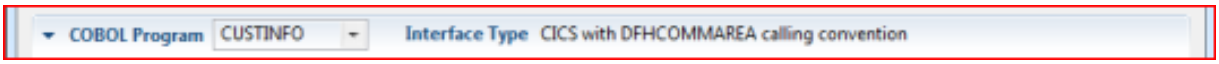
The user interface of the COBOL Mapping Editor is described below.

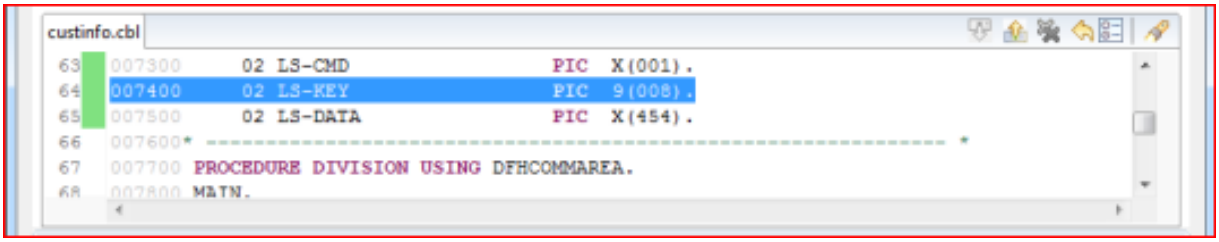
Mapping Editor User Interface

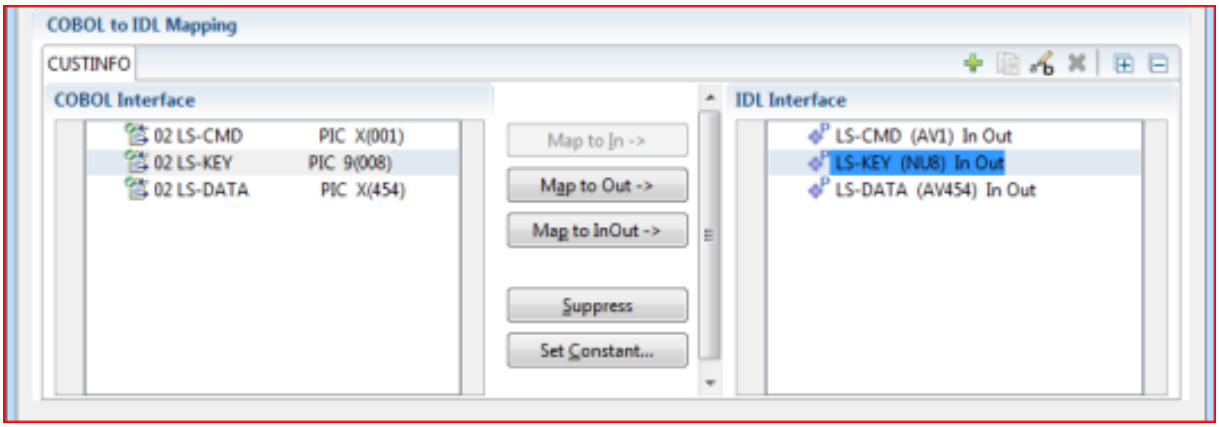
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface type CICS with DFHCOMMAREA interface, the user interface of the COBOL Mapping Editor looks like this:

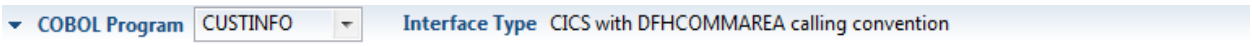
1. 

2. 

3. 

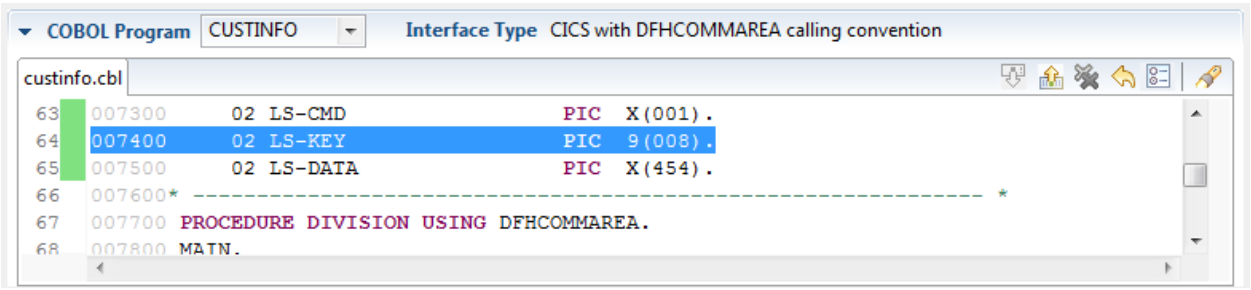
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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.
- 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

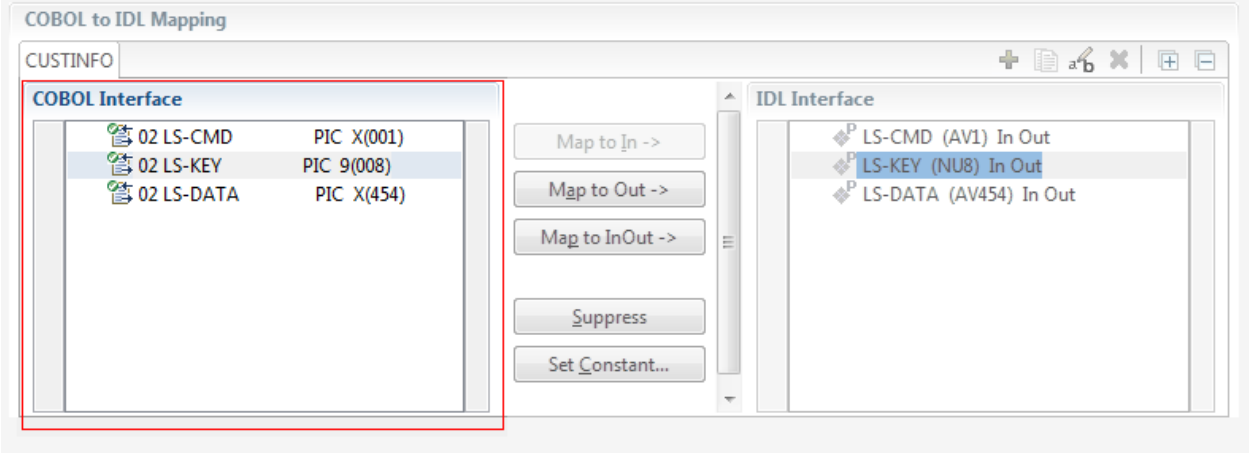
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

Map to In | Out | InOut	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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

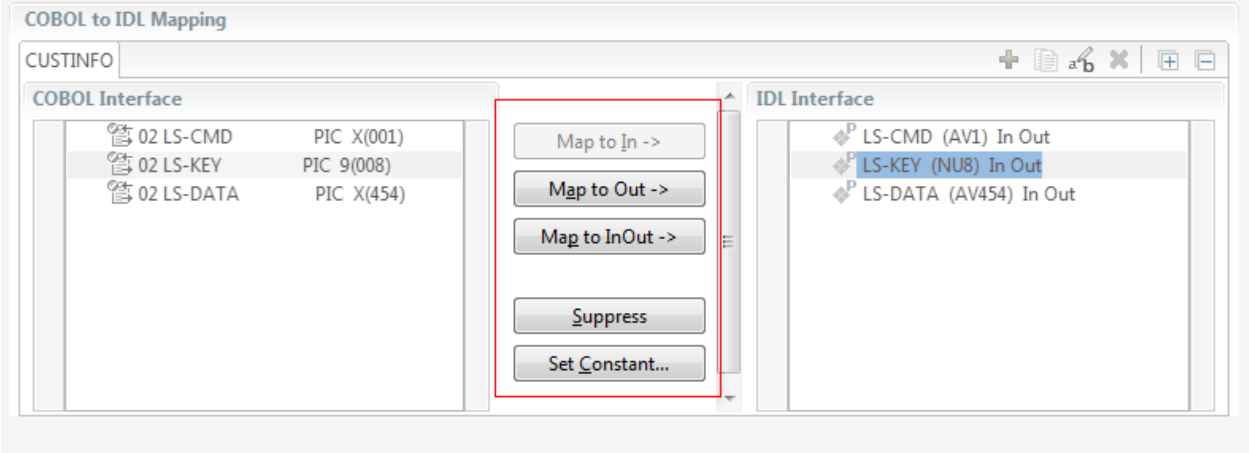
Mapping Icons

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

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

Mapping Buttons

The following buttons are available:



Map to In | Out | InOut ->

See *Map to In, Out, InOut*. 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*.

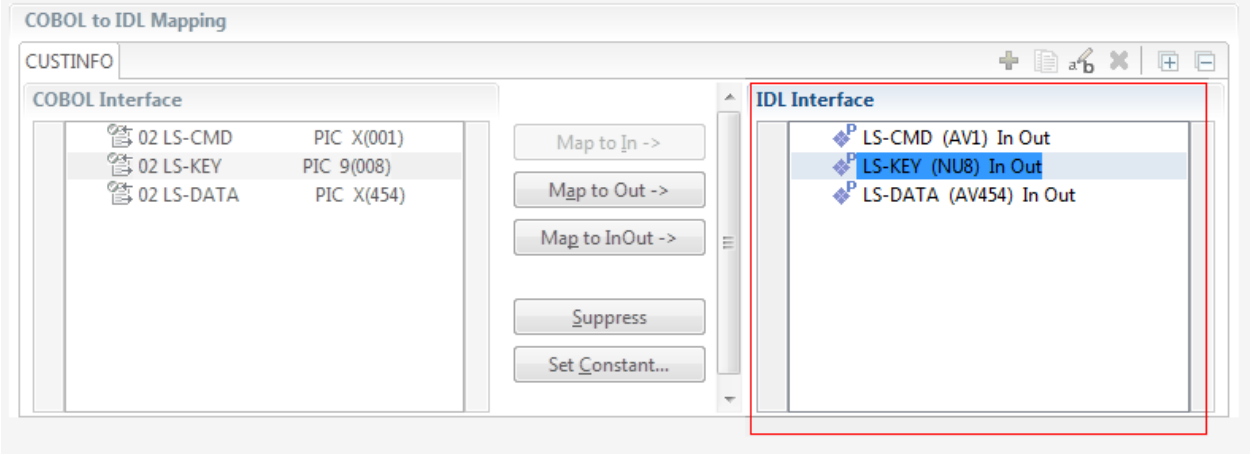
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

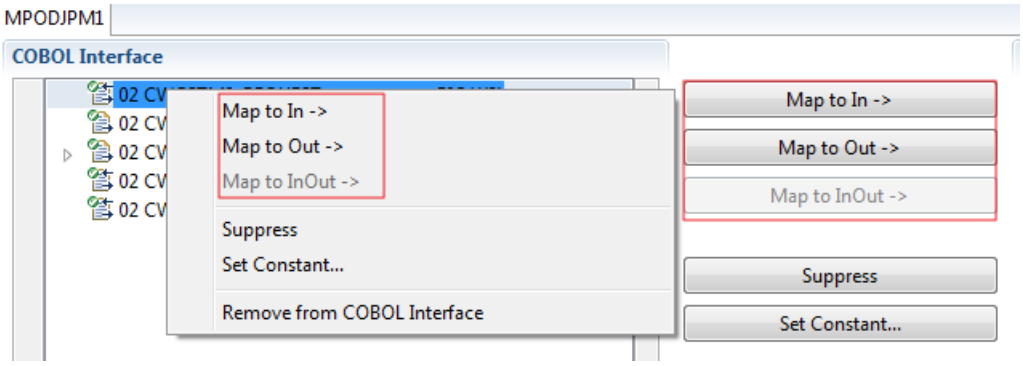
This section covers the following topics:

Map to In, Out, InOut

With the **Map to In**, **Out**, **InOut** functions you make a COBOL data item visible as an IDL parameter in the IDL interface. With correct IDL directions you design the IDL interface by defining input and output parameters. COBOL programs have no parameter directions, so you need to set IDL directions manually.

To provide IDL directions

- Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons to make the COBOL data items visible and provide IDL directions in the IDL interface:



Notes:

1. If a *top-level* COBOL *group* is mapped, the IDL direction is inherited by all subordinate COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subordinate COBOL data items can only be mapped to the same IDL direction as their *top-level* COBOL *group* data item.
3. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.
4. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Map OCCURS DEPENDING ON

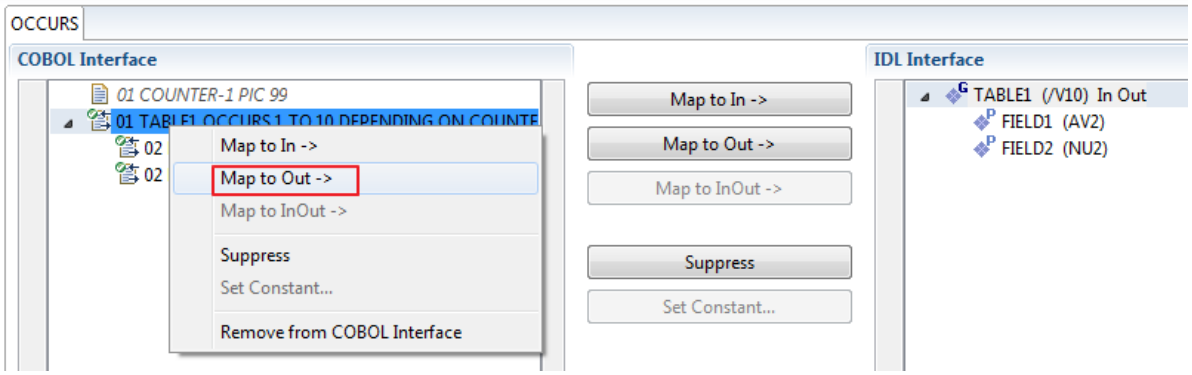
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):

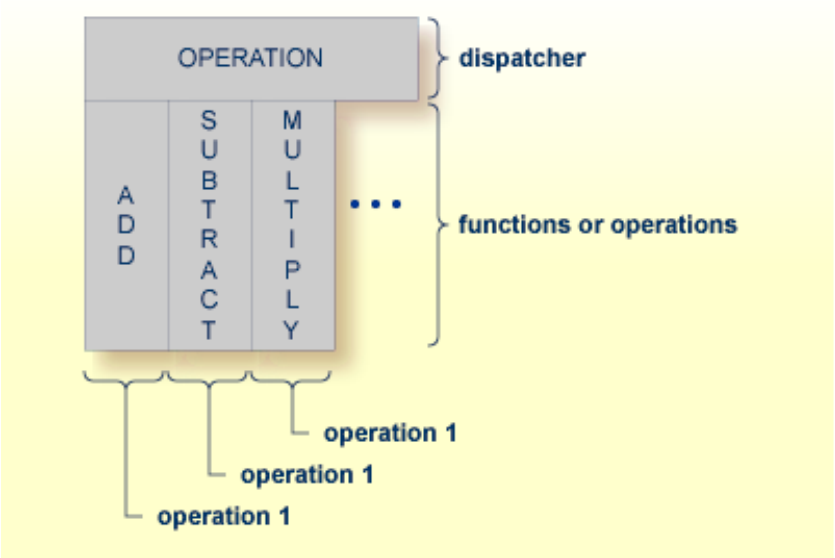


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "-"
```

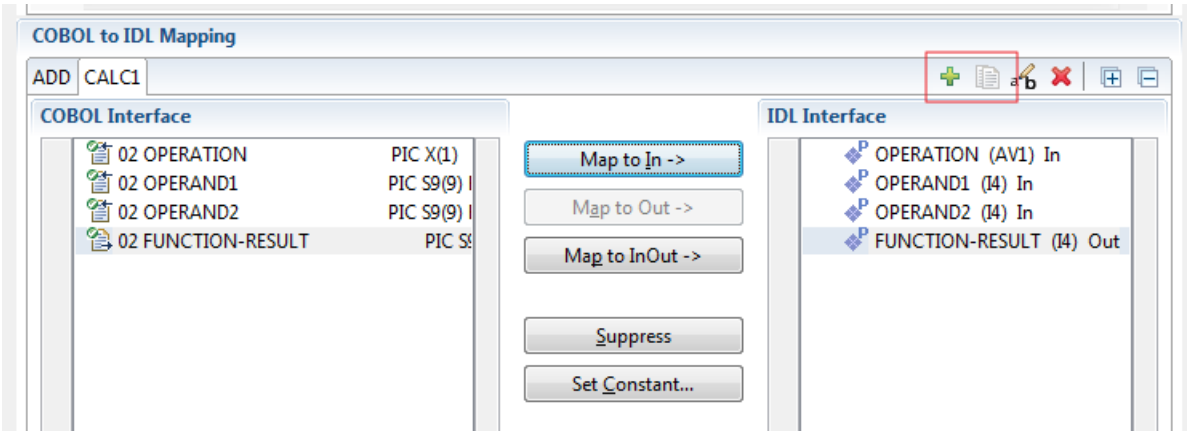
```
            SUBTRACT OPERAND2 FROM OPERAND1
            GIVING FUNCTION-RESULT
         WHEN "*"
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

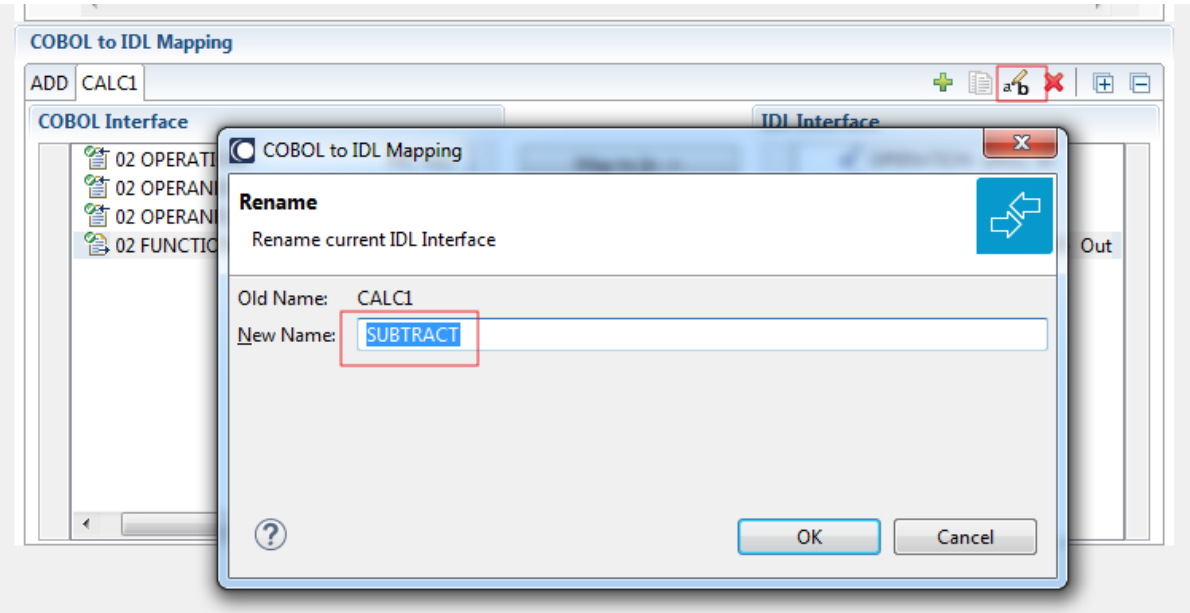
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

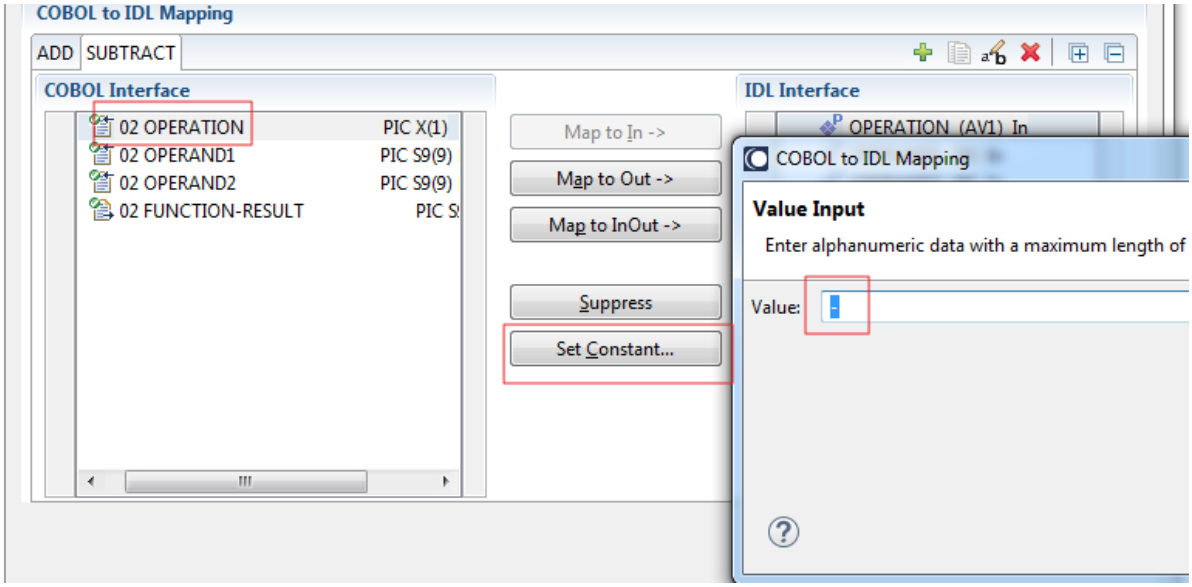
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

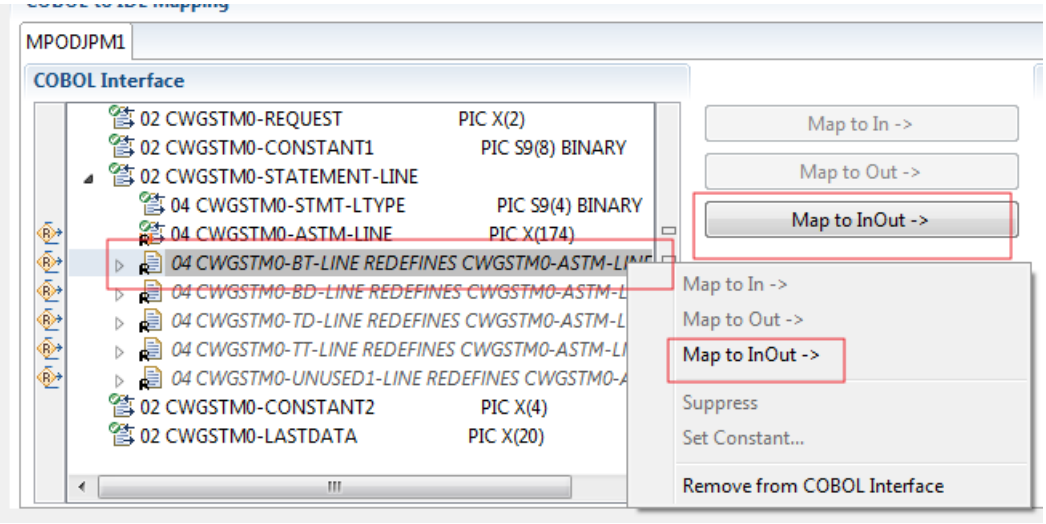
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

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

To select redefine paths

- Use the **Map to In**, **Out** or **InOut** functions available in the context menu of the COBOL interface and as mapping buttons 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:

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

Suppress Unneeded COBOL Data Items

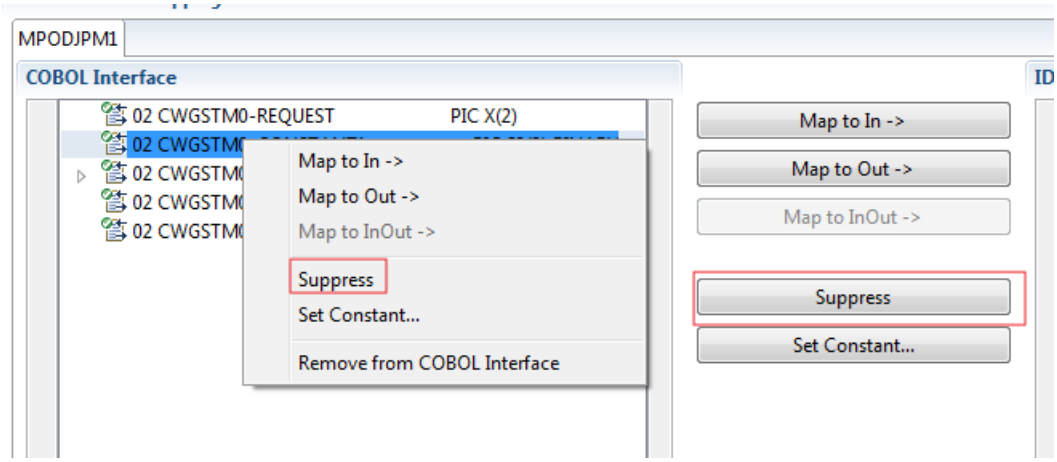
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface and as mapping button to make the COBOL data item invisible in the IDL interface:



Notes:

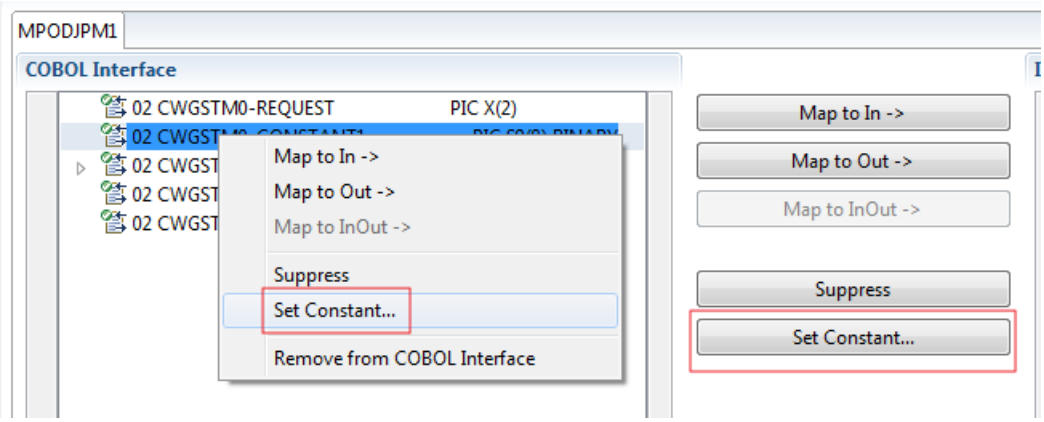
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subordinate COBOL data items are suppressed as well.
4. With the inverse functions **Map to In**, **Out** or **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item is made visible in the IDL interface again.

Set COBOL Data Items to Constants

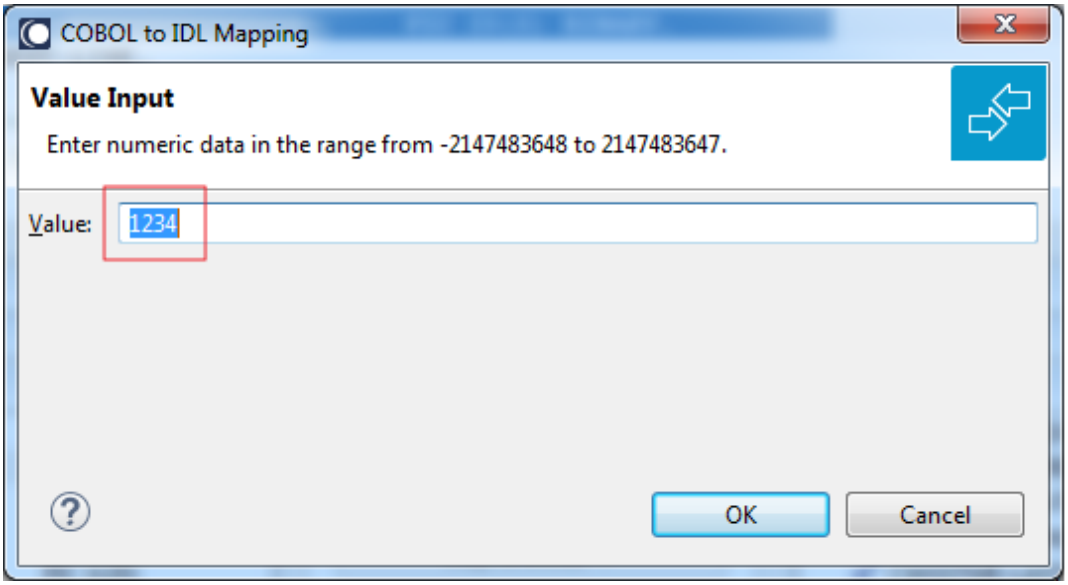
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter provides the defined constant in the COBOL data item to your COBOL server.
3. With the functions **Map to In**, **Out**, **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item can be made visible in the IDL interface again.

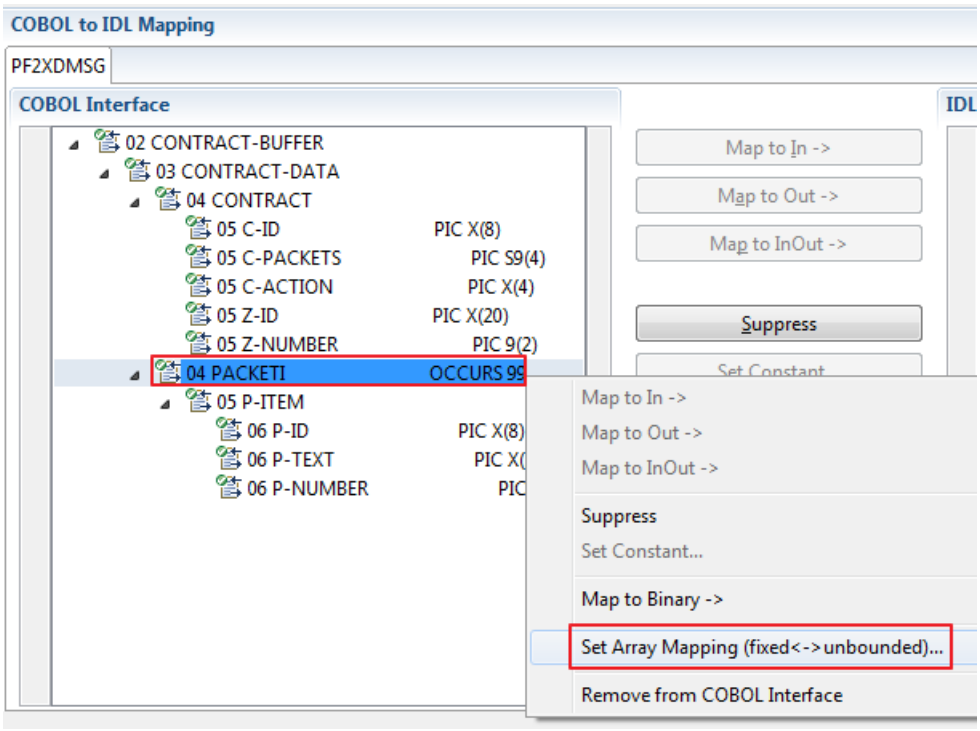
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

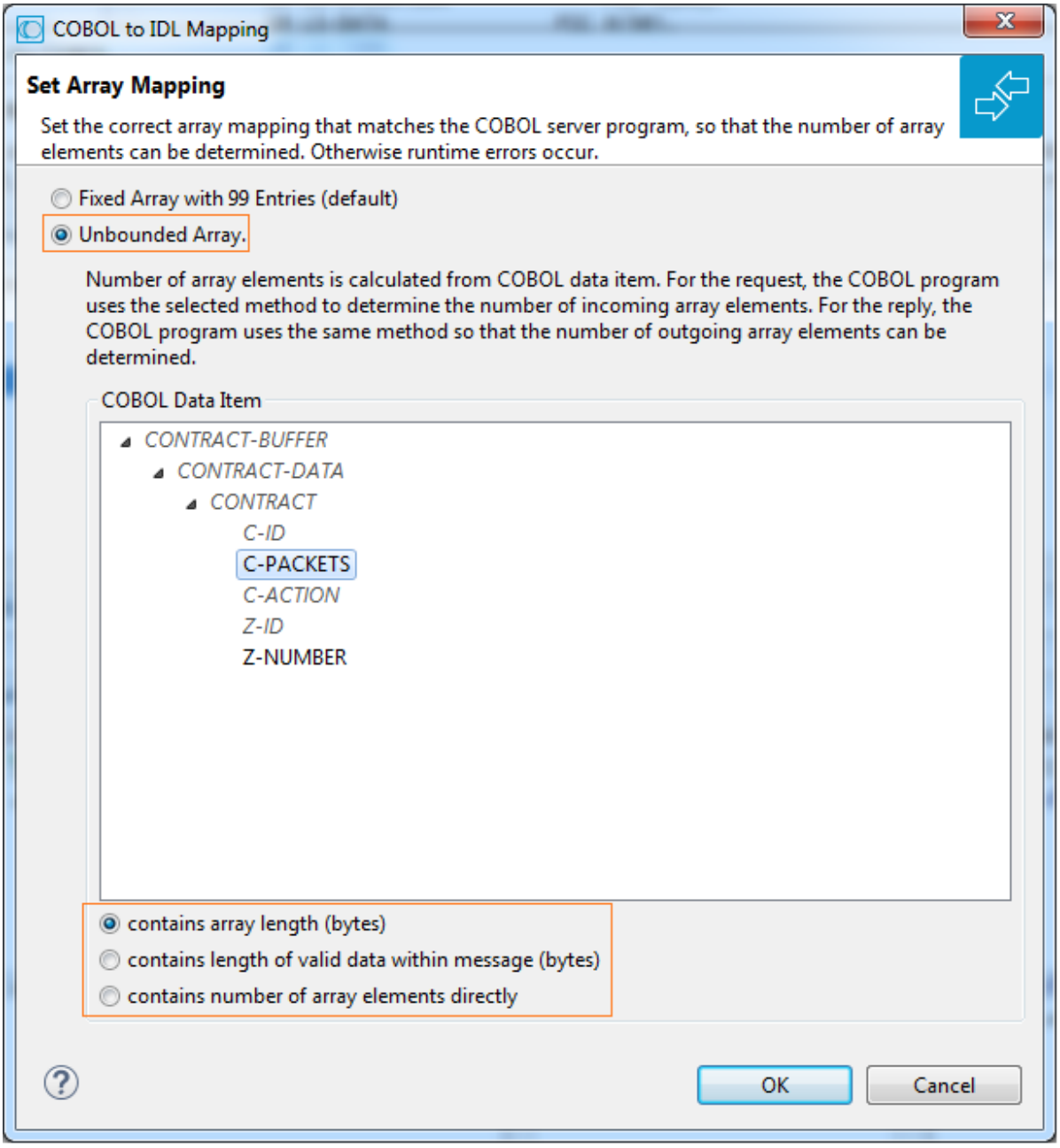
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The length of the fixed-size array `PACKETI` is contained in COBOL data item `C-BYTES`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-BYTES                            PIC S9(4).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
            05 Z-ID                               PIC X(20).
           04 PACKETI                             OCCURS 99.
            05 P-ITEM.
             06 P-ID                              PIC X(8).
             06 P-TEXT                            PIC X(30).
             06 P-NUMBER                          PIC 9(2).
      . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set table length
        COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT` is filled by the COBOL server on reply. COBOL data item `C-APPDATA` contains the length of the valid data of the reply message. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
         77 EPARM                                 PIC  9(2).
         77 EPARM2                                PIC  9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
           04 CONTRACT.
            05 C-ID                                PIC X(8).
            05 C-APPDATA                           PIC S9(4).
            05 C-ACTION                            PIC X(4).
            05 Z-ID                                PIC X(20).
            05 Z-NUMBER                            PIC 9(2).
           04 PACKETI                              OCCURS 99.
            05 P-ITEM.
             06 P-ID                               PIC X(8).
             06 P-TEXT                             PIC X(30).
             06 P-NUMBER                           PIC 9(2).
       . . .
     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set length
        COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                          +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**

  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                                PIC X(8).
            05 C-NUM                               PIC S9(4).
            05 C-ACTION                            PIC X(4).
           04 ZONE.
            05 Z-NUMBER                            PIC 9(2).
```

```
          05 Z-ID                                 PIC X(20).
        04 PACKETI                                OCCURS 99.
          05 P-ITEM.
            06 P-ID                               PIC X(8).
            06 P-TEXT                             PIC X(30).
            06 P-NUMBER                           PIC 9(2).
     . . .

    *  Fill variable output array
       MOVE 0 TO II.
       PERFORM RANDOMNUM TIMES
          ADD 1 TO II
          MOVE ... TO P-ID    (II)
          MOVE ... TO P-TEXT  (II)
          MOVE ... TO P-NUMBER(II)
       END-PERFORM.
    *  Set occurrences
       MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

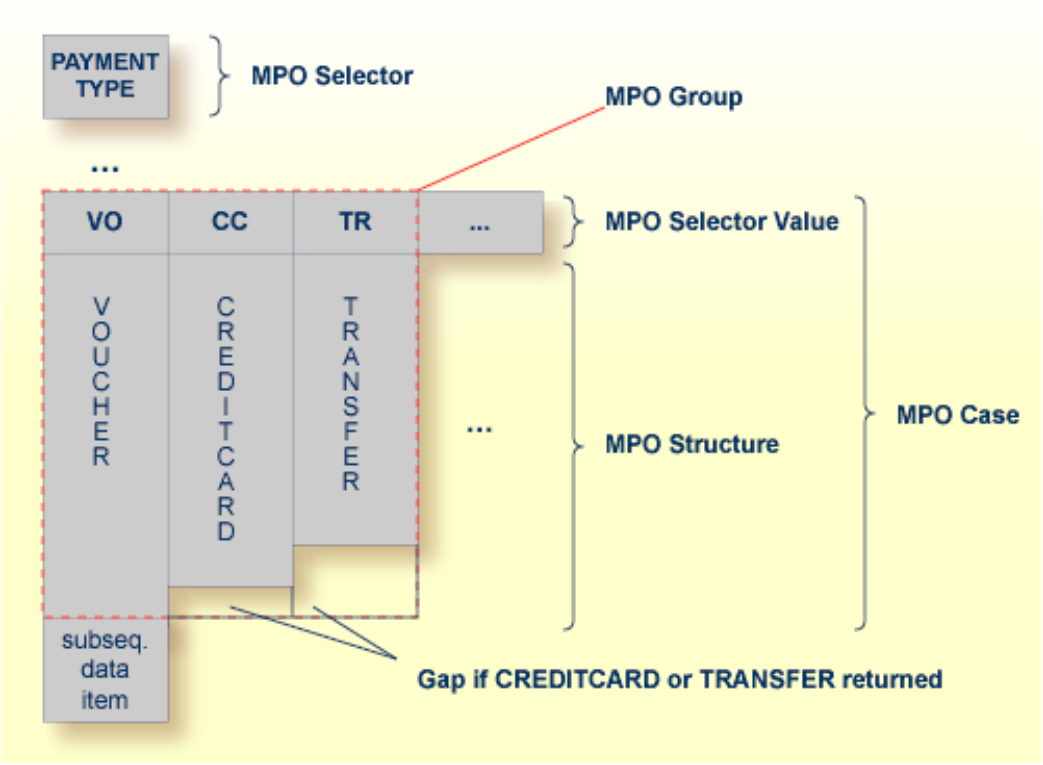
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>               PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>              PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

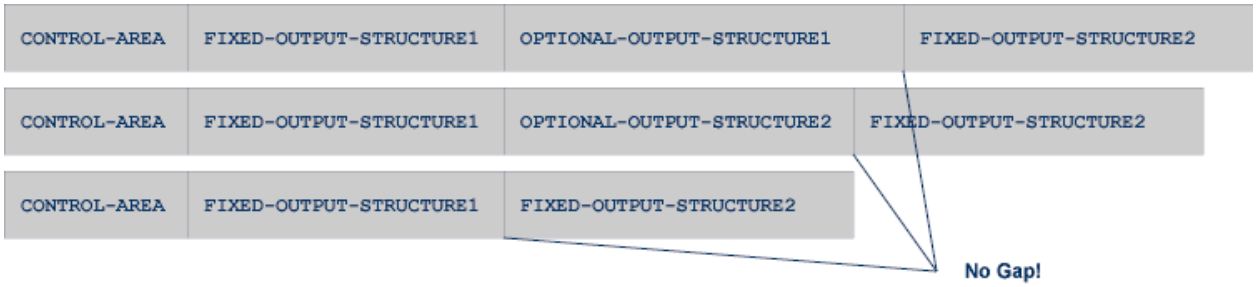
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

Note: If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

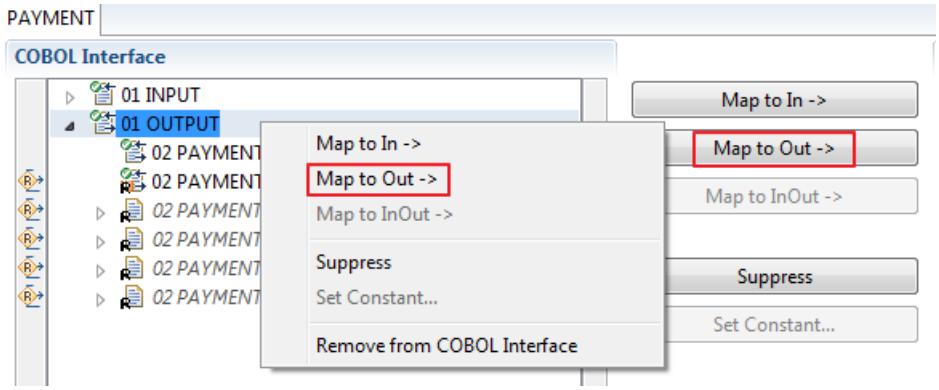
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

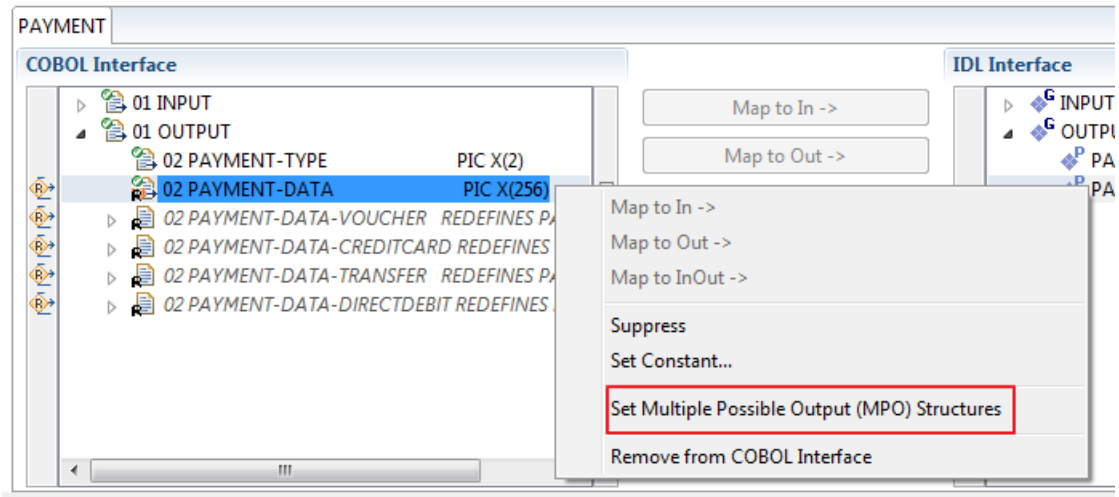
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

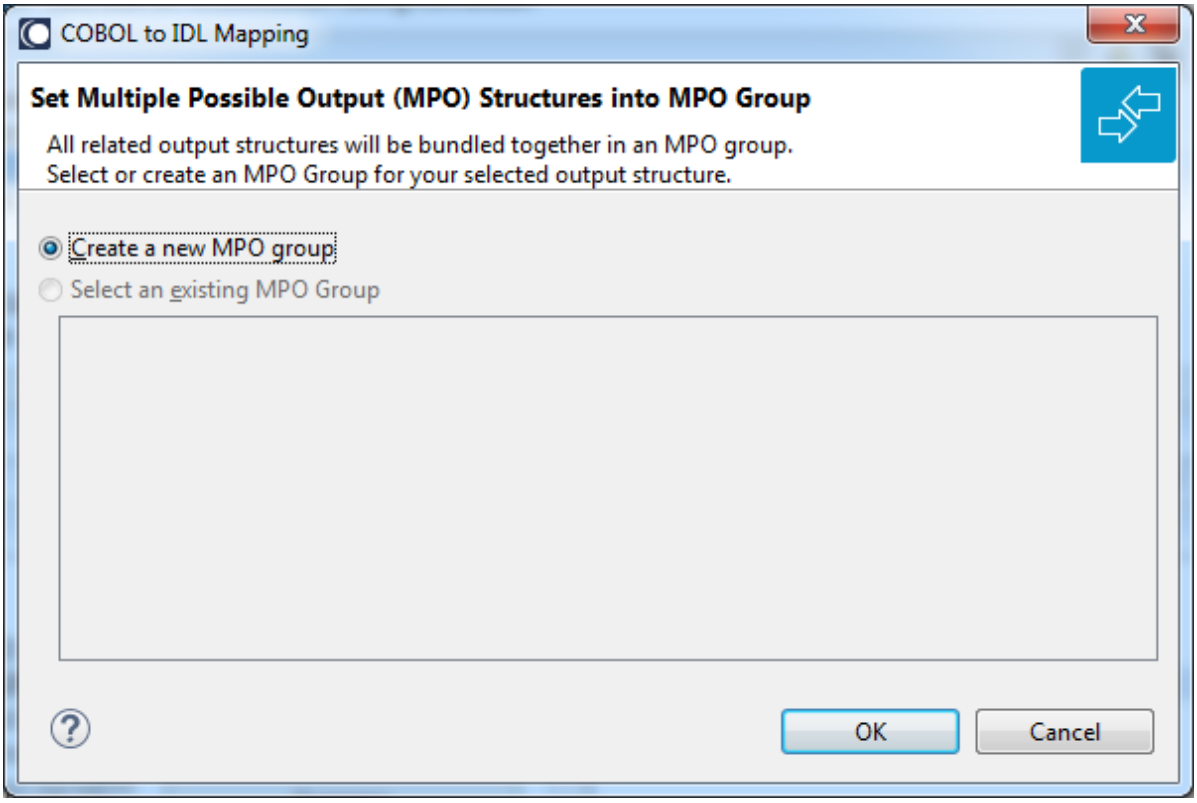
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



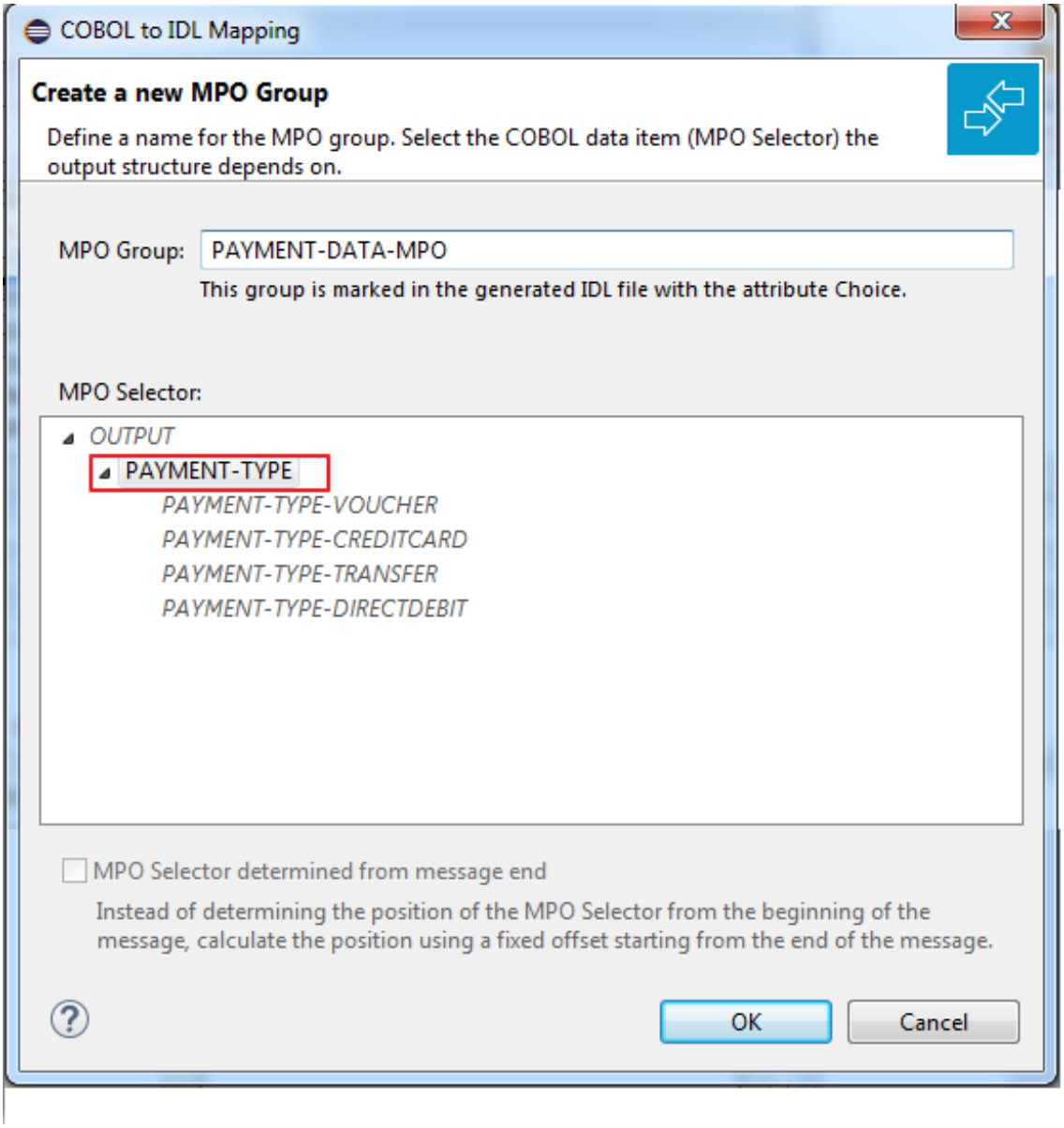
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



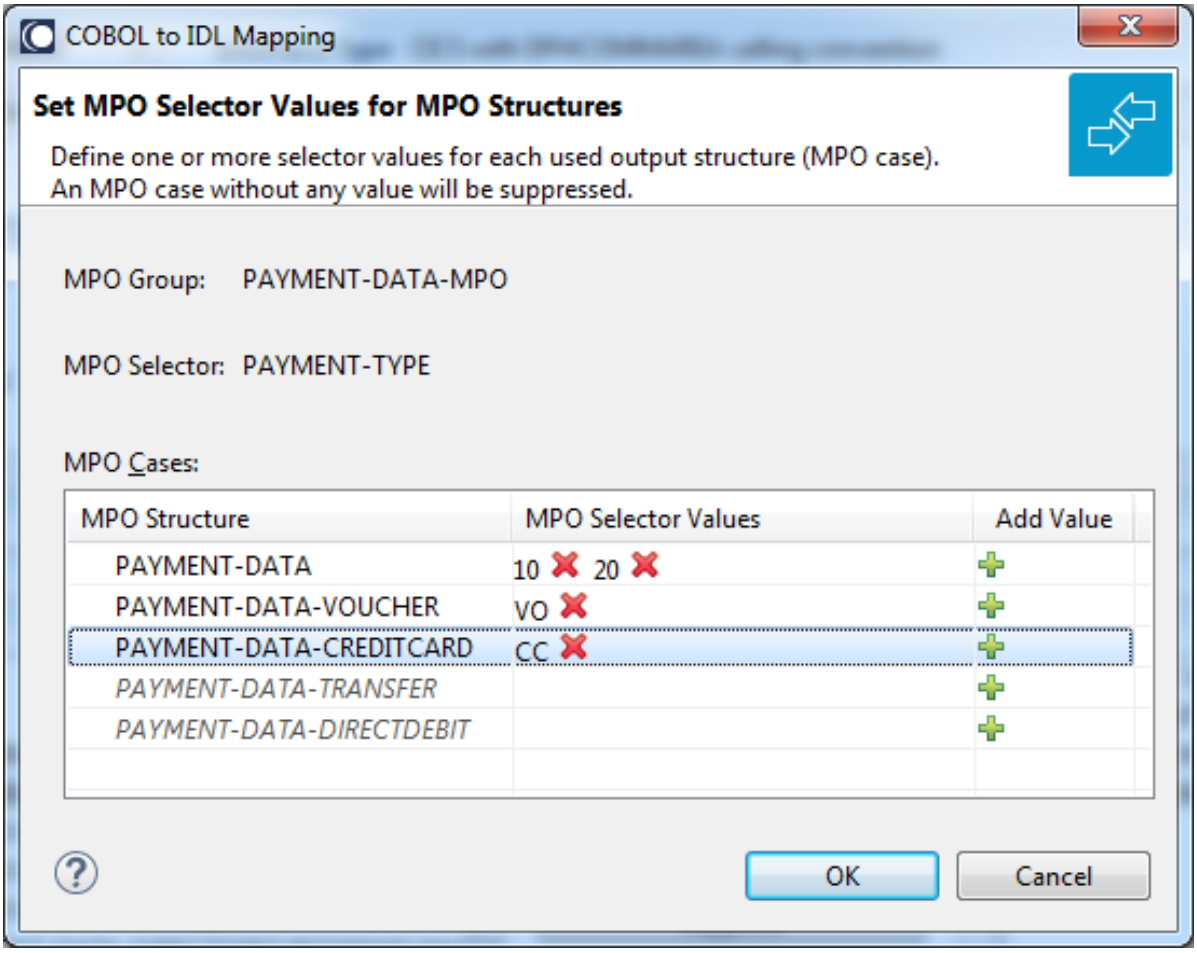
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



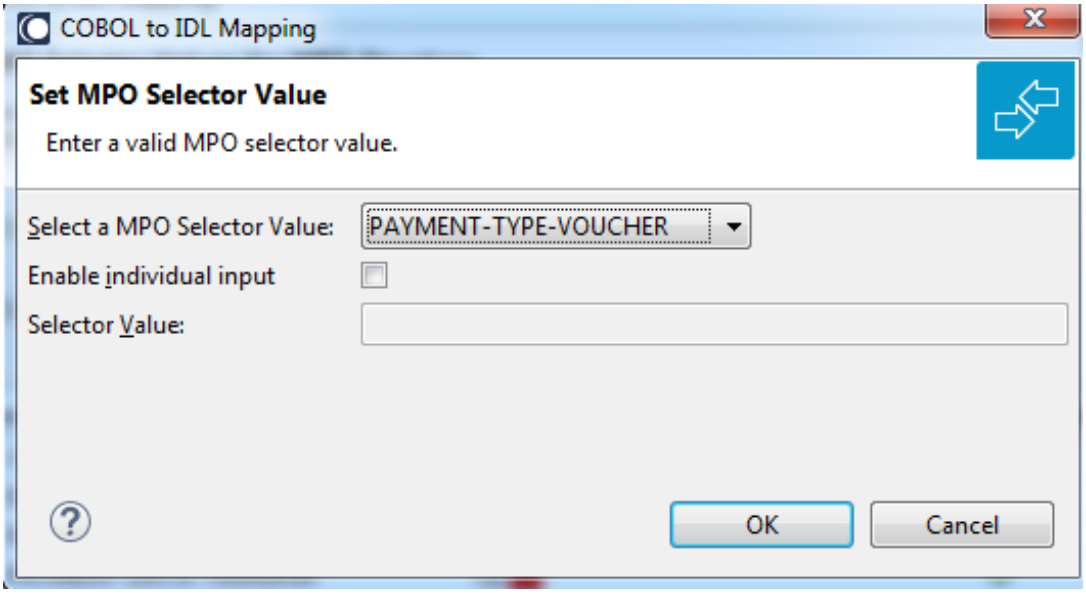
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

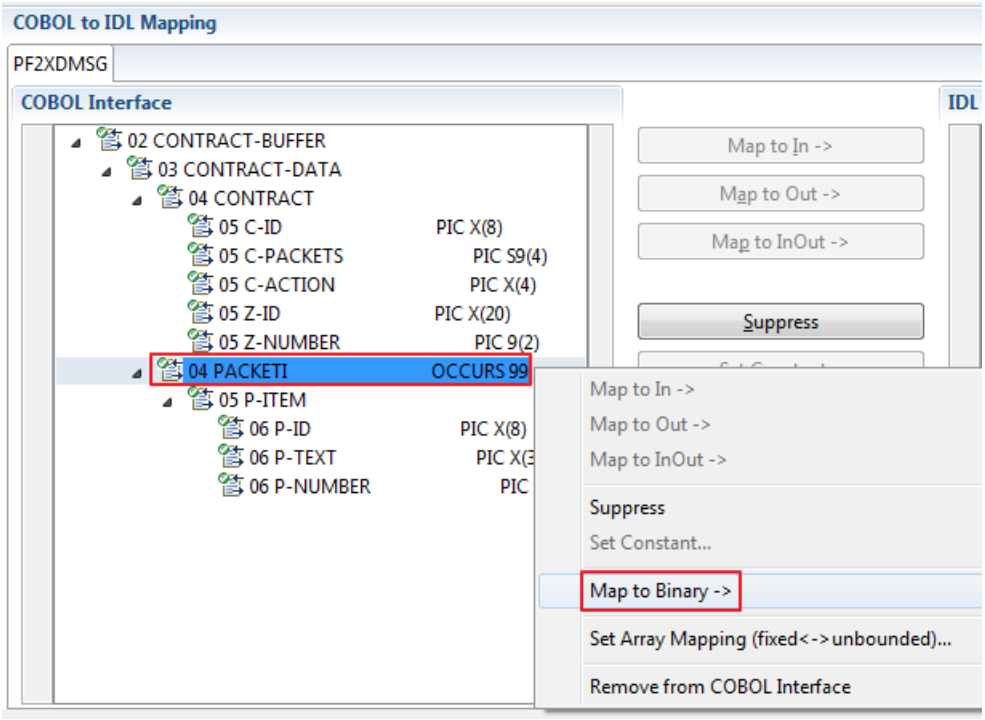
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

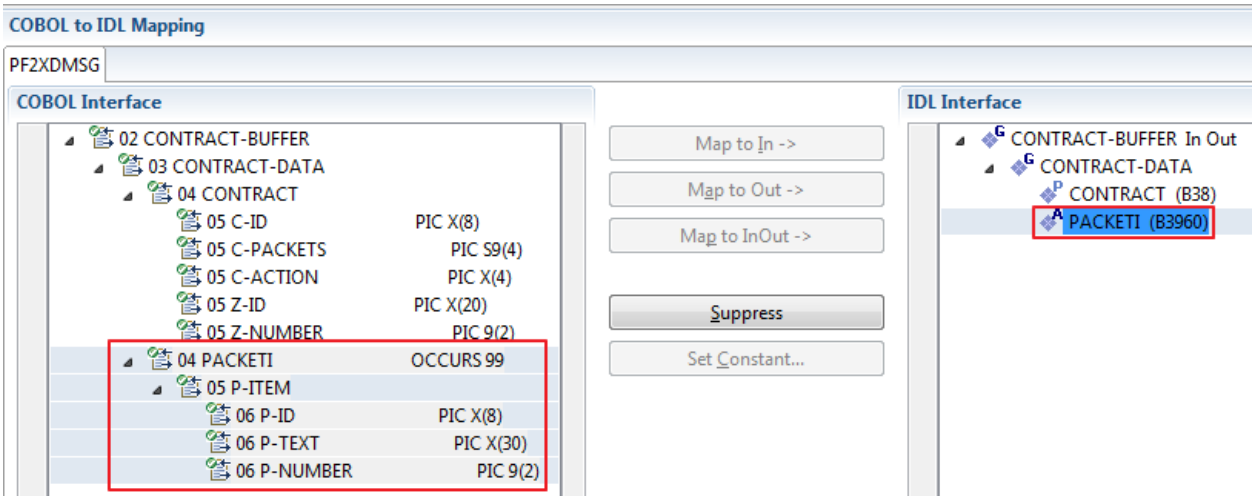
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.

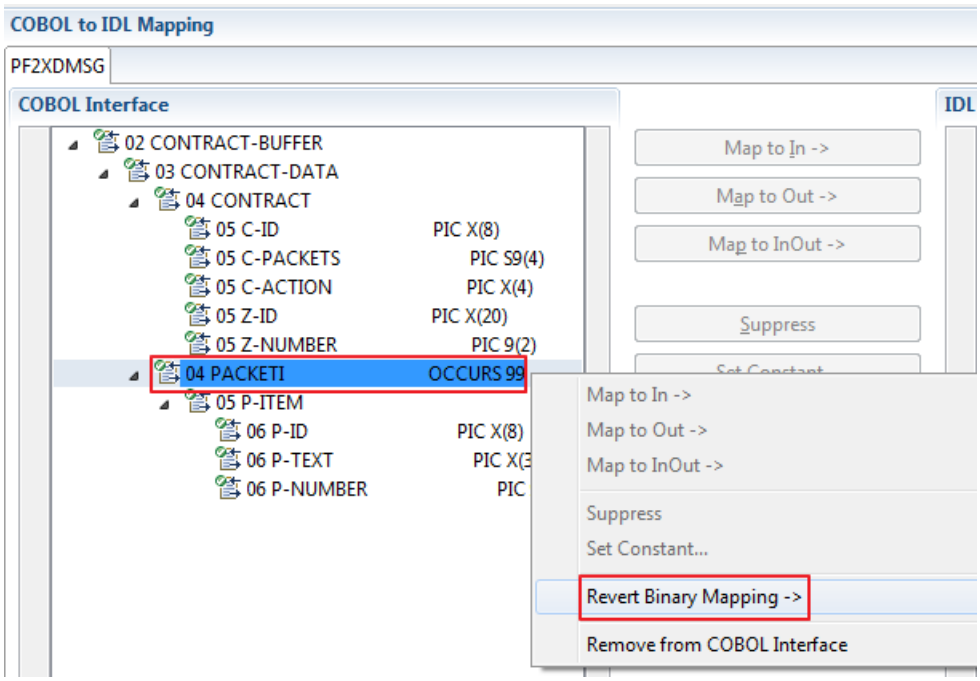


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

Programming Techniques

Example 1: Redefines

The input and output data is described with a `REDEFINE` as in the following example. In this case you need to select `REDEFINE` path `BUFFER2` for the COBOL interface.

```
 LINKAGE SECTION.
 01 DFHCOMMAREA.

 02 BUFFER1.
    03 FIELD-1                       PIC X(4).
    03 FIELD-2                       PIC X(2).
 . . .
 02 BUFFER2 REDEFINES BUFFER1.
    03 OPERATION                     PIC X(1).
    03 OPERAND-1                     PIC S9(9) BINARY.
    03 OPERAND-2                     PIC S9(9) BINARY.
    03 FUNCTION-RESULT               PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
* process BUFFER2
    EXEC CICS RETURN.
```

Often a similar looking technique is used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat: A `REDEFINE` is used to describe output data that overlays the input data, that is, the CICS input message is *different* to CICS output message. For more information see ***Example 1: Redefines*** in section *CICS with `DFHCOMMAREA` Calling Convention - In different to Out*.

Example 2: Buffer Technique

On entry, the server moves linkage section field(s) - often an entire buffer - into the working storage and processes the input data inside the working storage field(s). Before return, it moves the working storage field(s) - often an entire buffer - back to the linkage section. In this case, the relevant COBOL data items are described within the working storage section. You need to select `WS-BUFFER` for the COBOL interface.

```
 WORKING STORAGE SECTION.
 01 WS-BUFFER.
    02 OPERATION                    PIC X(1).
    02 OPERAND-1                    PIC S9(9) BINARY.
    02 OPERAND-2                    PIC S9(9) BINARY.
    02 FUNCTION-RESULT              PIC S9(9) BINARY.
 LINKAGE SECTION.
 01 DFHCOMMAREA.
    02 IO-BUFFER                    PIC X(9).
 . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
    MOVE IO-BUFFER TO WS-BUFFER.
* process the WS-BUFFER and provide result in WS-BUFFER
    MOVE WS-BUFFER TO IO-BUFFER.
    EXEC CICS RETURN.
```

A similar looking technique is used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat: The buffer technique may be used to describe output data that overlays the input data, that is, the CICS input message is *different* to CICS output message. For more information see ***Example 2: Buffer Technique*** in section *CICS with* `DFHCOMMAREA` *Calling Convention - In different to Out*.

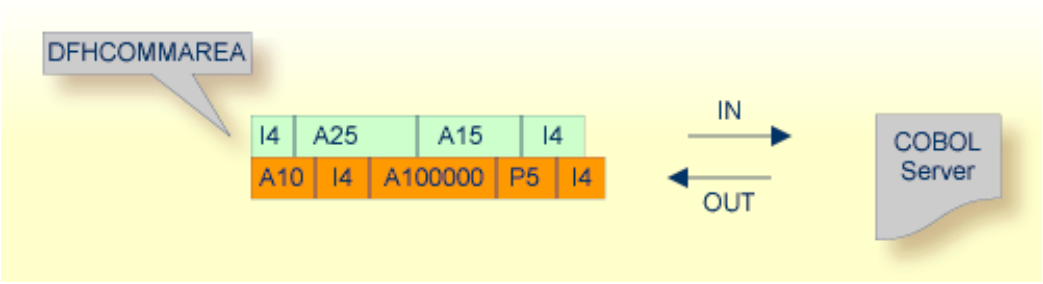
Example 3: COBOL SET ADDRESS Statements

COBOL `SET ADDRESS` statements are used to manipulate the interface of the CICS server. On entry, the server addresses the data with a (dummy) structure `LS-BUFFER` defined in the linkage section. You need to select `LS-BUFFER` for the COBOL interface.

```
 LINKAGE SECTION.
 01 LS-BUFFER.
    02 OPERATION                    PIC X(1).
    02 OPERAND-1                    PIC S9(9) BINARY.
    02 OPERAND-2                    PIC S9(9) BINARY.
    02 FUNCTION-RESULT              PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION.
    SET ADDRESS OF LS-BUFFER TO DFHCOMMAREA.
* process the LS-BUFFER and provide result.
    EXEC CICS RETURN.
```

A similar looking technique is used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat: COBOL `SET ADDRESS` statements may be used to describe output data that overlays the input data, that is, the CICS input message is *different* to CICS output message. For more information see ***Example 3: COBOL*** `SET ADDRESS` ***Statements*** in section *CICS with* `DFHCOMMAREA` *Calling Convention - In different to Out*.

7 CICS with DFHCOMMAREA Calling Convention - In different to Out



Introduction

Depending on the programming style used in the CICS program and the various different techniques for accessing the CICS `DFHCOMMAREA` interface, finding the relevant COBOL data structures can be a complex and time-consuming task that may require CICS COBOL programming knowledge. Note the following:

- A CICS program does not require a `PROCEDURE DIVISION` header, where parameters are normally defined. See *`PROCEDURE DIVISION` Mapping*.
- The `DFHCOMMAEA` can be omitted in the linkage section.
- If there is no `DFHCOMMAREA` in the linkage section or no `PROCEDURE DIVISION` header present in the `PROCEDURE DIVISION`, the CICS preprocessor completes the interface of the COBOL server and adds a `DFHCOMMAREA` and a `PROCEDURE DIVISON` header to the CICS program before compilation.

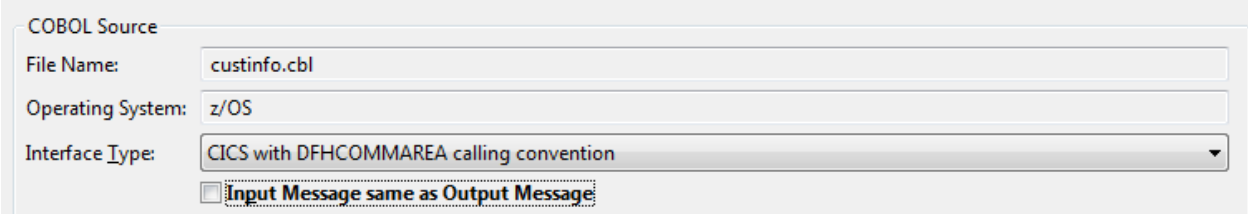
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

Extracting from a CICS DFHCOMMAREA Program

This section assumes **Input Message same as Output Message** is not checked. COBOL output and COBOL input parameters are different, that is, the DFHCOMMAREA on output is overlaid with a data structure that is different to the data structure on input. See the examples provided under *Programming Techniques*.

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

Make sure the interface type is correct and check box **Input Message same as Output Message** is cleared.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items of the CICS input message to **Input Message** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. See **Notes**.

2 Add the COBOL data items of the CICS output message to **Output Message** by using the context menu and toolbars available in the *COBOL Interface* and *IDL Interface*. See **Notes**.

3 Continue with *COBOL to IDL Mapping*.

Notes:

1. If a `DFHCOMMAREA` is present, the `DFHCOMMAREA` COBOL data item itself cannot be selected. In this case, select the COBOL data items directly subordinated to `DFHCOMMAREA` and map to IDL. See *Map to*.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL interface contains `REDEFINE`s, 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.

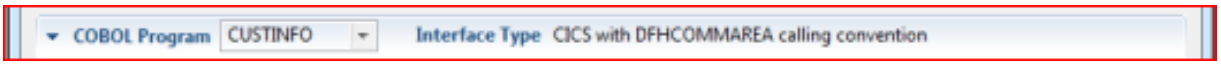
The user interface of the COBOL Mapping Editor is described below.

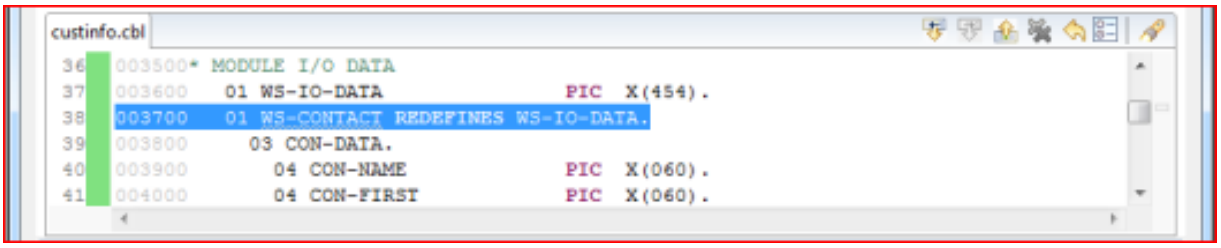
Mapping Editor User Interface

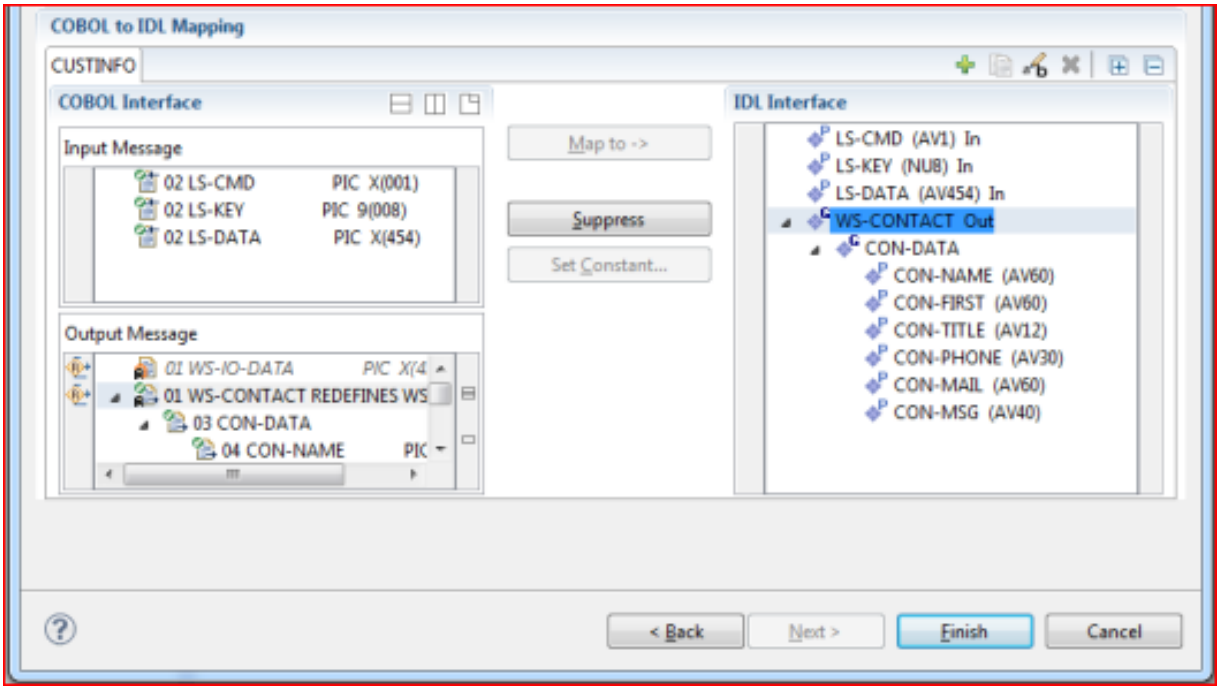
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface types where COBOL input and COBOL output parameters are different, the user interface of the COBOL Mapping Editor looks like this:

1. 

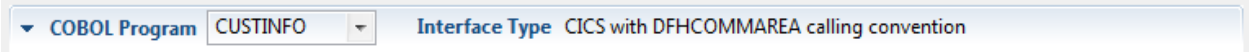
2. 

3. 

1. **COBOL Program Selection**. Currently selected program with interface type

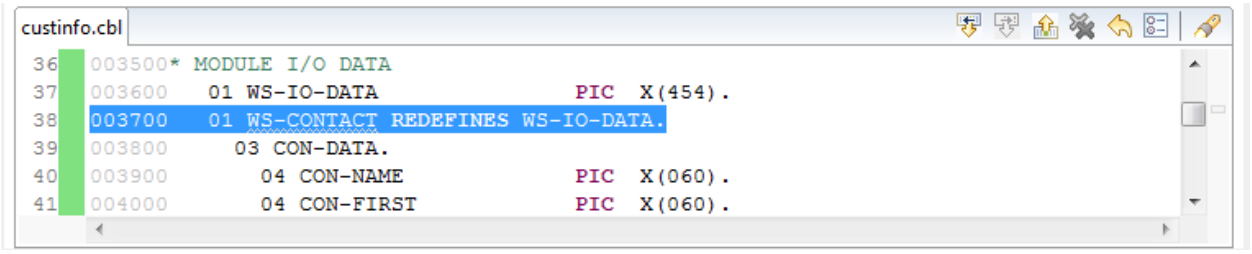
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

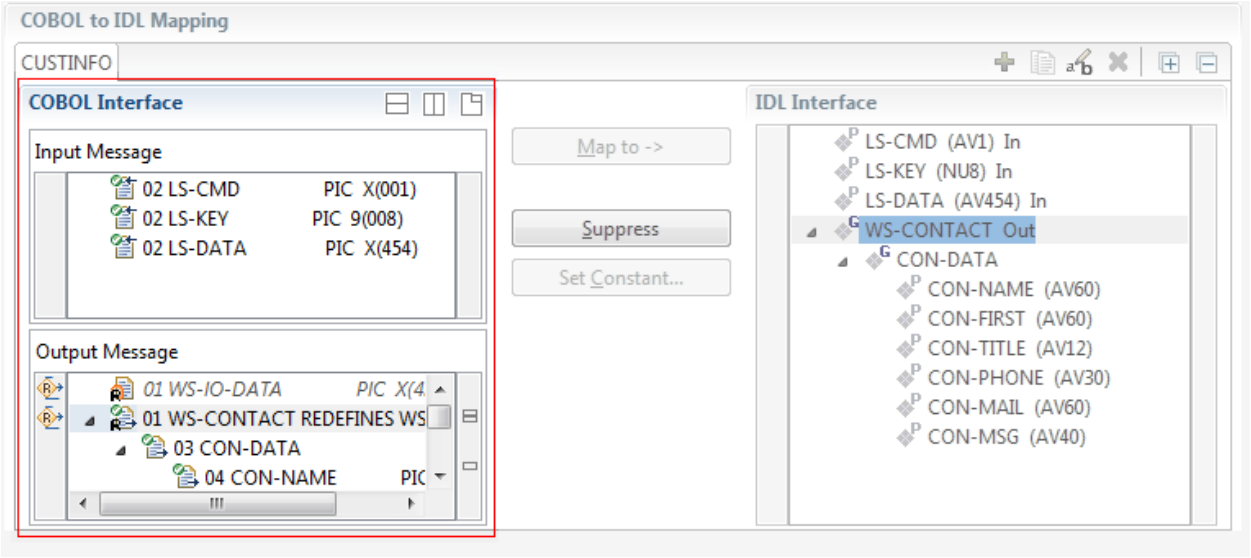
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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.

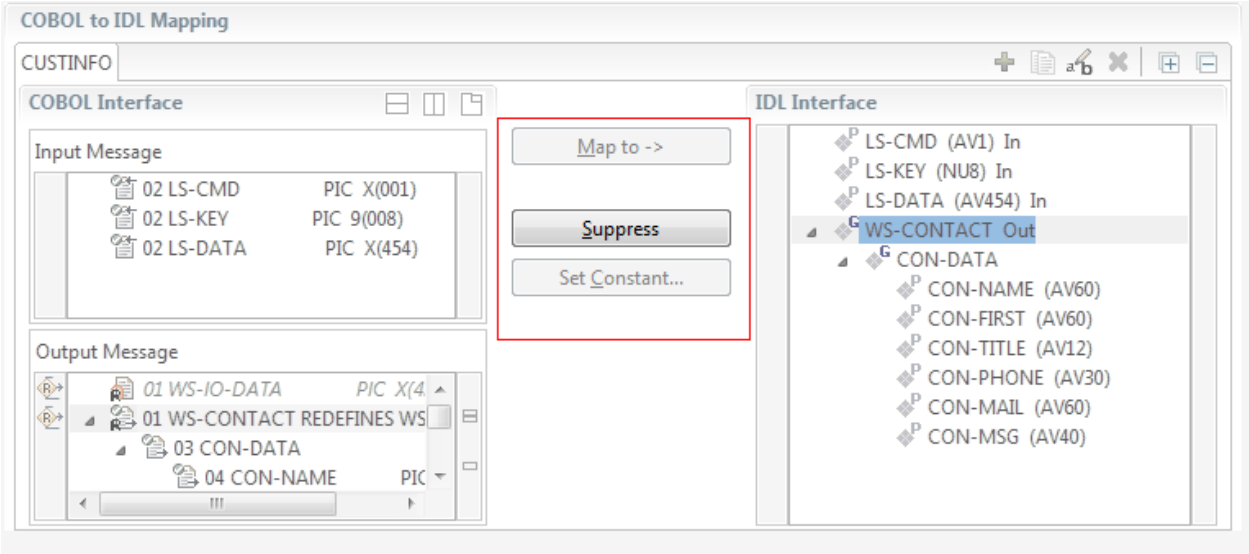
Group parameter, here mapped to In.

`REDEFINE` parameter, here mapped to Out.

Parameter set to Constant.

Mapping Buttons

The following buttons are available:



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*.

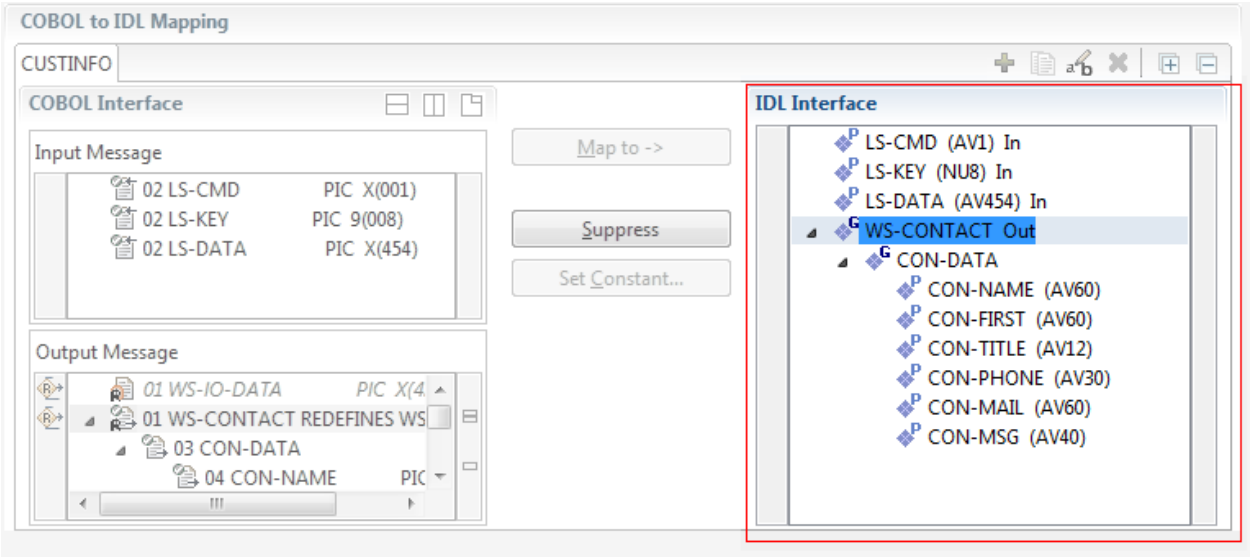
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

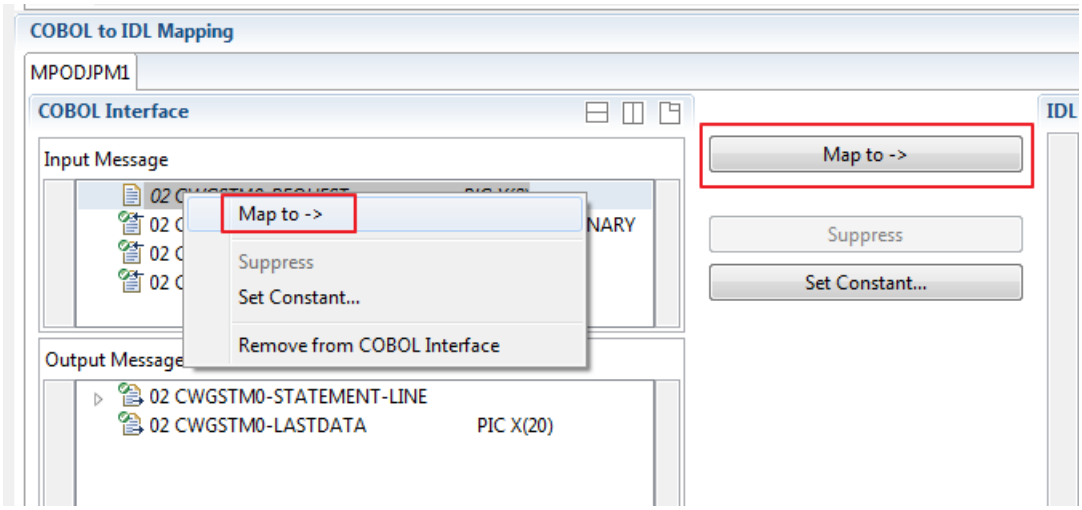
This section covers the following topics:

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 COBOL data items 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 of the COBOL interface and as mapping button to make a COBOL data item visible as an IDL parameter in the IDL interface:



2 Do the same for the output message of the COBOL interface.

Notes:

1. If a COBOL group is mapped, all subordinate 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 of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.

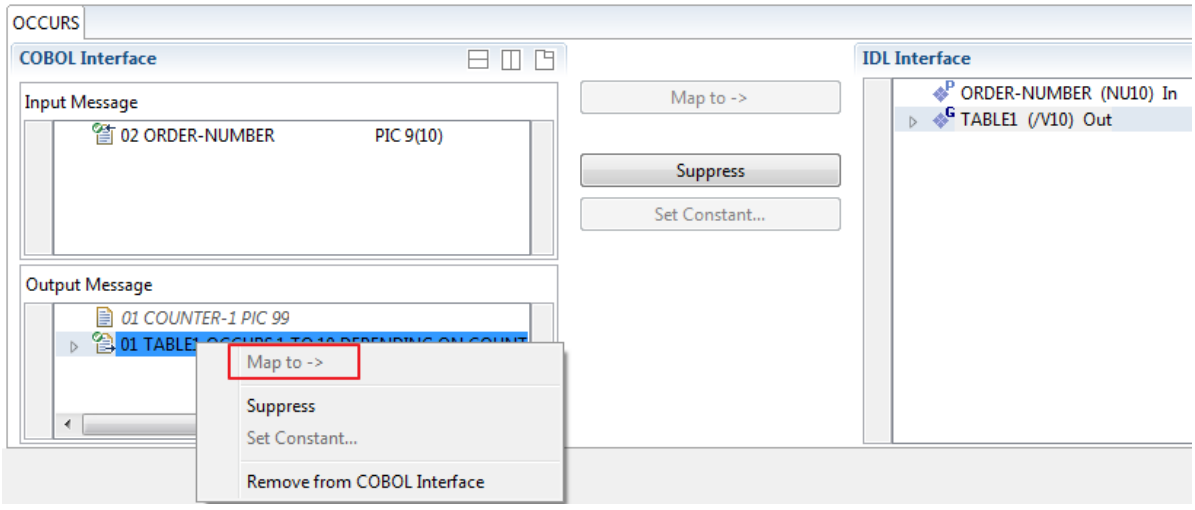
Map OCCURS DEPENDING ON

You can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see *COBOL Tables with Variable Size - `DEPENDING ON` Clause*) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

- Add the COBOL subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the input message or to the output message, wherever they belong. It is important both data items are always together per message direction (input or output).



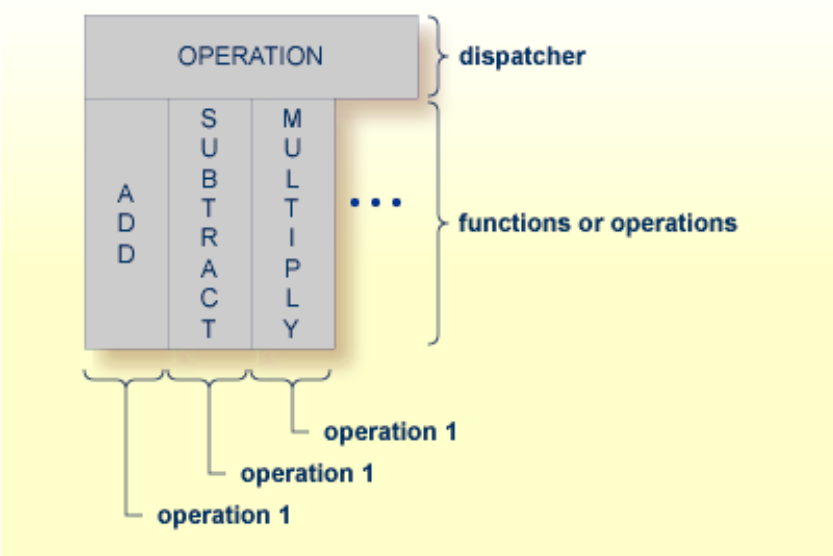
Notes:

1. The ODO subject can be mapped to the IDL interface.

2. The ODO object is always suppressed, but is required to be part of the same message direction (Input Message or Output Message) of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "*"
```

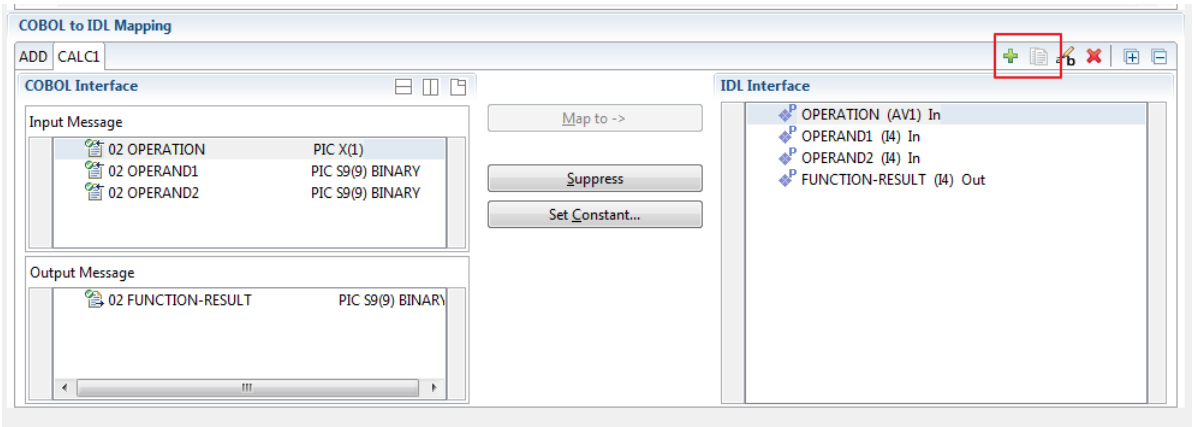
```
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

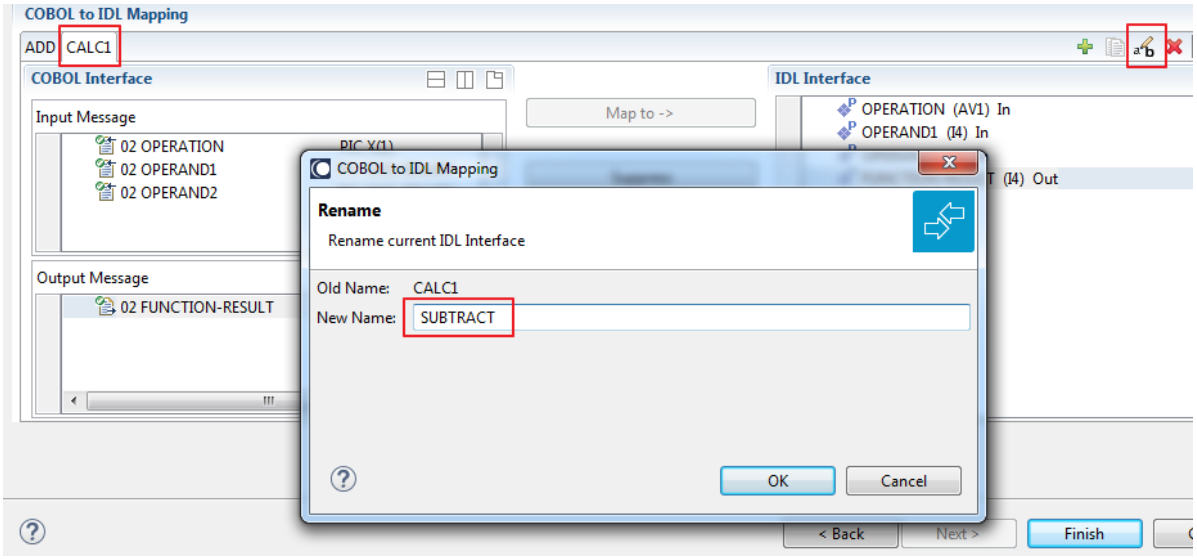
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

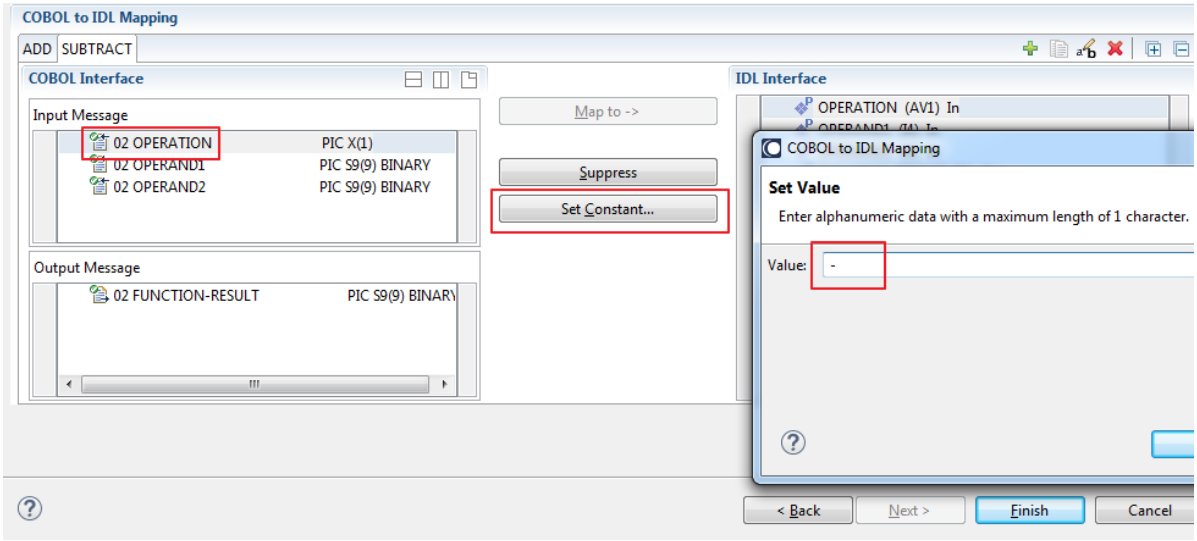
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.
- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

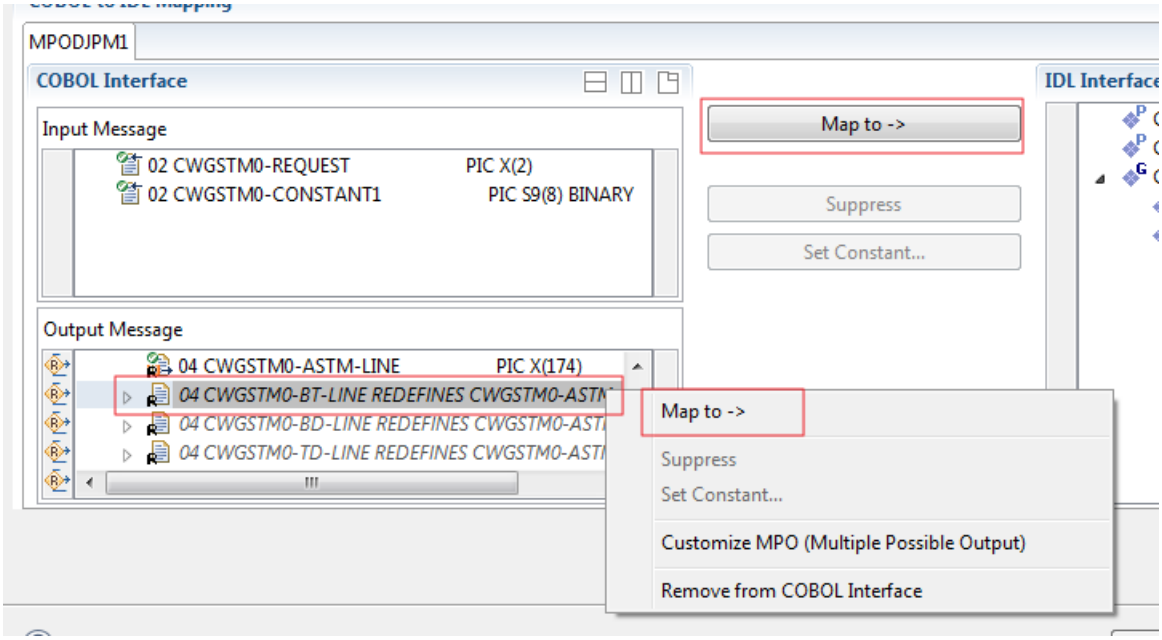
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

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 of the COBOL interface 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:

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

Suppress Unneeded COBOL Data Items

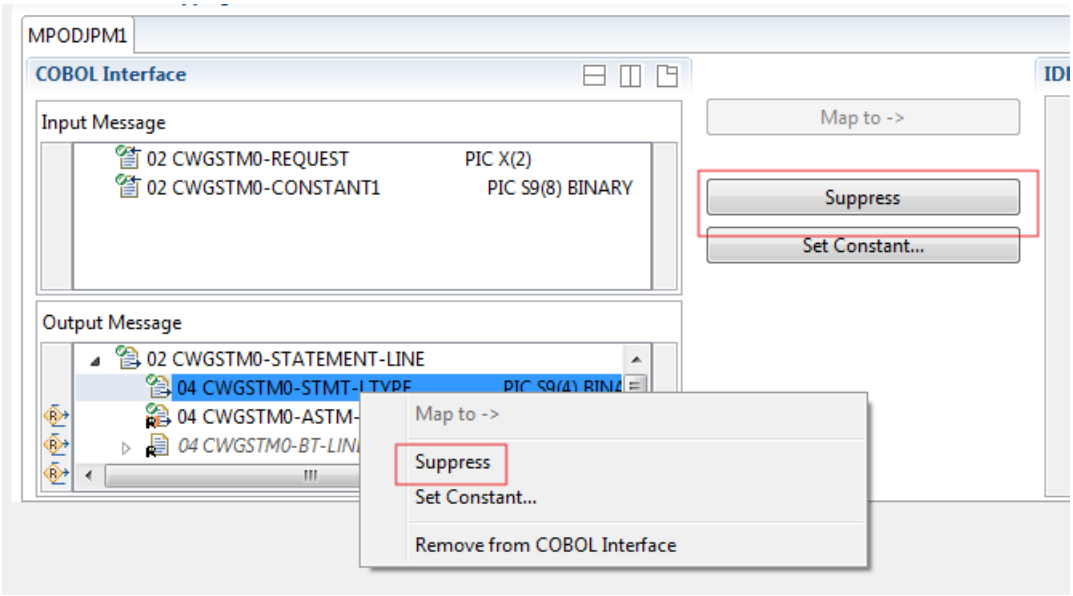
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface 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 consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* 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 subordinate COBOL data items are suppressed as well.

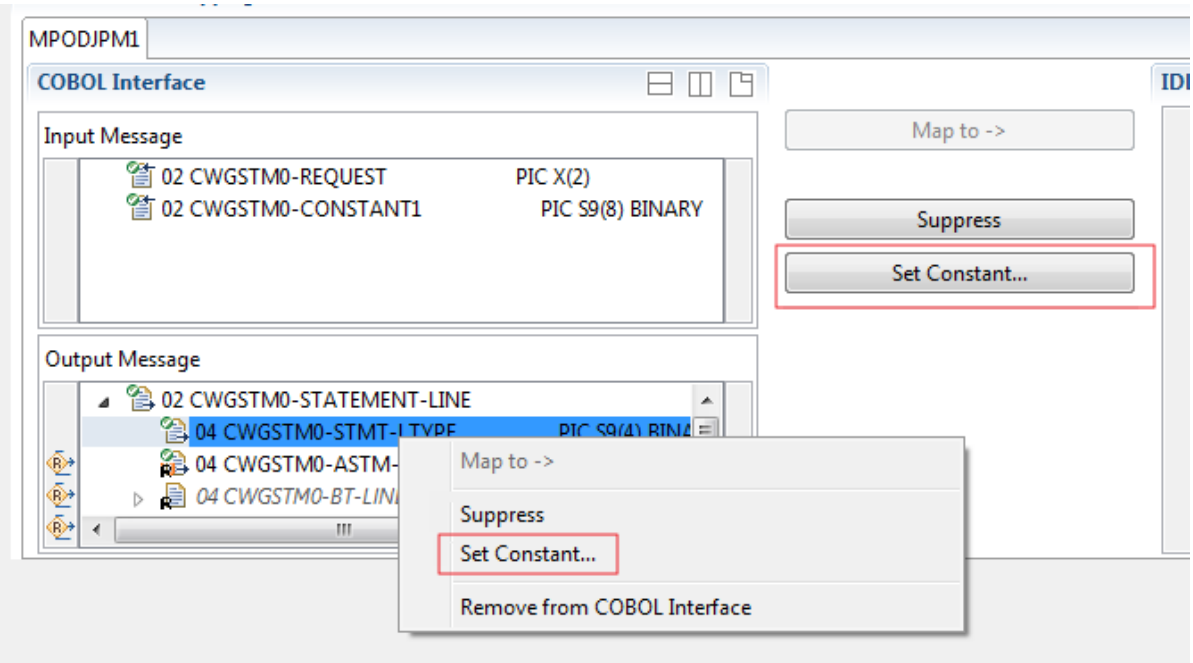
4. With the inverse function **Map to** (see above) available in the context menu of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

Set COBOL Data Items to Constants

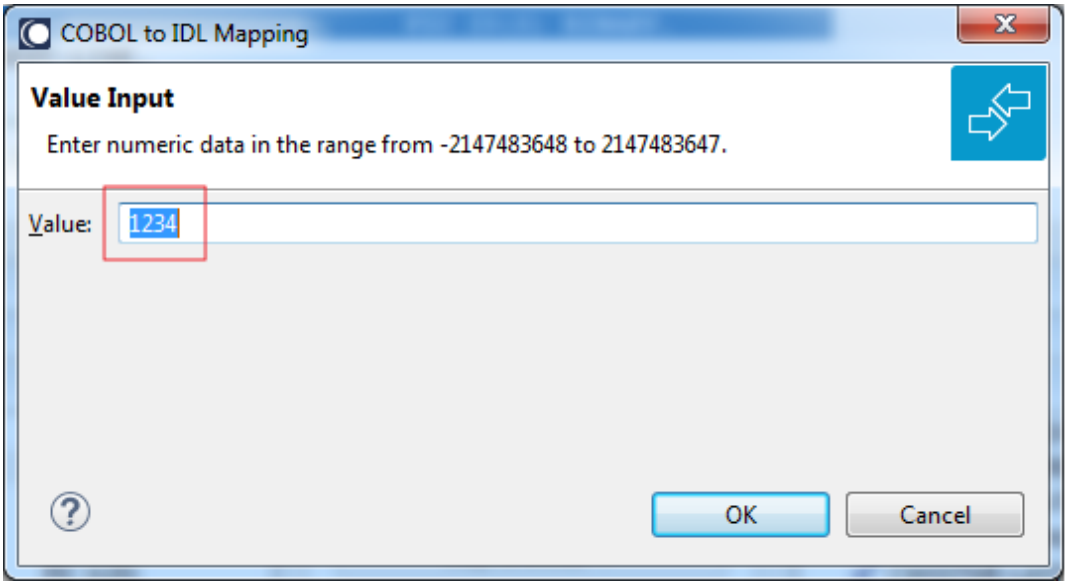
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter 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 of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

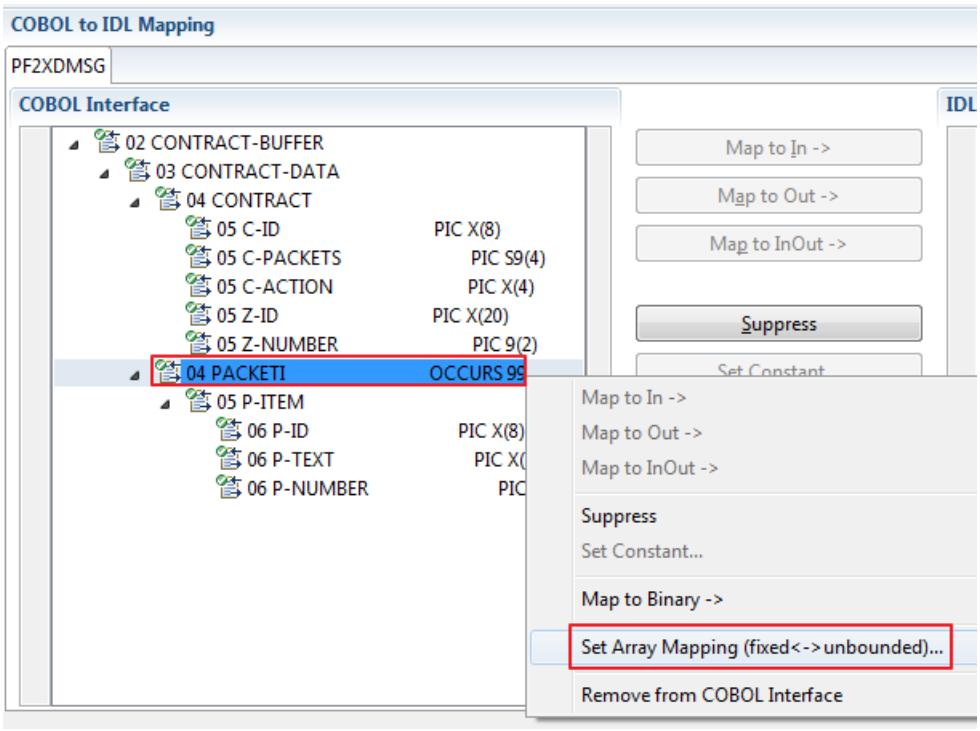
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

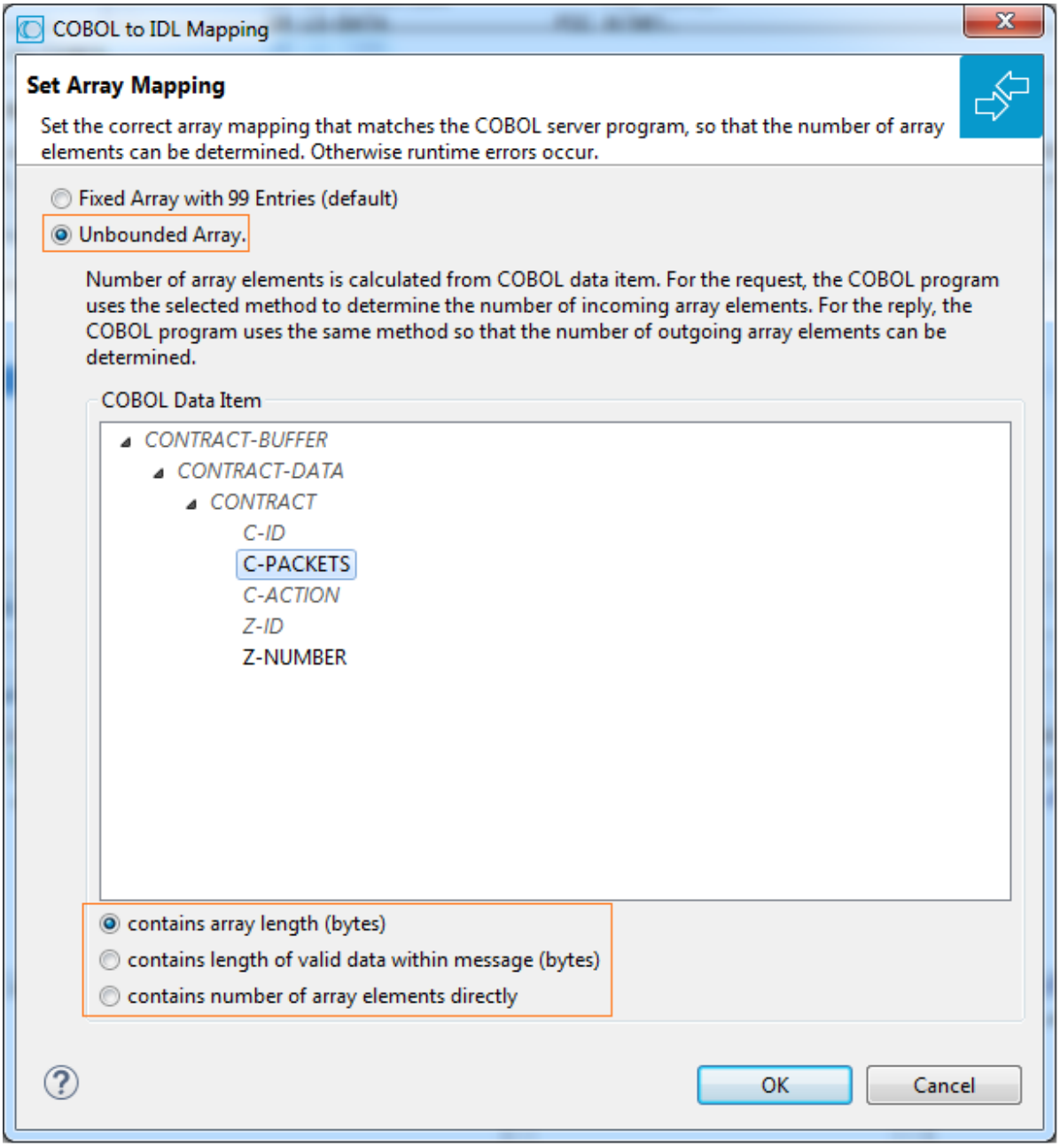
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The length of the fixed-size array `PACKETI` is contained in COBOL data item `C-BYTES`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-BYTES                            PIC S9(4).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
            05 Z-ID                               PIC X(20).
           04 PACKETI                             OCCURS 99.
            05 P-ITEM.
             06 P-ID                              PIC X(8).
             06 P-TEXT                            PIC X(30).
             06 P-NUMBER                          PIC 9(2).
      . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set table length
        COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT` is filled by the COBOL server on reply. COBOL data item `C-APPDATA` contains the length of the valid data of the reply message. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

```
       WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
        77 EPARM                                 PIC  9(2).
        77 EPARM2                                PIC  9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-APPDATA                         PIC S9(4).
            05 C-ACTION                          PIC X(4).
            05 Z-ID                              PIC X(20).
            05 Z-NUMBER                          PIC 9(2).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
       . . .
     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set length
        COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                          +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**

  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-NUM                             PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
```

```
           05 Z-ID                                  PIC X(20).
          04 PACKETI                                OCCURS 99.
           05 P-ITEM.
            06 P-ID                                 PIC X(8).
            06 P-TEXT                               PIC X(30).
            06 P-NUMBER                             PIC 9(2).
     . . .

   *  Fill variable output array
      MOVE 0 TO II.
      PERFORM RANDOMNUM TIMES
         ADD 1 TO II
         MOVE ... TO P-ID    (II)
         MOVE ... TO P-TEXT  (II)
         MOVE ... TO P-NUMBER(II)
      END-PERFORM.
   *  Set occurrences
      MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

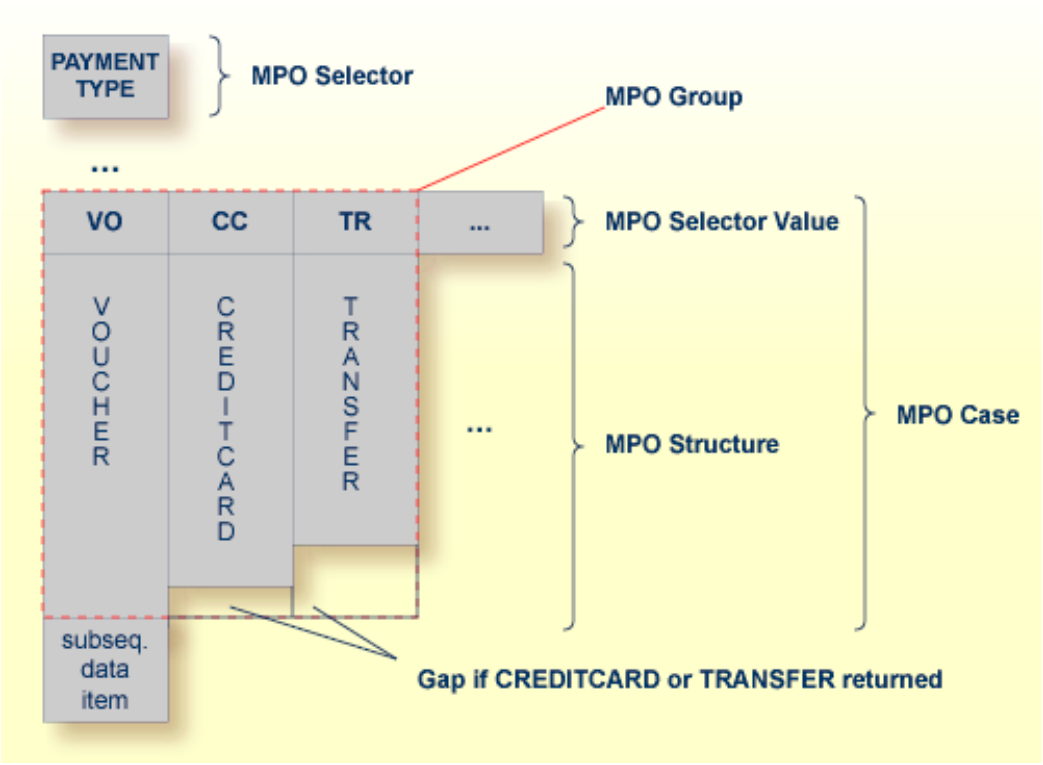
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>                PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>               PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

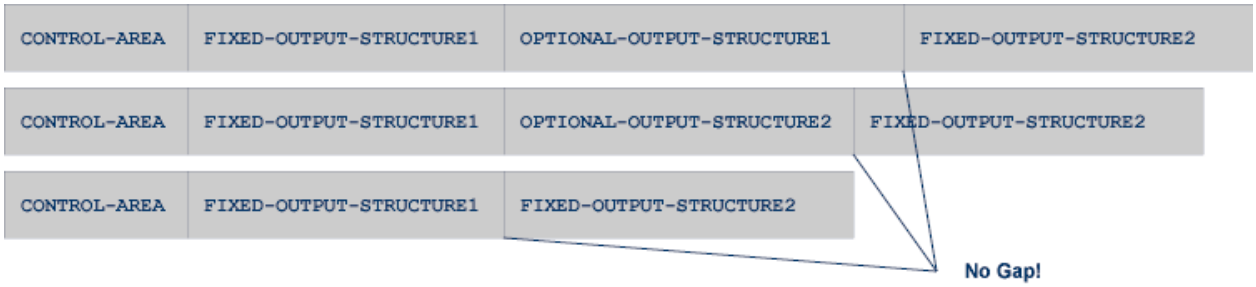
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

Note: If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

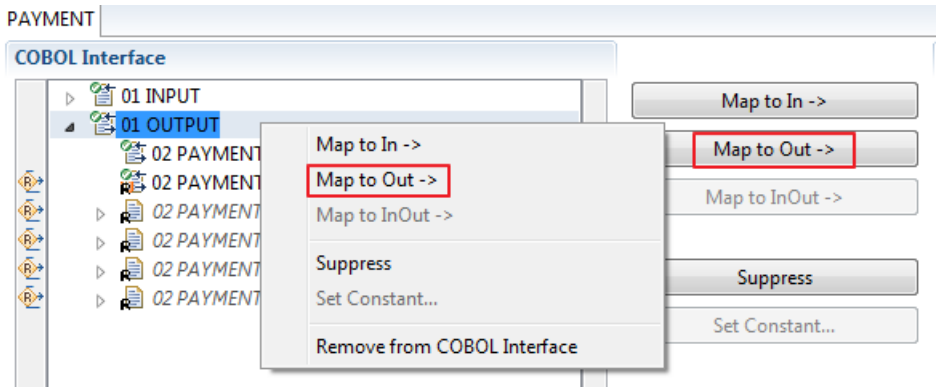
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

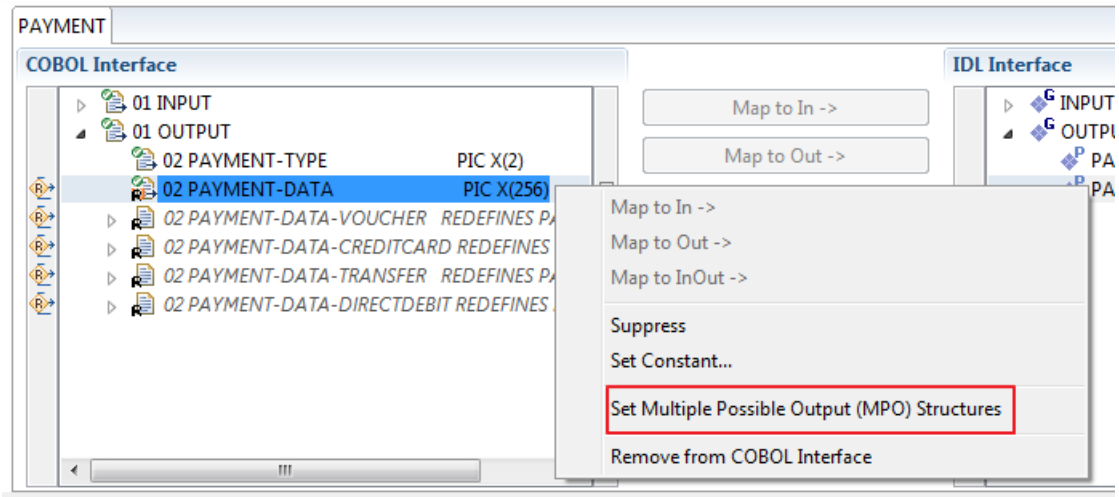
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

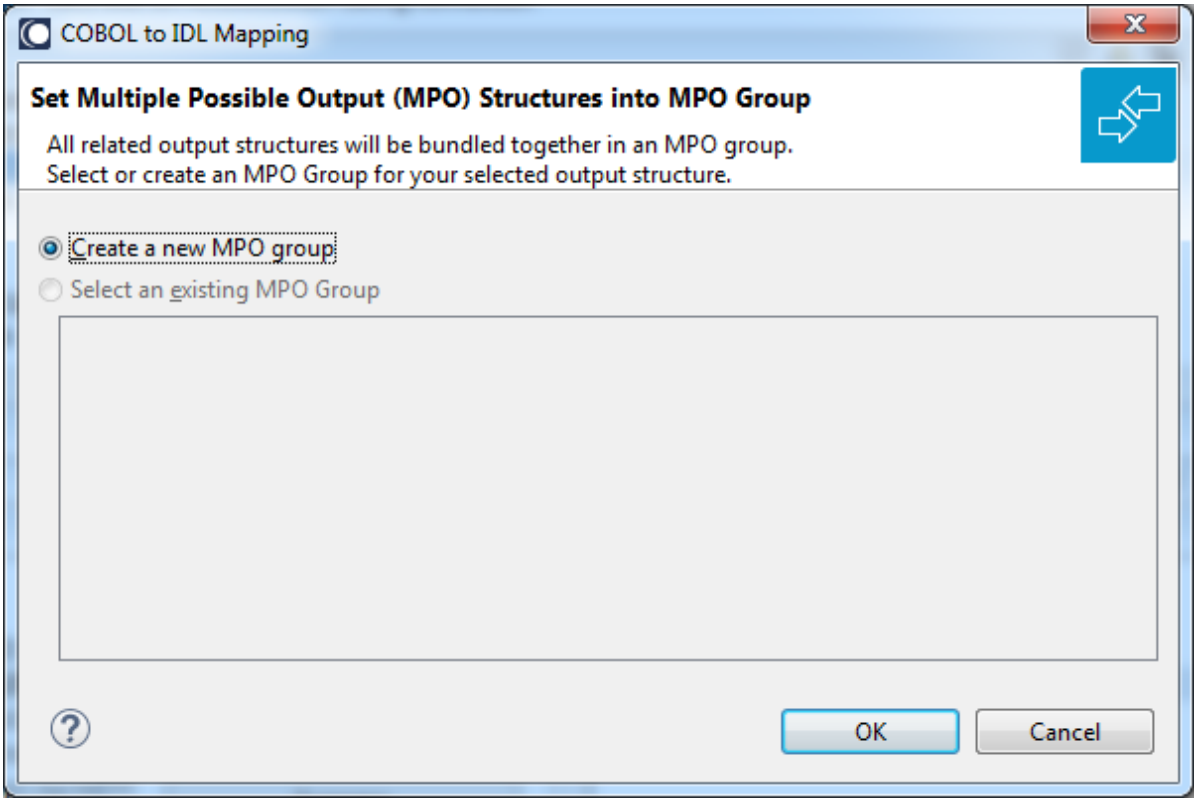
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



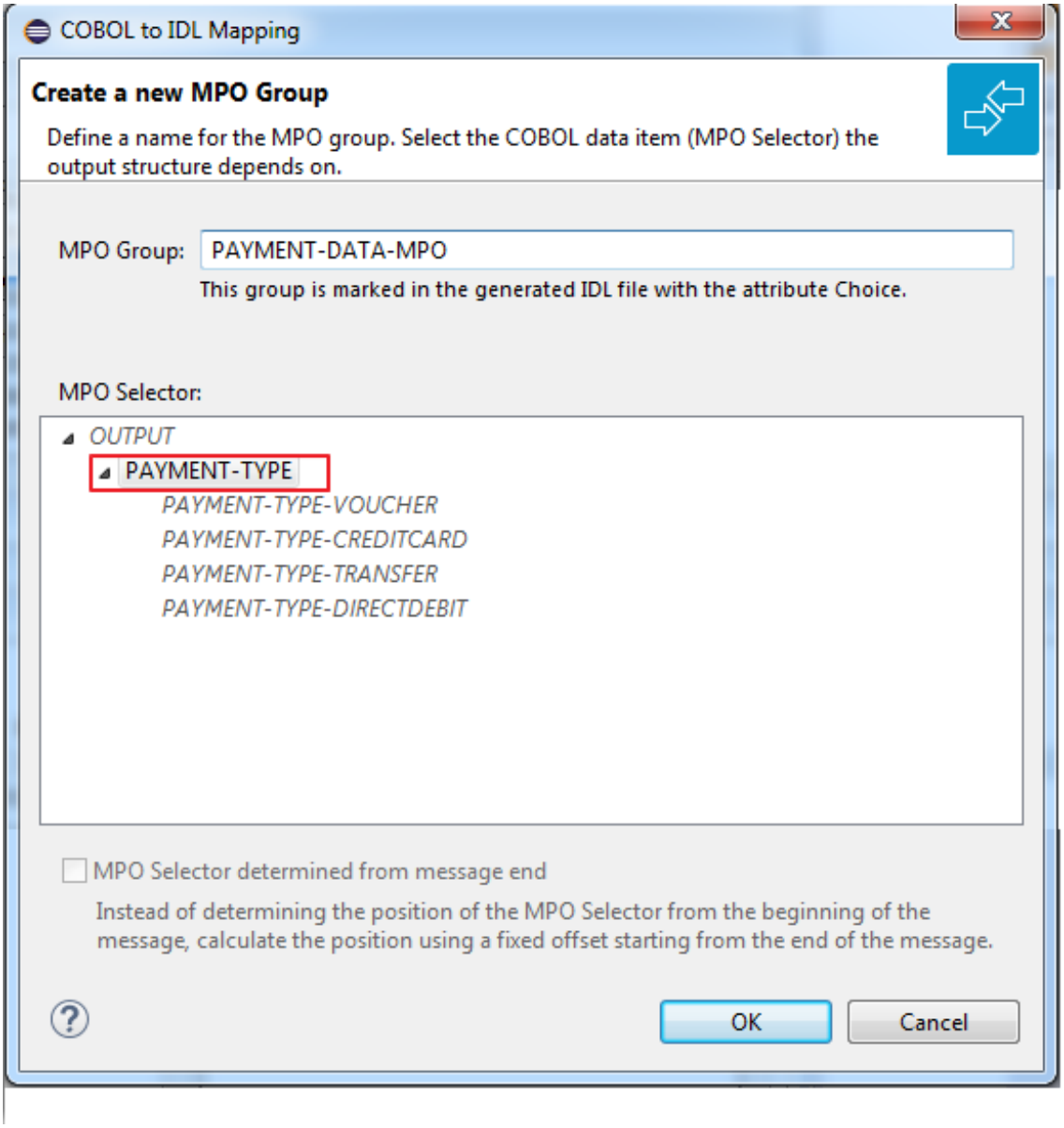
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



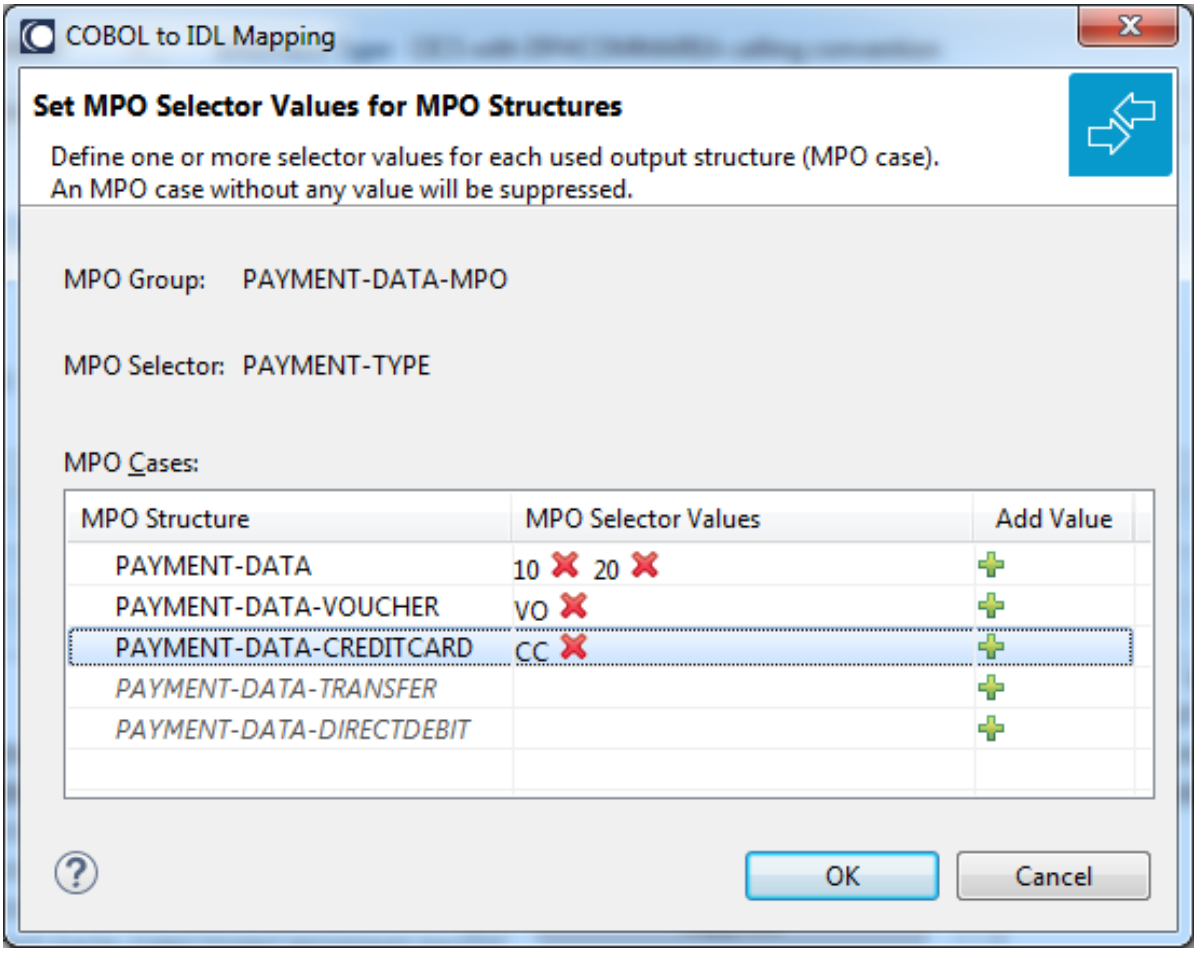
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



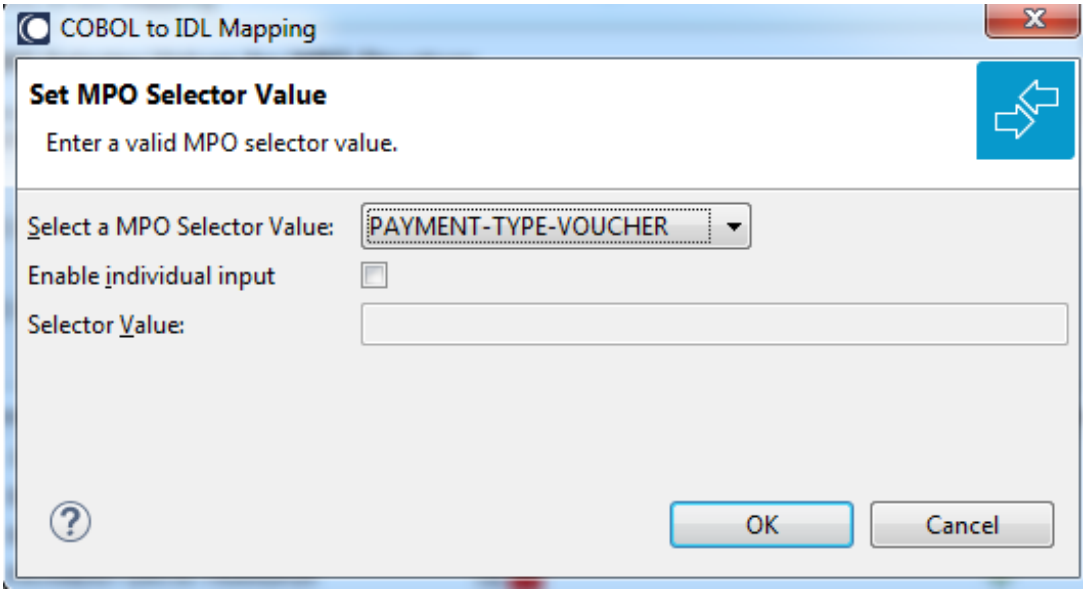
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

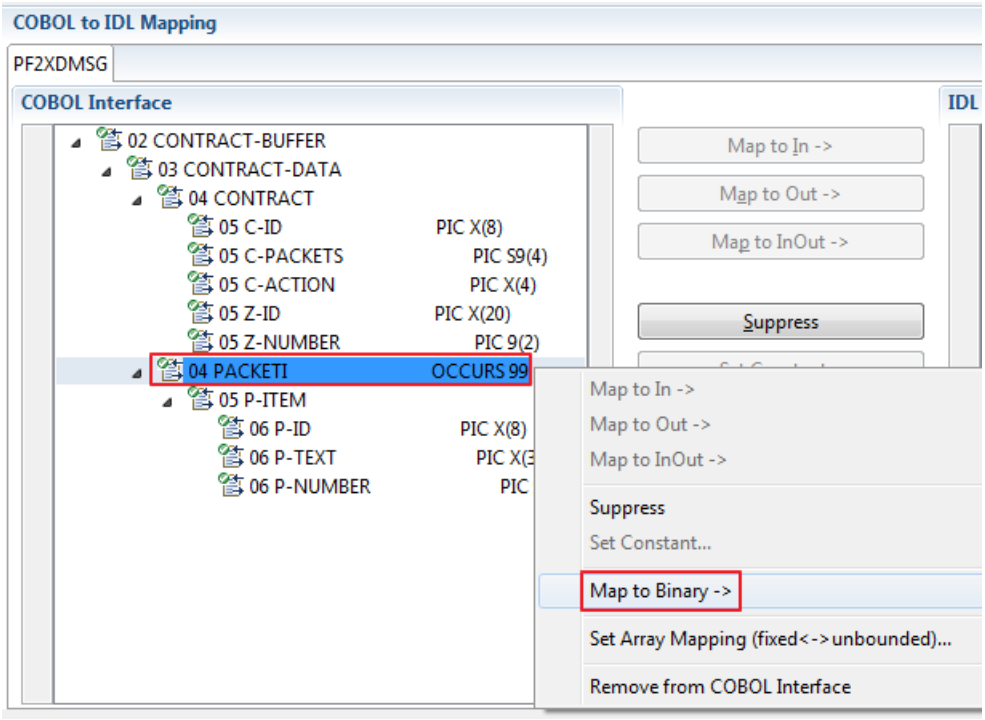
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

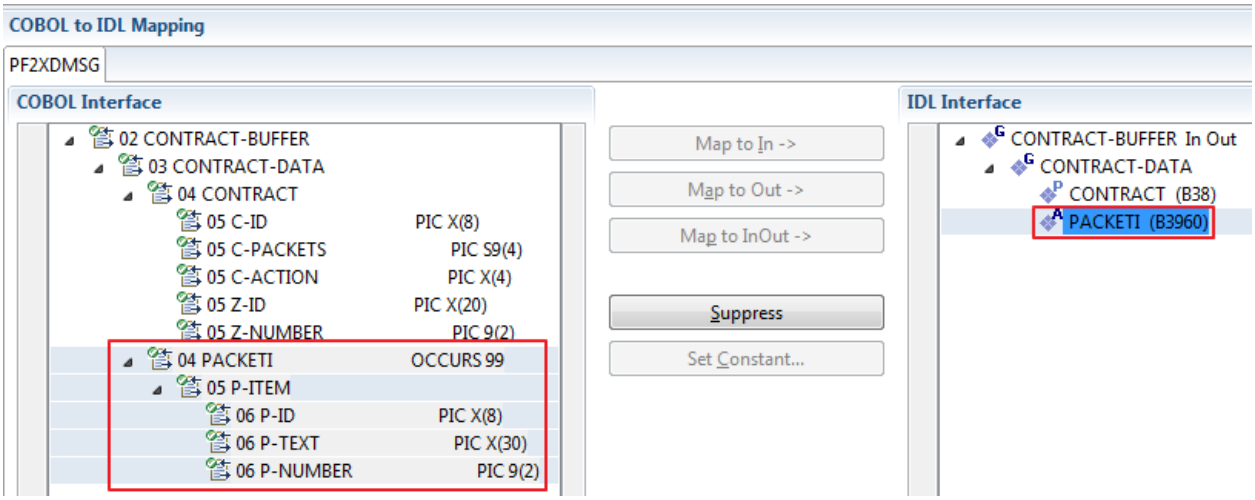
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.

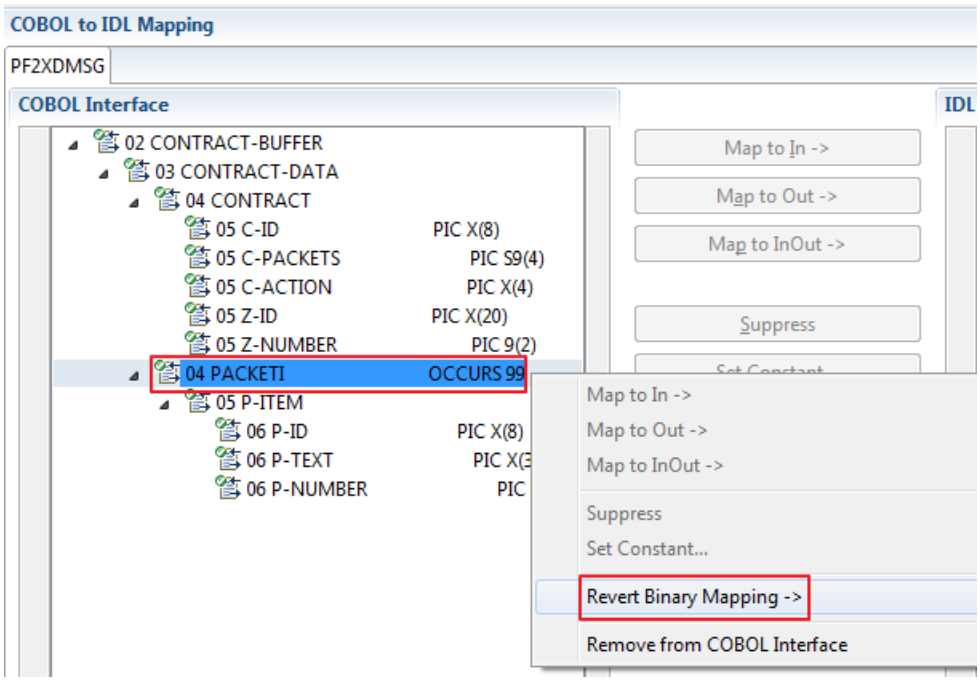


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

Programming Techniques

Example 1: Redefines

The output data is described with a `REDEFINE` that overlays the input data as in the following example. In this case you need to select `IN-BUFFER` for the input message and `OUT-BUFFER` for the output message of the COBOL interface. This technique is often used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat.

```
 LINKAGE SECTION.
 01 DFHCOMMAREA.

 02 IN-BUFFER.
    03 OPERATION                    PIC X(1).
    03 OPERAND-1                    PIC S9(9) BINARY.
    03 OPERAND-2                    PIC S9(9) BINARY.
 02 OUT-BUFFER REDEFINES IN-BUFFER.
    03 FUNCTION-RESULT              PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
* process the IN-BUFFER and provide result in OUT-BUFFER
    EXEC CICS RETURN.
```

`REDEFINE`s can also be used to describe a single buffer used for input and output, that is, the CICS input message is the same as the CICS output message. For more information see ***Example 1: Redefines*** in the section *CICS with `DFHCOMMAREA` Calling Convention - In same as Out*.

Example 2: Buffer Technique

On entry, the server moves linkage section field(s) - often an entire buffer - into the working storage and processes the input data inside the working storage field(s). Before return, it moves the working storage field(s) - often an entire buffer - back to the linkage section. In this case, the relevant COBOL data items are described within the working storage section. You need to select `IN-BUFFER` for the input message and `OUT-BUFFER` for the output message of the COBOL interface. This technique can be used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat.

```
 WORKING STORAGE SECTION
 01 IN-BUFFER.
    02 OPERATION                    PIC X(1).
    02 OPERAND-1                    PIC S9(9) BINARY.
    02 OPERAND-2                    PIC S9(9) BINARY.
 01 OUT-BUFFER.
    02 FUNCTION-RESULT              PIC S9(9) BINARY.
 LINKAGE SECTION
 01 DFHCOMMAREA.
    02 IO-BUFFER                    PIC X(9).
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
    MOVE IO-BUFFER TO IN-BUFFER.
* process the IN-BUFFER and provide result in OUT-BUFFER
    MOVE OUT-BUFFER TO IO-BUFFER.
    EXEC CICS RETURN.
```

The buffer technique can also be used to describe a single buffer used for input and output, that is, the CICS input message is the same as the CICS output message. For more information see ***Example 2: Buffer Technique*** in the section *CICS with* `DFHCOMMAREA` *Calling Convention - In same as Out*.

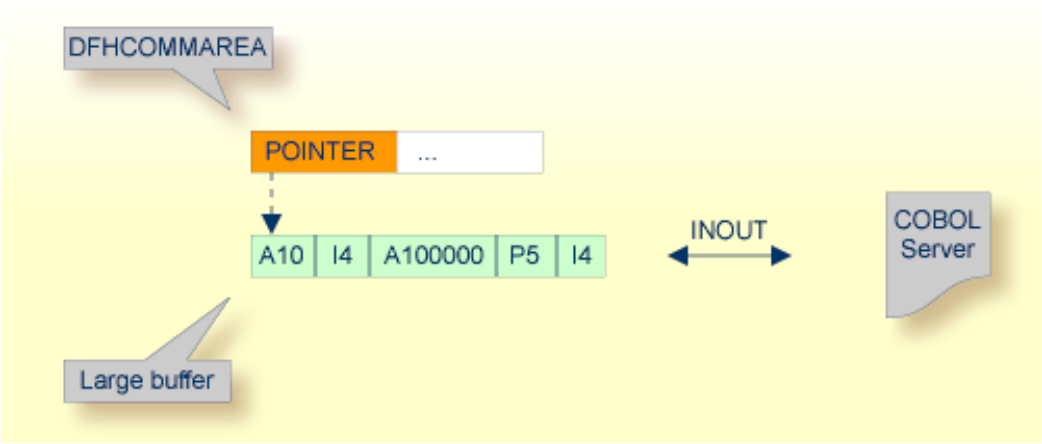
Example 3: COBOL SET ADDRESS Statements

COBOL `SET ADDRESS` statements are used to manipulate the interface of the CICS server. On entry, the server addresses the input data with a (dummy) structure `IN-BUFFER` defined in the linkage section. Upon return, the server addresses the output data again with a different (dummy) structure `OUT-BUFFER` defined in the linkage section. You need to select `IN-BUFFER` for the input message and `OUT-BUFFER` for the output message of the COBOL interface. This technique can be used to allow full 32K input and full 32K completely different output, thus circumventing CICS 32K restrictions somewhat.

```
 LINKAGE SECTION.
 01 IN-BUFFER.
    02 OPERATION                    PIC X(1).
    02 OPERAND-1                    PIC S9(9) BINARY.
    02 OPERAND-2                    PIC S9(9) BINARY.
 01 OUT-BUFFER.
    02 FUNCTION-RESULT              PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION.
    SET ADDRESS OF IN-BUFFER TO DFHCOMMAREA.
* process the IN-BUFFER and provide result in OUT-BUFFER
    SET ADDRESS OF OUT-BUFFER TO DFHCOMMAREA.
    EXEC CICS RETURN.
```

COBOL `SET ADDRESS` statements can also be used to describe a single buffer used for input and output, that is, the CICS input message is the same as the CICS output message. For more information see ***Example 3: COBOL*** `SET ADDRESS` ***Statements*** in the section *CICS with* `DFHCOMMAREA` *Calling Convention - In same as Out*.

8 CICS with DFHCOMMAREA Large Buffer Interface - In same as Out



Introduction

A `DFHCOMMAREA` Large Buffer Interface has the structure given below in the linkage section. The field subordinated under `DFHCOMMAREA` prefixed with `WM-LCB` describe this structure. The field names themselves can be different, but the COBOL data types (usage clauses) must match exactly. The COBOL server has one interface layout structure that is used for input as well as output.

```
 LINKAGE SECTION.

 01 DFHCOMMAREA.
    10 WM-LCB-MARKER                        PIC X(4).
    10 WM-LCB-INPUT-BUFFER                  POINTER.
    10 WM-LCB-INPUT-BUFFER-SIZE             PIC S9(8) BINARY.
    10 WM-LCB-OUTPUT-BUFFER                 POINTER.
    10 WM-LCB-OUTPUT-BUFFER-SIZE            PIC S9(8) BINARY.
    10 WM-LCB-FLAGS                         PIC X(1).
       88 WM-LCB-FREE-OUTPUT-BUFFER     VALUE 'F'.
    10 WM-LCB-RESERVED                      PIC X(3).
 01 INOUT-BUFFER.
    02 OPERATION                            PIC X(1).
    02 OPERAND-1                            PIC S9(9) BINARY.
    02 OPERAND-2                            PIC S9(9) BINARY.
    02 FUNCTION-RESULT                      PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
  . . .
    SET ADDRESS OF INOUT-BUFFER TO WM-LCB-INPUT-BUFFER.
*  process the INOUT-BUFFER and provide result
    EXEC CICS RETURN.
```

From a programming point of view, the COBOL server behaves as follows:

Variable	Description
WM-LCB-MARKER	Has eye-catcher "XXXX".
WM-LCB-INPUT-BUFFER	Has pointer to a buffer with COBOL server parameter data. This buffer is described by a COBOL layout structure.
WM-LCB-INPUT-BUFFER-SIZE	Contains size of COBOL server input parameter data.
WM-LCB-OUTPUT-BUFFER	Same as WM-LCB-INPUT-BUFFER.
WM-LCB-OUTPUT-BUFFER-SIZE	On input, same as WM-LCB-INPUT-BUFFER-SIZE. On return, the size must match the data length returned in the COBOL layout structure of the WM-LCB-OUTPUT-BUFFER. If the called COBOL server returns variable length data, that is, you have mapped *Map OCCURS DEPENDING ON* or ***Set Arrays (Fixed <-> Unbounded)***, and depending on your runtime architecture, consider the following: ■ **CICS Socket Listener (EntireX Adapter or RPC Server)** Providing a length considering the actual number of occurences instead of the maximum possible (which was provided on input), reduces network traffic and may improve performance. ■ **CICS RPC Server** Because in this architecture the marshalling is on-host, there will be no impact on network traffic, even if the provided length is set to the maximum possible number of occurences that was provided on input. If the called COBOL server returns fixed-length data, there is no need to change WM-LCB-OUTPUT-BUFFER-SIZE.
WM-LCB-FLAGS	On return, a value of 'F' in this flag indicates that the called COBOL server allocated an output buffer that had to be released by EntireX.

WM-LCB-OUTPUT-BUFFER and WM-LCB-FLAGS are normally not changed by the called COBOL server in this scenario.

If there is a need to return the output data in a different storage, this storage must be allocated by EXEC CICS GETMAIN. Return the new storage address in WM-LCB-OUTPUT-BUFFER. Indicate with WM-LCB-FLAGS='F' that the storage is released (EXEC CICS FREEMAIN) by EntireX.

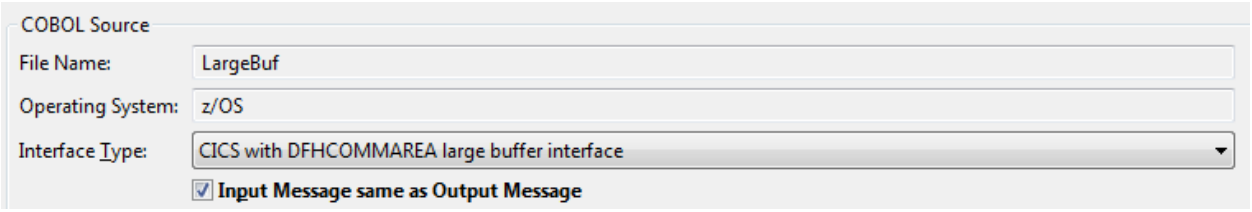
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with ***Mapping Editor User Interface***.

Extracting from a CICS DFHCOMMAREA Large Buffer Program

This section assumes **Input Message same as Output Message** is checked. COBOL output and COBOL input parameters are the same, that is, `WM-LCB-OUTPUT-BUFFER` is set to the same address as `WM-LCB-INPUT-BUFFER` (as in the `DFHCOMMAREA` large buffer example above).

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

Make sure the interface type is correct.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items of the large buffer to **COBOL Interface** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. To do this, locate in the `PROCEDURE DIVISION` the `SET ADDRESS OF <x> TO WM-LCB-INPUT-BUFFER` statement and the `SET ADDRESS OF <y> TO WM-LCB-OUTPUT-BUFFER` statement. The COBOL data items `<x>` and `<y>` are identical, and this is the large buffer you are looking for. See **Notes**.

2 Continue with *COBOL to IDL Mapping*.

Notes:

1. Do not select the pointers in the `DFHCOMMAREA` pointing to the large buffers, in the example above, `WM-LCB-INPUT-BUFFER` and `WM-LCB-OUTPUT-BUFFER`.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL interface contains `REDEFINE`s, 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.

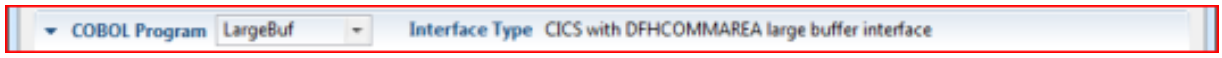
The user interface of the COBOL Mapping Editor is described below.

Mapping Editor User Interface

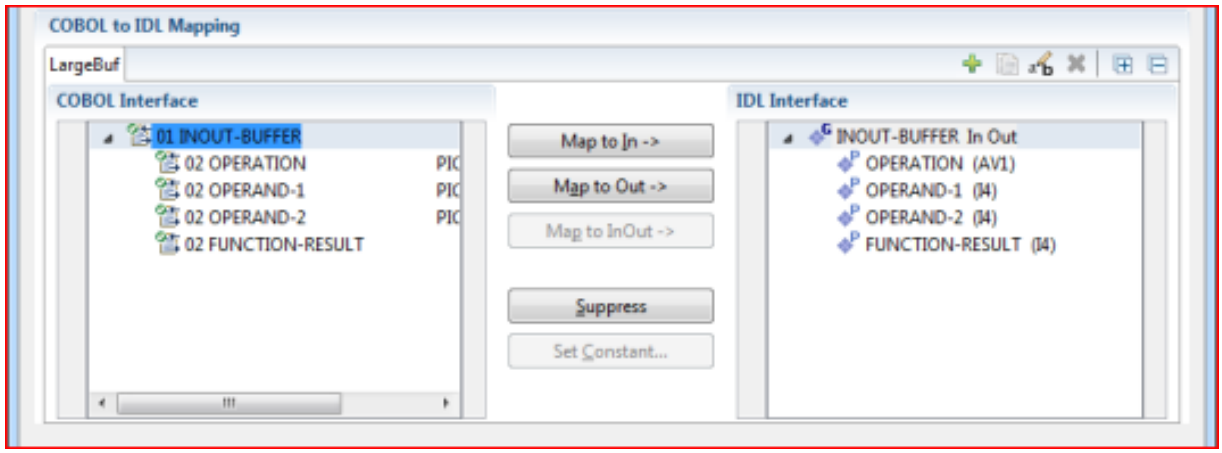
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface type CICS with DFHCOMMAREA large buffer interface, the user interface of the COBOL Mapping Editor looks like this:

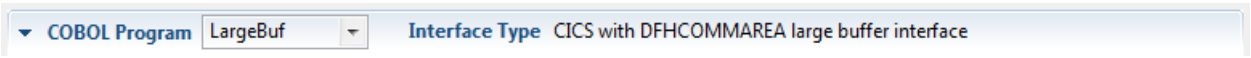
1. 

2. 

3. 

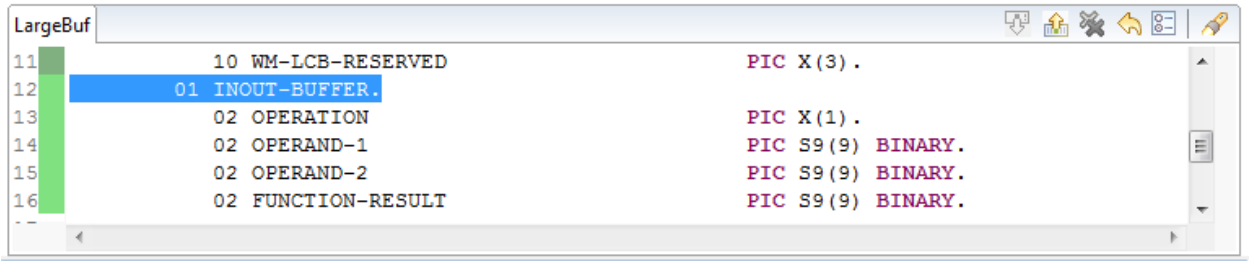
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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.
- 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

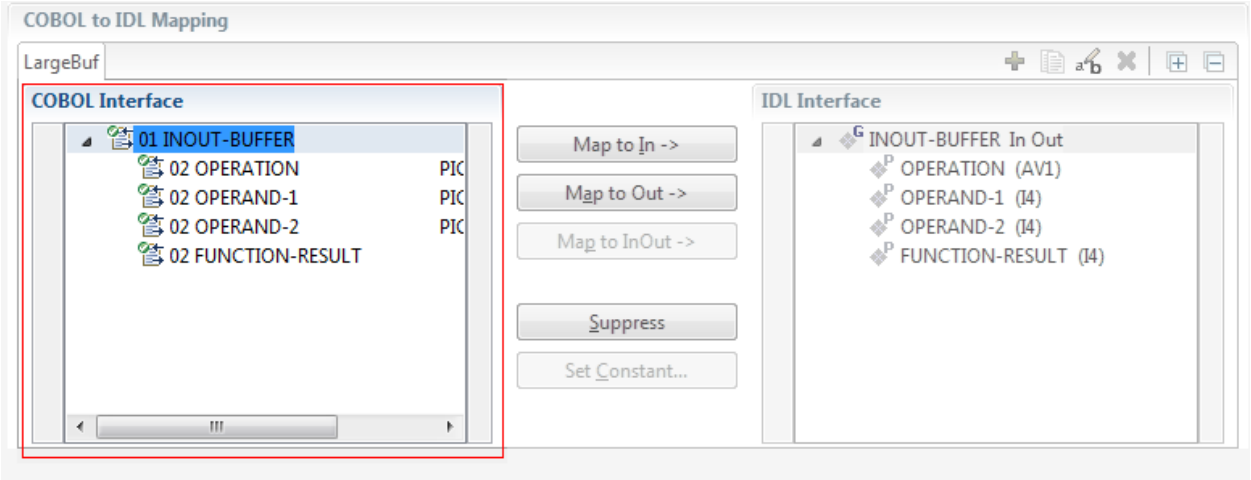
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

Map to In | Out | InOut	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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

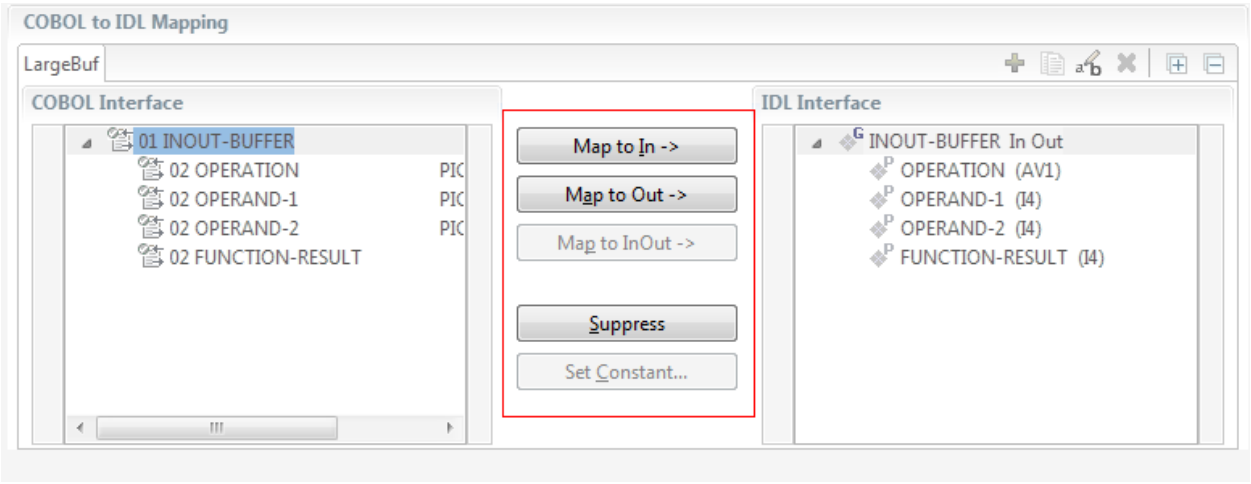
Mapping Icons

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

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

Mapping Buttons

The following buttons are available:



Map to In | Out | InOut ->

See *Map to In, Out, InOut*. 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*.

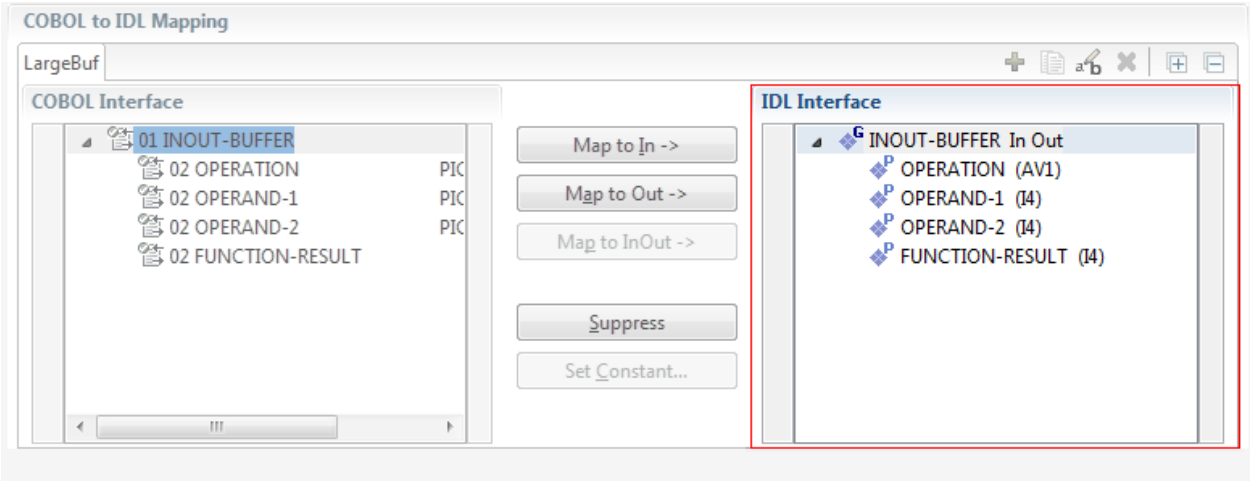
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

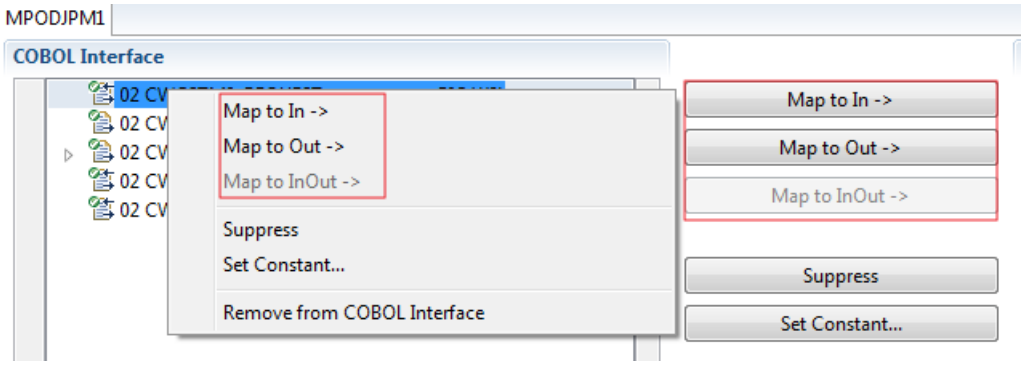
This section covers the following topics:

Map to In, Out, InOut

With the **Map to In**, **Out**, **InOut** functions you make a COBOL data item visible as an IDL parameter in the IDL interface. With correct IDL directions you design the IDL interface by defining input and output parameters. COBOL programs have no parameter directions, so you need to set IDL directions manually.

To provide IDL directions

- Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons to make the COBOL data items visible and provide IDL directions in the IDL interface:



Notes:

1. If a *top-level* COBOL *group* is mapped, the IDL direction is inherited by all subordinate COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subordinate COBOL data items can only be mapped to the same IDL direction as their *top-level* COBOL *group* data item.
3. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.
4. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Map OCCURS DEPENDING ON

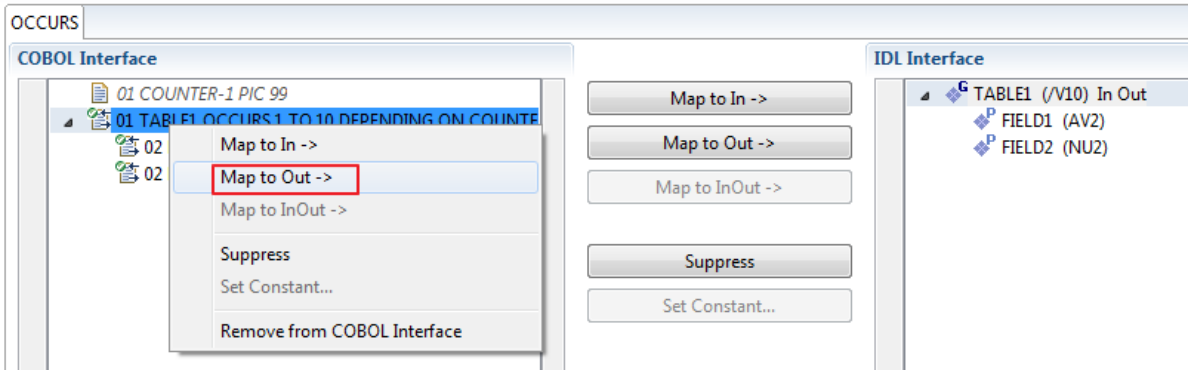
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - DEPENDING ON Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):

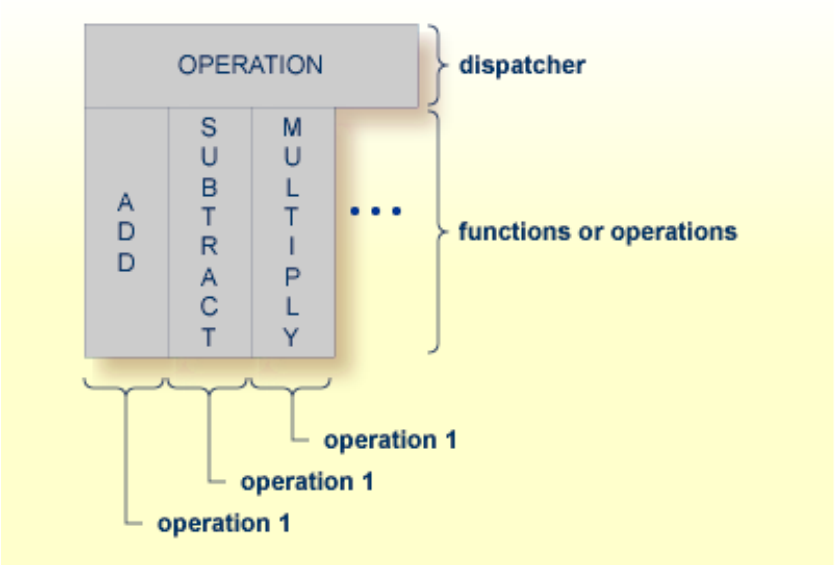


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "-"
```

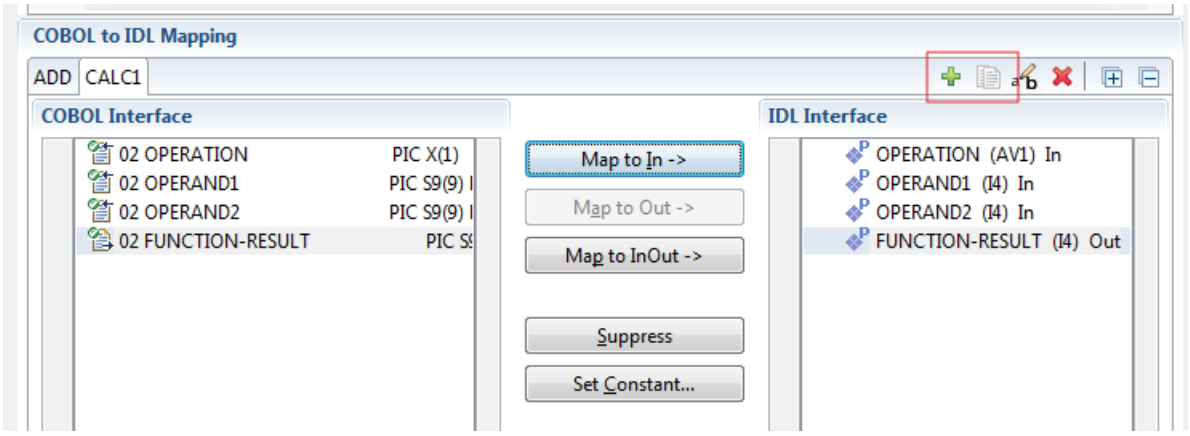
```
            SUBTRACT OPERAND2 FROM OPERAND1
            GIVING FUNCTION-RESULT
         WHEN "*"
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

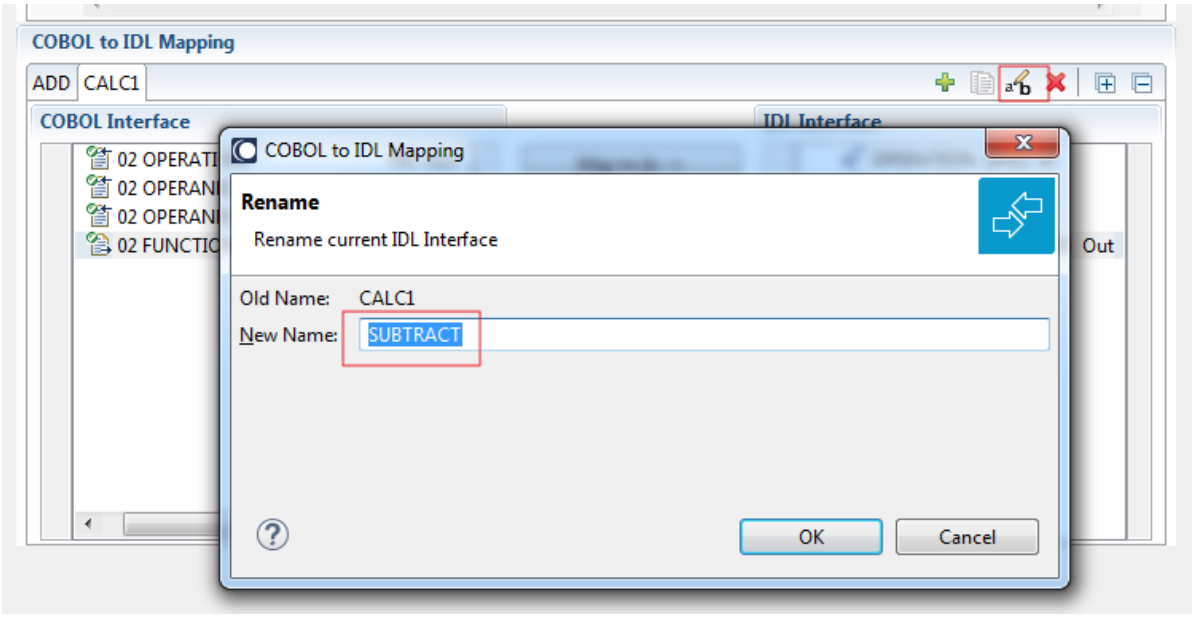
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

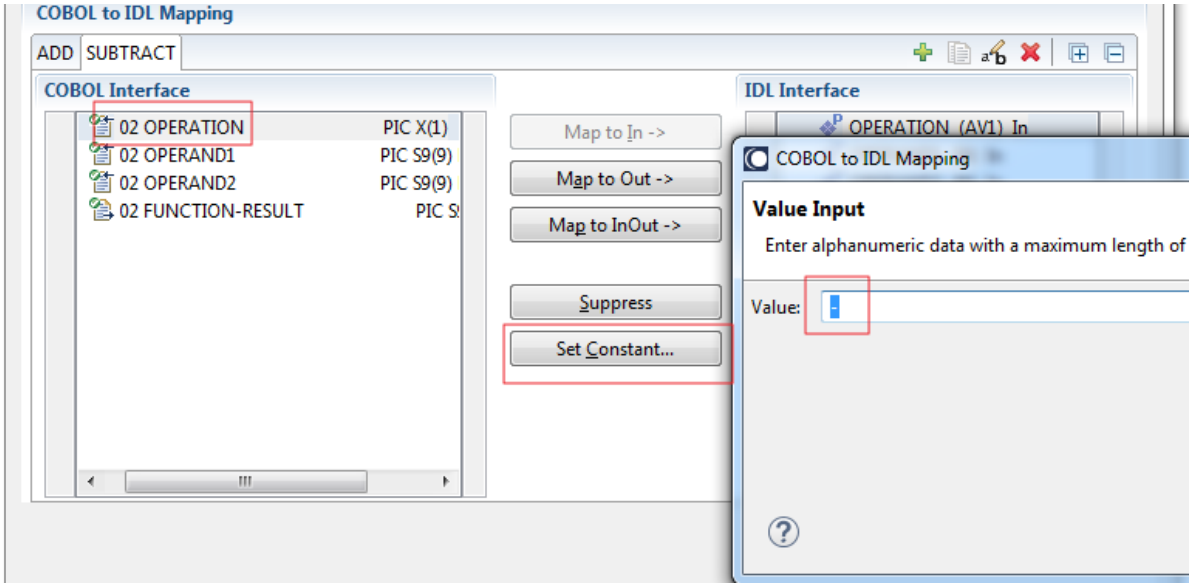
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

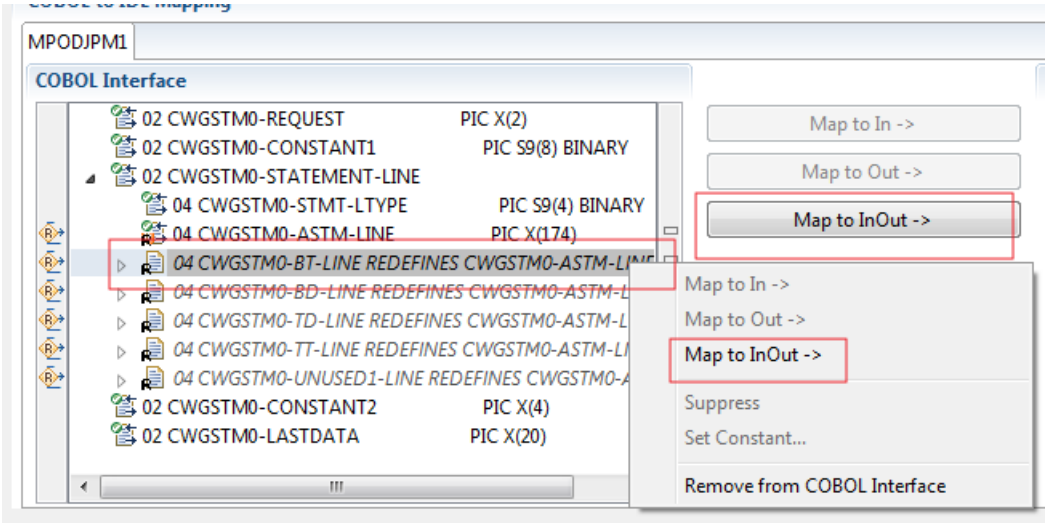
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

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

To select redefine paths

- Use the **Map to In**, **Out** or **InOut** functions available in the context menu of the COBOL interface and as mapping buttons 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:

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

Suppress Unneeded COBOL Data Items

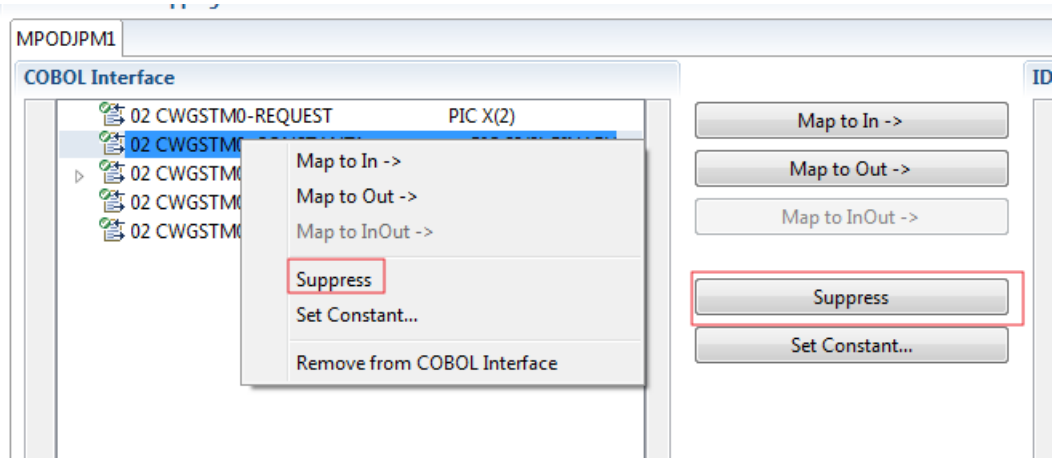
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface and as mapping button to make the COBOL data item invisible in the IDL interface:



Notes:

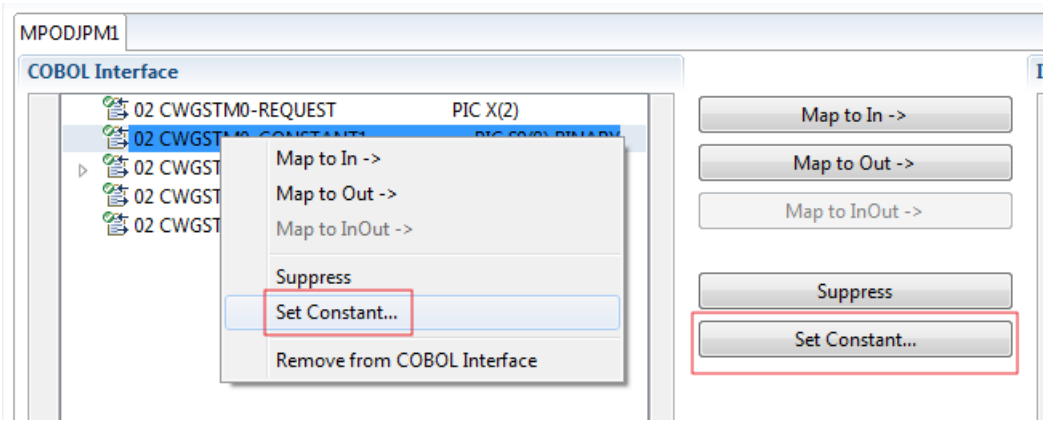
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subordinate COBOL data items are suppressed as well.
4. With the inverse functions **Map to In**, **Out** or **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item is made visible in the IDL interface again.

Set COBOL Data Items to Constants

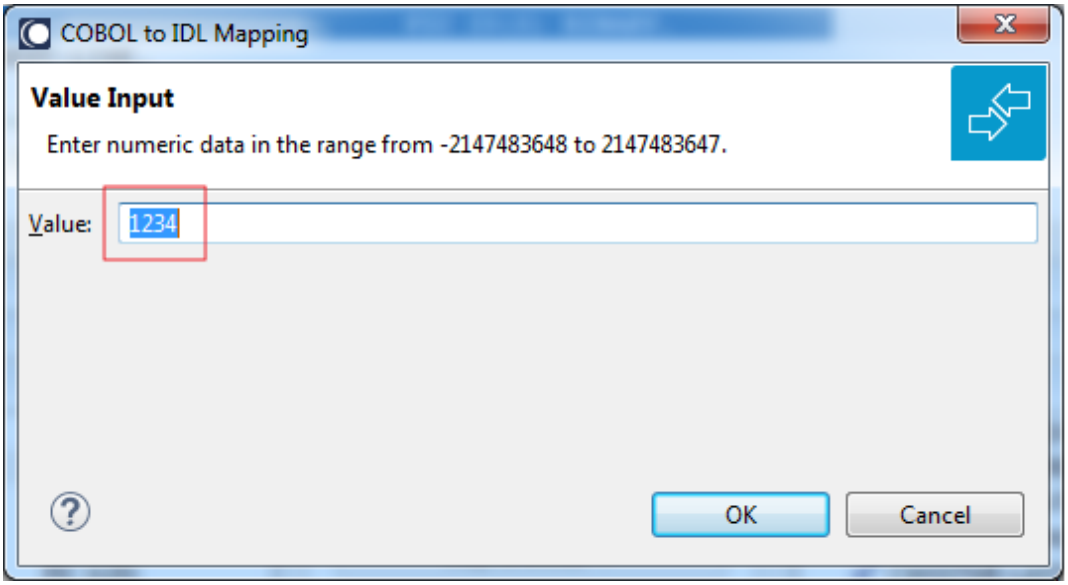
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter provides the defined constant in the COBOL data item to your COBOL server.
3. With the functions **Map to In**, **Out**, **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item can be made visible in the IDL interface again.

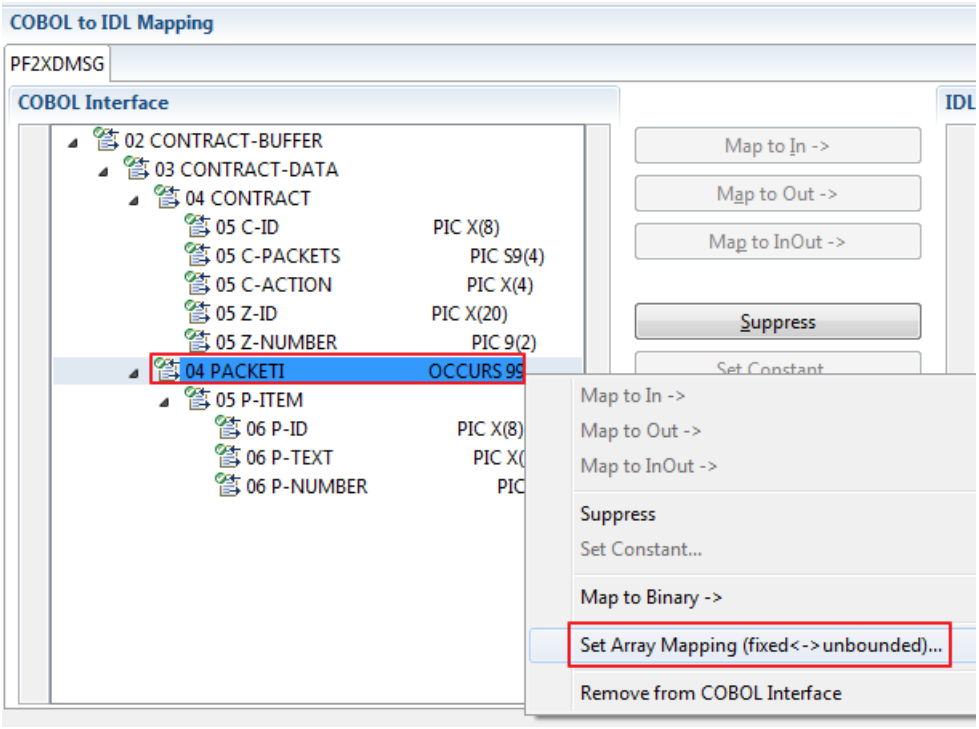
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

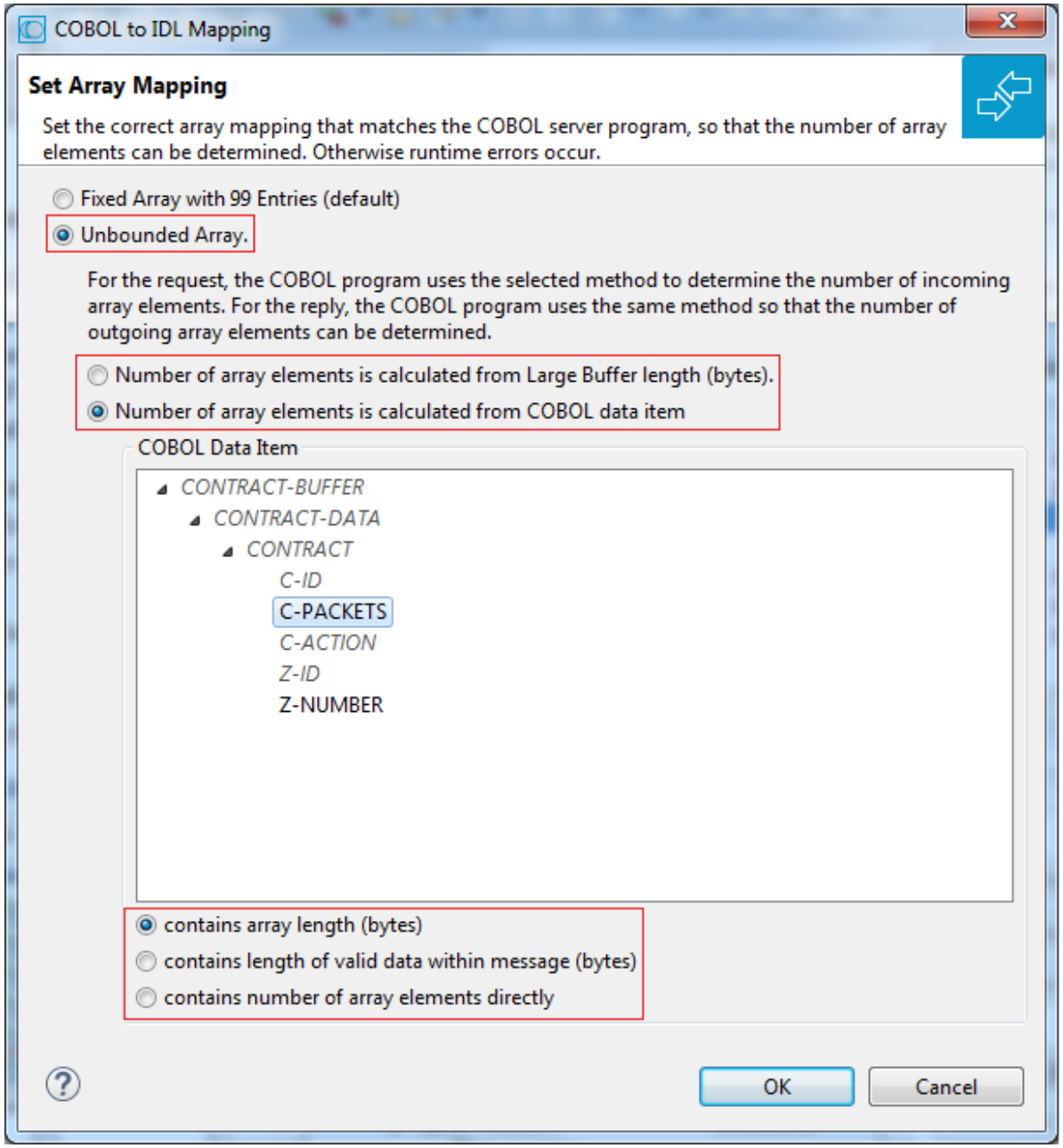
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **Large Buffer Length (bytes)**
  The COBOL server program inspects `WM-LCB-INPUT-BUFFER-SIZE` (large buffer length for input) for the request and sets `WM-LCB-OUTPUT-BUFFER-SIZE` (large buffer length for output) for the reply. To determine the number of array elements, the large buffer length is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the reply of a large buffer program. It assumes `CONTRACT-BUFFER` with fix array `PACKETI` is the large buffer.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
          10 WM-LCB-MARKER                        PIC X(4).
          10 WM-LCB-INPUT-BUFFER                  POINTER.
          10 WM-LCB-INPUT-BUFFER-SIZE             PIC S9(8) BINARY.
          10 WM-LCB-OUTPUT-BUFFER                 POINTER.
          10 WM-LCB-OUTPUT-BUFFER-SIZE            PIC S9(8) BINARY.
          10 WM-LCB-FLAGS                         PIC X(1).
             88 WM-LCB-FREE-OUTPUT-BUFFER         VALUE "F".
          10 WM-LCB-RESERVED                      PIC X(3).
         01 CONTRACT-BUFFER.
            04 CONTRACT.
             05 C-ID                              PIC X(8).
             05 C-ACTION                          PIC X(4).
            04 ZONE.
             05 Z-NUMBER                          PIC 9(2).
             05 Z-ID                              PIC X(20).
            04 PACKETI                            OCCURS 99.
             05 P-ID                              PIC X(8).
             05 P-TEXT                            PIC X(30).
             05 P-NUMBER                          PIC 9(2).
          . . .

          *  Fill variable output array
             MOVE 0 TO II.
             PERFORM RANDOMNUM TIMES
                ADD 1 TO II
                MOVE ... TO P-ID    (II)
                MOVE ... TO P-TEXT  (II)
                MOVE ... TO P-NUMBER(II)
             END-PERFORM.
          *  Set large buffer length depending on number of elements
             COMPUTE WM-LCB-OUTPUT-BUFFER-SIZE =
                    (LENGTH OF P-ID  +
                     LENGTH OF P-TEXT   +
                     LENGTH OF P-NUMBER) * II.
             ADD LENGTH OF CONTRACT TO WM-LCB-OUTPUT-BUFFER-SIZE.
             ADD LENGTH OF ZONE     TO WM-LCB-OUTPUT-BUFFER-SIZE.

             EXEC CICS RETURN END-EXEC.
```

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The length of the fixed-size array `PACKETI` is contained in COBOL data item `C-BYTES`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-BYTES                            PIC S9(4).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
            05 Z-ID                               PIC X(20).
           04 PACKETI                             OCCURS 99.
            05 P-ITEM.
             06 P-ID                              PIC X(8).
             06 P-TEXT                            PIC X(30).
             06 P-NUMBER                          PIC 9(2).
      . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set table length
        COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT` is filled by the COBOL server on reply. COBOL data item `C-APPDATA` contains the length of the valid data of the reply message. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

```
       WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
        77 EPARM                                 PIC  9(2).
        77 EPARM2                                PIC  9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-APPDATA                         PIC S9(4).
            05 C-ACTION                          PIC X(4).
            05 Z-ID                              PIC X(20).
            05 Z-NUMBER                          PIC 9(2).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
       . . .
      *  Fill variable output array
         MOVE 0 TO II.
         PERFORM RANDOMNUM TIMES
            ADD 1 TO II
            MOVE ... TO P-ID    (II)
            MOVE ... TO P-TEXT  (II)
            MOVE ... TO P-NUMBER(II)
         END-PERFORM.
      *  Set length
         COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                           +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**

  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-NUM                             PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
```

```
           05 Z-ID                                  PIC X(20).
          04 PACKETI                               OCCURS 99.
           05 P-ITEM.
            06 P-ID                                PIC X(8).
            06 P-TEXT                              PIC X(30).
            06 P-NUMBER                            PIC 9(2).
     . . .

    *  Fill variable output array
       MOVE 0 TO II.
       PERFORM RANDOMNUM TIMES
          ADD 1 TO II
          MOVE ... TO P-ID    (II)
          MOVE ... TO P-TEXT  (II)
          MOVE ... TO P-NUMBER(II)
       END-PERFORM.
    *  Set occurrences
       MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

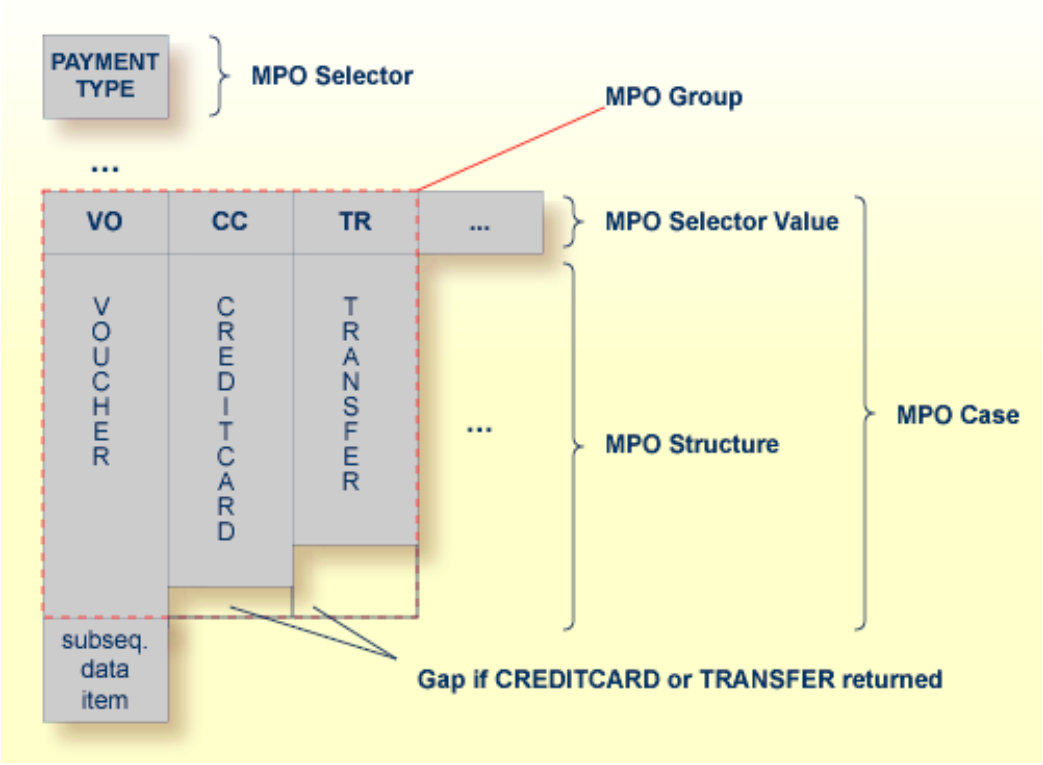
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>               PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>              PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

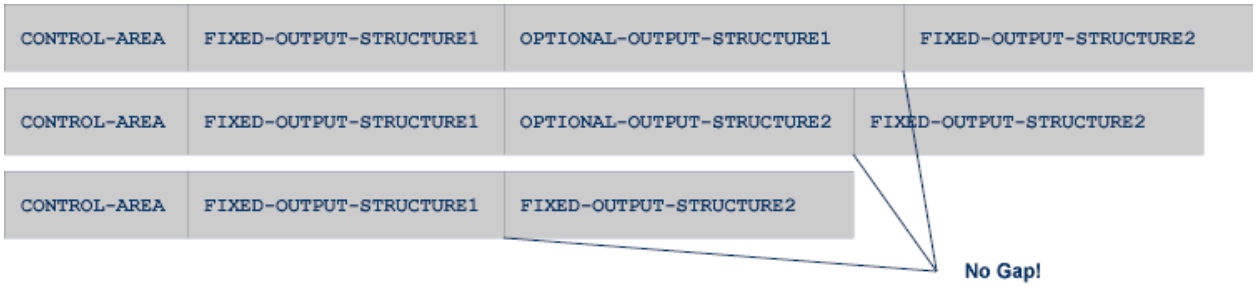
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

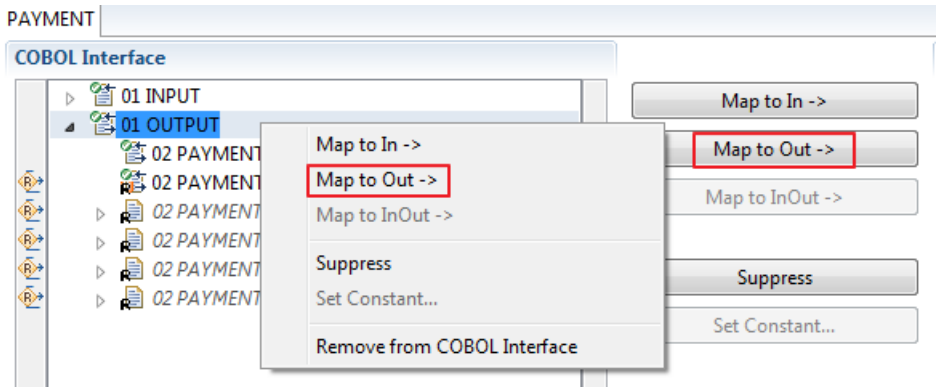
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

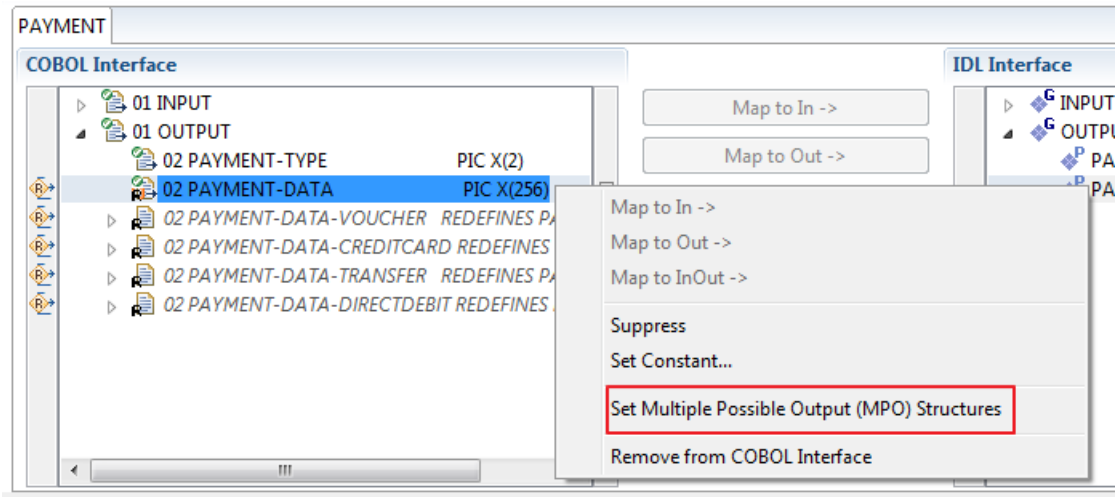
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

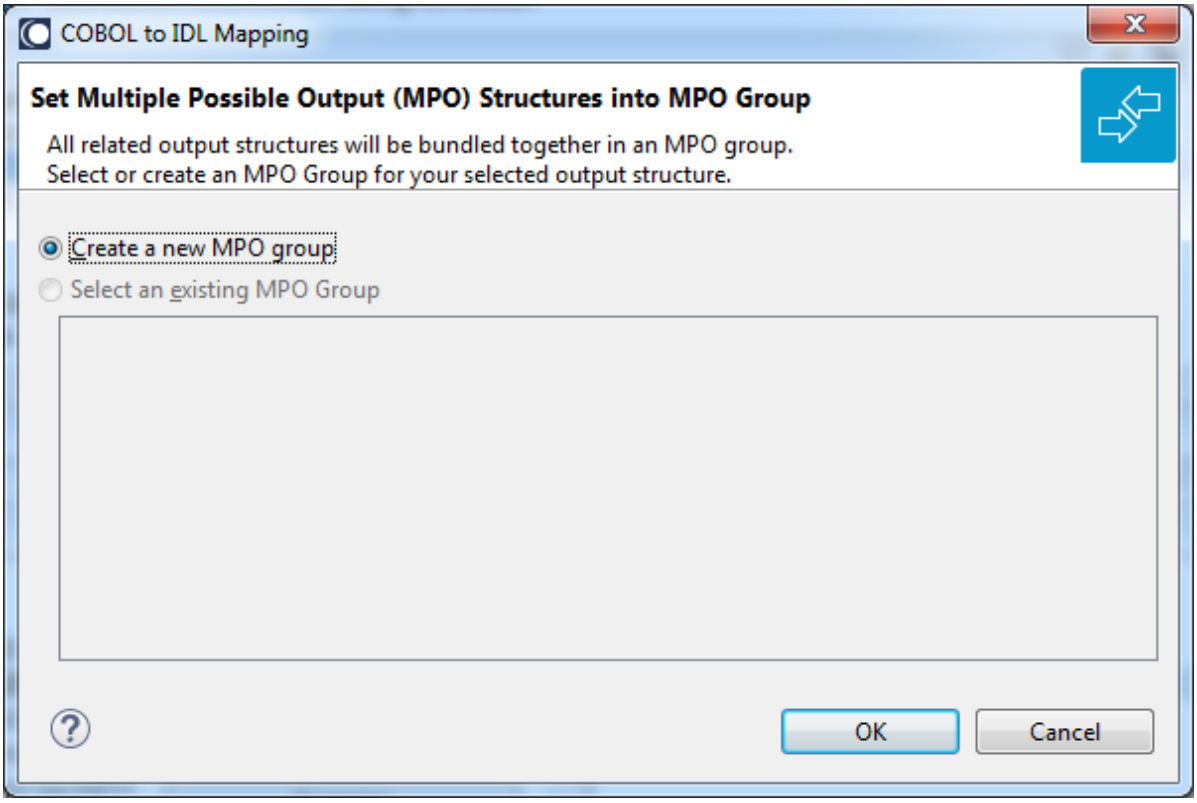
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



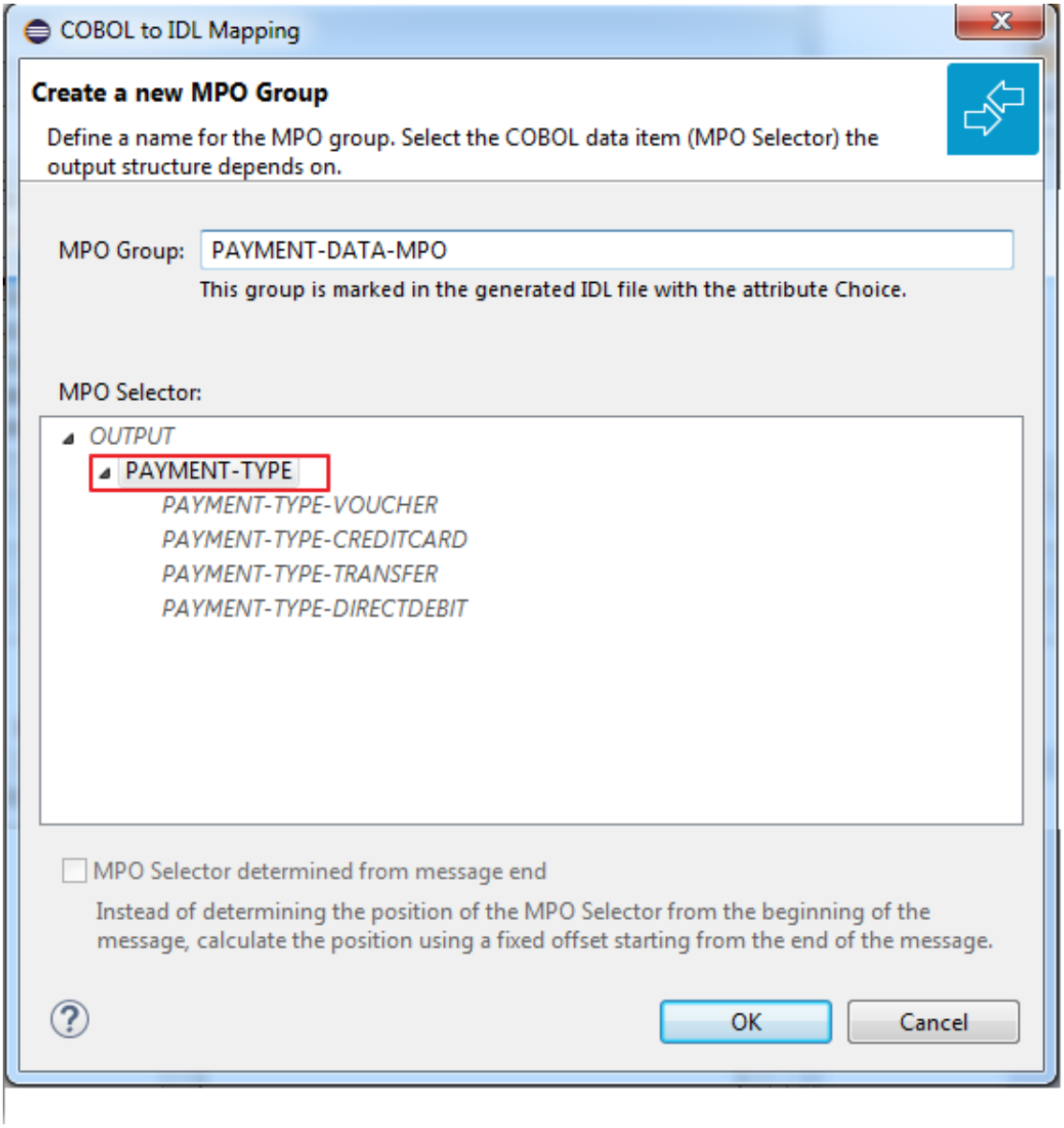
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



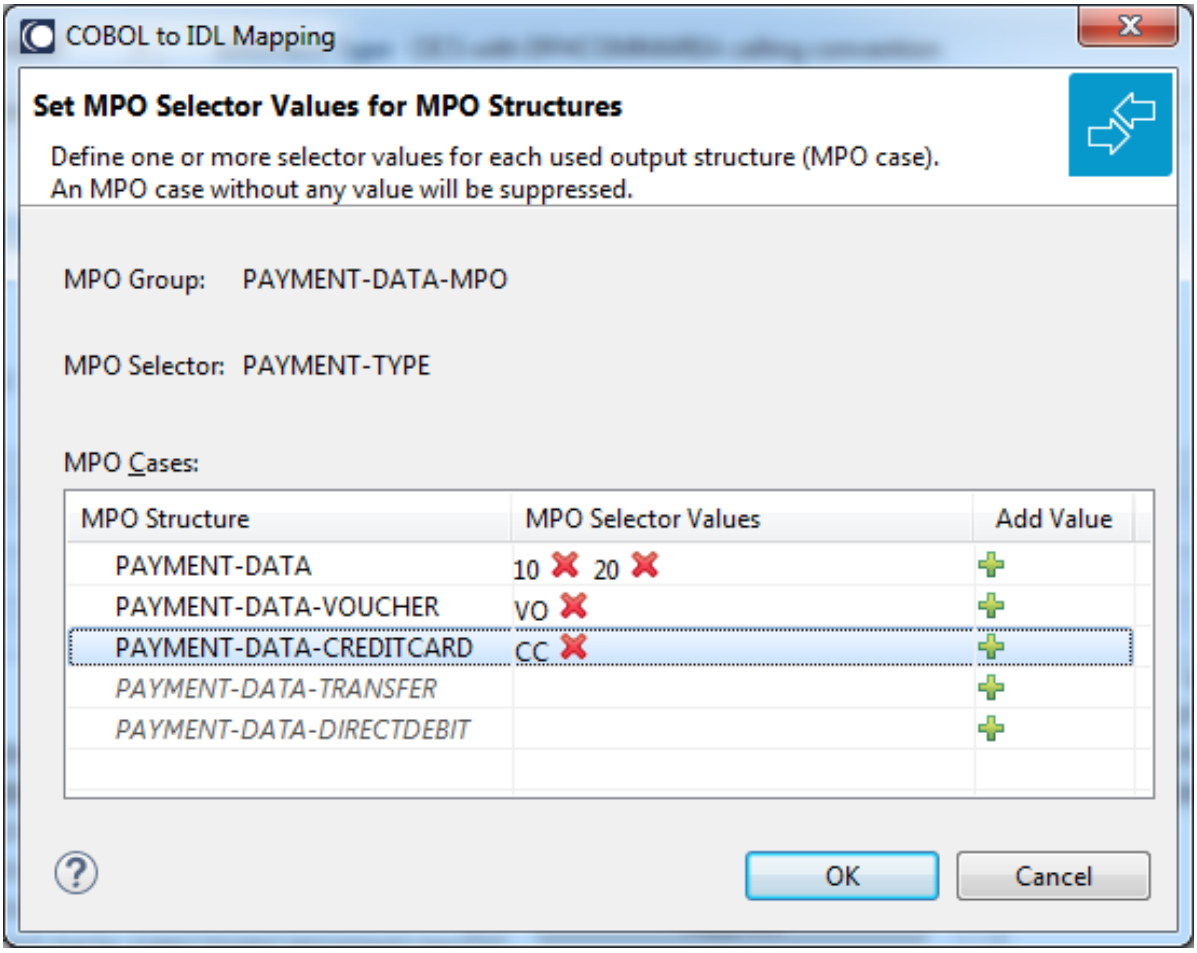
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



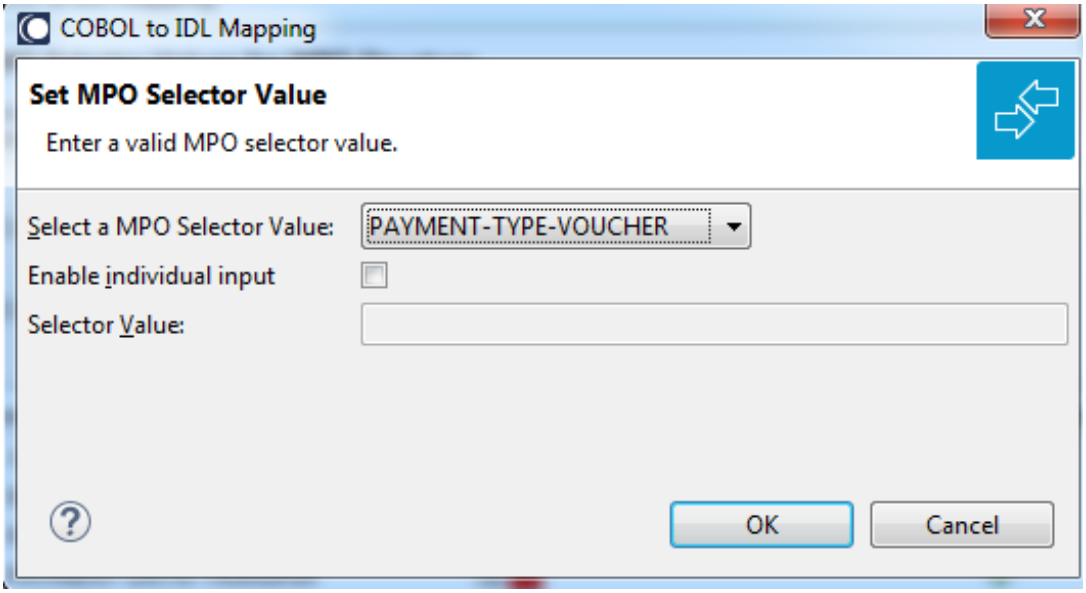
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

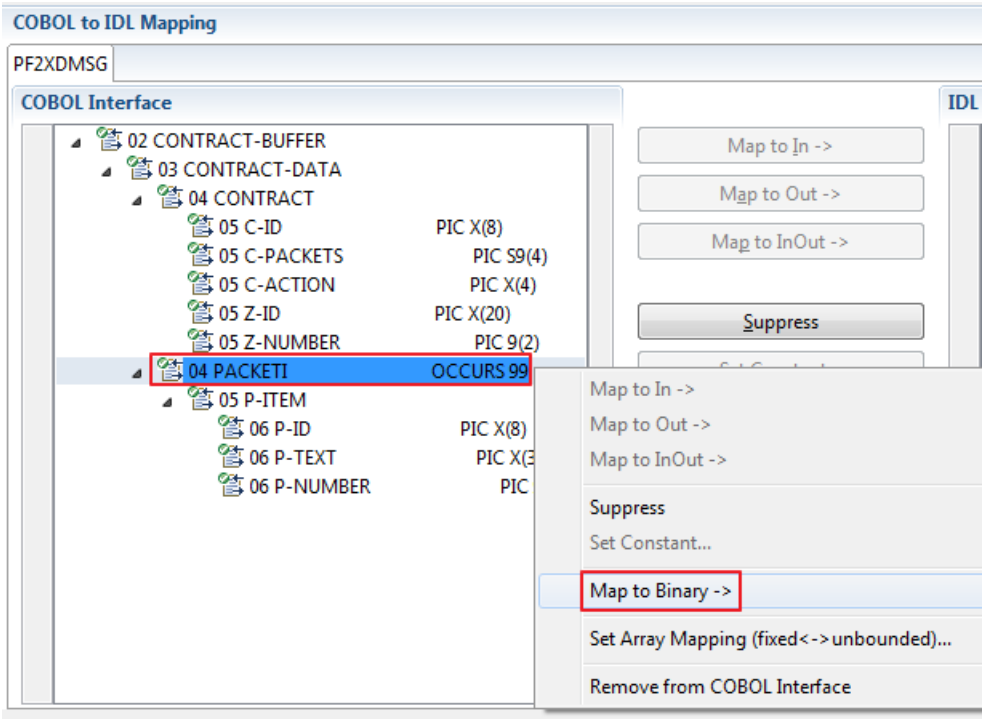
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

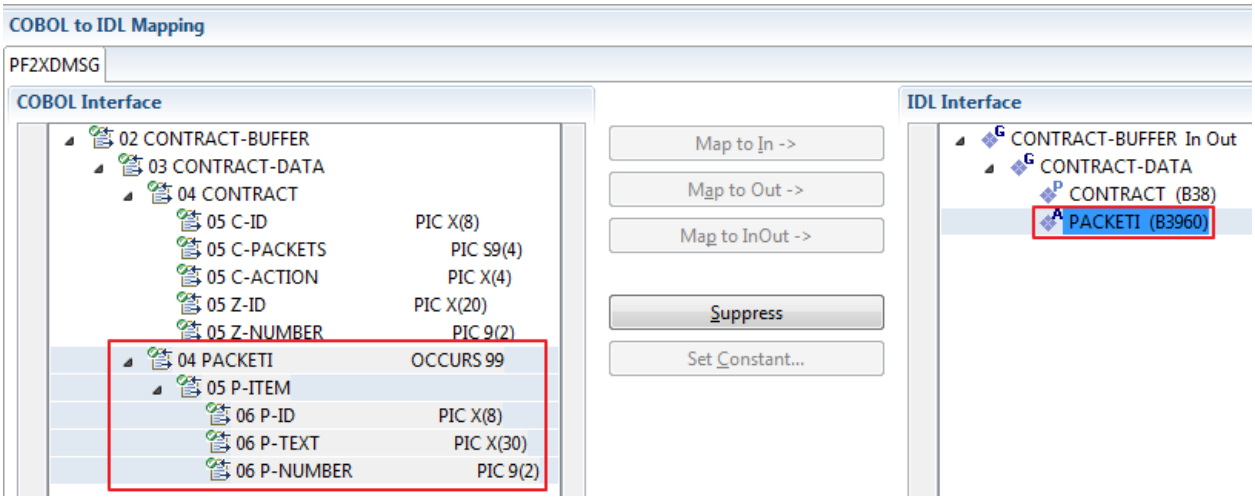
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.

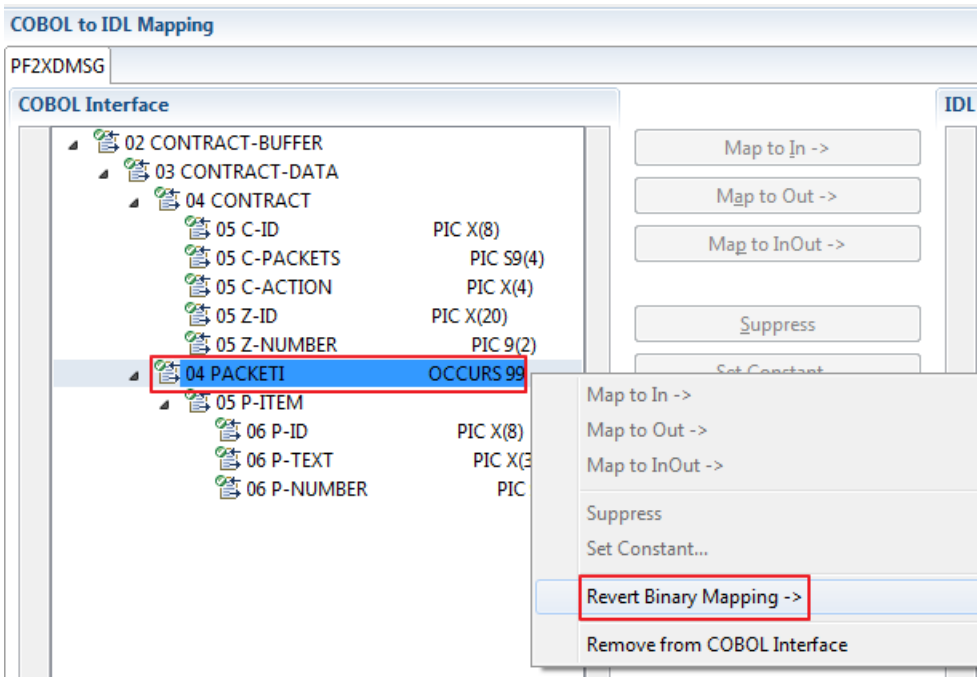


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

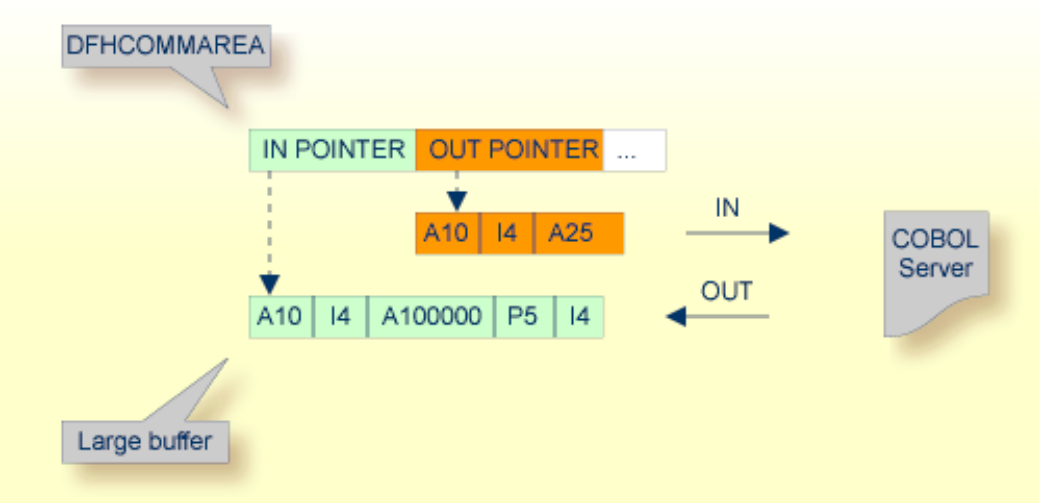
When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

9 CICS with DFHCOMMAREA Large Buffer Interface - In different to Out



Introduction

A DFHCOMMAREA Large Buffer Interface has the structure given below in the linkage section. The field subordinated under DFHCOMMAREA prefixed with WM-LCB describe this structure. The field names themselves can be different, but the COBOL data types (usage clauses) must match exactly. The COBOL server has two fixed interface layout structures: one for input, the other for output.

```
LINKAGE SECTION.
 01 DFHCOMMAREA.
   10 WM-LCB-MARKER                    PIC X(4).
   10 WM-LCB-INPUT-BUFFER              POINTER.
   10 WM-LCB-INPUT-BUFFER-SIZE         PIC S9(8) BINARY.
   10 WM-LCB-OUTPUT-BUFFER             POINTER.
   10 WM-LCB-OUTPUT-BUFFER-SIZE        PIC S9(8) BINARY.
   10 WM-LCB-FLAGS                     PIC X(1).
      88 WM-LCB-FREE-OUTPUT-BUFFER  VALUE 'F'.
   10 WM-LCB-RESERVED                  PIC X(3).
 01 IN-BUFFER.
    02 OPERATION                       PIC X(1).
    02 OPERAND-1                       PIC S9(9) BINARY.
    02 OPERAND-2                       PIC S9(9) BINARY.
 01 OUT-BUFFER.
    02 FUNCTION-RESULT                 PIC S9(9) BINARY.
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
  . . .
    SET ADDRESS OF IN-BUFFER  TO WM-LCB-INPUT-BUFFER.
    SET ADDRESS OF OUT-BUFFER TO WM-LCB-OUTPUT-BUFFER.
* process the IN-BUFFER and provide result in OUT-BUFFER
    EXEC CICS RETURN.
```

From a programming point of view, the COBOL server behaves as follows:

Variable	Description
WM-LCB-MARKER	Has eye-catcher "XXXX".
WM-LCB-INPUT-BUFFER	Has pointer to a buffer with COBOL server input parameter data. This buffer is described by a COBOL layout structure.
WM-LCB-INPUT-BUFFER-SIZE	Contains size of COBOL server input parameter data.
WM-LCB-OUTPUT-BUFFER	Has pointer to a buffer with length WM-LCB-OUTPUT-BUFFER-SIZE, where the COBOL server writes its output parameter data. This buffer is described by a COBOL layout structure.
WM-LCB-OUTPUT-BUFFER-SIZE	On input, size of WM-LCB-OUTPUT-BUFFER. On return, the size must match the data length returned in the COBOL layout structure of the WM-LCB-OUTPUT-BUFFER. If the called COBOL server returns variable length data, that is, you have mapped *Map OCCURS DEPENDING ON* or ***Set Arrays (Fixed <-> Unbounded)***, and depending on your runtime architecture, consider the following: ■ **CICS Socket Listener (EntireX Adapter or RPC Server)** Providing a length considering the actual number of occurences instead of the maximum possible (which was provided on input), reduces network traffic and may improve performance. ■ **CICS RPC Server** Because in this architecture the marshalling is on-host, there will be no impact on network traffic, even if the provided length is set to the maximum possible number of occurences that was provided on input. If the called COBOL server returns fixed-length data, there is no need to change WM-LCB-OUTPUT-BUFFER-SIZE.
WM-LCB-FLAGS	On return, a value of 'F' in this flag indicates that the called COBOL server allocated an output buffer which had to be released by EntireX

WM-LCB-OUTPUT-BUFFER and WM-LCB-FLAGS are normally not changed by the called COBOL server in this scenario.

If there is a need to return the output data in a different storage, this storage must be allocated by EXEC CICS GETMAIN. Return the new storage address in WM-LCB-OUTPUT-BUFFER. Indicate with WM-LCB-FLAGS='F' that the storage is released (EXEC CICS FREEMAIN) by EntireX.

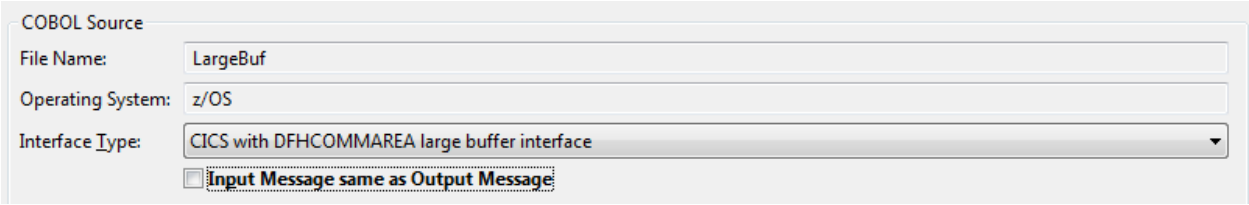
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with ***Mapping Editor User Interface***.

Extracting from a CICS DFHCOMMAREA Large Buffer Program

This section assumes **Input Message same as Output Message** is not checked. COBOL output and COBOL input parameters are different, that is, the `WM-LCB-OUTPUT-BUFFER` (as in the large buffer example above) is set to an address that is different to `WM-LCB-INPUT-BUFFER`.

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

Make sure the interface type is correct and check box **Input Message same as Output Message** is cleared.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items of the input large buffer to **Input Message** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. To do this, locate in the `PROCEDURE DIVISION` the `SET ADDRESS OF <x> TO WM-LCB-INPUT-BUFFER` statement. The COBOL data item `<x>` is the input large buffer you are looking for. See **Notes**.

2 Add the COBOL data items of the output large buffer to **Output Message** by using the context menu and toolbars available in the *COBOL Interface* and *IDL Interface*. To do this, locate in the `PROCEDURE DIVISION` the `SET ADDRESS OF <y> TO WM-LCB-OUTPUT-BUFFER` statement. The COBOL data item `<y>` is the output large buffer you are looking for. See **Notes**.

3 Continue with *COBOL to IDL Mapping*.

Notes:

1. Do not select the pointers in the `DFHCOMMAREA` pointing to the large buffers, in the example above, `WM-LCB-INPUT-BUFFER` and `WM-LCB-OUTPUT-BUFFER`.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).

3. If your COBOL interface contains `REDEFINE`s, 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.

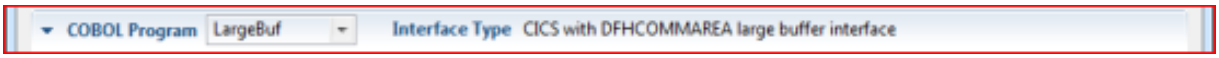
The user interface of the COBOL Mapping Editor is described below.

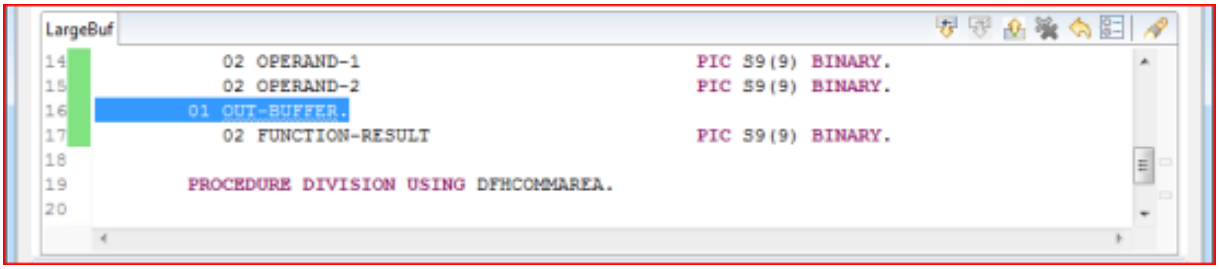
Mapping Editor User Interface

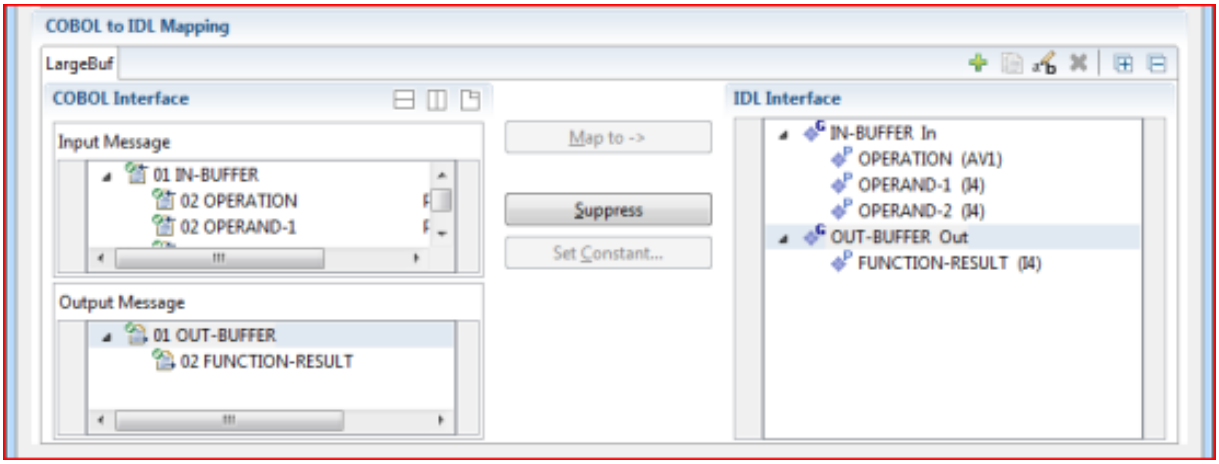
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface types where COBOL input and COBOL output parameters are different, the user interface of the COBOL Mapping Editor looks like this:

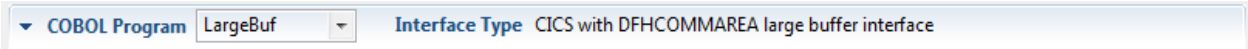
1. 

2. 

3. 

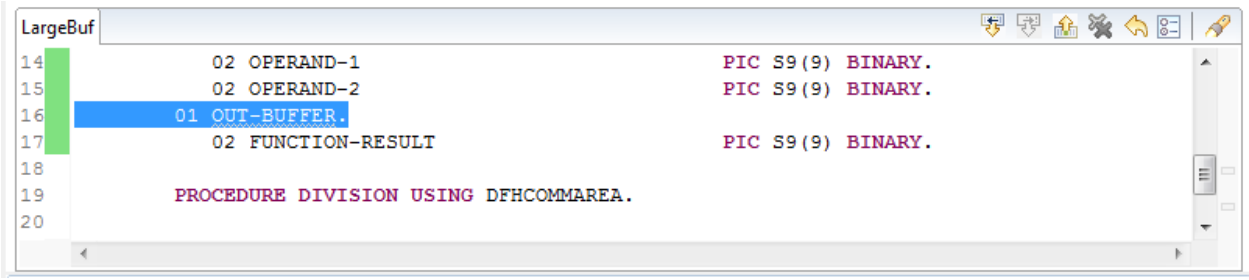
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

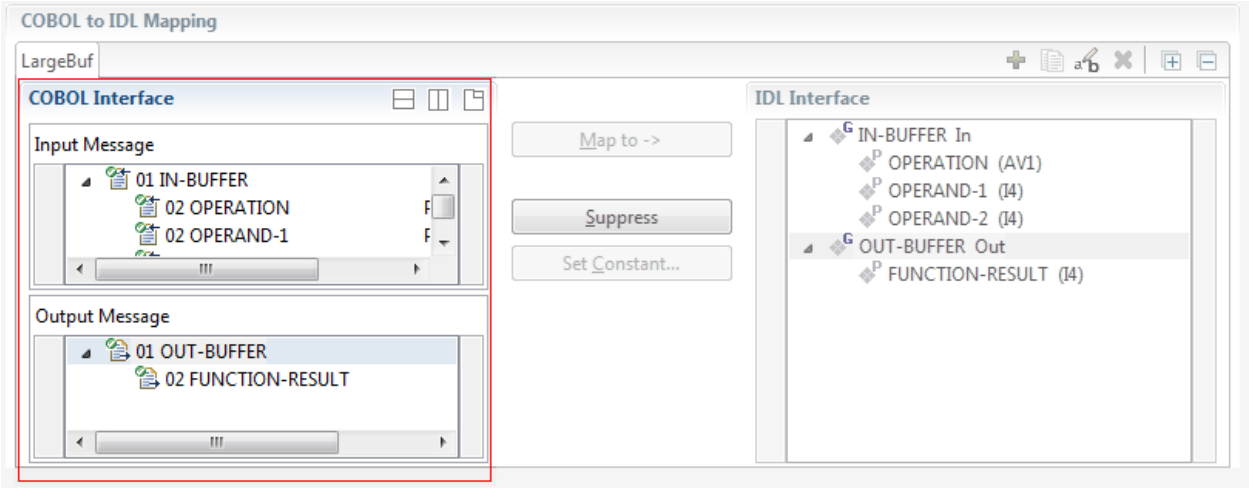
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

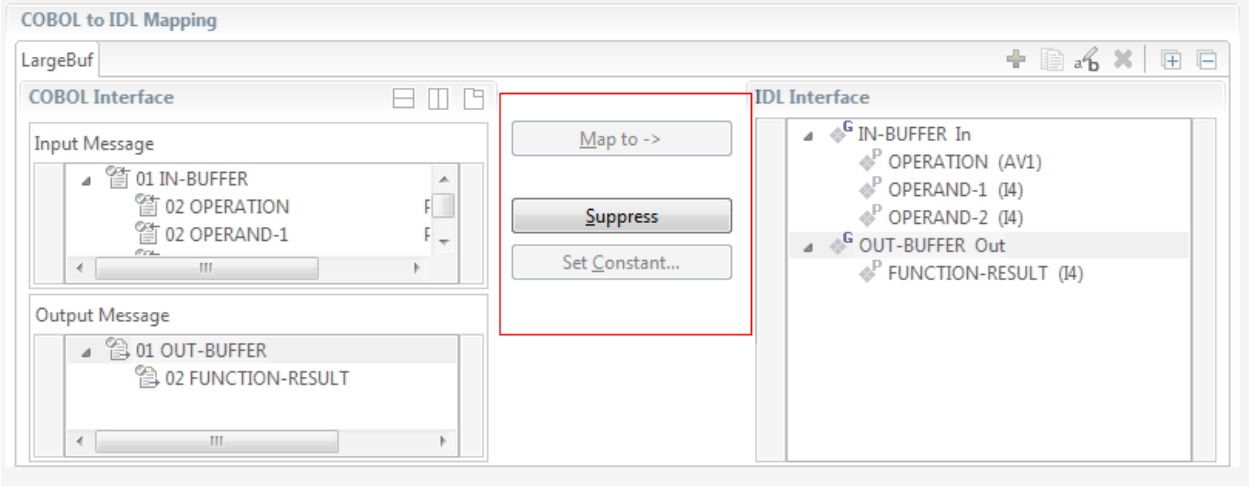
Mapping Icons

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:



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*.

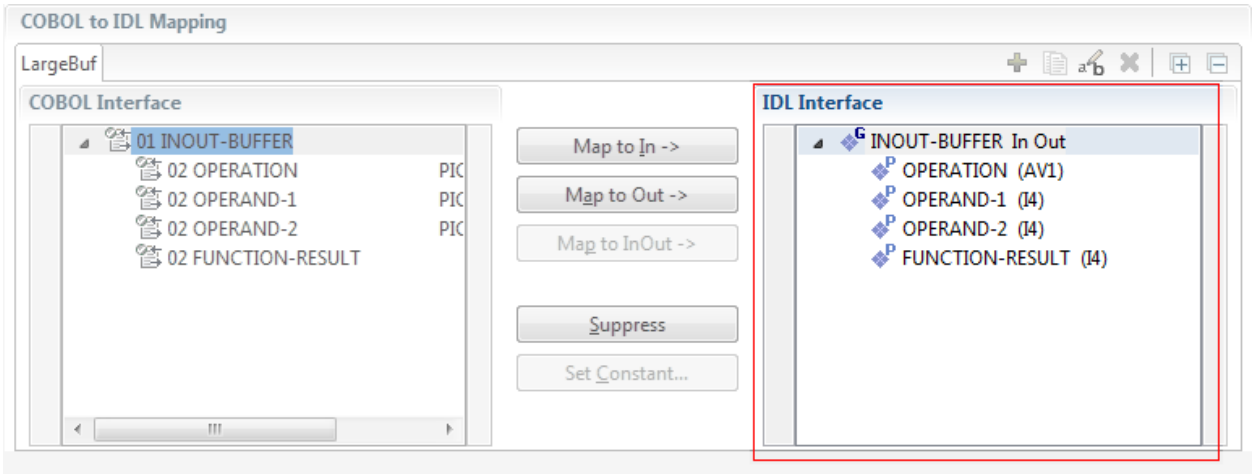
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

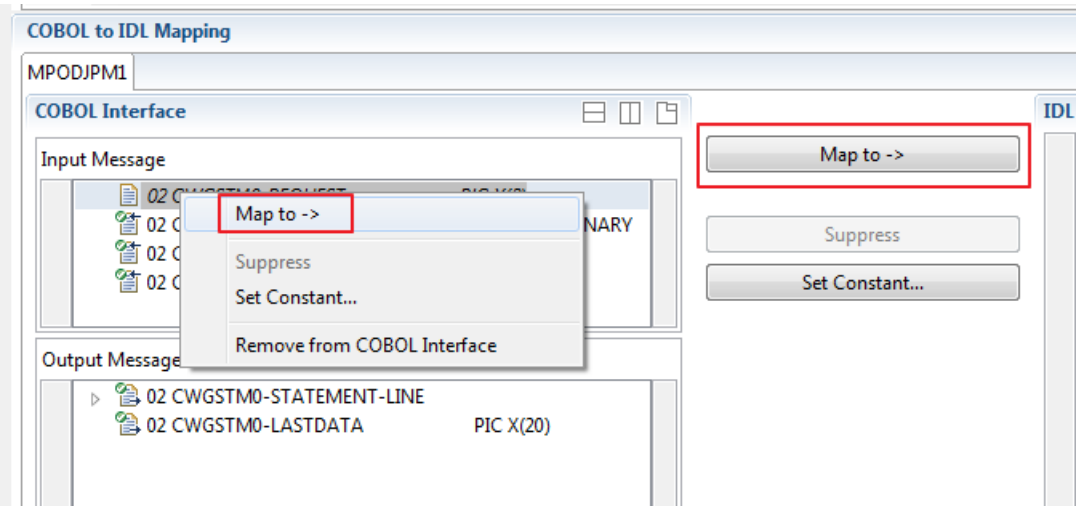
This section covers the following topics:

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 COBOL data items 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 of the COBOL interface and as mapping button to make a COBOL data item visible as an IDL parameter in the IDL interface:



2 Do the same for the output message of the COBOL interface.

Notes:

1. If a COBOL group is mapped, all subordinate 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 of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.

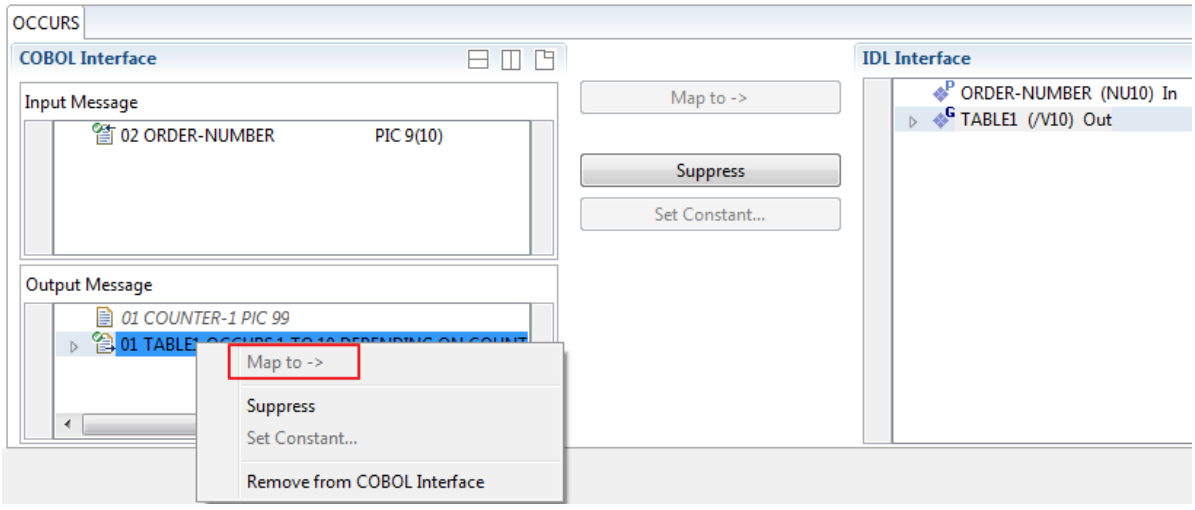
Map OCCURS DEPENDING ON

You can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

- Add the COBOL subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the input message or to the output message, wherever they belong. It is important both data items are always together per message direction (input or output).



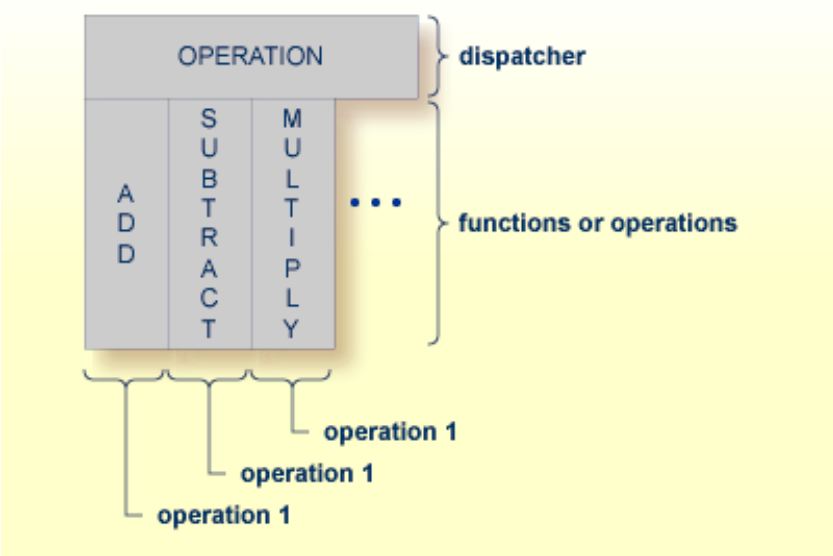
Notes:

1. The ODO subject can be mapped to the IDL interface.

2. The ODO object is always suppressed, but is required to be part of the same message direction (Input Message or Output Message) of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "*"
```

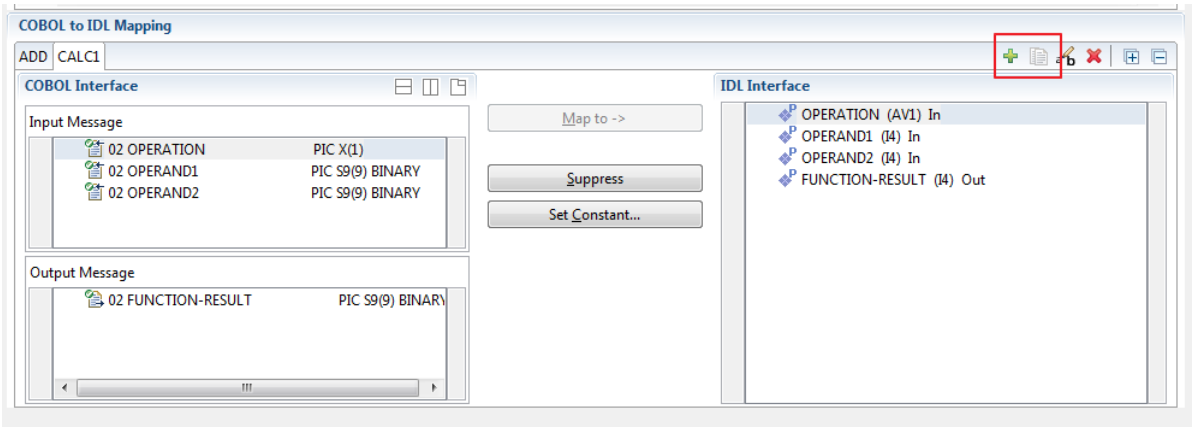
```
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

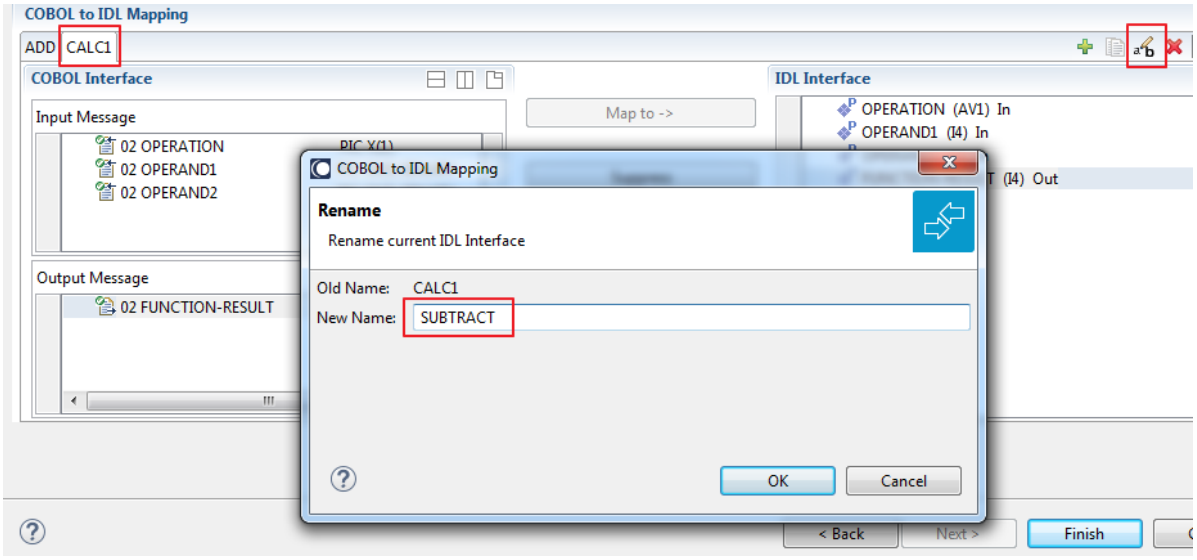
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

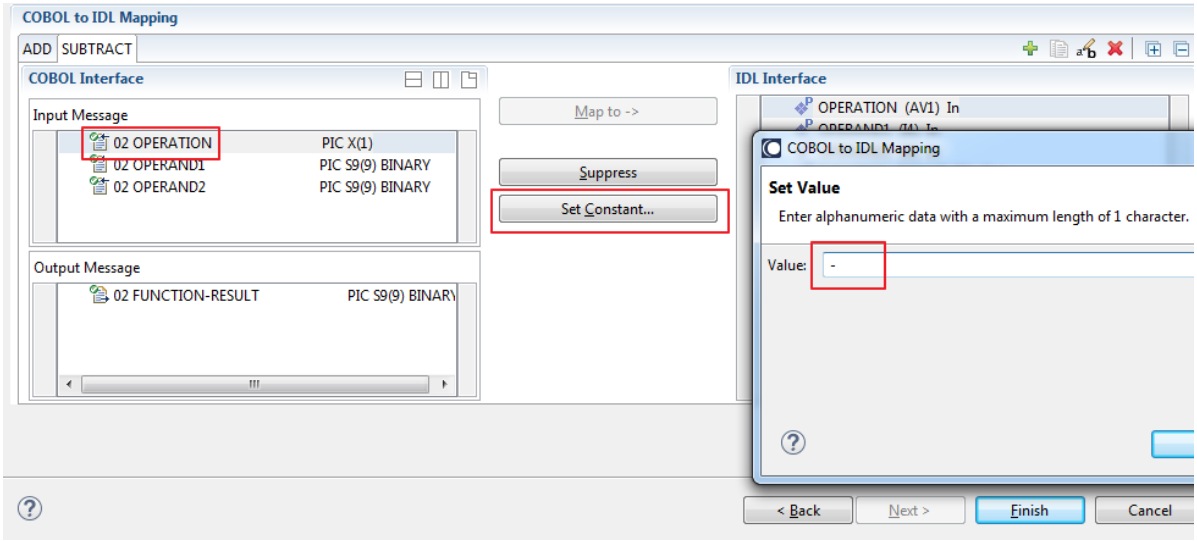
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.
- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

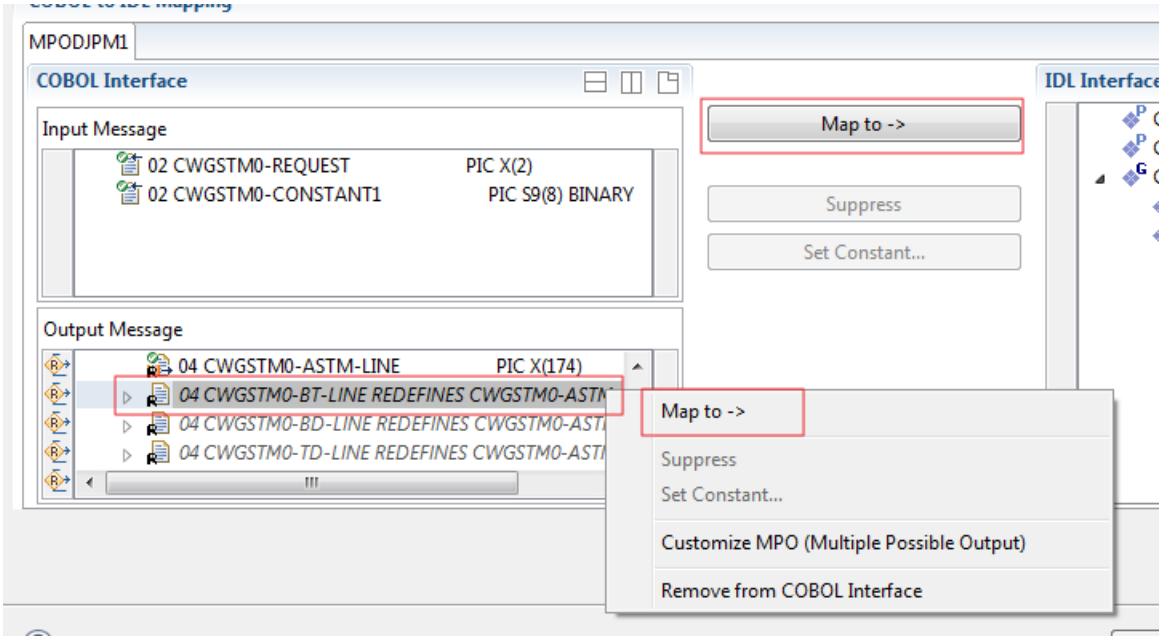
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

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 of the COBOL interface 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:

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

Suppress Unneeded COBOL Data Items

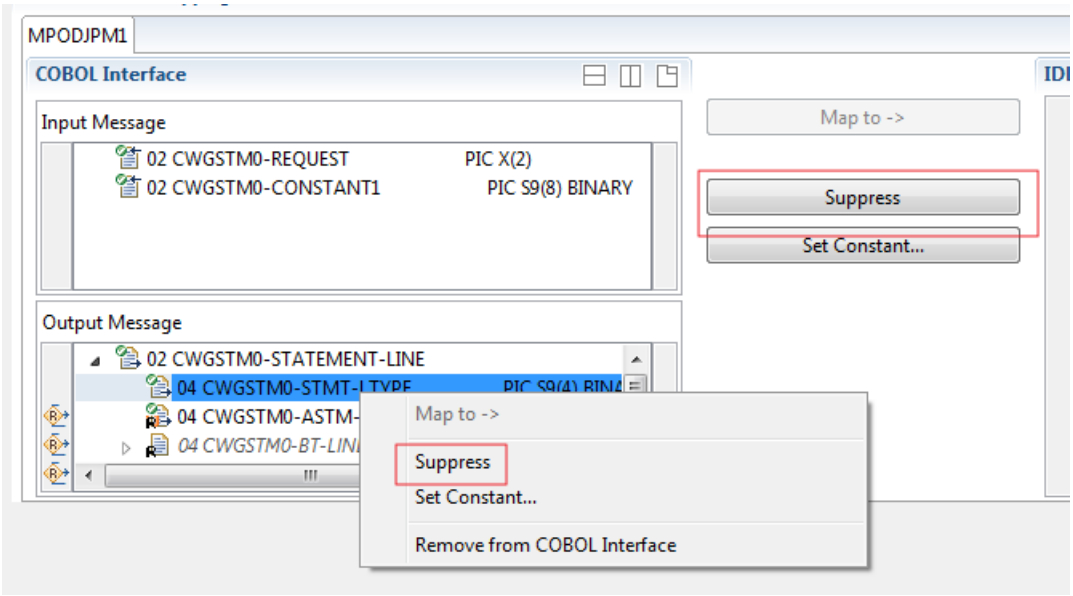
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface 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 consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* 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 subordinate COBOL data items are suppressed as well.

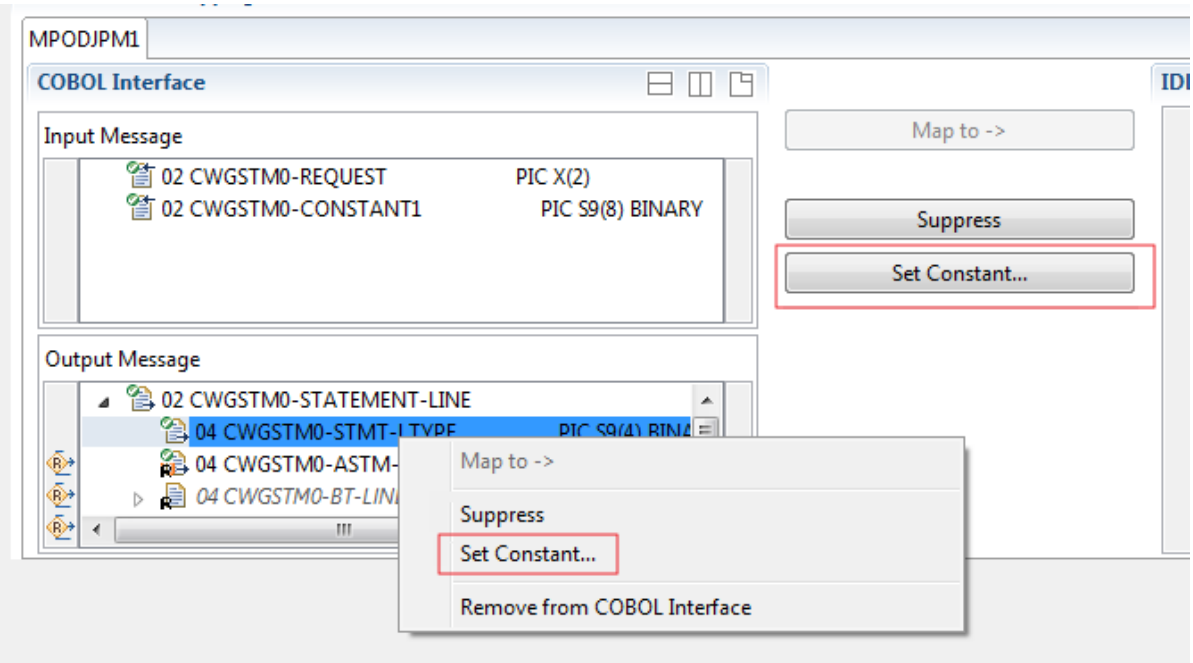
4. With the inverse function **Map to** (see above) available in the context menu of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

Set COBOL Data Items to Constants

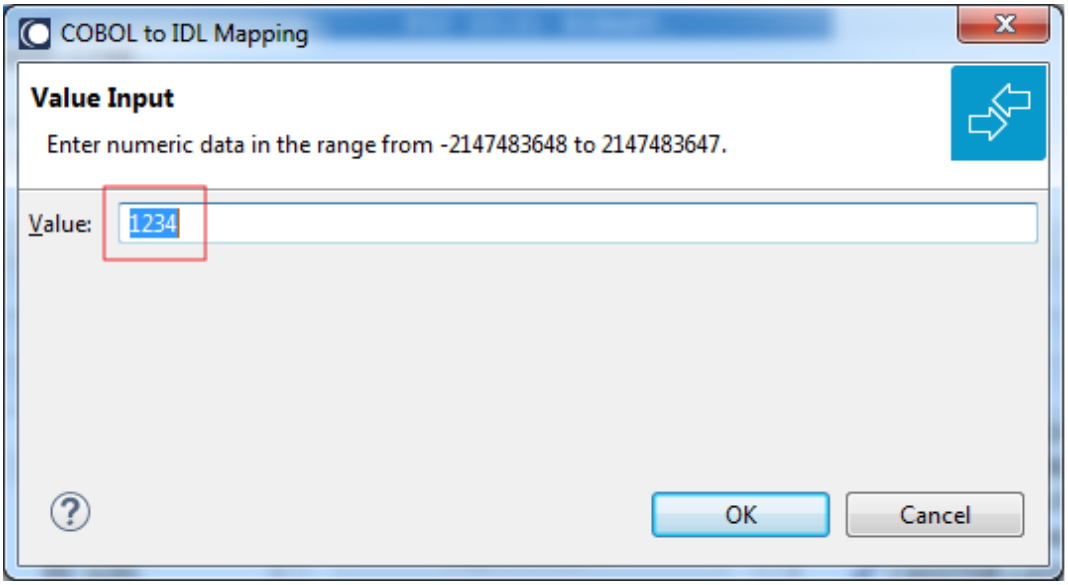
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter 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 of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

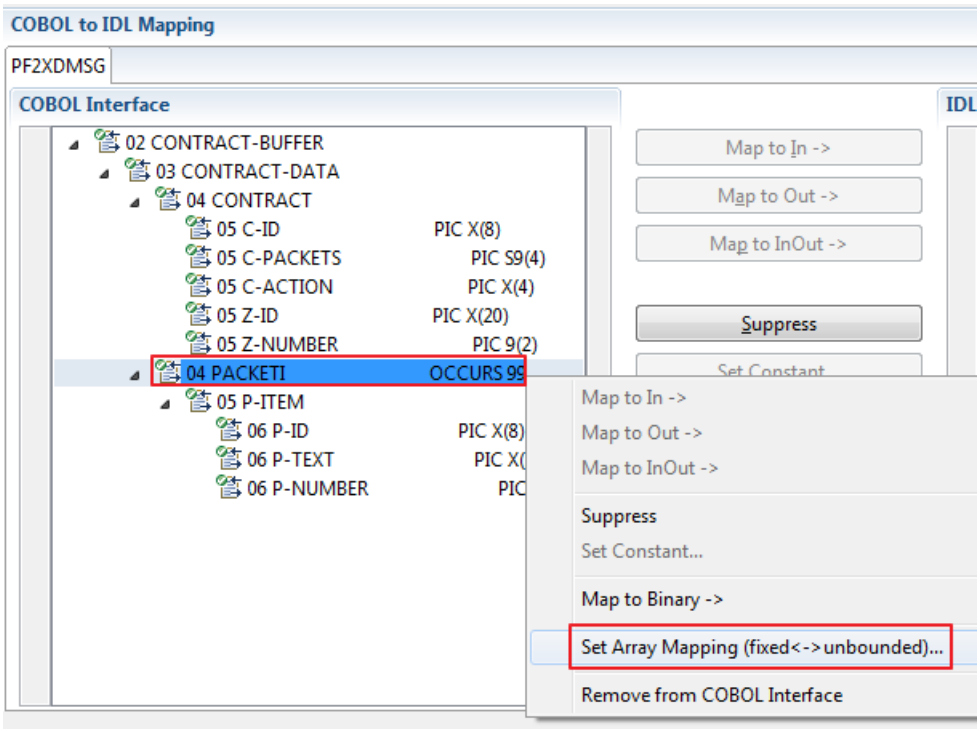
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

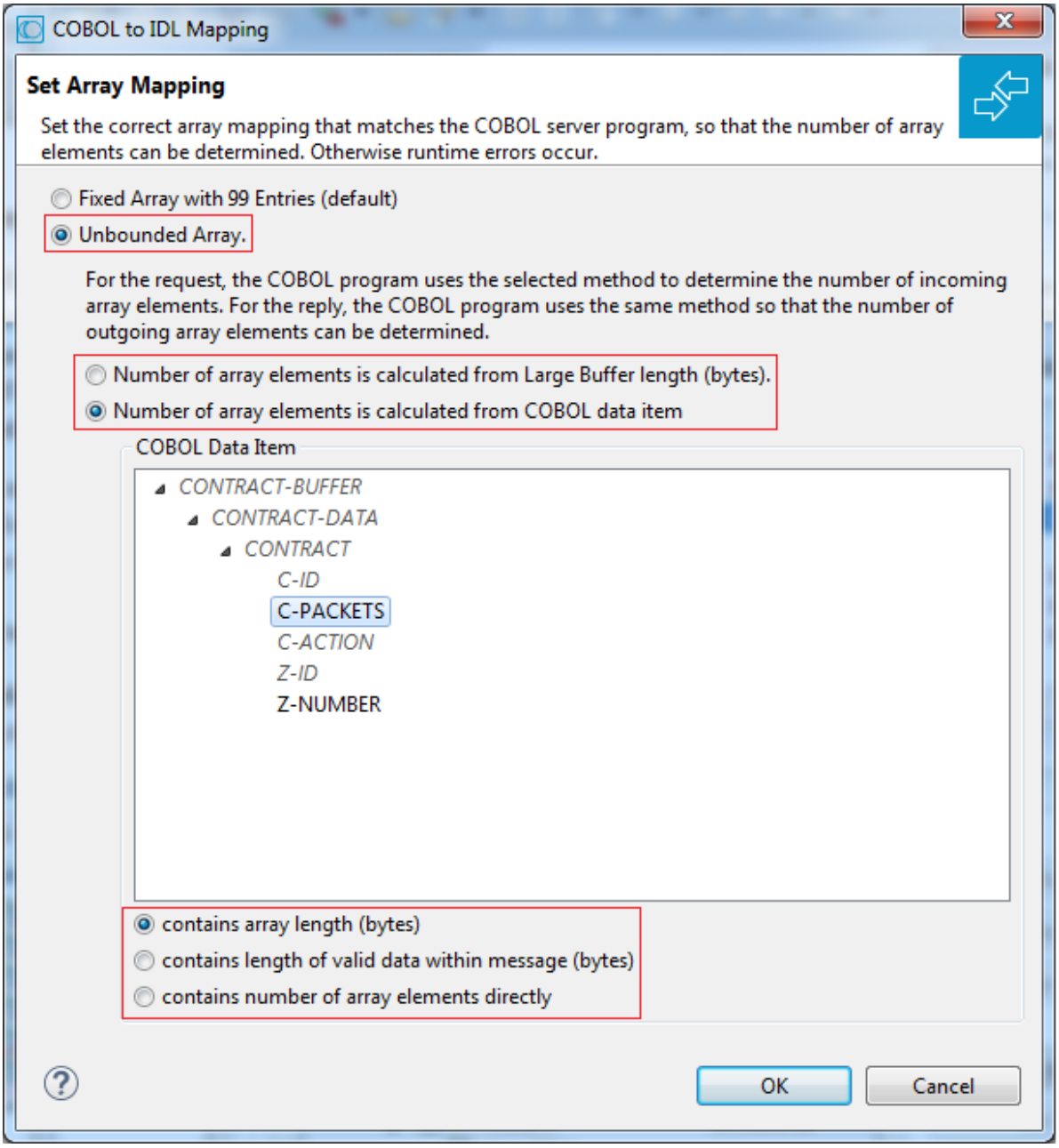
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **Large Buffer Length (bytes)**
  The COBOL server program inspects `WM-LCB-INPUT-BUFFER-SIZE` (large buffer length for input) for the request and sets `WM-LCB-OUTPUT-BUFFER-SIZE` (large buffer length for output) for the reply. To determine the number of array elements, the large buffer length is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the reply of a large buffer program. It assumes `CONTRACT-BUFFER` with fix array `PACKETI` is the large buffer.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
         10 WM-LCB-MARKER                         PIC X(4).
         10 WM-LCB-INPUT-BUFFER                   POINTER.
         10 WM-LCB-INPUT-BUFFER-SIZE              PIC S9(8) BINARY.
         10 WM-LCB-OUTPUT-BUFFER                  POINTER.
         10 WM-LCB-OUTPUT-BUFFER-SIZE             PIC S9(8) BINARY.
         10 WM-LCB-FLAGS                          PIC X(1).
            88 WM-LCB-FREE-OUTPUT-BUFFER          VALUE "F".
         10 WM-LCB-RESERVED                       PIC X(3).
        01 CONTRACT-BUFFER.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
            05 Z-ID                               PIC X(20).
           04 PACKETI                             OCCURS 99.
            05 P-ID                               PIC X(8).
            05 P-TEXT                             PIC X(30).
            05 P-NUMBER                           PIC 9(2).
       . . .

        *  Fill variable output array
           MOVE 0 TO II.
           PERFORM RANDOMNUM TIMES
              ADD 1 TO II
              MOVE ... TO P-ID    (II)
              MOVE ... TO P-TEXT  (II)
              MOVE ... TO P-NUMBER(II)
           END-PERFORM.
        *  Set large buffer length depending on number of elements
           COMPUTE WM-LCB-OUTPUT-BUFFER-SIZE =
                  (LENGTH OF P-ID  +
                   LENGTH OF P-TEXT   +
                   LENGTH OF P-NUMBER) * II.
           ADD LENGTH OF CONTRACT TO WM-LCB-OUTPUT-BUFFER-SIZE.
           ADD LENGTH OF ZONE     TO WM-LCB-OUTPUT-BUFFER-SIZE.

           EXEC CICS RETURN END-EXEC.
```

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The length of the fixed-size array `PACKETI` is contained in COBOL data item `C-BYTES`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
           03 CONTRACT-DATA.
            04 CONTRACT.
             05 C-ID                               PIC X(8).
             05 C-BYTES                            PIC S9(4).
             05 C-ACTION                           PIC X(4).
            04 ZONE.
             05 Z-NUMBER                           PIC 9(2).
             05 Z-ID                               PIC X(20).
            04 PACKETI                             OCCURS 99.
             05 P-ITEM.
              06 P-ID                              PIC X(8).
              06 P-TEXT                            PIC X(30).
              06 P-NUMBER                          PIC 9(2).
       . . .

      *  Fill variable output array
         MOVE 0 TO II.
         PERFORM RANDOMNUM TIMES
            ADD 1 TO II
            MOVE ... TO P-ID    (II)
            MOVE ... TO P-TEXT  (II)
            MOVE ... TO P-NUMBER(II)
         END-PERFORM.
      *  Set table length
         COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT` is filled by the COBOL server on reply. COBOL data item `C-APPDATA` contains the length of the valid data of the reply message. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
         77 EPARM                                 PIC  9(2).
         77 EPARM2                                PIC  9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
            04 CONTRACT.
             05 C-ID                              PIC X(8).
             05 C-APPDATA                         PIC S9(4).
             05 C-ACTION                          PIC X(4).
             05 Z-ID                              PIC X(20).
             05 Z-NUMBER                          PIC 9(2).
            04 PACKETI                            OCCURS 99.
             05 P-ITEM.
              06 P-ID                             PIC X(8).
              06 P-TEXT                           PIC X(30).
              06 P-NUMBER                         PIC 9(2).
       . . .
      *  Fill variable output array
         MOVE 0 TO II.
         PERFORM RANDOMNUM TIMES
            ADD 1 TO II
            MOVE ... TO P-ID    (II)
            MOVE ... TO P-TEXT  (II)
            MOVE ... TO P-NUMBER(II)
         END-PERFORM.
      *  Set length
         COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                           +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
           03 CONTRACT-DATA.
            04 CONTRACT.
             05 C-ID                              PIC X(8).
             05 C-NUM                             PIC S9(4).
             05 C-ACTION                          PIC X(4).
            04 ZONE.
             05 Z-NUMBER                          PIC 9(2).
```

```
           05 Z-ID                                  PIC X(20).
          04 PACKETI                               OCCURS 99.
           05 P-ITEM.
            06 P-ID                                PIC X(8).
            06 P-TEXT                              PIC X(30).
            06 P-NUMBER                            PIC 9(2).
     . . .

    *  Fill variable output array
       MOVE 0 TO II.
       PERFORM RANDOMNUM TIMES
          ADD 1 TO II
          MOVE ... TO P-ID    (II)
          MOVE ... TO P-TEXT  (II)
          MOVE ... TO P-NUMBER(II)
       END-PERFORM.
    *  Set occurrences
       MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

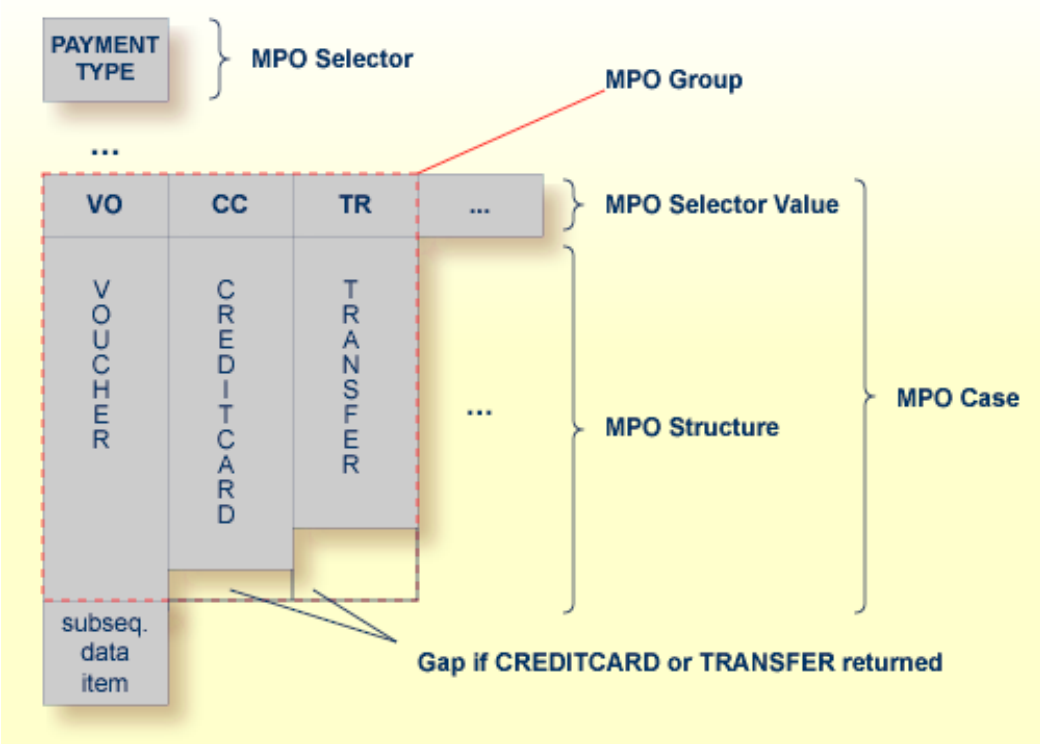
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>               PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>              PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
             . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

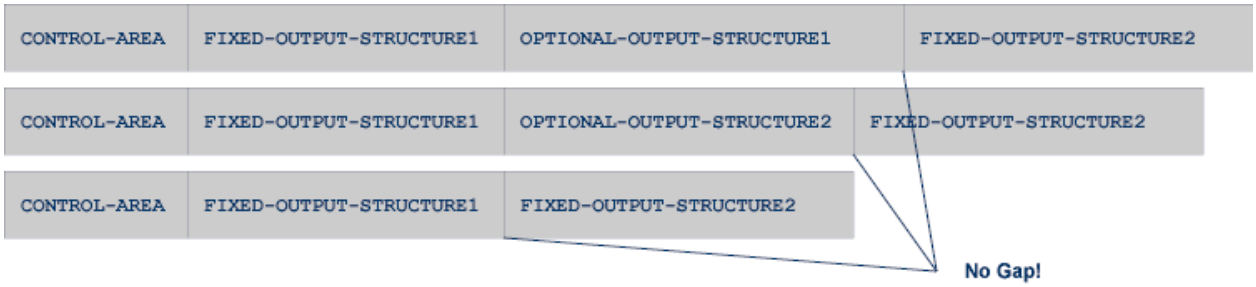
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

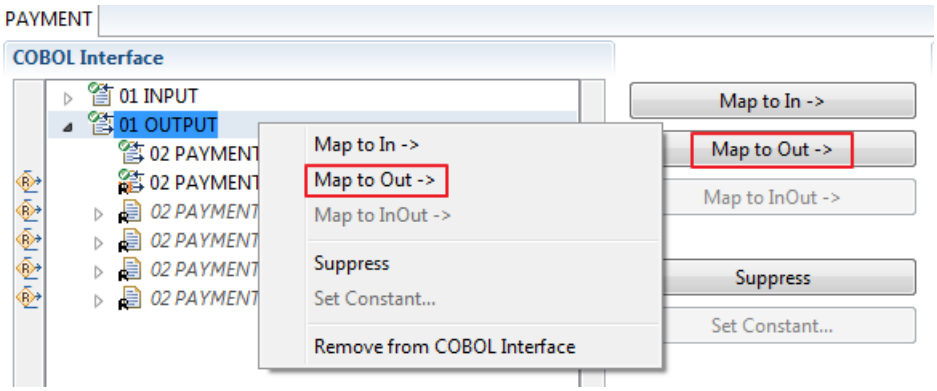
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

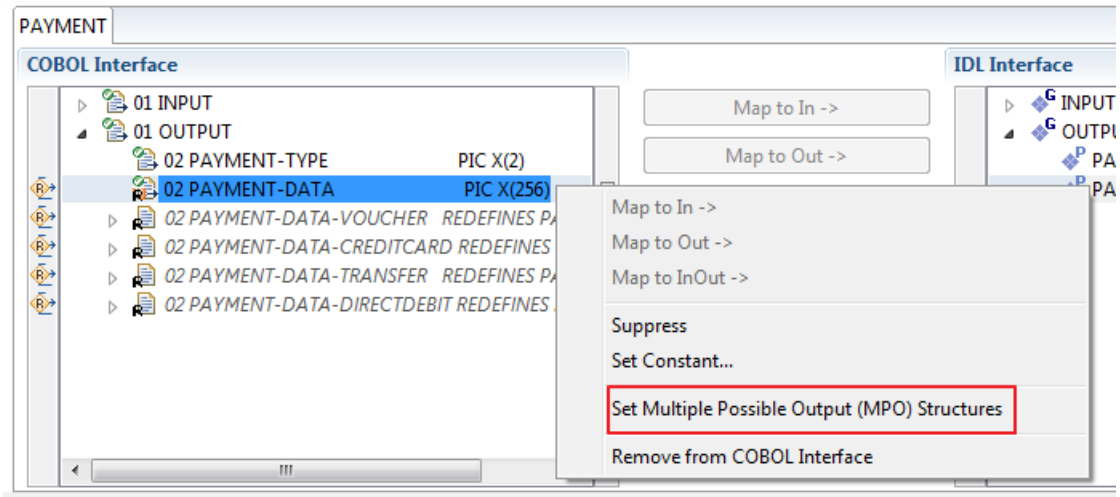
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

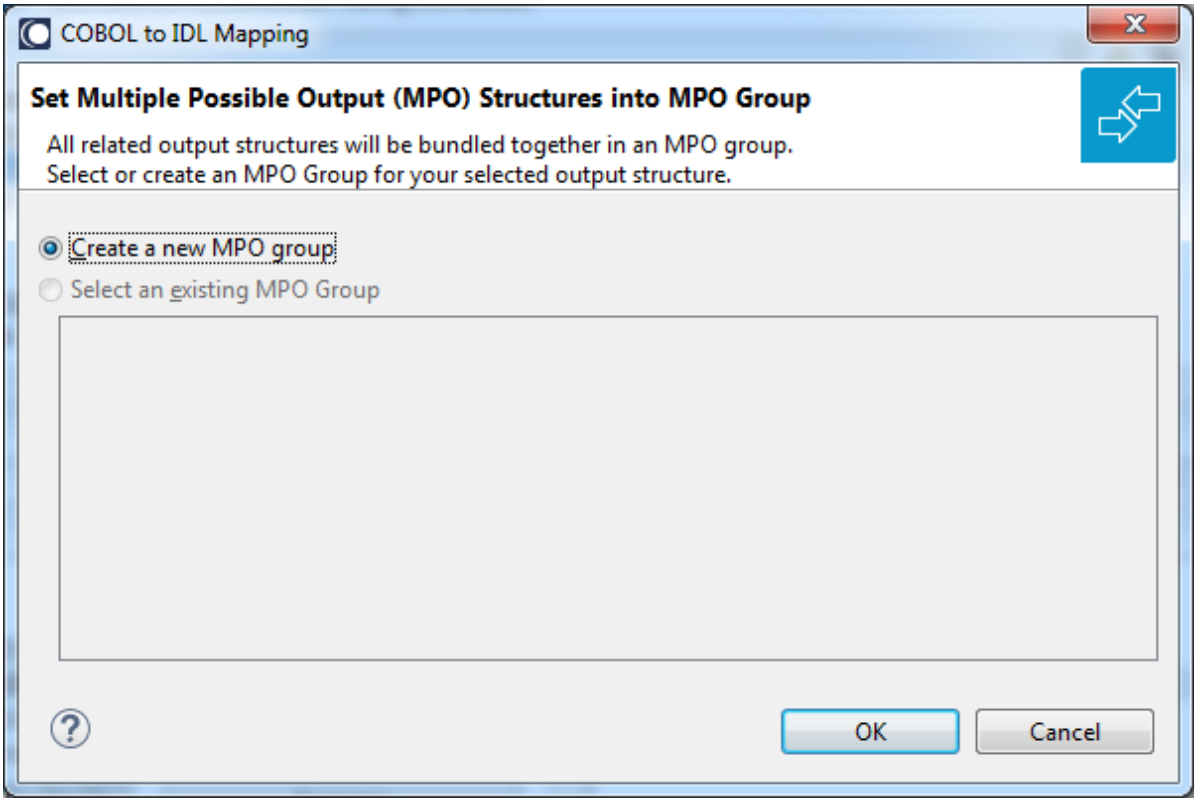
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



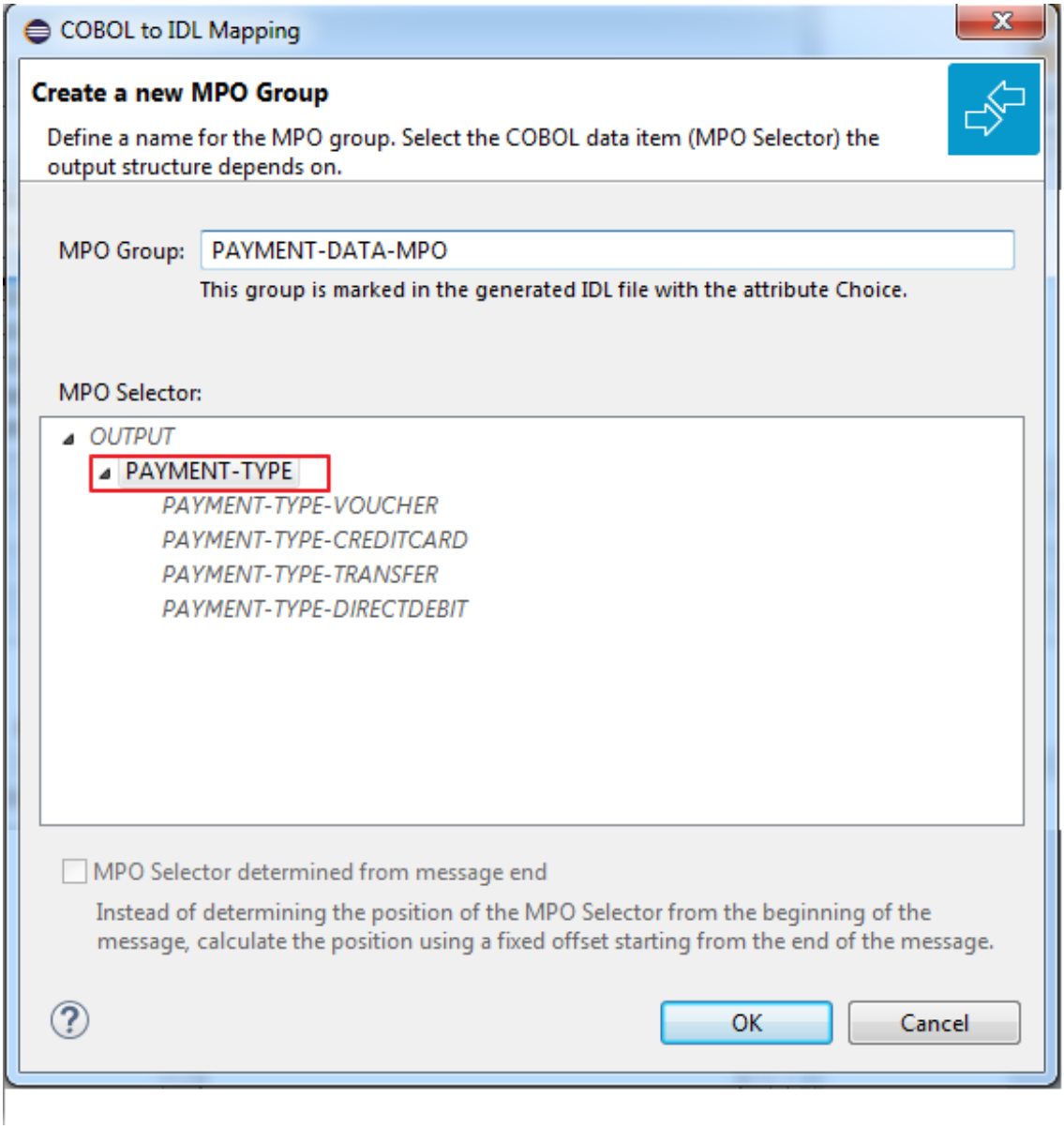
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



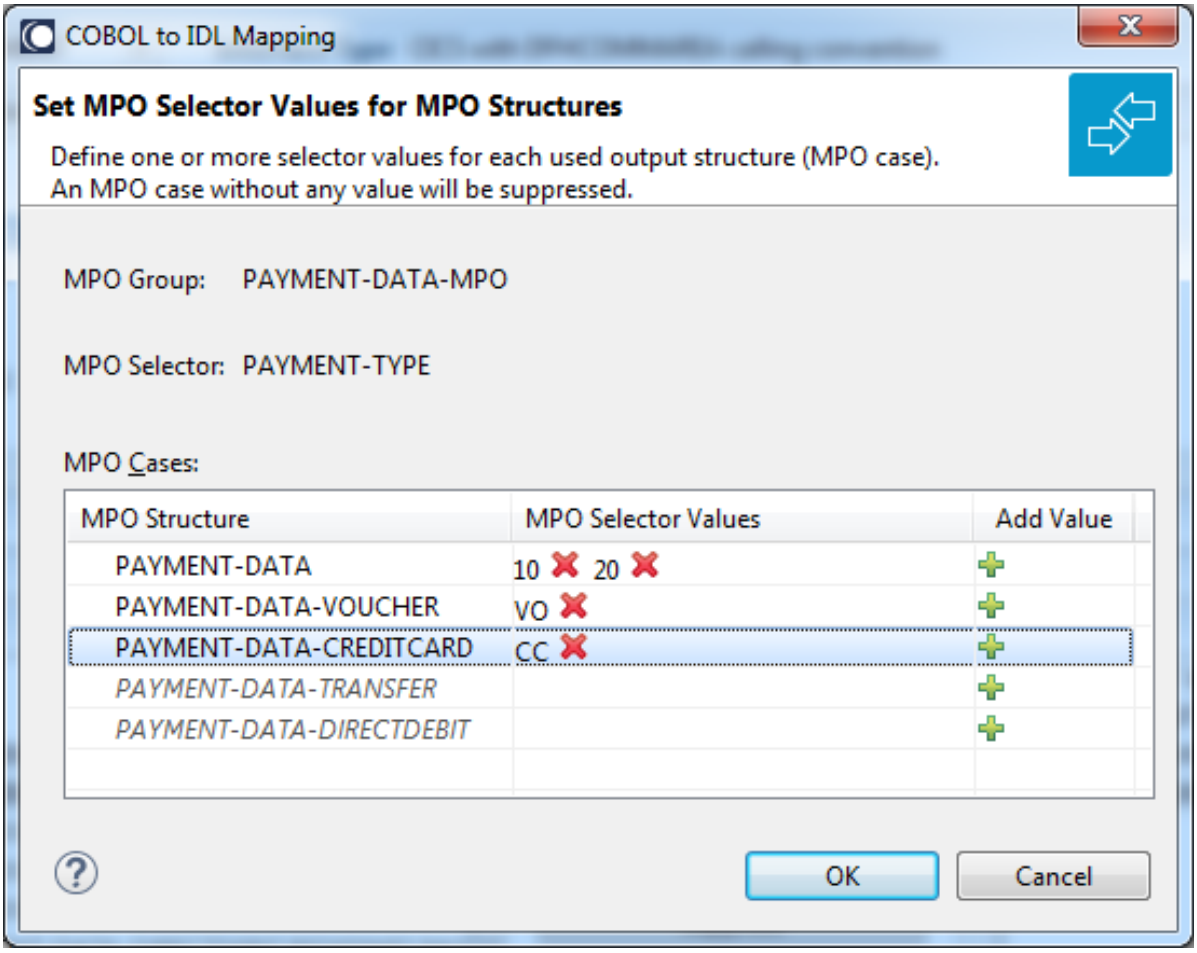
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



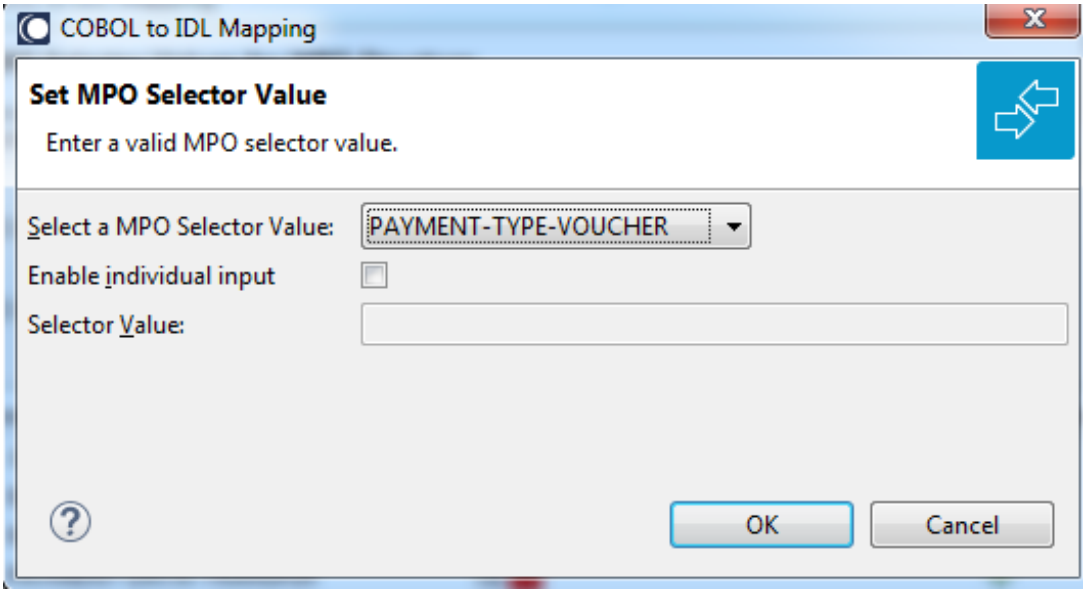
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

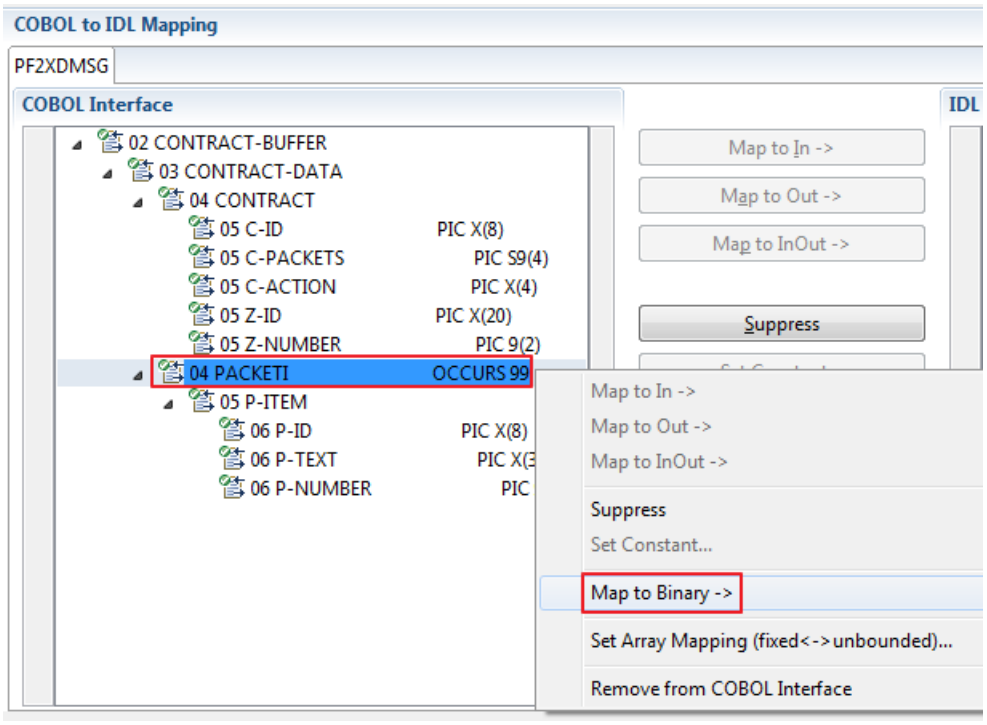
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER              (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE              (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA             (/V1)
      4 PAYMENT-DATA            (AV256)
     3 PAYMENT-DATA-VOUCHER     (/V1)
      4 VOUCHER-ORIGIN          (AV128)
      4 VOUCHER-SERIES          (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER       (NU18)
      4 CREDITCARD-CODE         (NU12)
      4 CREDITCARD-VALIDITY     (AV8)
  end-define
```

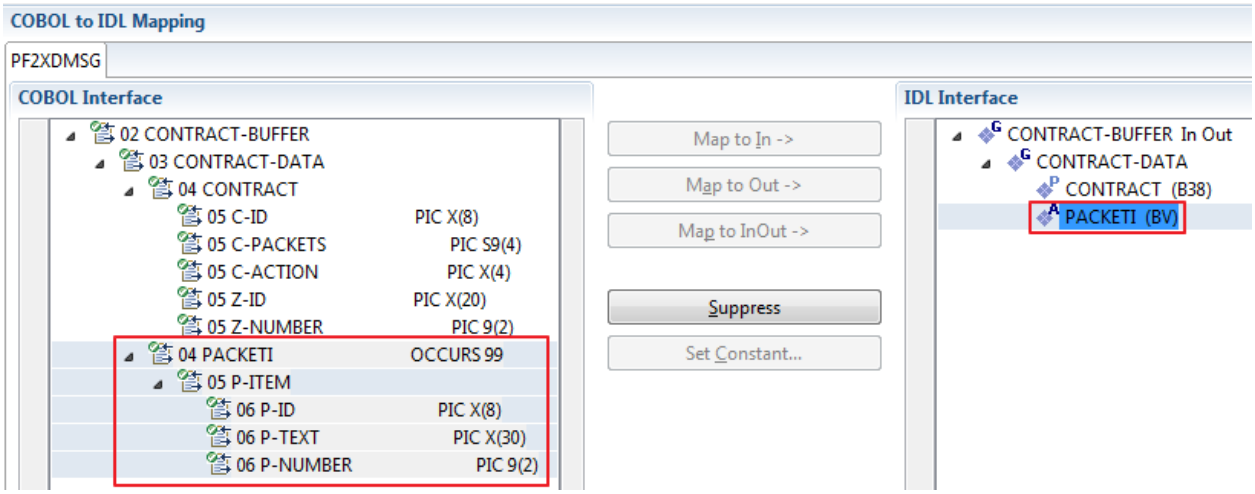
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.



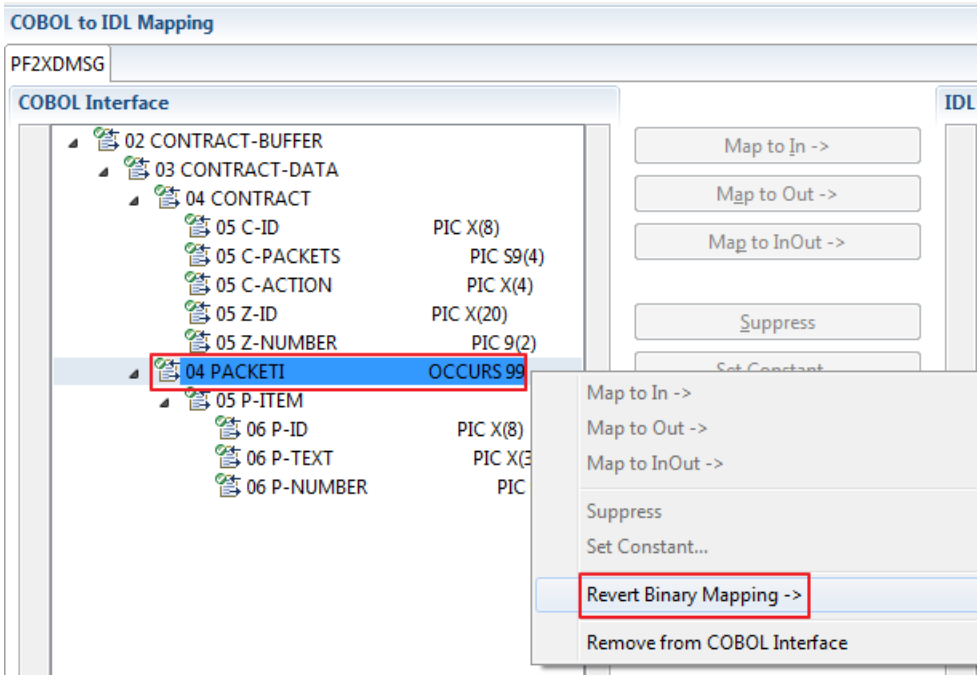
The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.

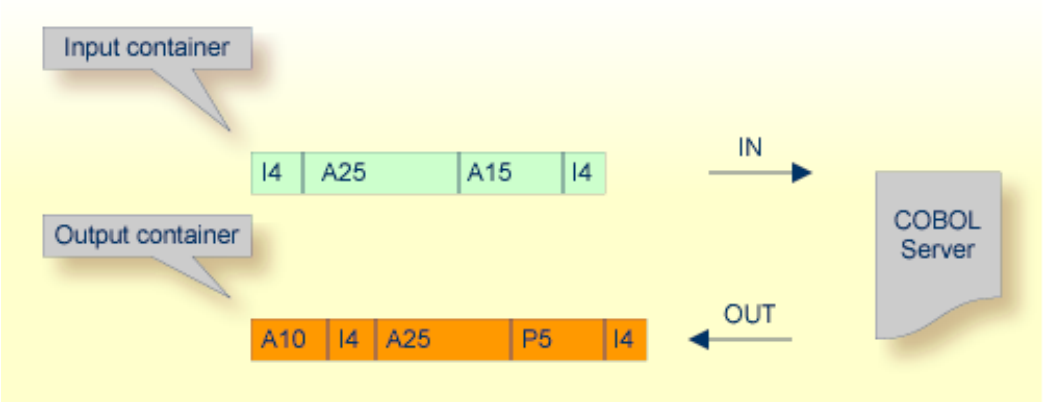


Note: The last COBOL data items are mapped to IDL data type BV instead of B*n* (`PACKETI (BV)` in this example).

To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.



10 CICS with Channel Container Calling Convention



Introduction

Modern CICS programs may use the CICS channels and containers model. During extraction, containers are mapped to IDL structures. See `structure-parameter-definition` (IDL) under *Software AG IDL Grammar* in the IDL Editor documentation.

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 a CICS Channel Container Program

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

Make sure the interface type is correct and, if required, that the channel name (max. 16 characters) is provided. If you do not provide a channel name, "EntireXChannel" is used as the default value.

Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Define all the CICS input containers, one after another: in the **Source View**, use the toolbar icon **Find text in Source** and enter "EXEC CICS" to find a GET call containing "EXEC CICS GET", function "CONTAINER" etc. Example:

```
EXEC CICS GET
      CONTAINER(<container name constant>)
      CHANNEL  (<channel>)
      INTO     (<container>)
      NOHANDLE
END-EXEC
```

The COBOL data item <container> is the item you are looking for. Add the COBOL data item <container> to **Input Message** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. In the **Input Message** pane, select the corresponding COBOL data item <container>. Enter the container name, found in the value of <container name constant>. You can select multiple CICS input containers. See **Notes**.

2 Define all the CICS output containers that are created in any case by your COBOL server using the steps as above, but look for "EXEC CICS PUT". Example:

```
EXEC CICS PUT
      CONTAINER(<container name constant>)
      CHANNEL  (<channel>)
      FROM     (<container>)
      FLENGTH  (LENGTH OF <container>)
      NOHANDLE
END-EXEC
```

Add the corresponding COBOL data item <container> to **Output Message**. In the **Output Message** pane, select the corresponding COBOL data item <container>. Enter the container name, found in the value of <container name constant>. You can select multiple CICS output containers. See **Notes**.

3 Optional. If your COBOL server creates multiple output containers, all with the same COBOL layout, map them as an array. See *Map Array of Containers*.

4 Optional. If your COBOL server creates an output container under certain conditions only, map this container as an optional container. See *Map Optional Containers*.

5 Continue with *COBOL to IDL Mapping*.

Notes:

1. It is very important to select the right COBOL data items describing the COBOL interface 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 interface contains `REDEFINE`s, 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.
3. The container name length is restricted to 16 characters by CICS.

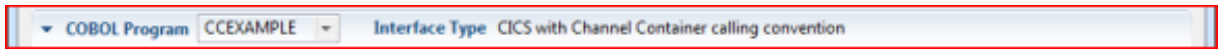
The user interface of the COBOL Mapping Editor is described below.

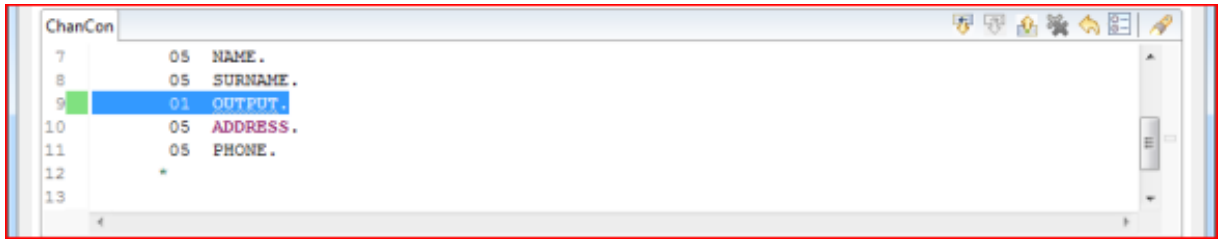
Mapping Editor User Interface

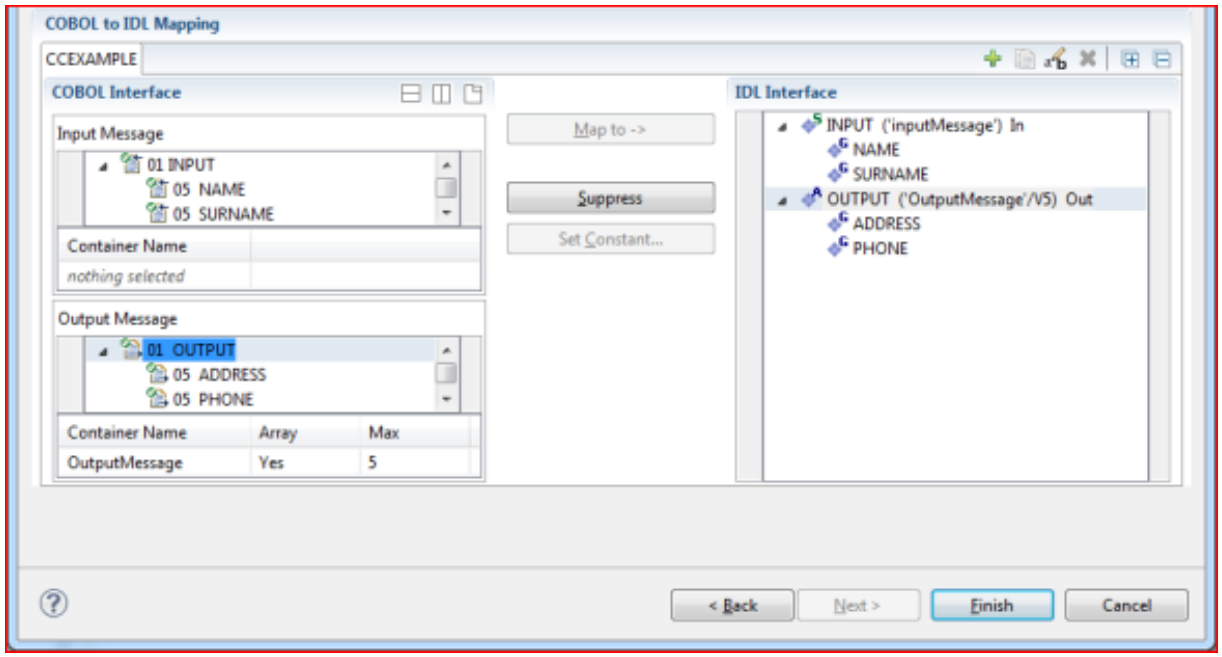
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL server programs with CICS channel container interface, the user interface of the COBOL Mapping Editor looks like this:

1. 

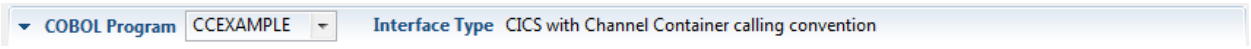
2. 

3. 

1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program

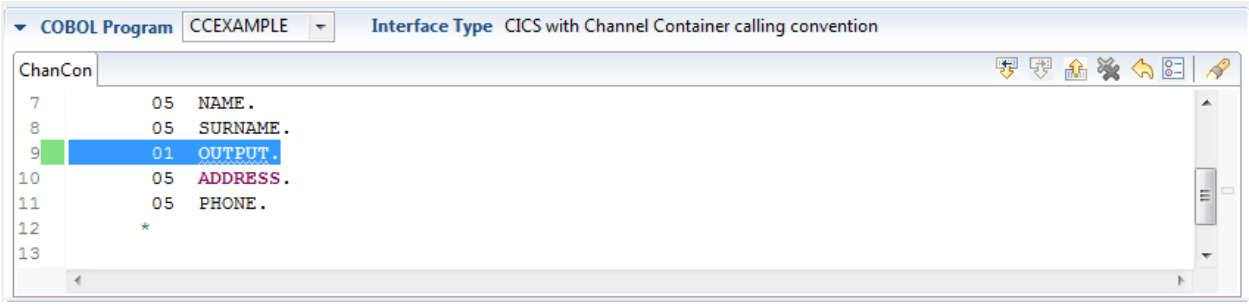
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

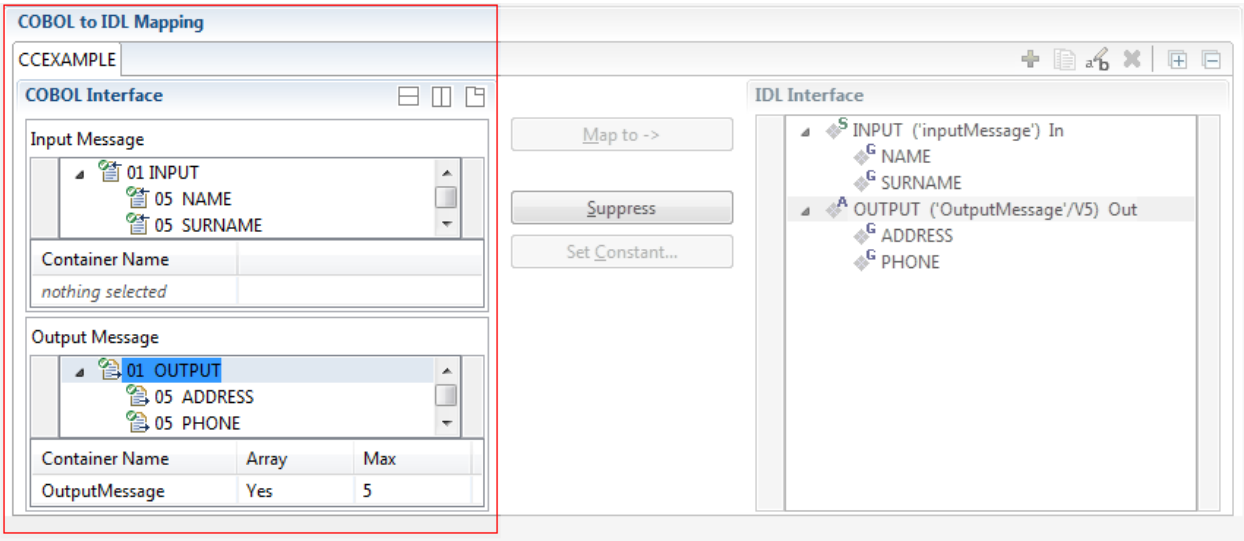
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

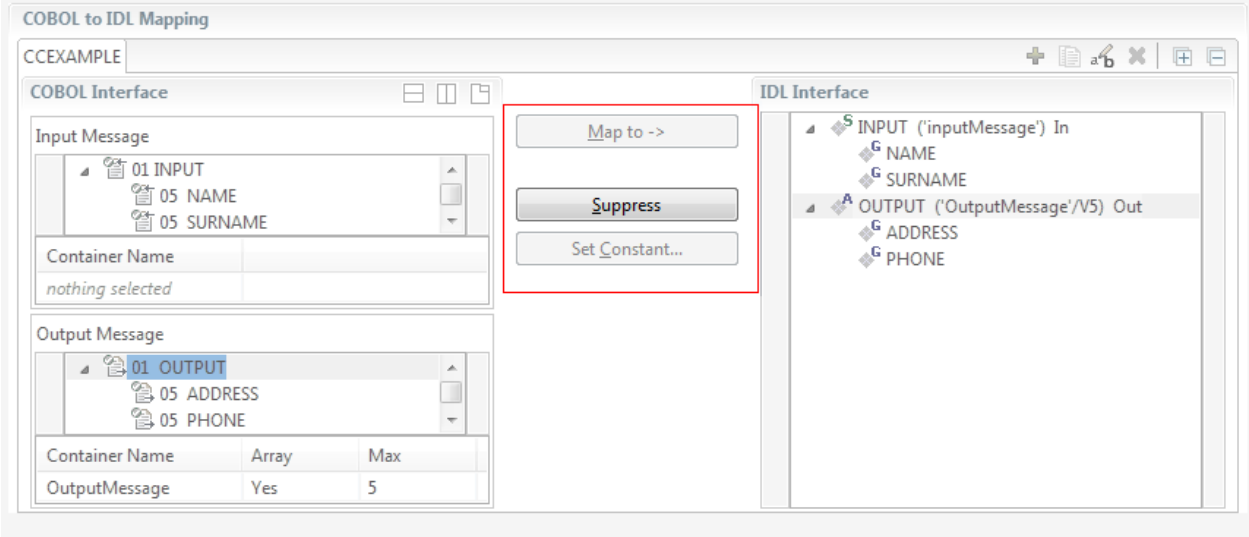
Mapping Icons

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:



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*.

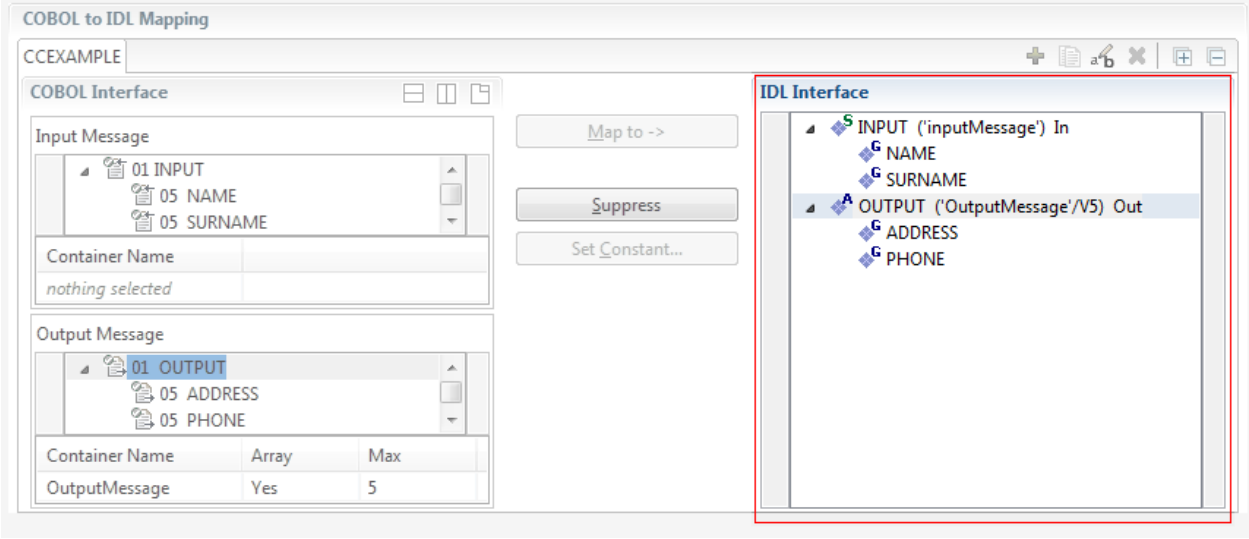
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

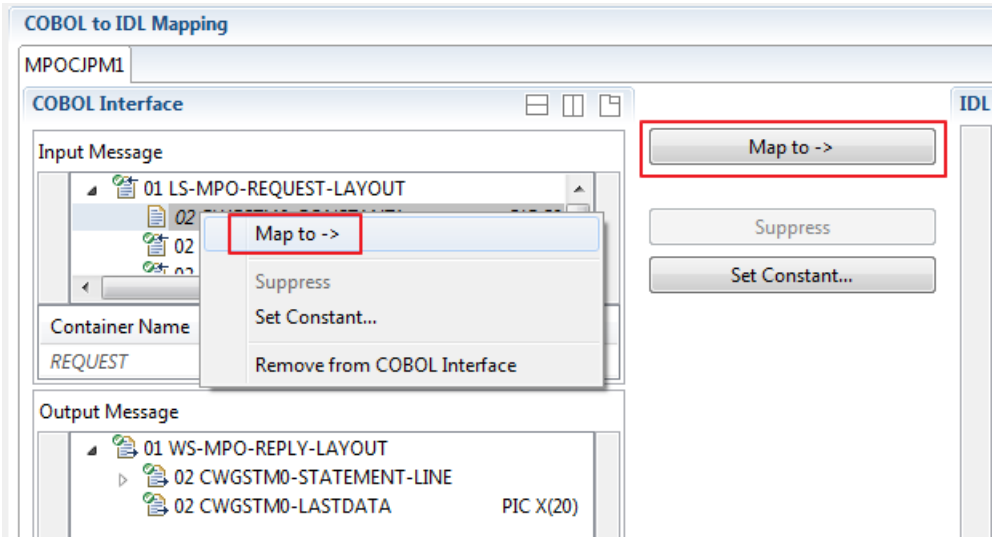
This section covers the following topics:

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 COBOL data items 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 of the COBOL interface and as mapping button to make a COBOL data item visible as an IDL parameter in the IDL interface:



2 Do the same for the output message of the COBOL interface.

Notes:

1. If a COBOL group is mapped, all subordinate 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 of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.

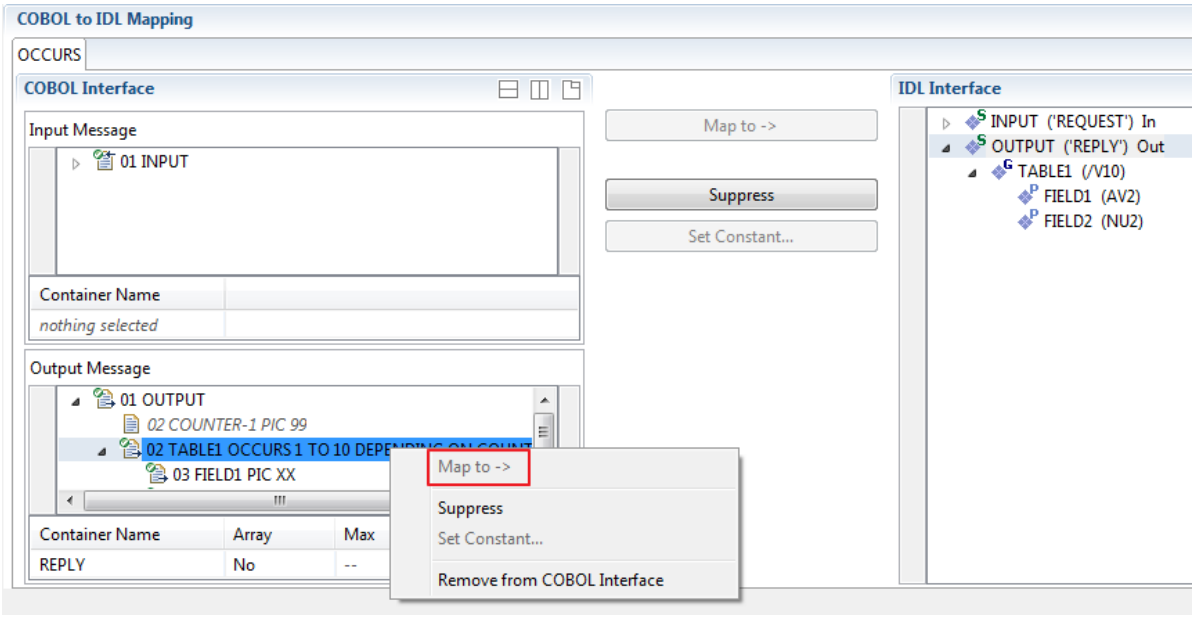
Map OCCURS DEPENDING ON

You can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

- Add the COBOL subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the input message or to the output message, wherever they belong. It is important both data items are always together per message direction (input or output).

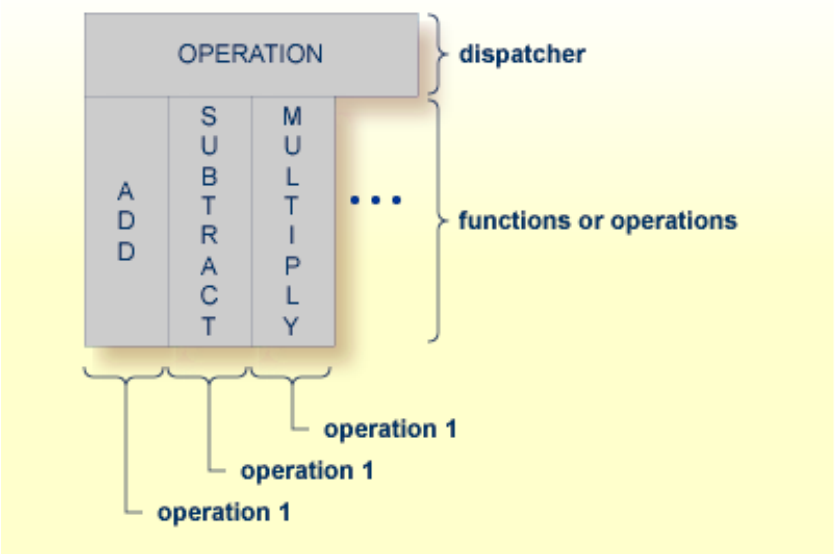


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the same message direction (Input Message or Output Message) of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

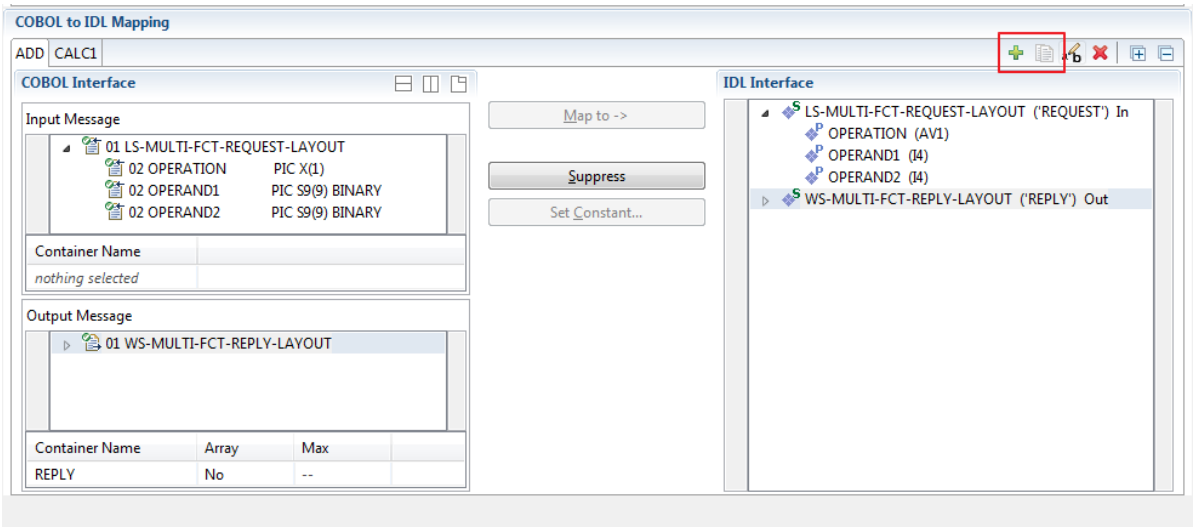
```
. . .

    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.
. . .
```

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. See the following examples, depending on your target endpoint:

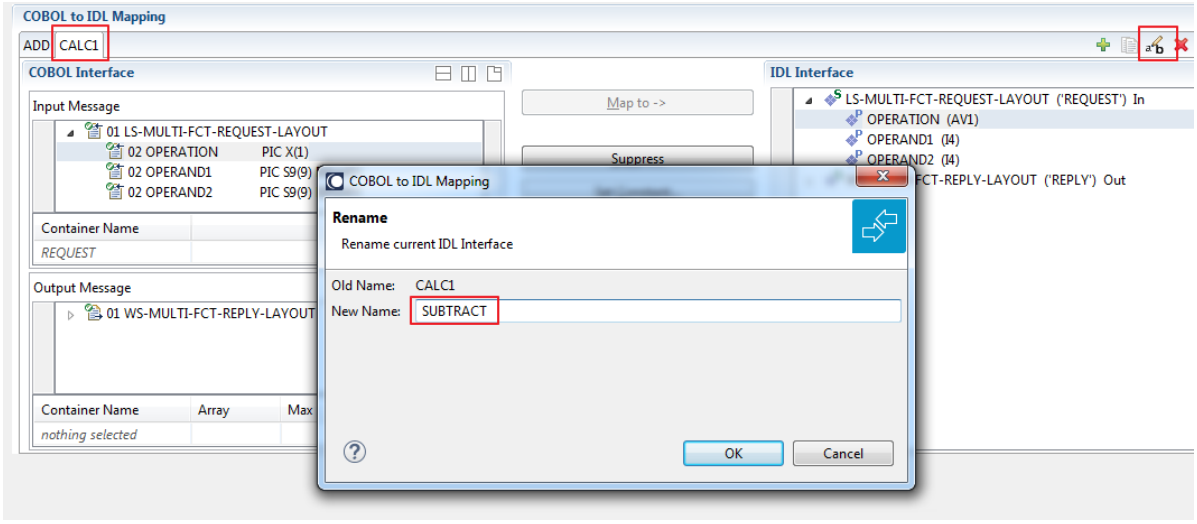
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

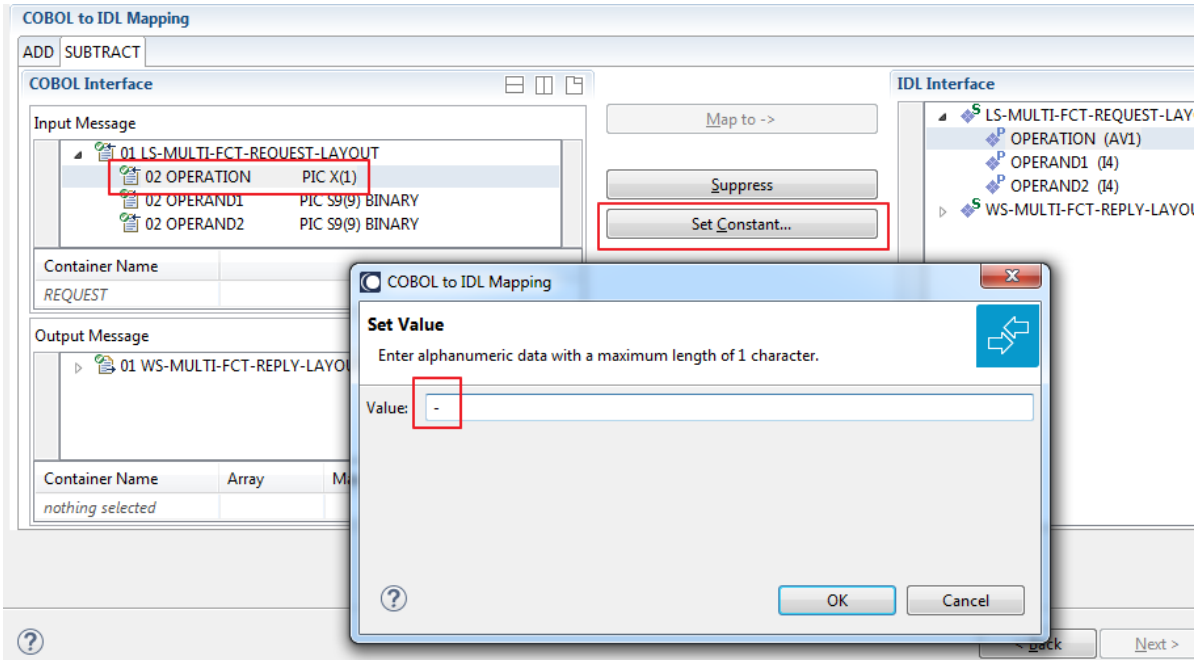
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 INPUT ('REQUEST') In
   1 OUTPUT ('REPLY')  Out
  end-define

 struct 'REQUEST' is
  define data parameter
   1 OPERAND1 (I4)
   1 OPERAND2 (I4)
  end-define

 struct 'REPLY' is
  define data parameter
   1 FUNCTION-RESULT (I4)
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 INPUT ('REQUEST') In
   1 OUTPUT ('REPLY')  Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 INPUT ('REQUEST') In
   1 OUTPUT ('REPLY')  Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see ***Step 4: Define the Extraction Settings and Start Extraction***.
	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.

Icon	Function	Description
	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.

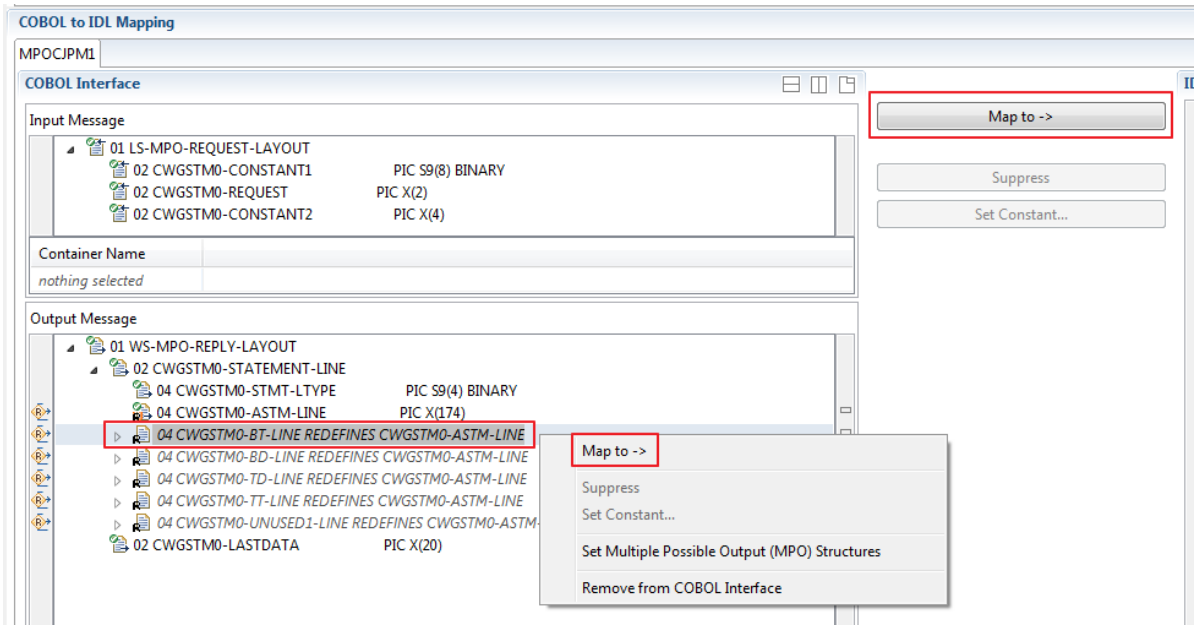
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

For COBOL server programs containing COBOL `REDEFINE`s, 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 of the COBOL interface 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:

1. Only one `REDEFINE` path of a COBOL `REDEFINE` can be mapped to the IDL interface. All COBOL `REDEFINE` siblings are suppressed.

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

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 consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

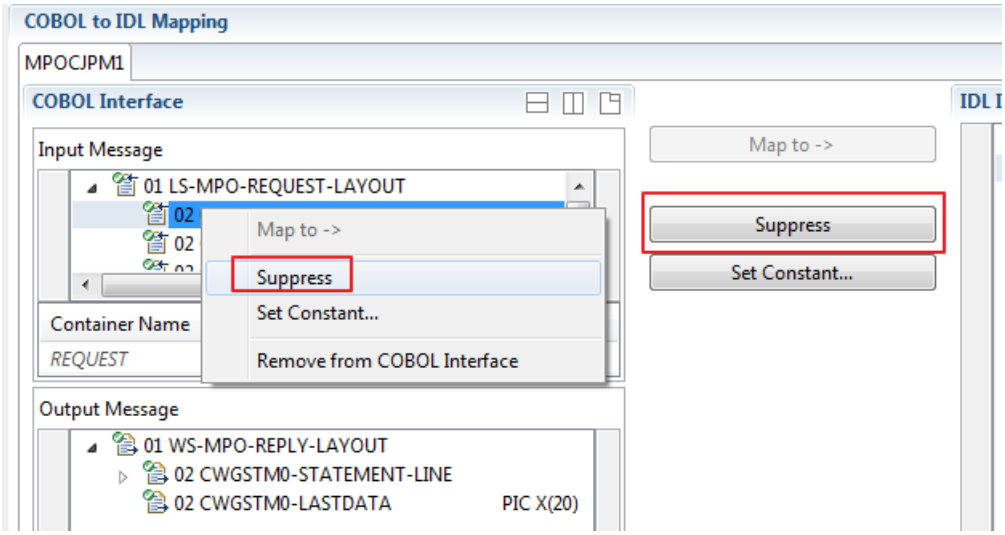
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for FILLER data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface and as mapping button to make the COBOL data item invisible in the IDL interface:



Notes:

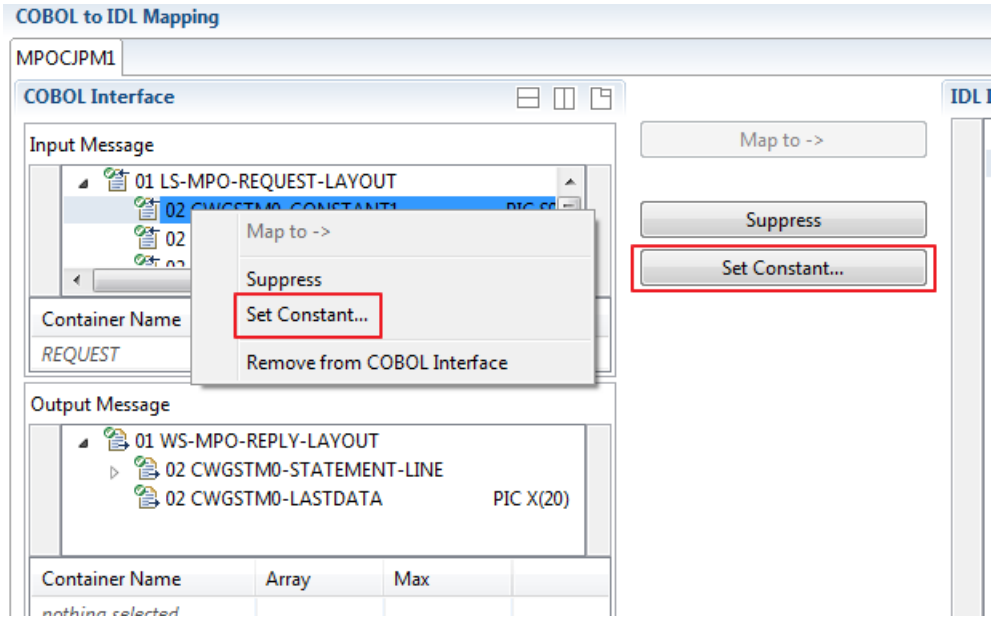
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed,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

Set COBOL Data Items to Constants

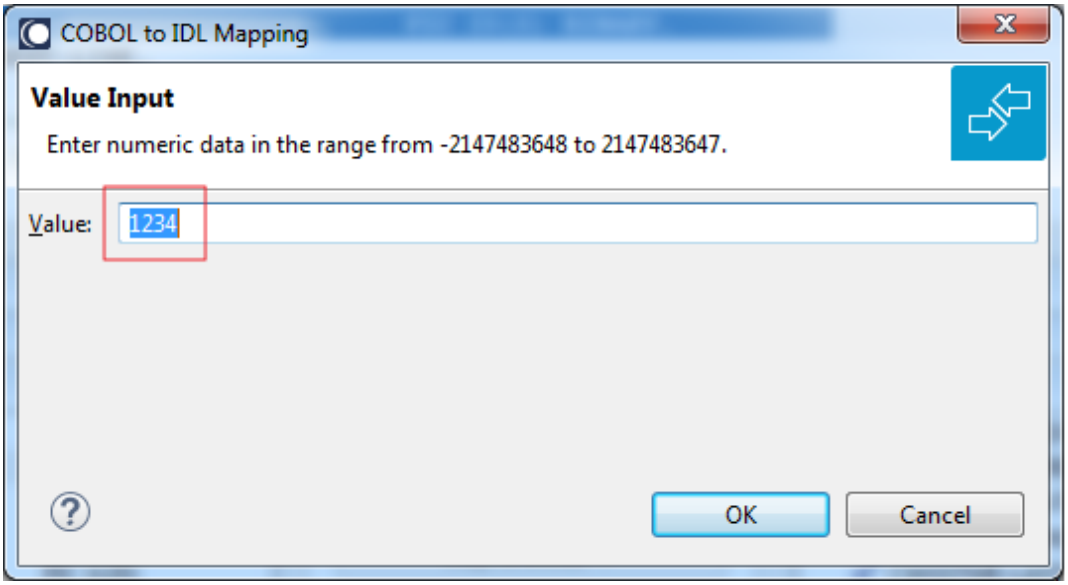
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter 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 of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

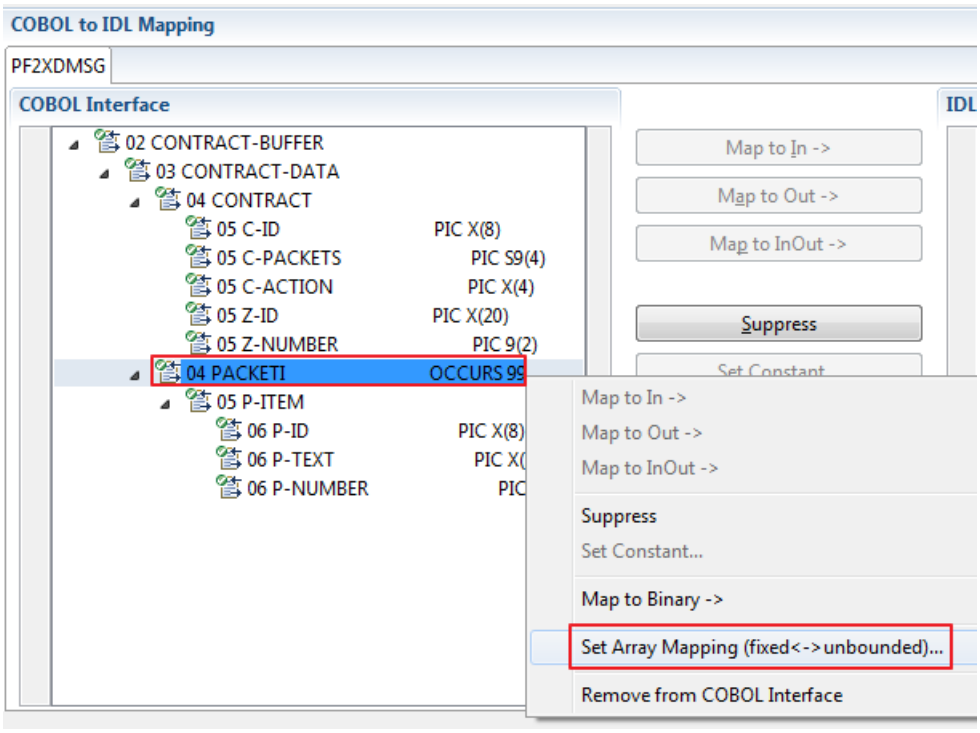
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

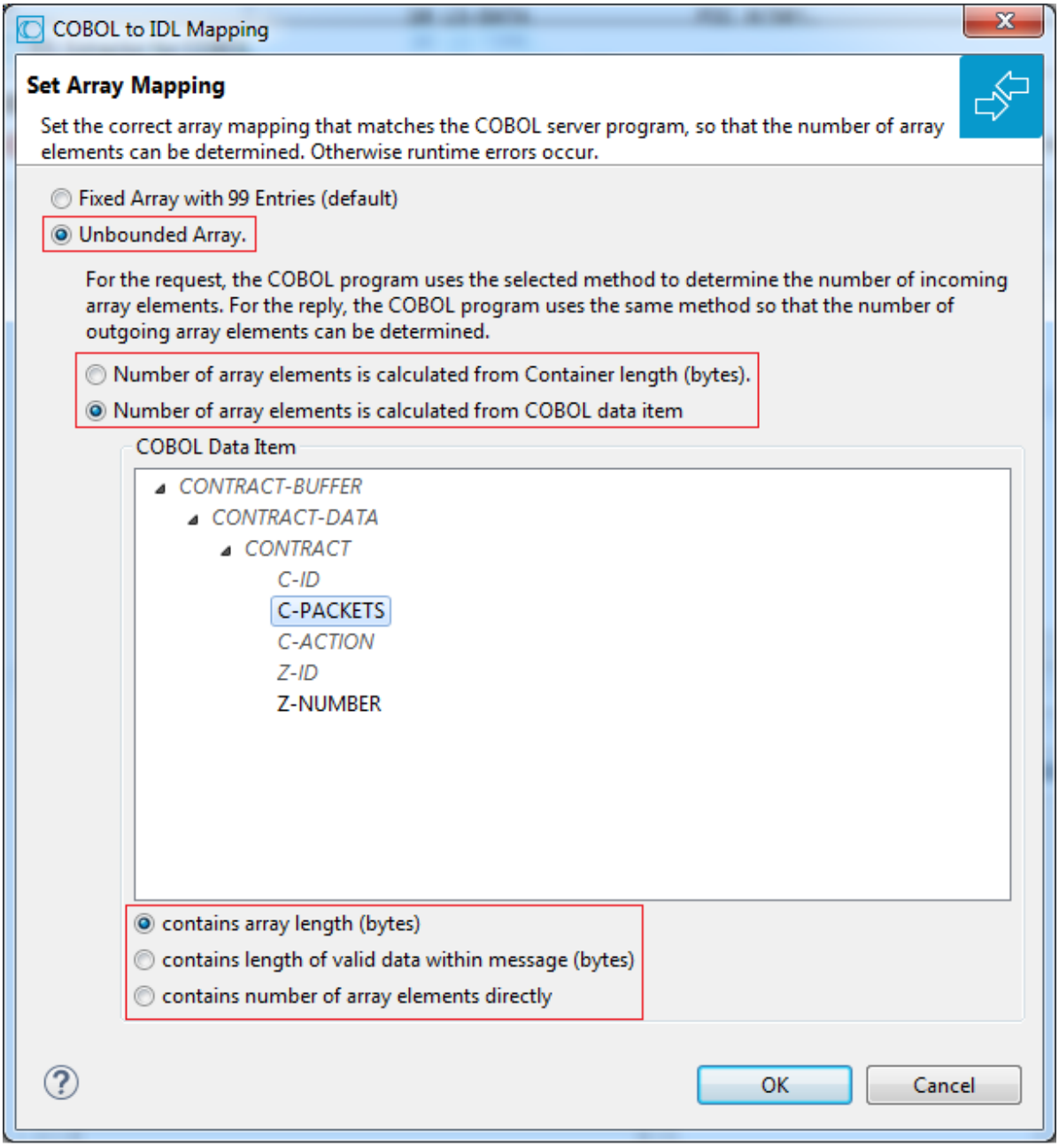
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **Container Length (bytes)**
  The COBOL server program inspects `CICS GET CONTAINER` parameter `FLENGTH` (container length for input) of the input container for the request and sets `CICS PUT CONTAINER` parameter `FLENGTH` (container length for output) of the output container for the reply. To determine the number of array elements, the container length is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL

snippet shows the reply of a CICS container. It assumes `LS-CONTRACT-BUFFER-LAYOUT` with fixed array `PACKETI` is the CICS container.

```
       WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
        01 LS-CONTRACT-BUFFER-LAYOUT.
          03 CONTRACT.
           04 C-ID                                PIC X(8).
           04 C-ACTION                            PIC X(4).
          03 ZONE.
           04 Z-NUMBER                            PIC 9(2).
           04 Z-ID                                PIC X(20).
          03 PACKETI                              OCCURS 99.
           04 P-ID                                PIC X(8).
           04 P-TEXT                              PIC X(30).
           04 P-NUMBER                            PIC 9(2).
       . . .

     *  Fill variable output array
           MOVE 0 TO II.
           PERFORM RANDOMNUM TIMES
              ADD 1 TO II
              MOVE ... TO P-ID    (II)
              MOVE ... TO P-TEXT  (II)
              MOVE ... TO P-NUMBER(II)
           END-PERFORM.
     *  Set container length depending on number of elements
           COMPUTE WS-LENGTH =
                  (LENGTH OF P-ID  +
                   LENGTH OF P-TEXT   +
                   LENGTH OF P-NUMBER) * II.
           ADD LENGTH OF CONTRACT TO WS-LENGTH.
           ADD LENGTH OF ZONE     TO WS-LENGTH.

     *  Reply CICS container
           EXEC CICS PUT
                CONTAINER (WS-CONTRACT-BUFFER-NAME)
                FROM      (LS-CONTRACT-BUFFER-LAYOUT)
                FLENGTH   (WS-LENGTH)
                RESP      (WS-RESP)
                RESP2
            END-EXEC.
```

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following

COBOL snippet shows how the COBOL interface CONTRACT-DATA is filled by the COBOL server on reply. The length of the fixed-size array PACKETI is contained in COBOL data item C-BYTES.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
           03 CONTRACT-DATA.
            04 CONTRACT.
             05 C-ID                               PIC X(8).
             05 C-BYTES                            PIC S9(4).
             05 C-ACTION                           PIC X(4).
            04 ZONE.
             05 Z-NUMBER                           PIC 9(2).
             05 Z-ID                               PIC X(20).
            04 PACKETI                             OCCURS 99.
             05 P-ITEM.
              06 P-ID                              PIC X(8).
              06 P-TEXT                            PIC X(30).
              06 P-NUMBER                          PIC 9(2).
       . . .

      *  Fill variable output array
         MOVE 0 TO II.
         PERFORM RANDOMNUM TIMES
            ADD 1 TO II
            MOVE ... TO P-ID    (II)
            MOVE ... TO P-TEXT  (II)
            MOVE ... TO P-NUMBER(II)
         END-PERFORM.
      *  Set table length
         COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (ZONED, PACKED or BINARY COBOL type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface CONTRACT is filled by the COBOL server on reply. COBOL data item C-APPDATA contains the length of the valid data of the reply message. The number of array elements of the fixed-size array PACKETI is implicitly contained in COBOL data item C-APPDATA.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
         77 EPARM                                 PIC  9(2).
         77 EPARM2                                PIC  9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
            04 CONTRACT.
             05 C-ID                              PIC X(8).
             05 C-APPDATA                         PIC S9(4).
             05 C-ACTION                          PIC X(4).
             05 Z-ID                              PIC X(20).
             05 Z-NUMBER                          PIC 9(2).
            04 PACKETI                            OCCURS 99.
             05 P-ITEM.
              06 P-ID                             PIC X(8).
              06 P-TEXT                           PIC X(30).
              06 P-NUMBER                         PIC 9(2).
       . . .
      *  Fill variable output array
         MOVE 0 TO II.
         PERFORM RANDOMNUM TIMES
            ADD 1 TO II
            MOVE ... TO P-ID    (II)
            MOVE ... TO P-TEXT  (II)
            MOVE ... TO P-NUMBER(II)
         END-PERFORM.
      *  Set length
         COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                           +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
       . . .
       LINKAGE SECTION.
         01 DFHCOMMAREA.
           03 CONTRACT-DATA.
            04 CONTRACT.
             05 C-ID                              PIC X(8).
             05 C-NUM                             PIC S9(4).
             05 C-ACTION                          PIC X(4).
            04 ZONE.
             05 Z-NUMBER                          PIC 9(2).
```

```
           05 Z-ID                                  PIC X(20).
          04 PACKETI                                OCCURS 99.
           05 P-ITEM.
            06 P-ID                                 PIC X(8).
            06 P-TEXT                               PIC X(30).
            06 P-NUMBER                             PIC 9(2).
     . . .

    *  Fill variable output array
       MOVE 0 TO II.
       PERFORM RANDOMNUM TIMES
          ADD 1 TO II
          MOVE ... TO P-ID    (II)
          MOVE ... TO P-TEXT  (II)
          MOVE ... TO P-NUMBER(II)
       END-PERFORM.
    *  Set occurrences
       MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

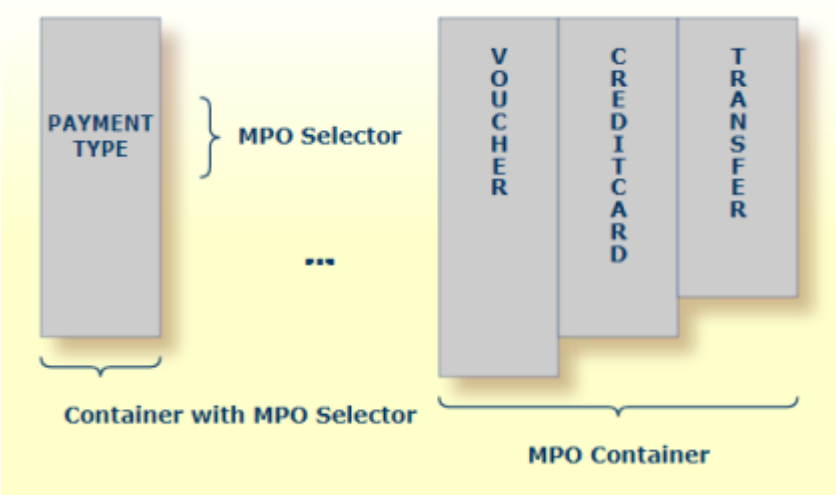
- Multiple Possible Output with Containers
- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with Containers

Containers can be used to describe the possible output structures. The COBOL server program decides at runtime which container is created and returned. In this way the output varies.

Containers are ideal for mapping MPO cases. The MPO selector and its value are contained in a different container, this is created by your COBOL server in any case. In this way, the caller of your COBOL server can investigate the MPO selector first to find out which MPO container was returned.



To map MPO with containers

1 Map the container with the MPO selector as a simple output container. See *Extracting from a CICS Channel Container Program*.

2 Map all MPO containers as optional containers, see *Map Optional Containers*.

Multiple Possible Output with REDEFINES

A COBOL `REDEFINES` *Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

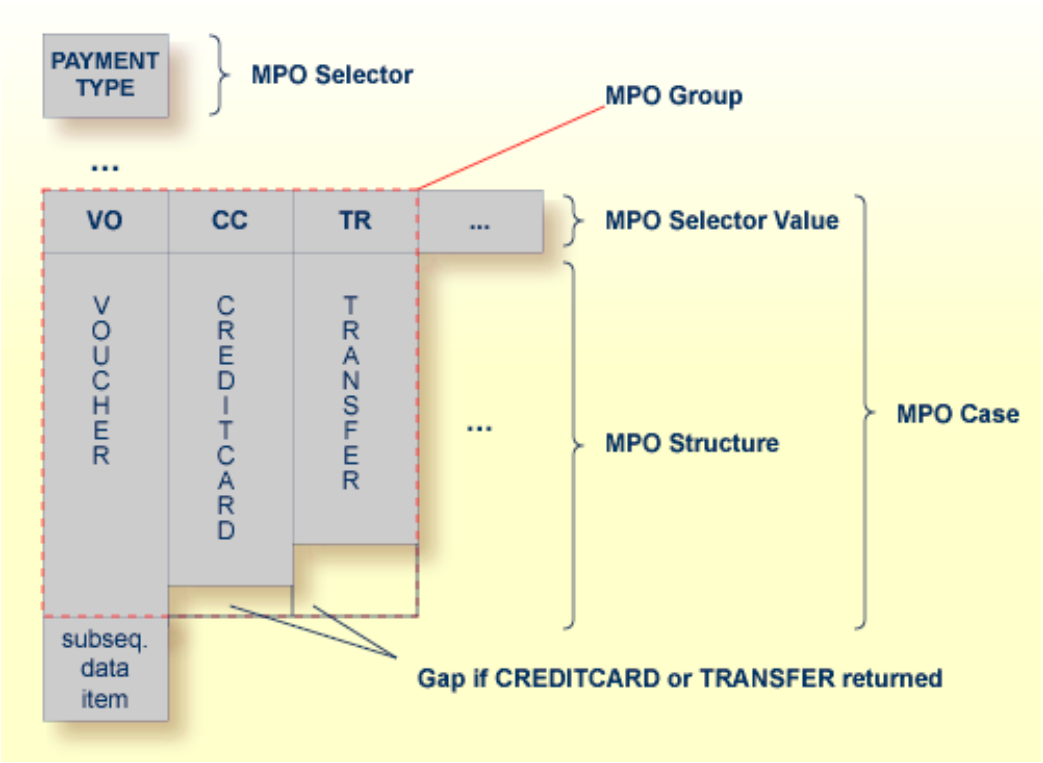
Similar to COBOL data item `PAYMENT-DATA` in the example below; for this purpose, `PAYMENT-DATA` is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item `PAYMENT-TYPE` (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

```
  . . .
    01 INPUT-DATA.
       02 ORDER-NUMBER                          PIC 9(10).

    . . .
    01 OUTPUT-DATA.
        02 <some fields>                        PIC <clause>.
       . . .

        02 PAYMENT-TYPE                         PIC X(2).
           88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
           88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
           88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
           88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
          . . .
        02 <preceding data items>                PIC <clause>.
       . . .
        02 PAYMENT-DATA                         PIC X(256).
        02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
           04 VOUCHER-ORIGIN                    PIC X(128).
           04 VOUCHER-SERIES                    PIC X(128).
        02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
           04 CREDITCARD-NUMBER                 PIC 9(18).
           04 CREDITCARD-COMPANY                PIC X(128).
           04 CREDITCARD-CODE                   PIC 9(12).
           04 CREDITCARD-VALIDITY               PIC X(8).
        02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
           04 TRANSFER-NAME                     PIC X(128).
           04 TRANSFER-IBAN                     PIC X(34).
           04 TRANSFER-BIC                      PIC X(11).
        02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
           04 DIRECTDEBIT-IBAN                  PIC X(34).
           04 DIRECTDEBIT-NAME                  PIC X(128).
           04 DIRECTDEBIT-EXPIRES               PIC 9(8).
          . . .
        02 <subsequent data items>               PIC <clause>.
       . . .

    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
```

```
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.
- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

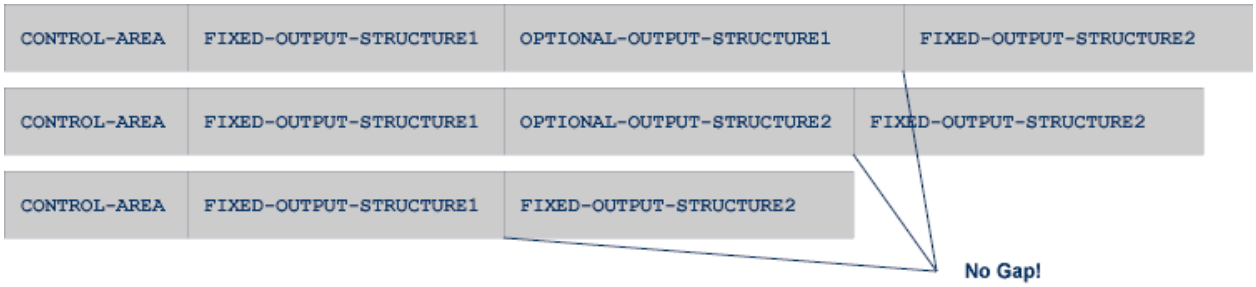
Optional Output with Groups

COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.
- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
      02 FIX-OUTPUT-ITEM13                  PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
```

```
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

Each value indicates exactly one output structure. An output structure can be indicated by further values.

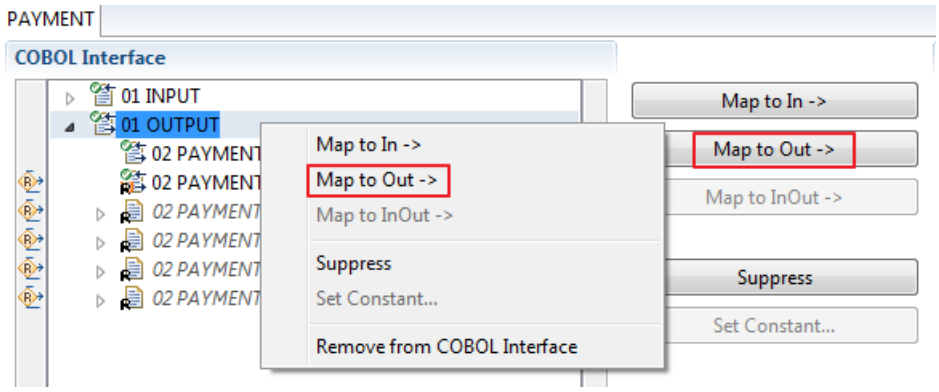
Steps

This section is for multiple possible output (MPO) with REEFINES or groups only. For multiple possible output (MPO) with containers, see *Multiple Possible Output with Containers*.

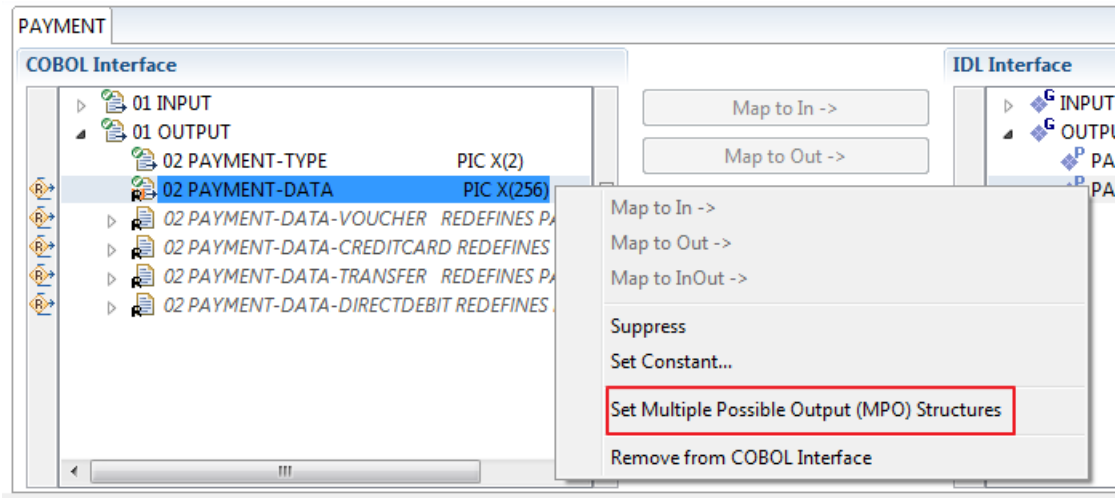
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

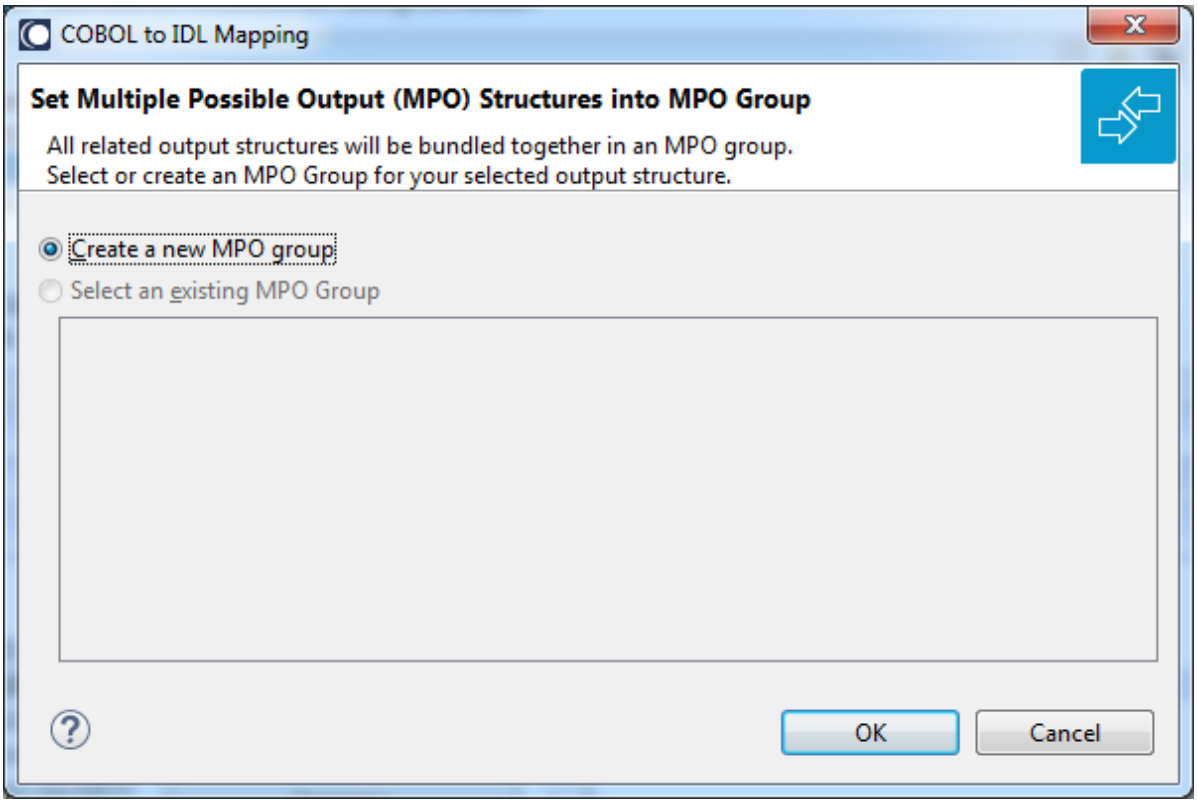
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



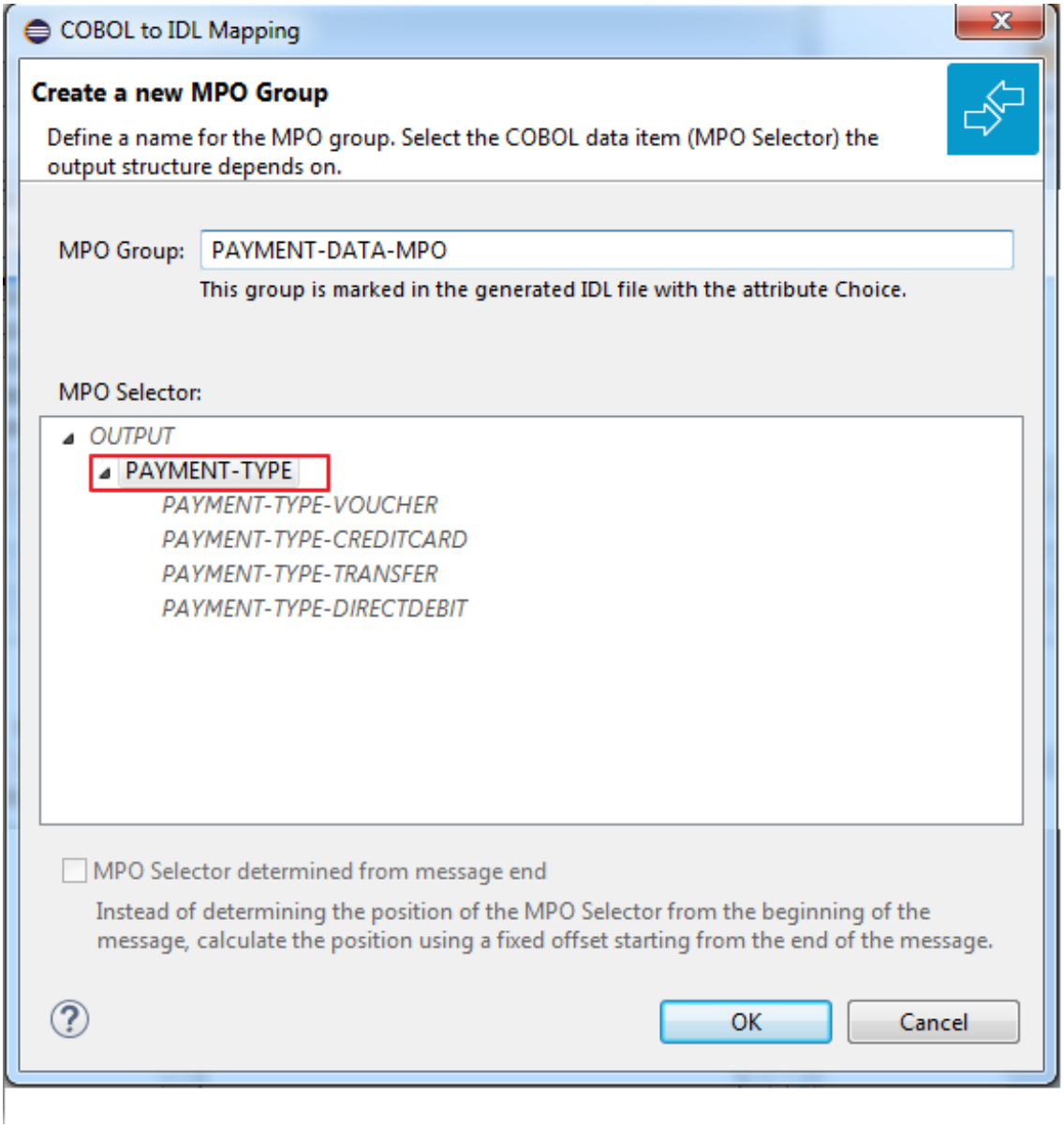
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



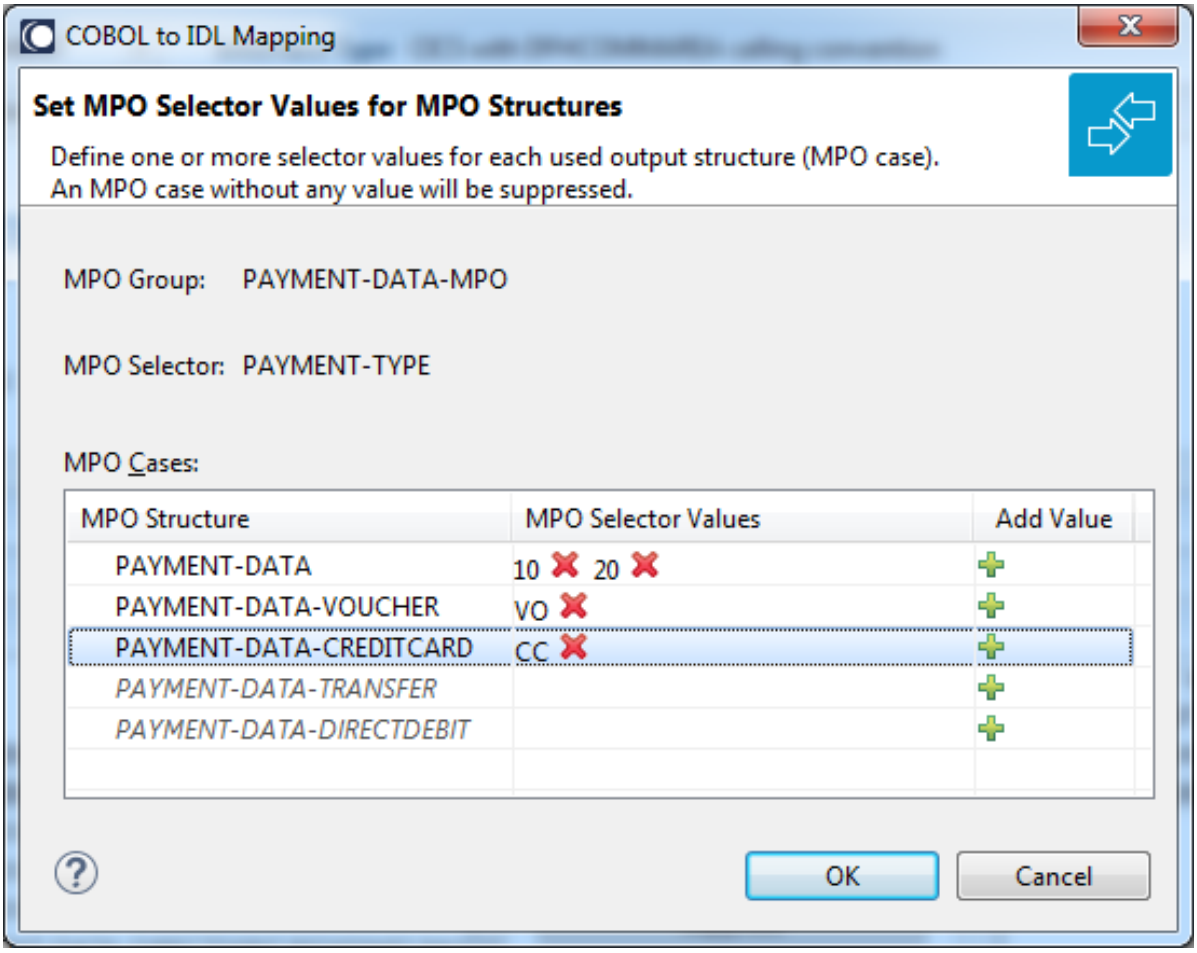
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



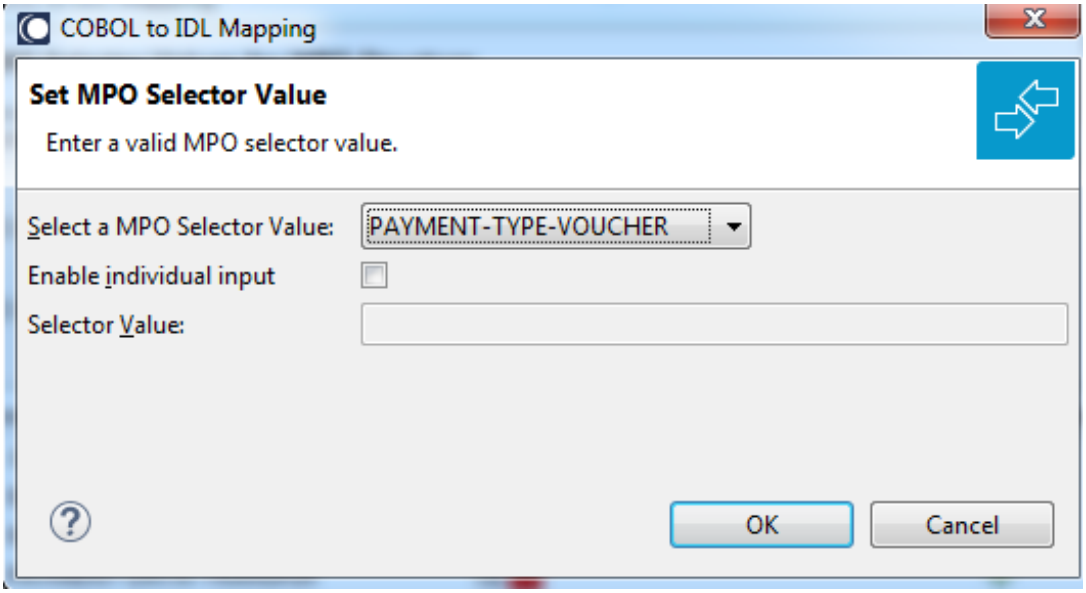
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

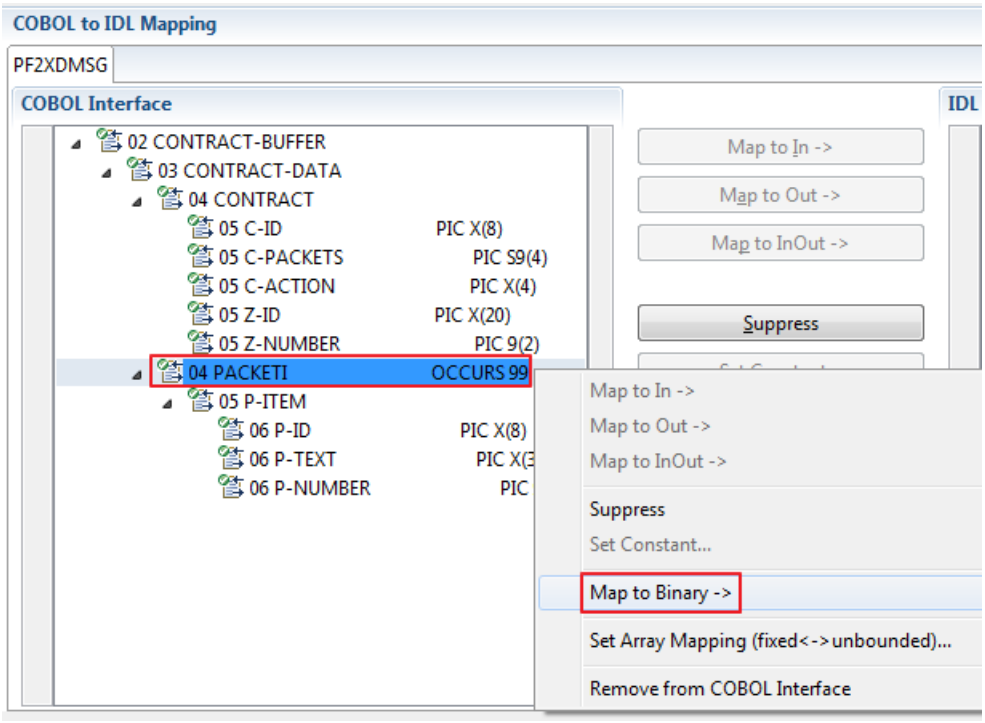
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT ('REQUEST') In
   1 OUTPUT ('REPLY')  Out
  end-define

 struct 'REQUEST' is
  define data parameter
   1 ORDER-NUMBER (NU10)
  end-define

 struct 'REPLY' is
  define data parameter
   1 PAYMENT-TYPE              (AV2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
     end-define
```

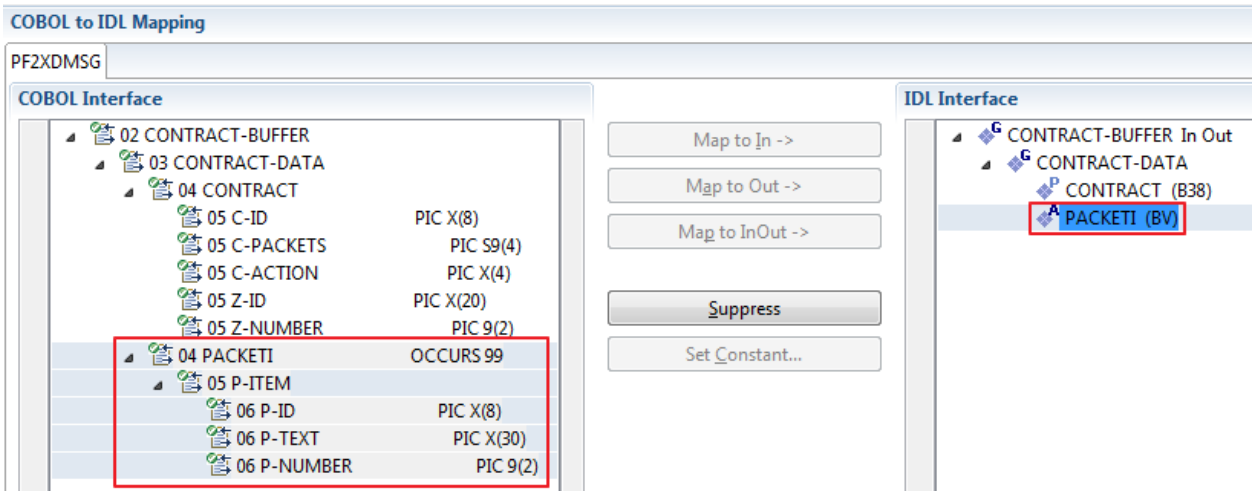
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage *Complex MPO Selections*.



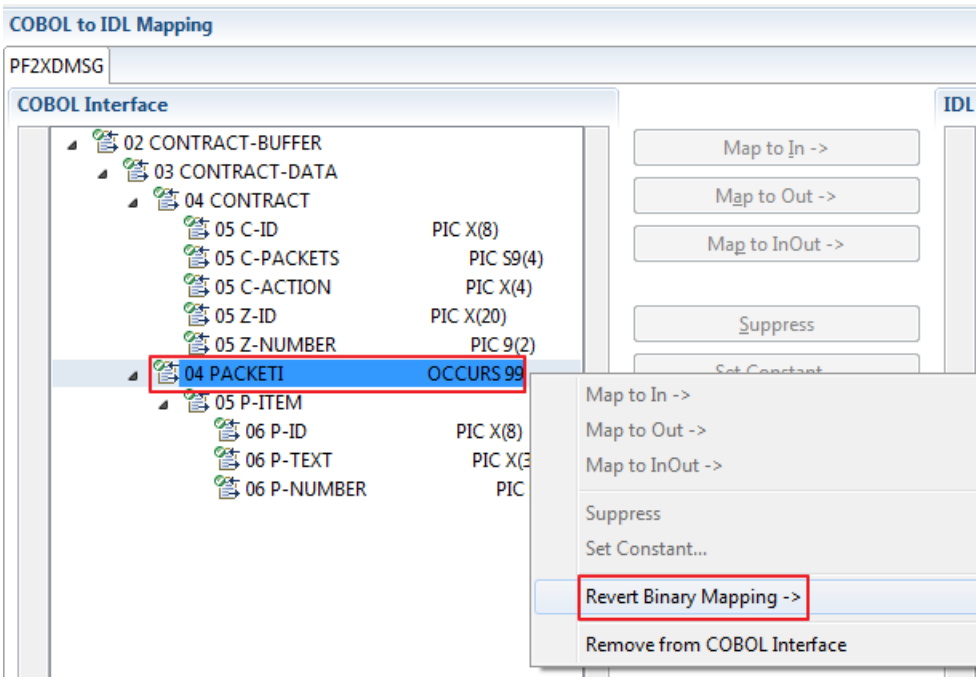
The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



Note: The last COBOL data items are mapped to IDL data type BV instead of B*n* (`PACKETI (BV)` in this example).

To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.



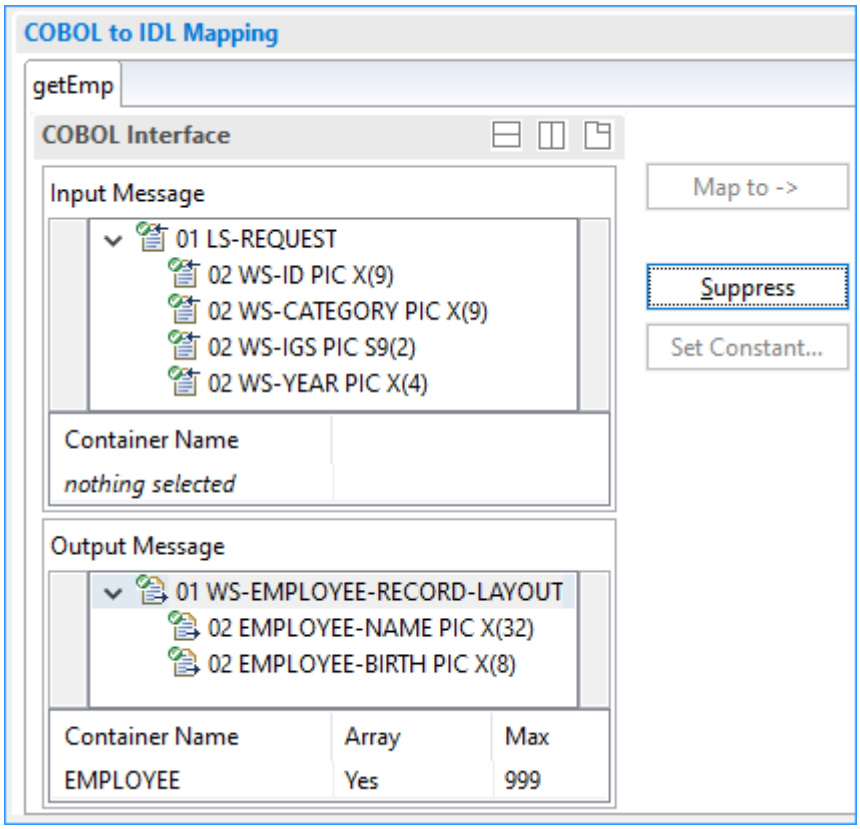
Map Array of Containers

If your COBOL server program creates a series of output containers, you can map them to an array of containers if the COBOL layout used is the same for each container and the container names are formed using a prefix.

In the example below, each container layout is described with COBOL data item `WS-EMPLOYEE-RECORDS-LAYOUT`, "`EMPLOYEE`" is the prefix used for each container name and 0 to 999 containers may be created.

```
 DATA DIVISION.
   WORKING-STORAGE-SECTION.
*  Direction OUT Container Names
   01 WS-EMPLOYEE-RECORDS-NAME PIC X(16) VALUE "EMPLOYEE000".
   01 REDEFINES WS-EMPLOYEE-RECORDS-NAME.
    02 WS-EMPLOYEE-RECORDS-PRE PIC X(8).
    02 WS-EMPLOYEE-RECORDS-NUM PIC 3.
    02 FILLER                  PIC X(5).
  . . .
  01 WS-EMPLOYEE-RECORDS-LAYOUT.
    02 EMPLOYEE-NAME           PIC X(32).
    02 EMPLOYEE-BIRTH          PIC 9(8).
  . . .
 PROCEDURE DIVISION.
   MAIN SECTION.
. . .
       PERFORM 0 ... 999 TIMES
           PERFORM 9100-REPLY-EMPLOYEE-RECORDS
       END-PERFORM.
 . . .
   9100-REPLY-EMPLOYEE-RECORDS.
       IF (WS-EMPLOYEE-RECORDS-NUM IS >= 999) THEN
* Overflow
          EXEC CICS ABEND
              ABCODE(ABEND-CONTAINER-OVF)
          END-EXEC
       END-IF.
       ADD 1 to WS-EMPLOYEE-RECORDS-NUM.
       EXEC CICS PUT
            CONTAINER (WS-EMPLOYEE-RECORDS-NAME)
            FROM      (WS-EMPLOYEE-RECORDS-LAYOUT)
            FLENGTH   (LENGTH OF WS-EMPLOYEE-RECORDS-LAYOUT)
            RESP      (WS-RESP)
            RESP2     (WS-RESP2)
        END-EXEC.
        IF (WS-RESP NOT = DFHRESP(NORMAL)) THEN
          EXEC CICS ABEND
               ABCODE(ABEND-CONTAINER-MOV)
          END-EXEC
        END-IF.
```

To map an array of containers



1 Locate the container layout and container name as described under *Extracting from a CICS Channel Container Program*.

2 Add the container layout to the **Output Message**.

3 Enter the prefix as the container name, set the column **Array** in the wizard to "Yes" and enter the maximum number of occurrences for the container in the **Max** column.

Notes:

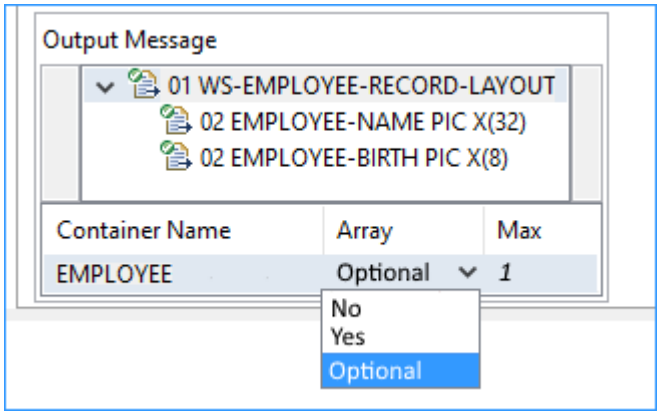
1. The container name length is restricted to 16 characters by CICS.
2. Make sure all containers related to this array created by your COBOL server program can be uniquely identified by its prefix and the number of created containers fit into the array boundaries. Set the **Max** column accordingly.
3. The easiest way is to use numbers as the suffix as in the COBOL program example above. If you do so, your COBOL program builds a container name using the prefix and enlarges it with the suffix. In the example, the COBOL server creates a maximum of 999 containers, resulting in container names `EMPLOYEE001` - `EMPLOYEE999`. In the **Max** column, 999 is set to be able to hold all possibly created containers.

4. Containers are delivered in the array in lexicographical order of their names. Because each container name has the same prefix, the lexicographical order of the suffix is relevant for it.
5. As a minimum, 2 can be specified in the **Max** column. If you enter 1, "Optional" will be forced in the column **Array**. Technically, an optional container is the same as an array of containers with upper limit 1. See *Map Optional Containers*.

Map Optional Containers

If your COBOL server program creates an output container under certain conditions only, map it as an optional container.

To map an optional container



1 Locate the container layout and container name as described under *Extracting from a CICS Channel Container Program*.

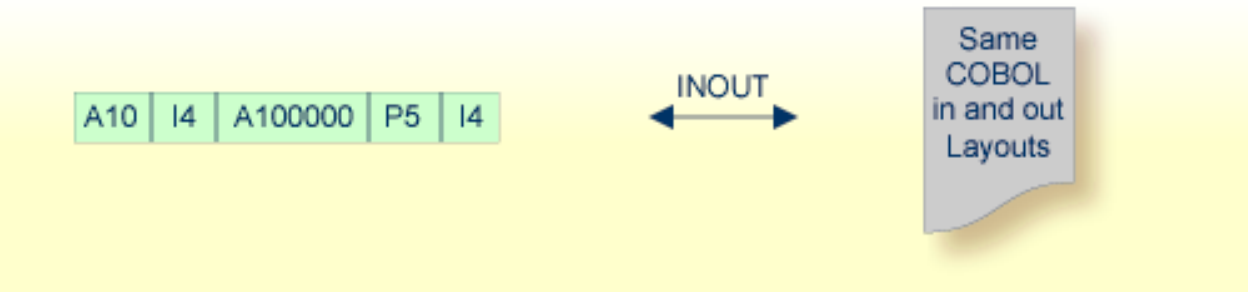
2 Add the container layout to the **Output Message**.

3 Enter the container name and set the column **Array** in the wizard to "Optional". This will force the maximum number of occurrences for container in the **Max** column to 1.

Notes:

1. Make sure your COBOL server program either creates the container with the name defined or does not create it.
2. Technically, an optional container is the same as an array of containers with 1 as upper limit (1 set in the **Max** column).
3. At runtime, the container array with upper limit 1 either contains the container if created by your COBOL server, or is empty (no item) if it not created.

11 COBOL Converter - In same as Out



Introduction

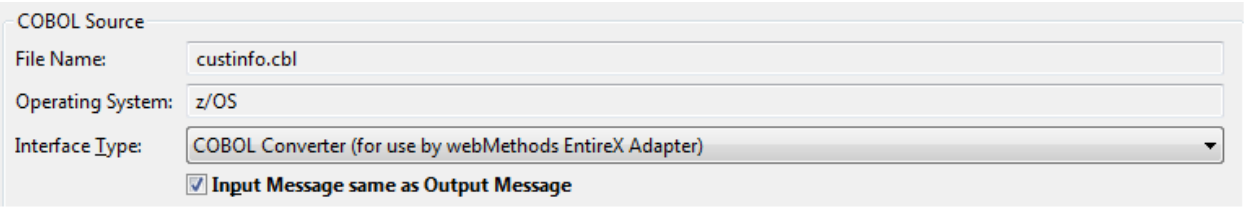
A file containing valid COBOL data items describing the COBOL payload can be used to extract a COBOL converter for the EntireX Adapter. If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

Extracting a COBOL Converter

This section assumes **Input Message same as Output Message** is checked. COBOL output and COBOL input parameters are the same, that is, the output is not overlaid with a data structure different to the data structure on input.

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

Make sure the interface type is correct and checkbox **Input Message same as Output Message** is not cleared.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items to the **COBOL Interface** by using the context menu or toolbar available in the ***COBOL Source View*** and ***COBOL Interface***. See **Notes**.

2 Continue with ***COBOL to IDL Mapping***.

Notes:

1. It is very important to select the right COBOL data items describing the COBOL interface 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 interface contains `REDEFINE`s, 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.

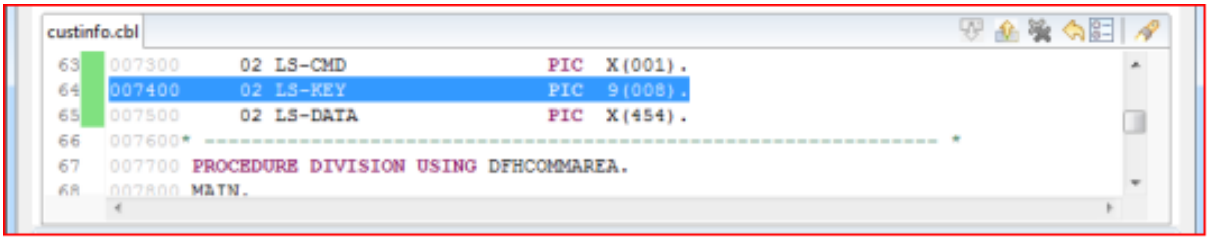
Mapping Editor User Interface

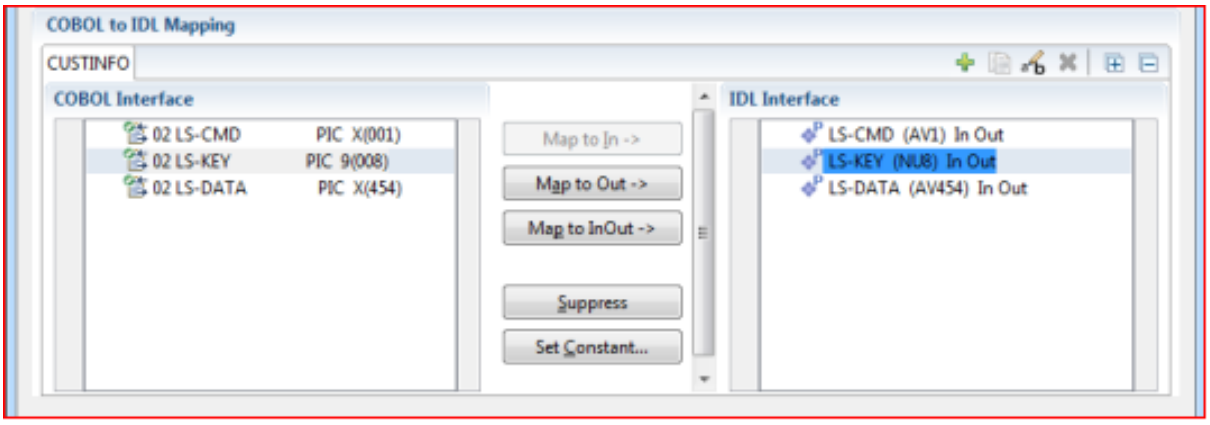
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface type COBOL Converter, the user interface of the COBOL Mapping Editor looks like this:

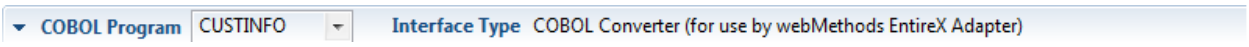
1. 

2. 

3. 

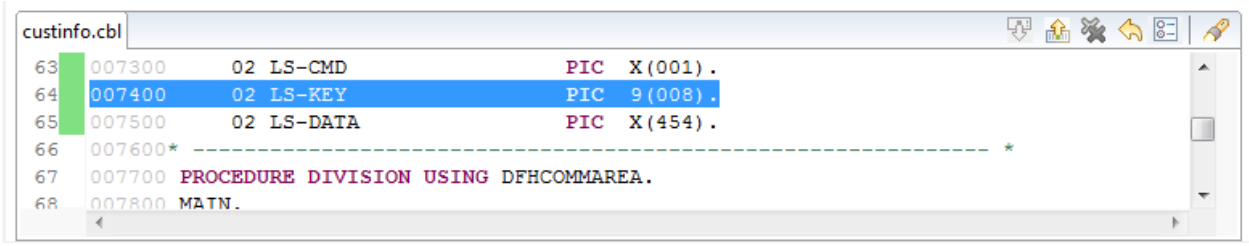
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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.
- 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

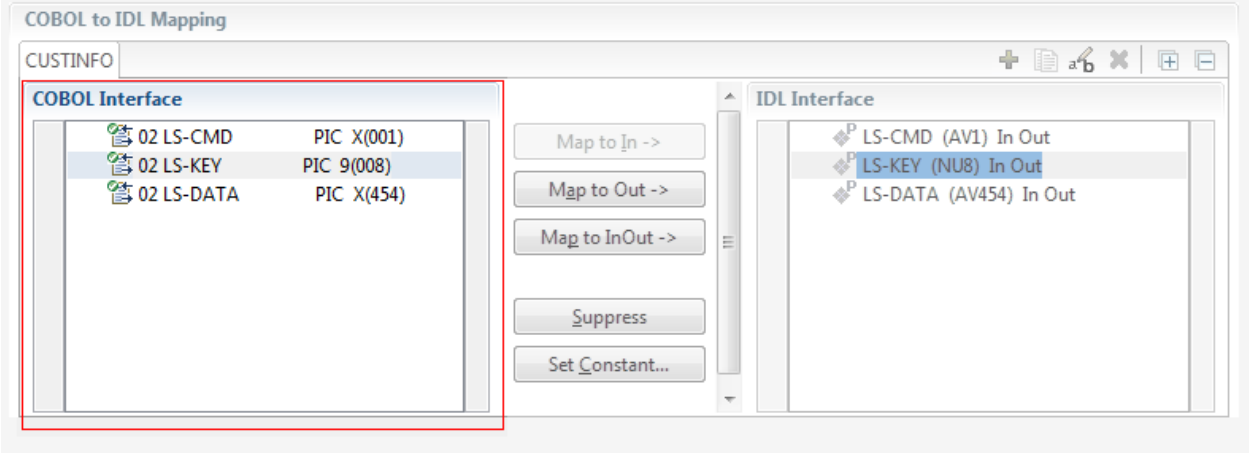
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

Map to In | Out | InOut	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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

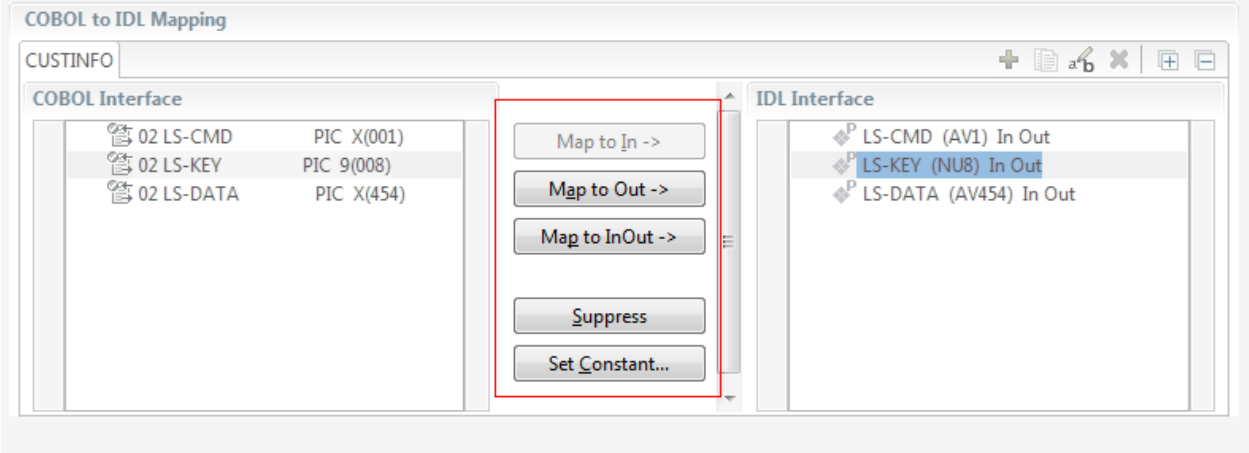
Mapping Icons

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

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

Mapping Buttons

The following buttons are available:



Map to In | Out | InOut ->

See *Map to In, Out, InOut*. 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*.

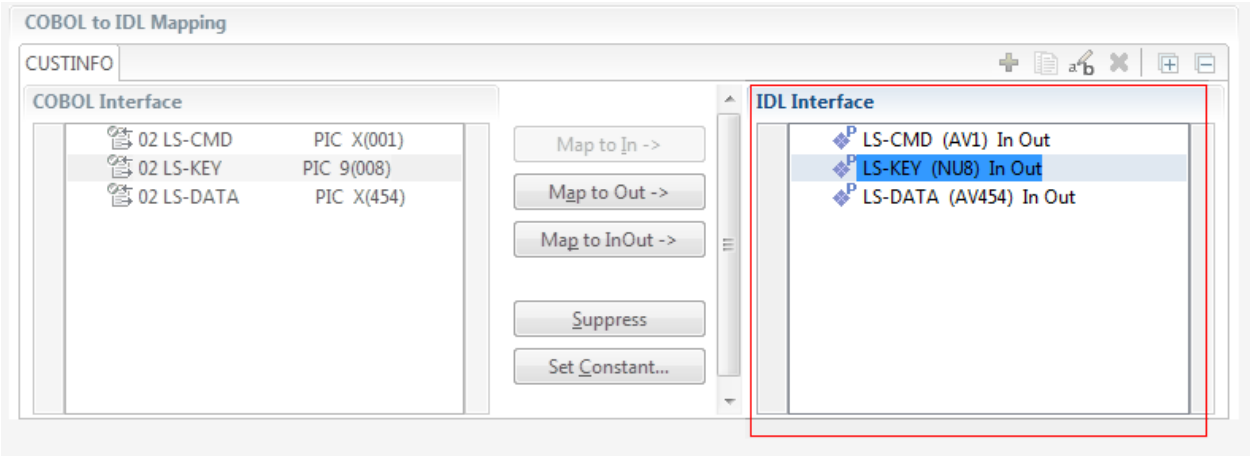
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

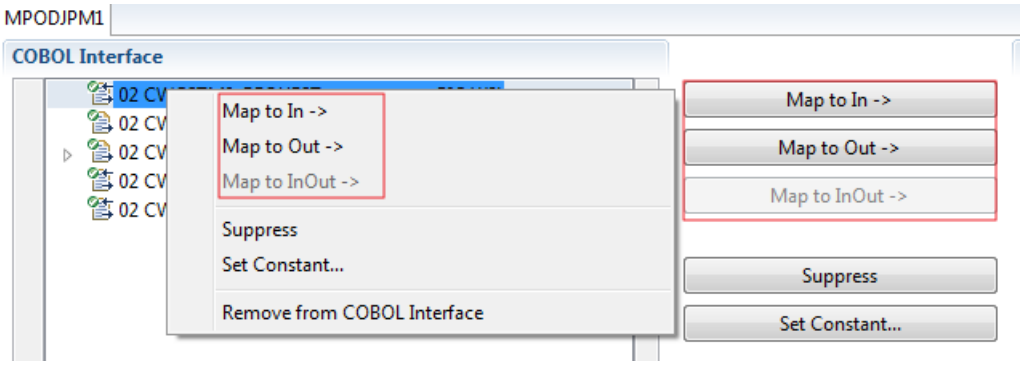
This section covers the following topics:

Map to In, Out, InOut

With the **Map to In**, **Out**, **InOut** functions you make a COBOL data item visible as an IDL parameter in the IDL interface. With correct IDL directions you design the IDL interface by defining input and output parameters. COBOL programs have no parameter directions, so you need to set IDL directions manually.

To provide IDL directions

- Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons to make the COBOL data items visible and provide IDL directions in the IDL interface:



Notes:

1. If a *top-level* COBOL *group* is mapped, the IDL direction is inherited by all subordinate COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subordinate COBOL data items can only be mapped to the same IDL direction as their *top-level* COBOL *group* data item.
3. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.
4. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map OCCURS DEPENDING ON

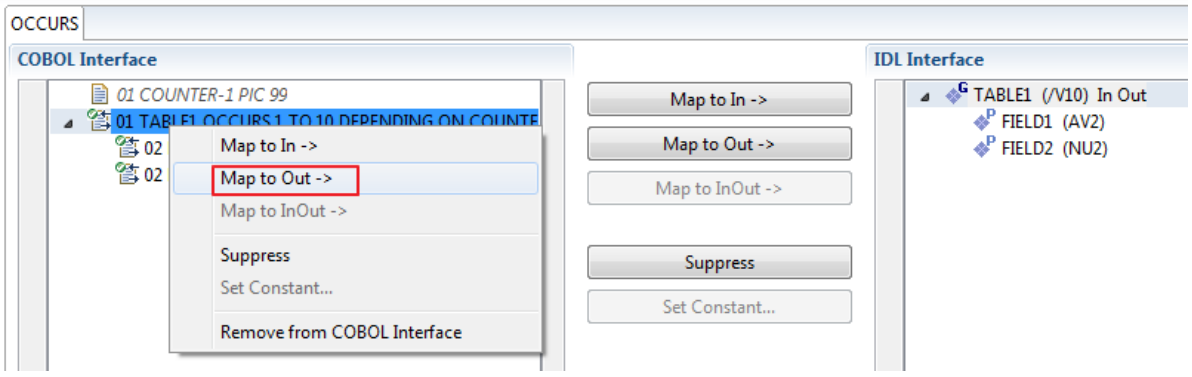
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - DEPENDING ON Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):

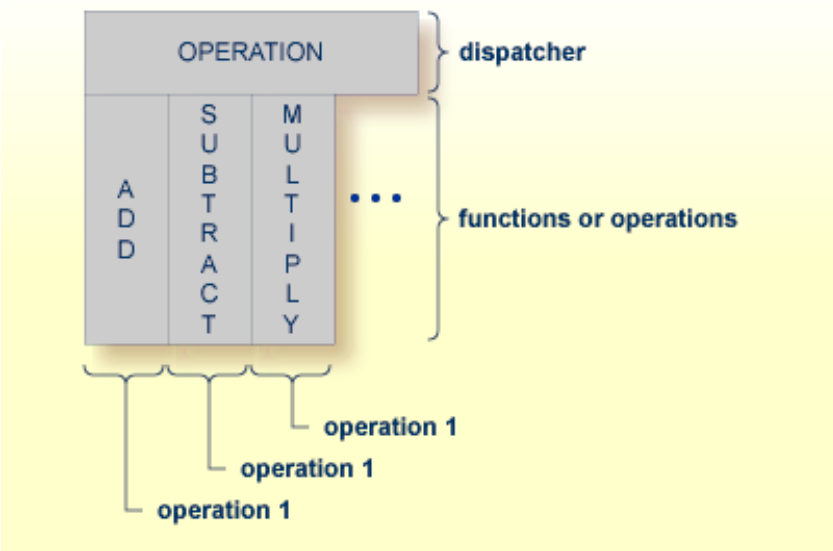


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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
```

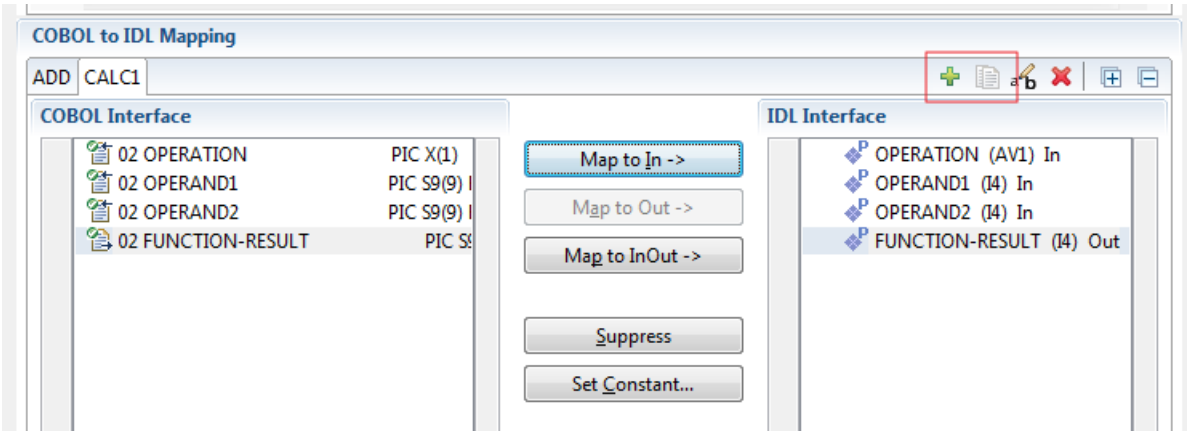
```
         WHEN "*"
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

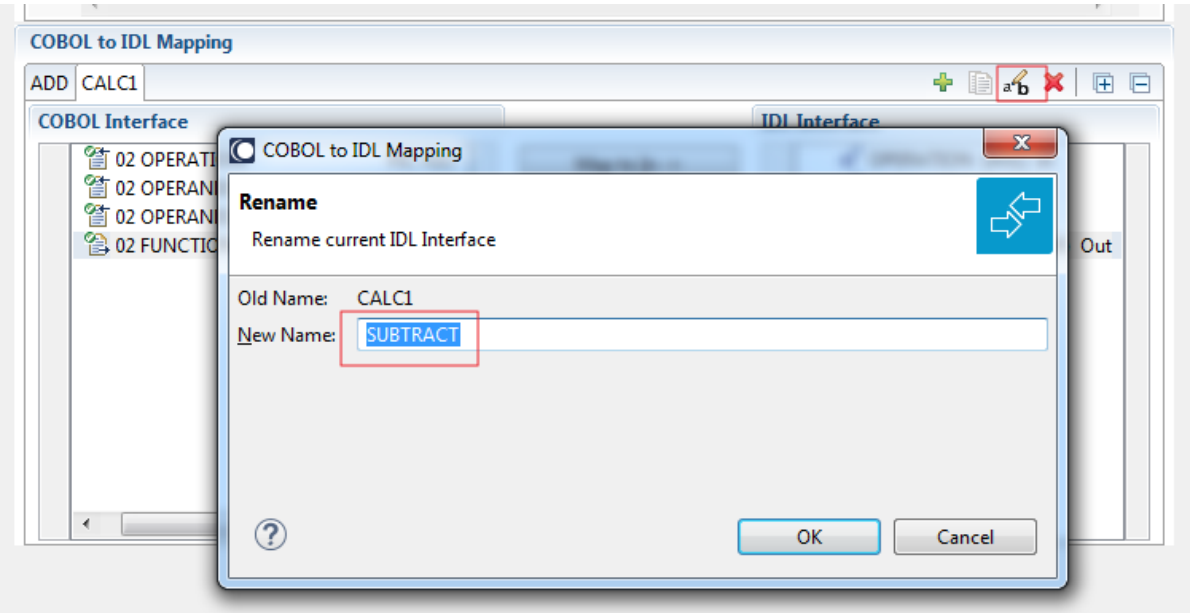
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

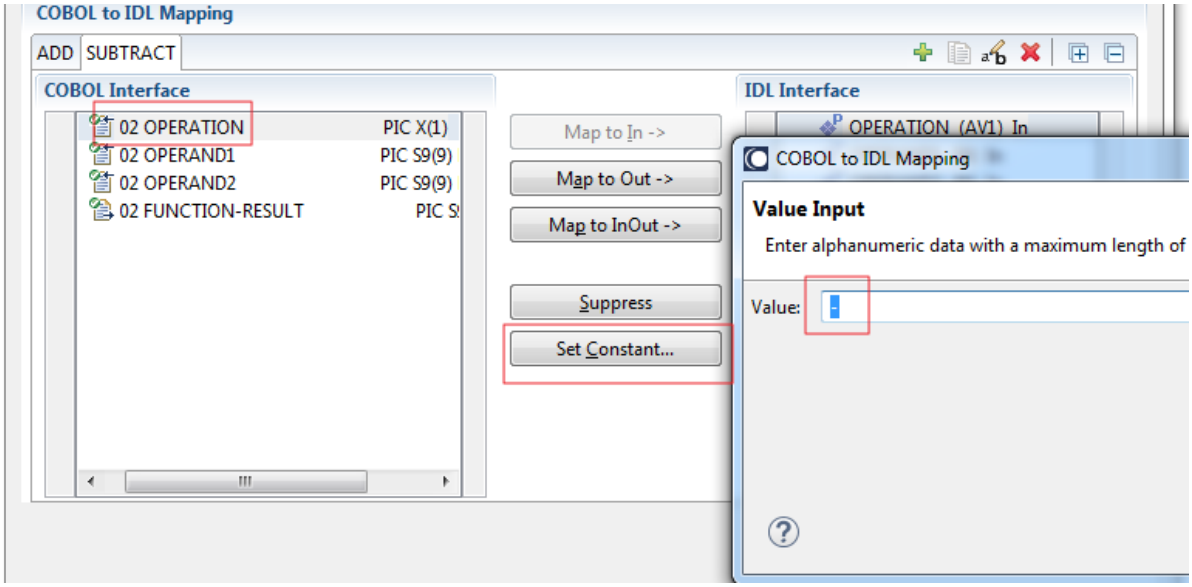
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions  or  :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

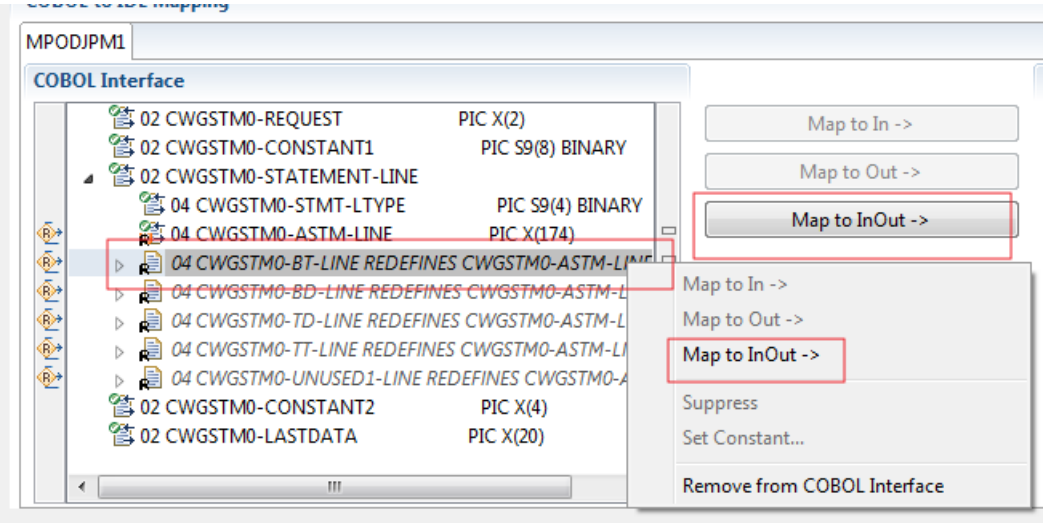
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

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

To select redefine paths

- Use the **Map to In**, **Out** or **InOut** functions available in the context menu of the COBOL interface and as mapping buttons 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:

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

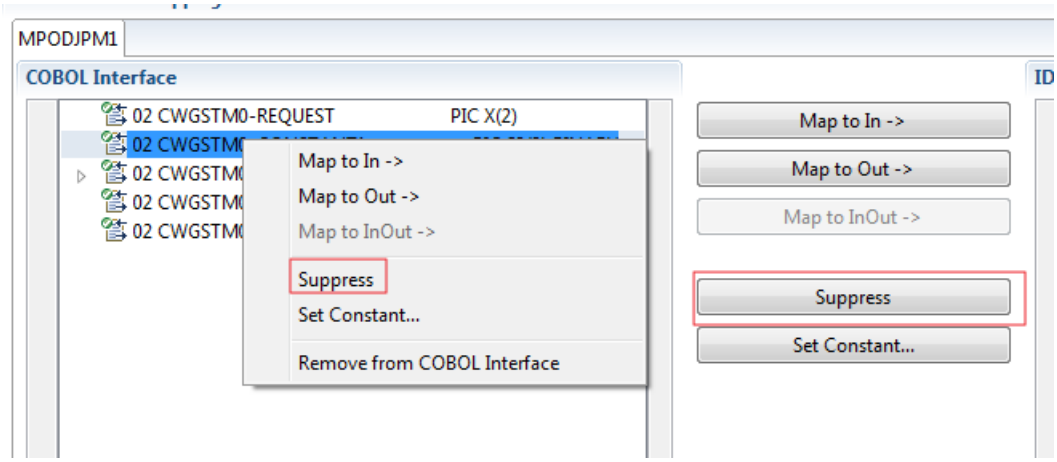
Suppress Unneeded COBOL Data Items

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 consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

To suppress unneeded COBOL data items

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



Notes:

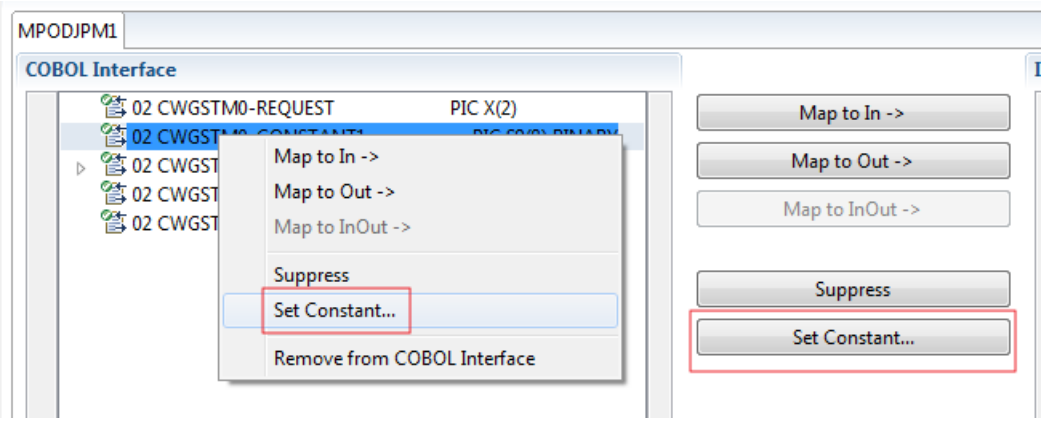
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subordinate COBOL data items are suppressed as well.
4. With the inverse functions **Map to In**, **Out** or **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item is made visible in the IDL interface again.

Set COBOL Data Items to Constants

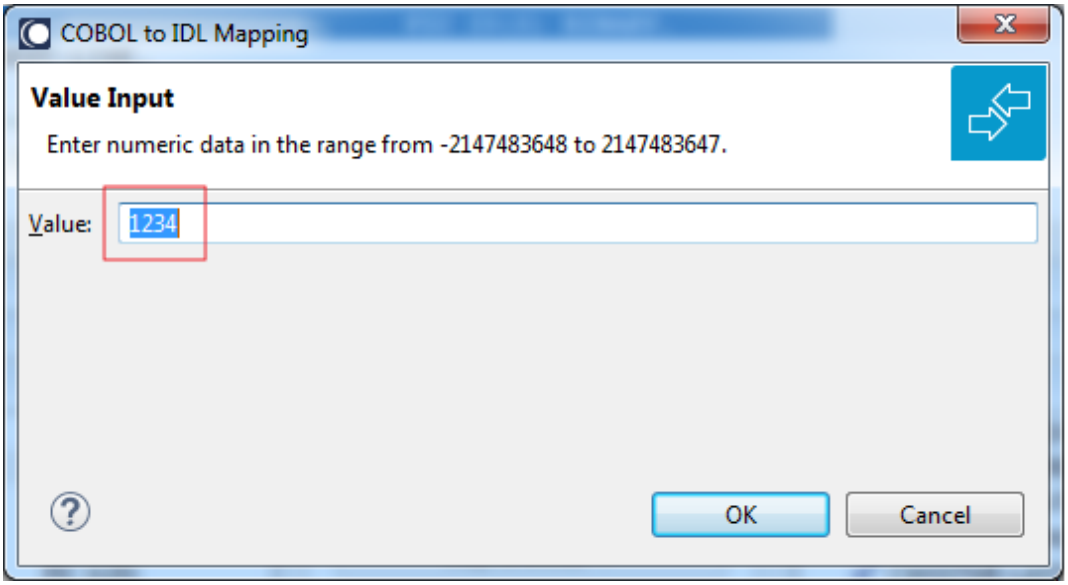
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter provides the defined constant in the COBOL data item to your COBOL server.
3. With the functions **Map to In**, **Out**, **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item can be made visible in the IDL interface again.

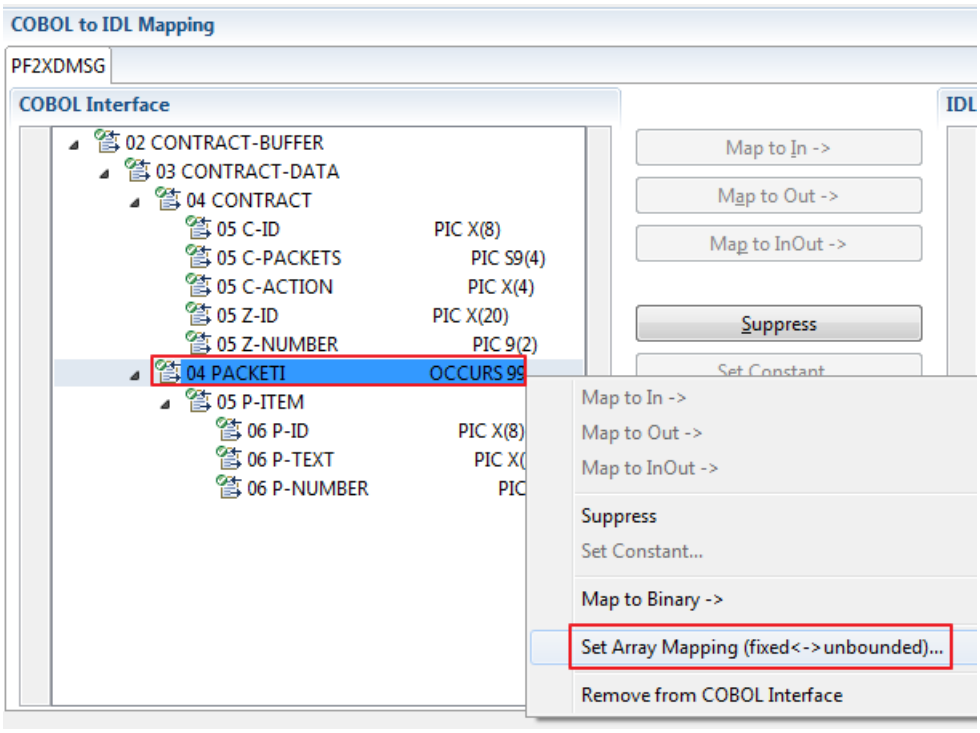
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

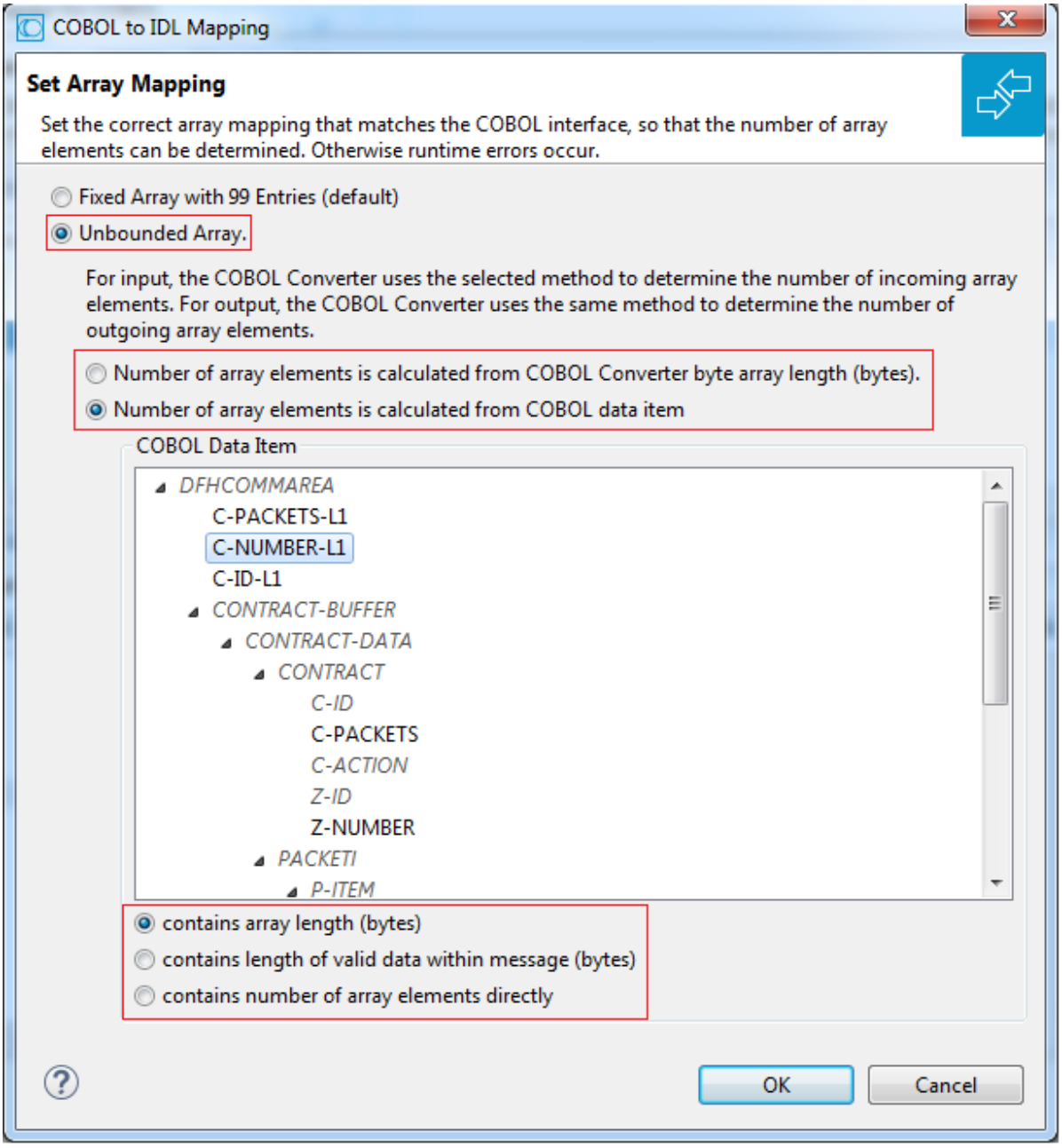
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **COBOL Converter length (bytes)**
  For input, the COBOL Converter uses the length of byte array `cobolInput` as COBOL Converter length. For output, the length of byte array `cobolOutput` has to be used accordingly. To determine the number of array elements, the length of the byte array is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` used in this manner:

```
          01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-BYTES                           PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
```

Assume the array `PACKETI` is filled with 15 occurrences. The length of byte array `cobolInput` is calculated then as follows: (`LENGTH OF ZONE`) + (`LENGTH OF CONTRACT`) + (`LENGTH OF P-ITEM`) * 15.
The number of array elements of the fixed-size array `PACKETI` is implicitly contained in the COBOL converter length.

- **COBOL data item contains array length (bytes)**
  For input, The COBOL Converter inspects the COBOL data item in byte array `cobolInput` and sets it accordingly for output in byte array `cobolOutput`. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` and `C-BYTES` as the COBOL data item used in this manner:

```
          01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-BYTES                           PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
```

Assume the array `PACKETI` is filled with 7 occurrences. The contents of COBOL data item `C-BYTES` are calculated as follows: `(LENGTH OF P-ITEM) * 7`.
The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-BYTES`.

- **COBOL data item contains length of valid data within messages (bytes)**
  For input, The COBOL converter inspects the COBOL data item in byte array `cobolInput` and sets it accordingly for output in byte array `cobolOutput`. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than COBOL converter length. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` and `C-APPDATA` as the COBOL data item used in this manner:

```
        01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                                PIC X(8).
            05 C-APPDATA                           PIC S9(4).
            05 C-ACTION                            PIC X(4).
            05 Z-ID                                PIC X(20).
            05 Z-NUMBER                            PIC 9(2).
           04 PACKETI                              OCCURS 99.
            05 P-ITEM.
             06 P-ID                               PIC X(8).
             06 P-TEXT                             PIC X(30).
             06 P-NUMBER                           PIC 9(2).
```

  Assume the array `PACKETI` is filled with 31 occurrences. The contents of COBOL data item `C-APPDATA` are calculated as follows: `(LENGTH OF CONTRACT) + (LENGTH OF P-ITEM) * 31`. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

- **COBOL data item contains number of array elements directly**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                   PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-NUM                             PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
```

```
           06 P-ID                                 PIC X(8).
           06 P-TEXT                               PIC X(30).
           06 P-NUMBER                             PIC 9(2).
     . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set occurrences
        MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

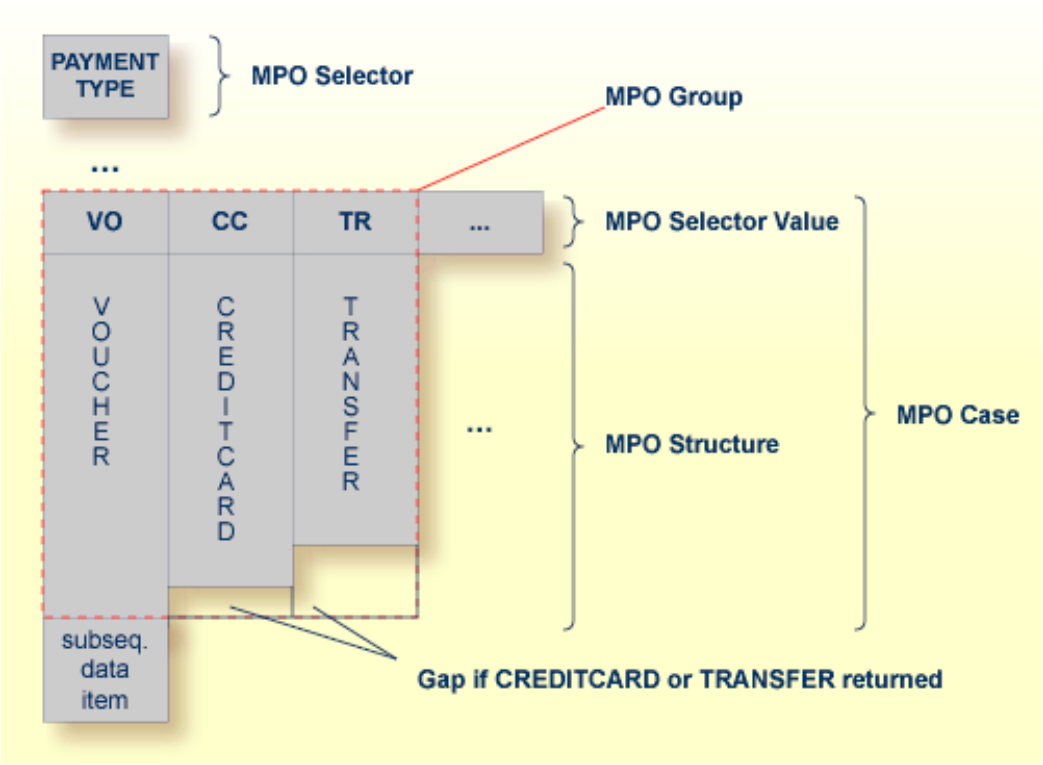
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>               PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>              PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

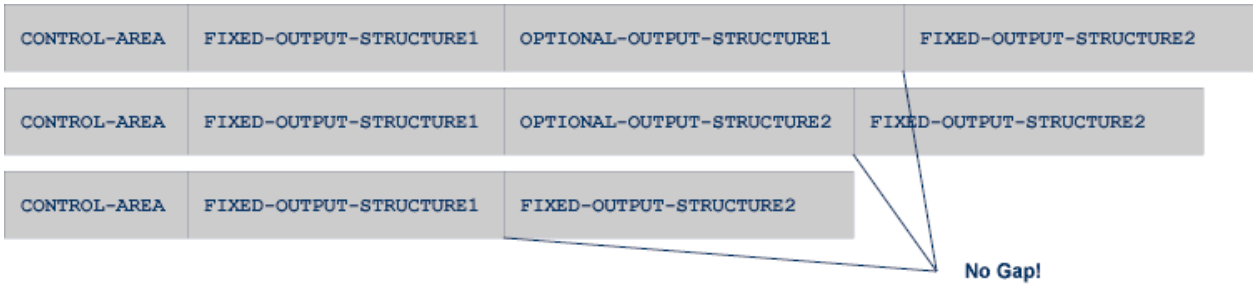
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

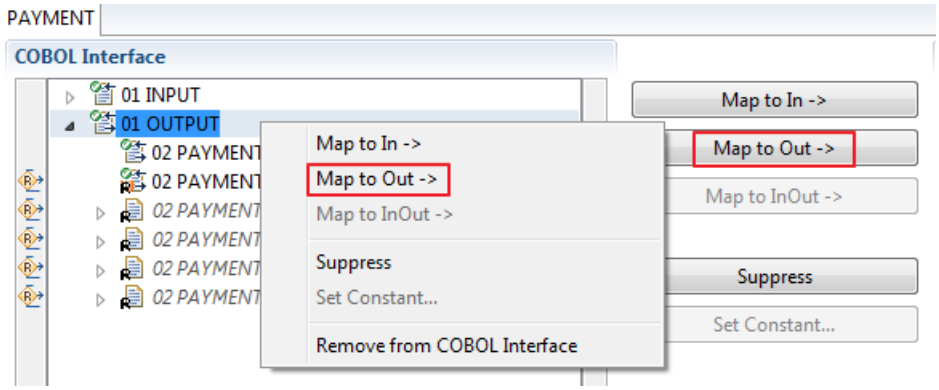
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

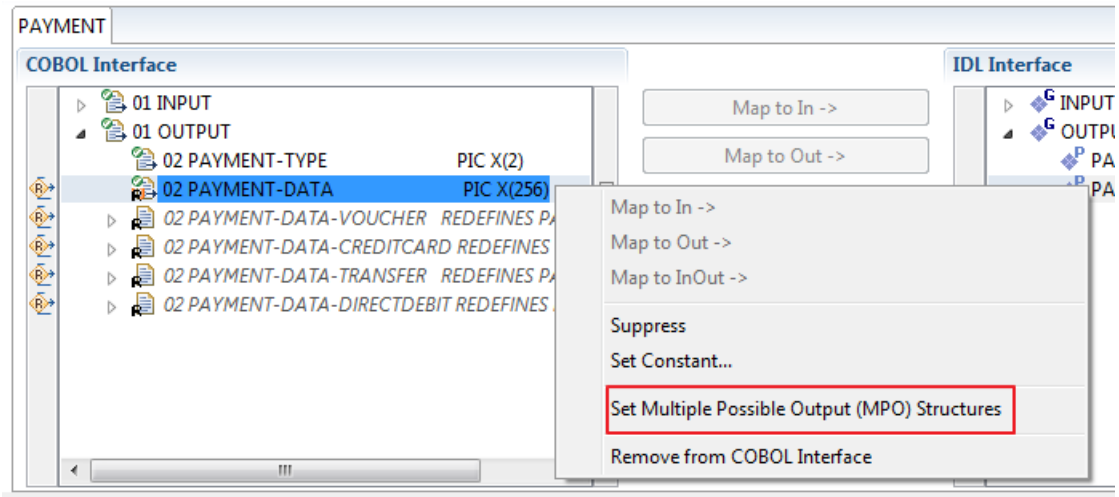
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

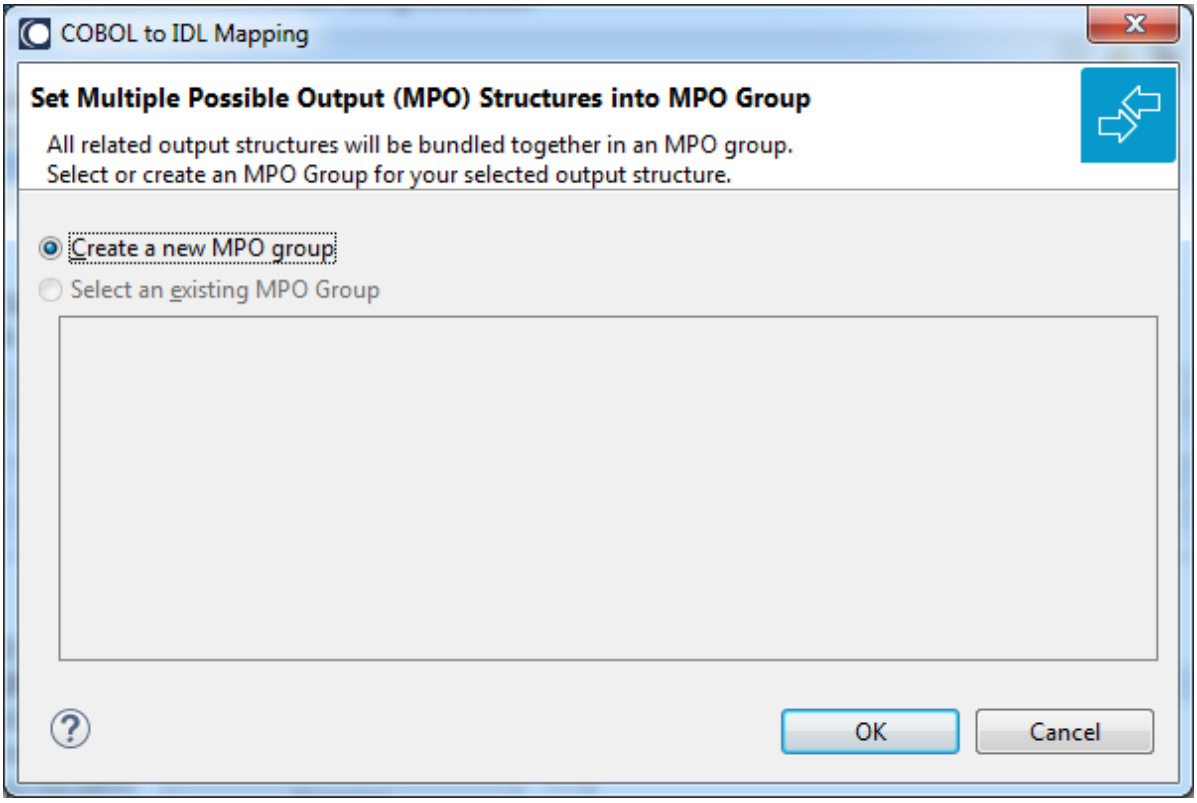
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



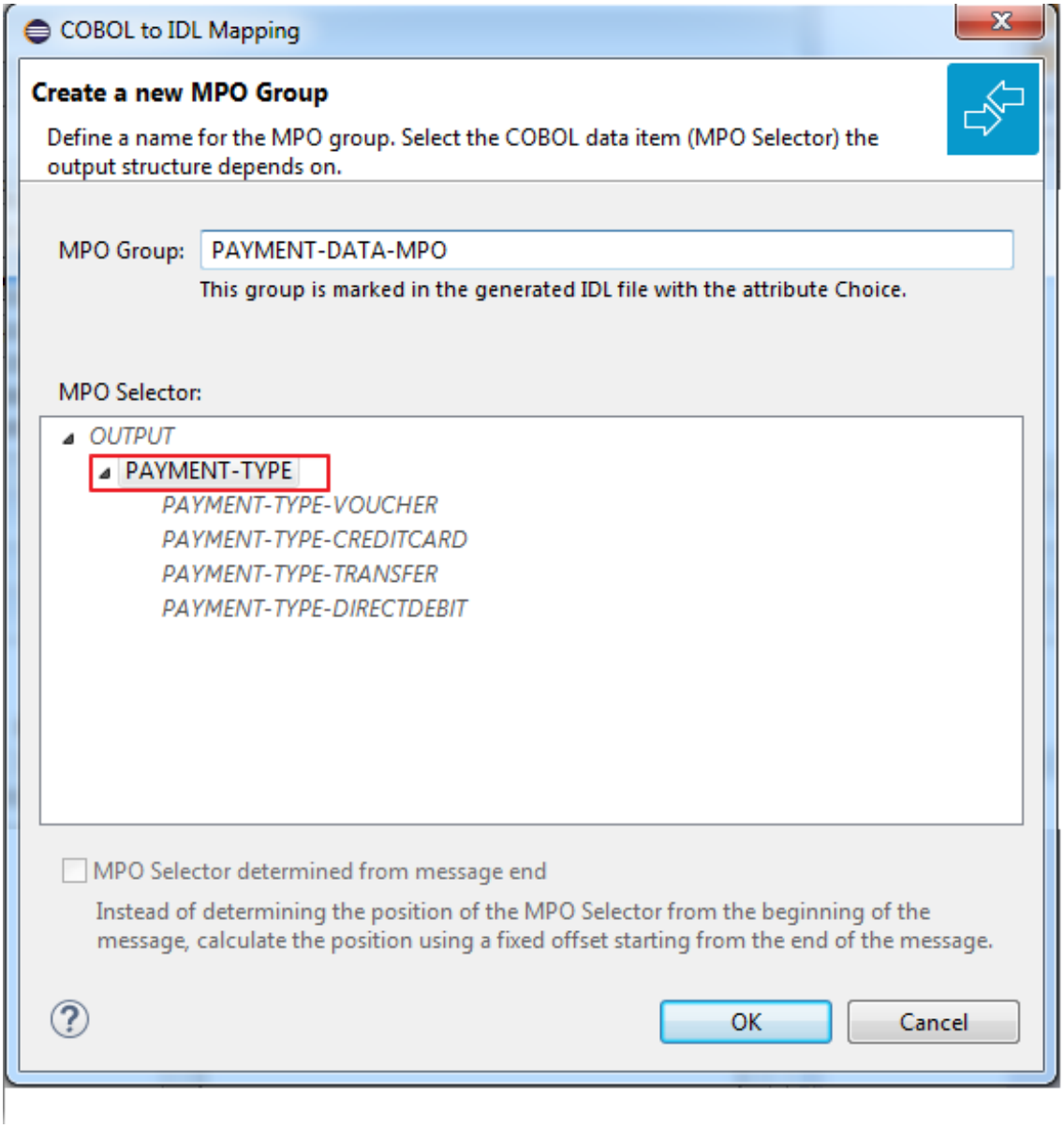
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



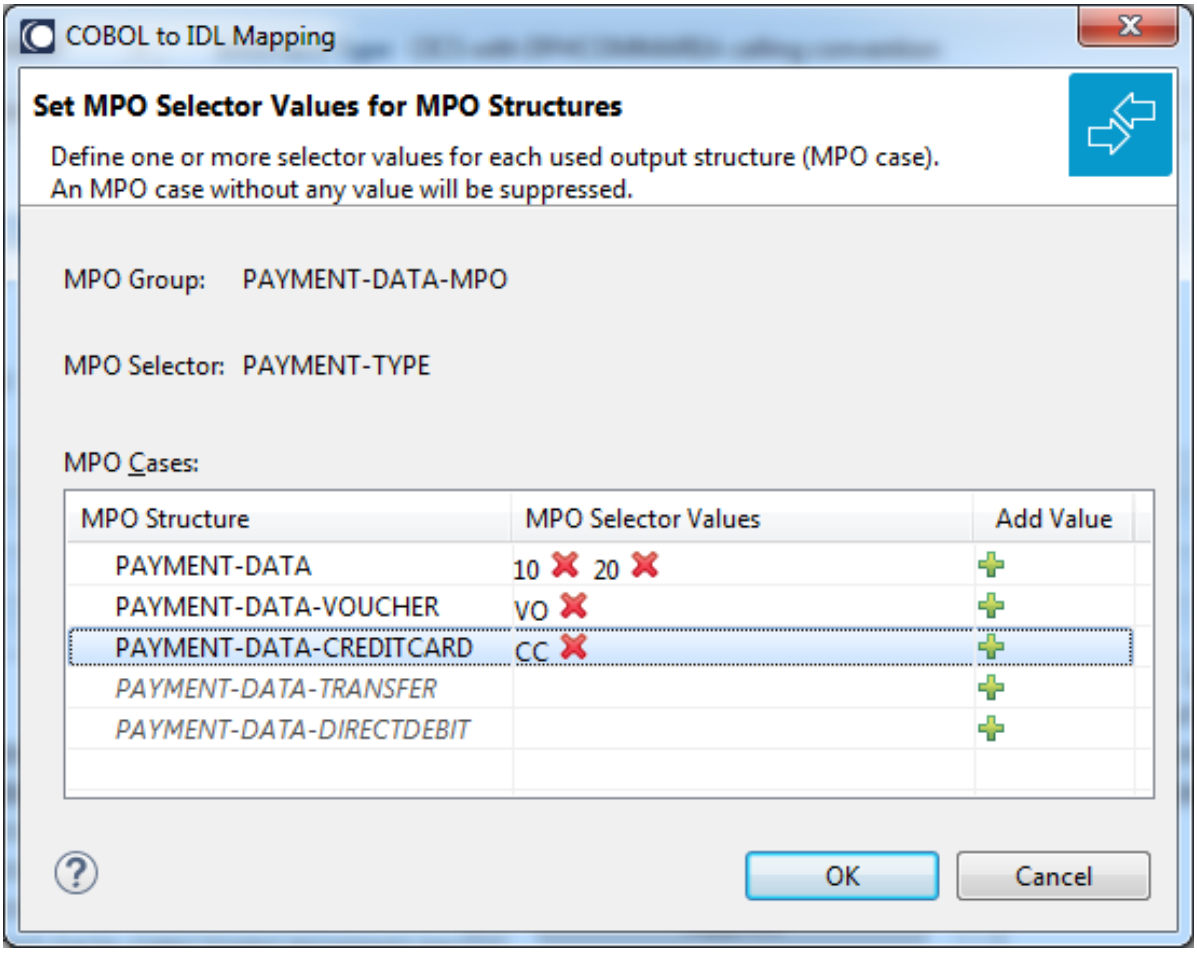
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



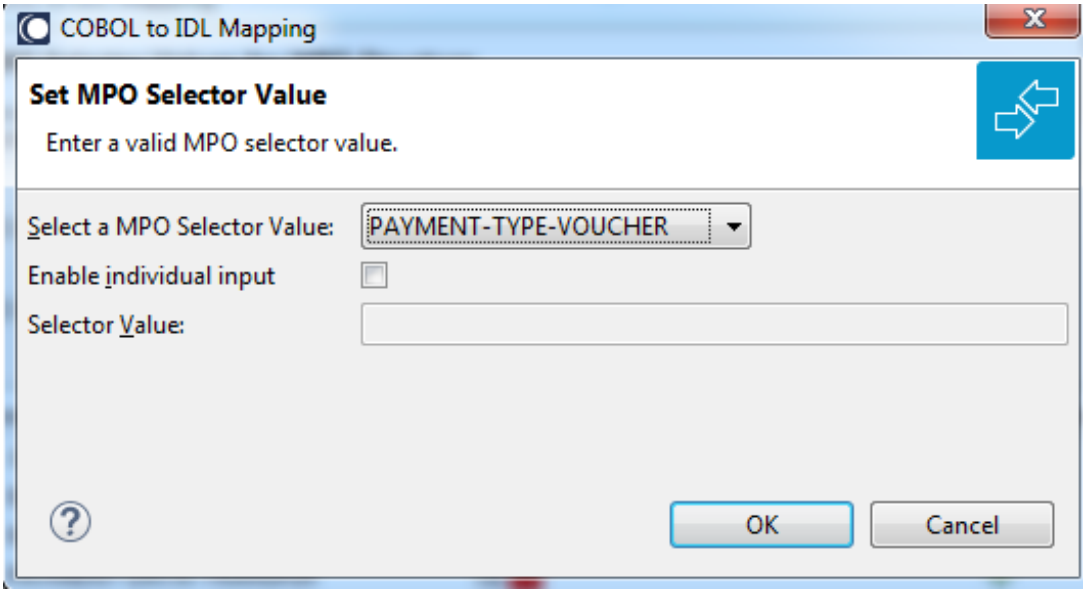
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

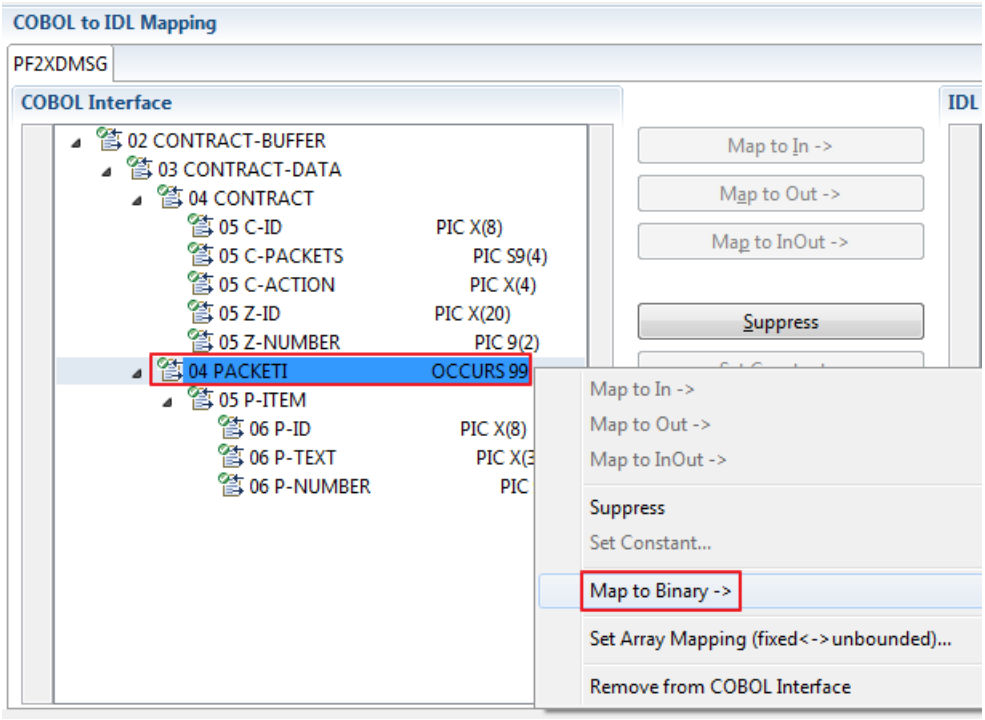
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

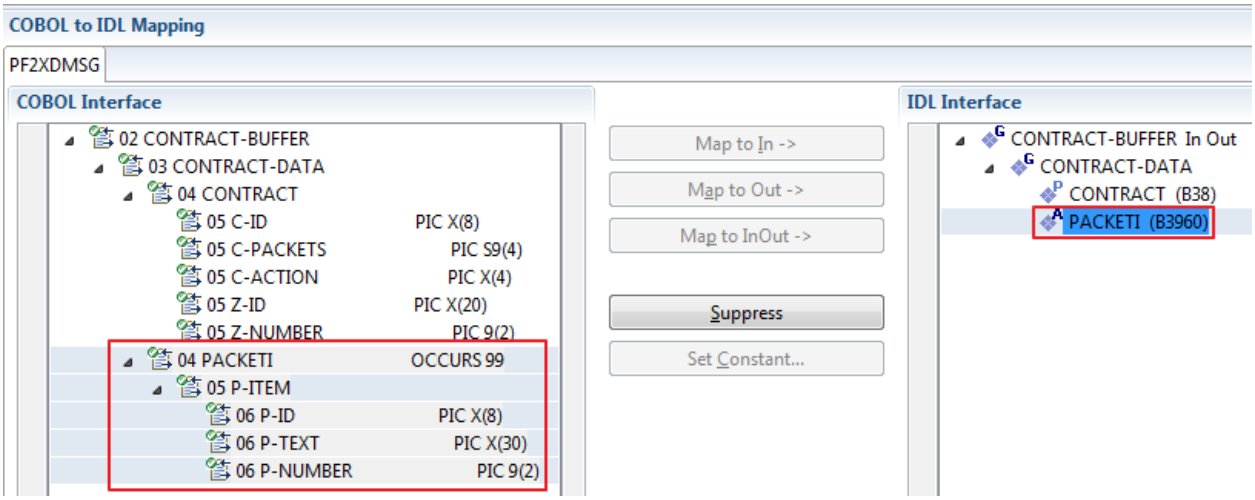
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.

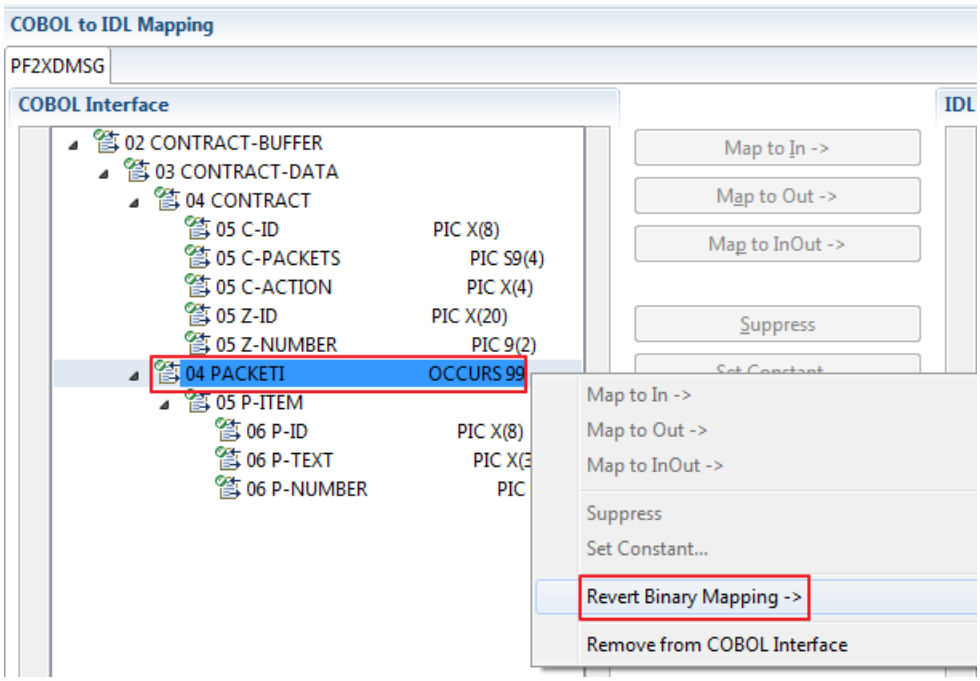


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

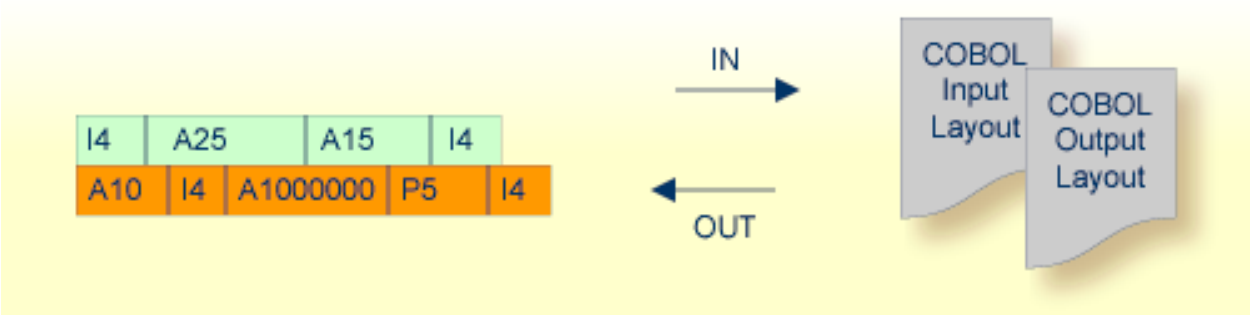
When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

12 COBOL Converter - In different to Out



Introduction

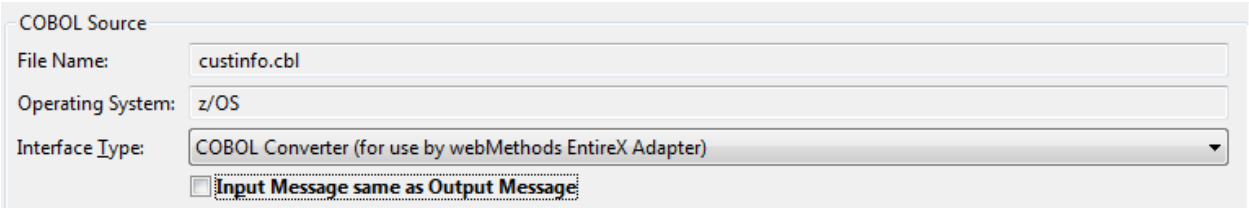
A file containing valid COBOL data items describing the COBOL payload can be used to extract a COBOL converter for the EntireX Adapter. If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

Extracting a COBOL Converter

This section assumes **Input Message same as Output Message** is not checked. COBOL output and COBOL input parameters are different, that is, the output is overlaid with a data structure that is different to the data structure on input.

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

Make sure the interface type is correct and check box **Input Message same as Output Message** is cleared.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items of the input message to **Input Message** by using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. See **Notes**.

2 Add the COBOL data items of the output message to **Output Message** by using the context menu and toolbars available in the *COBOL Interface* and *IDL Interface*. See **Notes**.

3 Continue with *COBOL to IDL Mapping*.

Notes:

1. It is very important to select the right COBOL data items describing the COBOL interface 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 interface contains `REDEFINE`s, 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.

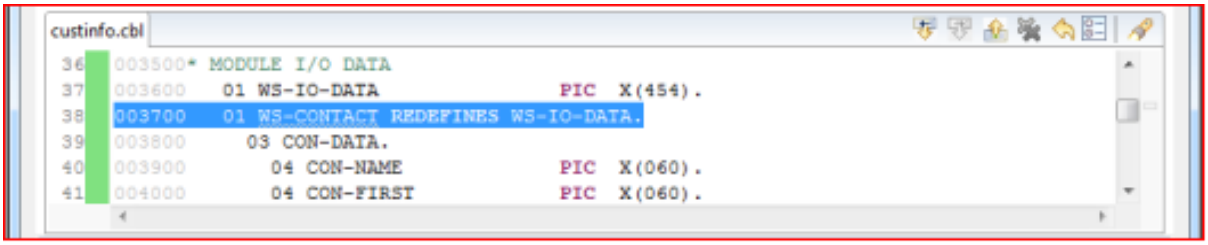
Mapping Editor User Interface

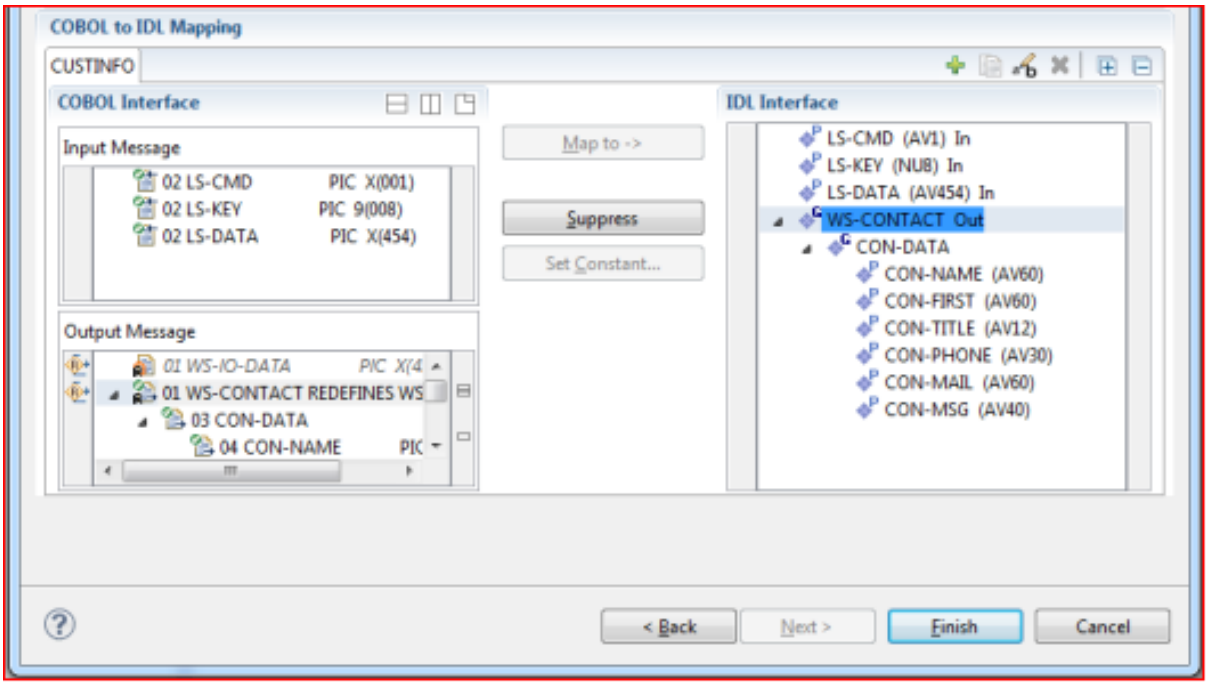
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL interface types where COBOL input and COBOL output parameters are different, the user interface of the COBOL Mapping Editor looks like this:

1. 

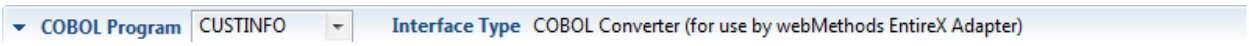
2. 

3. 

1. **COBOL Program Selection**. Currently selected program with interface type

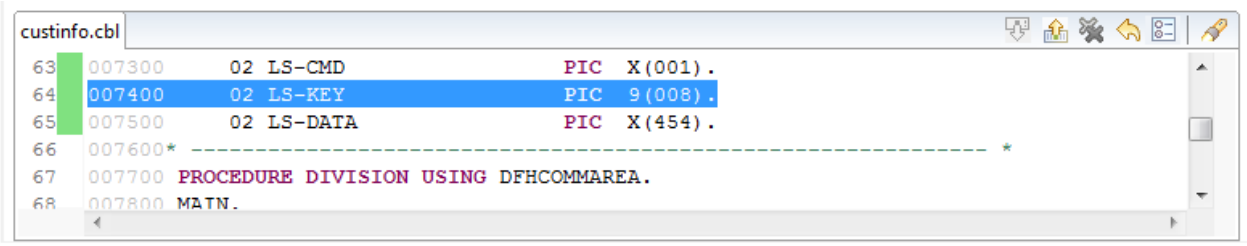
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

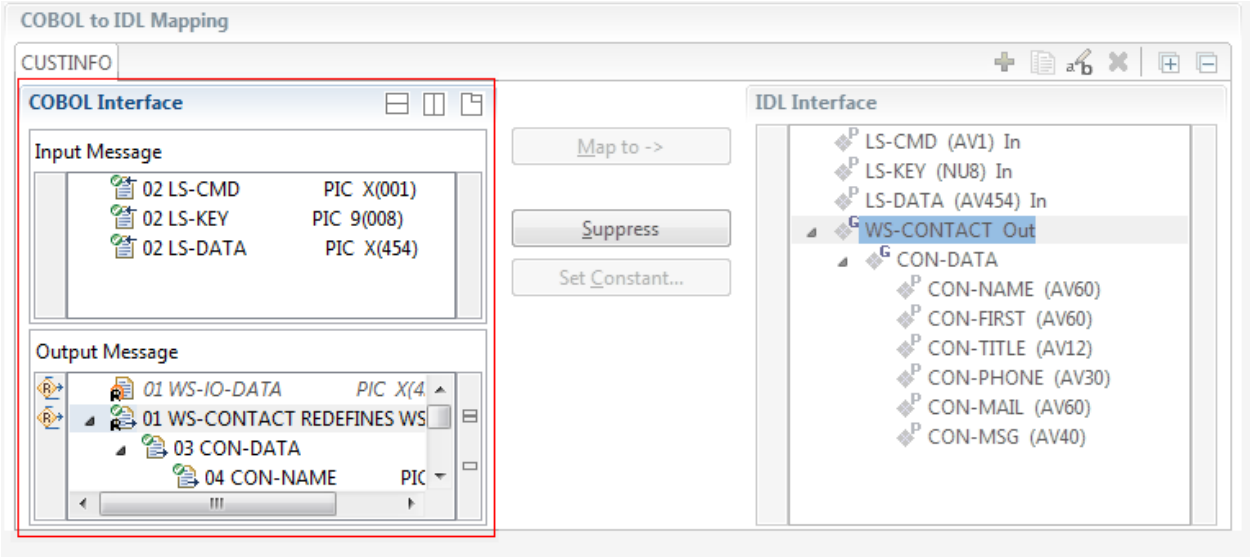
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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.

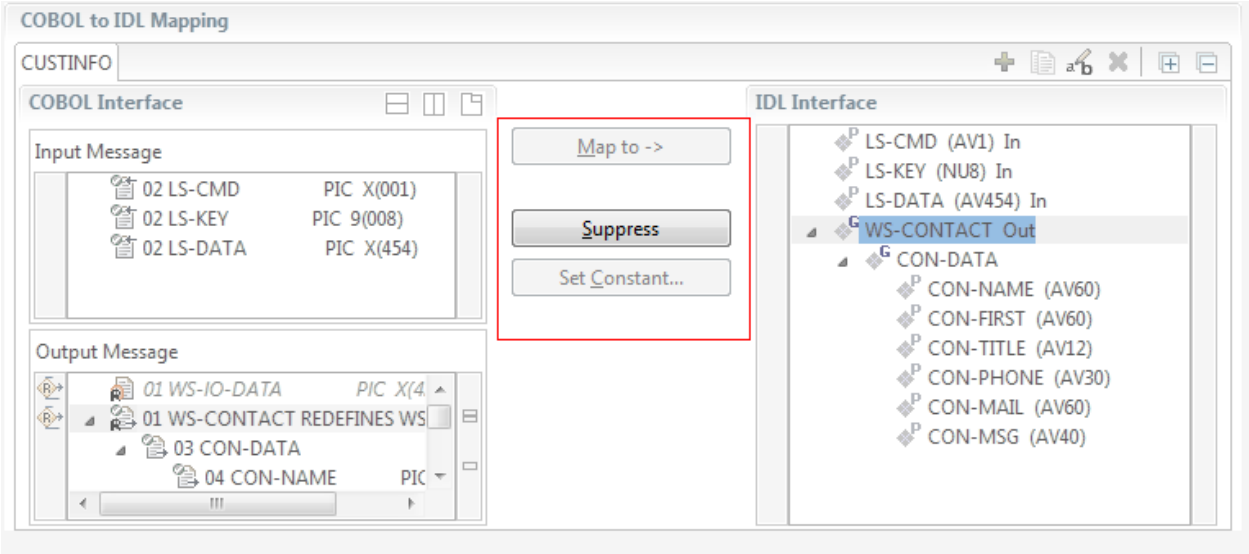
Group parameter, here mapped to In.

`REDEFINE` parameter, here mapped to Out.

Parameter set to Constant.

Mapping Buttons

The following buttons are available:



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*.

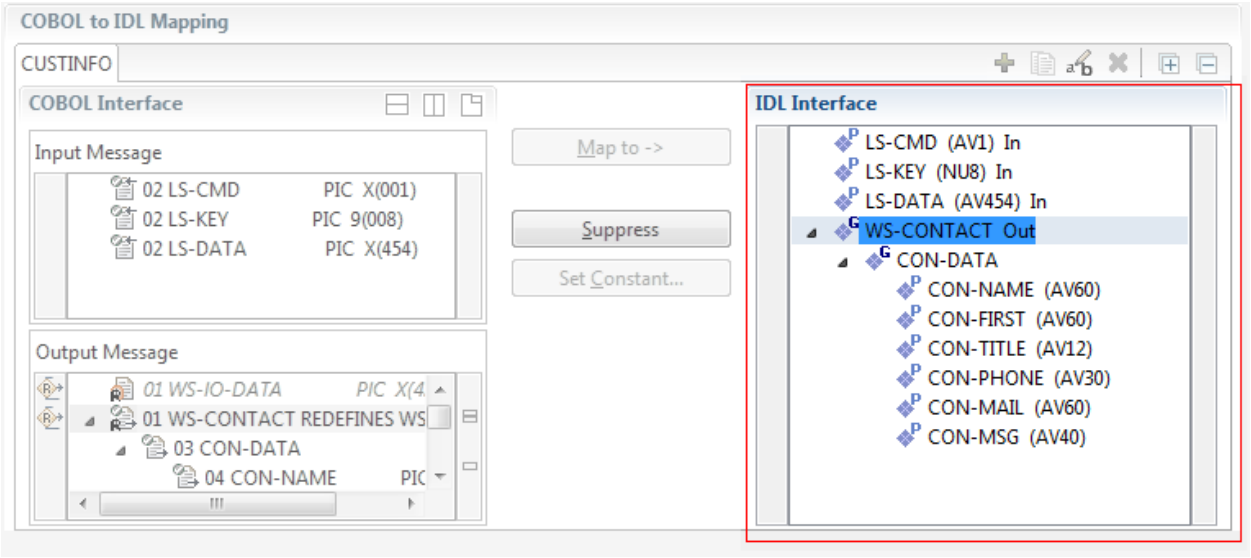
Set Constant...

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

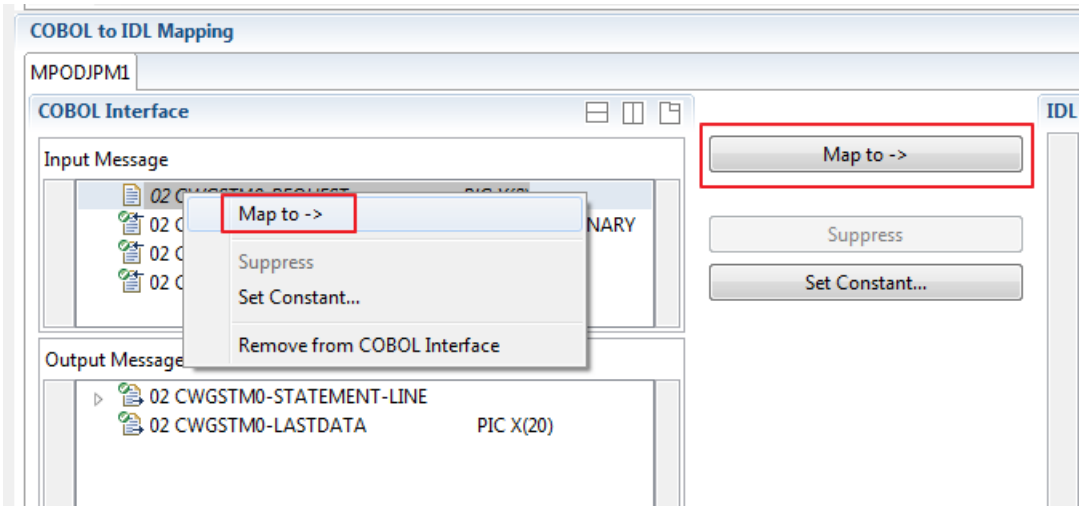
This section covers the following topics:

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 COBOL data items 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 of the COBOL interface and as mapping button to make a COBOL data item visible as an IDL parameter in the IDL interface:



2 Do the same for the output message of the COBOL interface.

Notes:

1. If a COBOL group is mapped, all subordinate 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 of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.

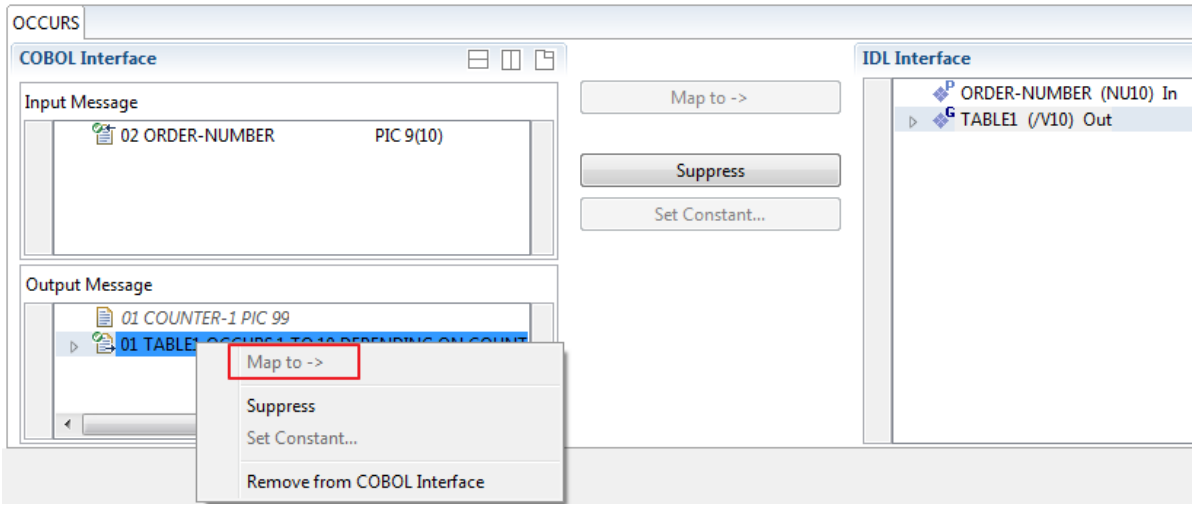
Map OCCURS DEPENDING ON

You can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

- Add the COBOL subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the input message or to the output message, wherever they belong. It is important both data items are always together per message direction (input or output).



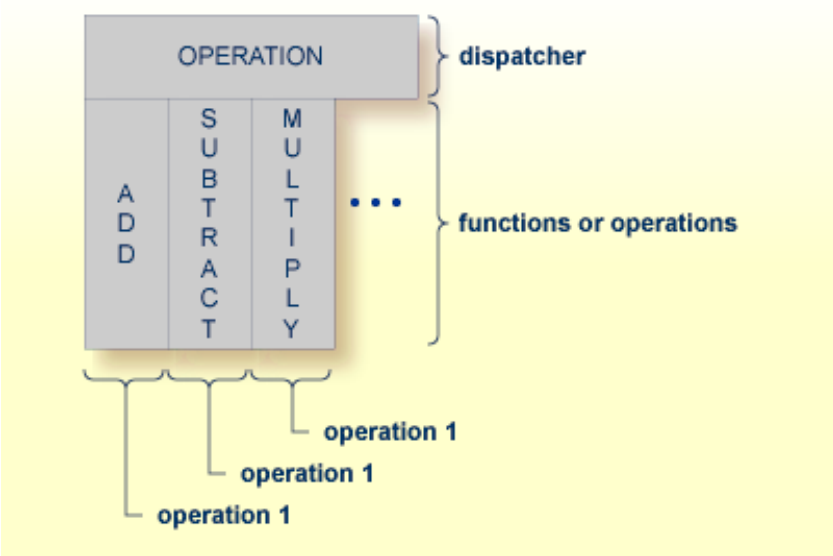
Notes:

1. The ODO subject can be mapped to the IDL interface.

2. The ODO object is always suppressed, but is required to be part of the same message direction (Input Message or Output Message) of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "*"
```

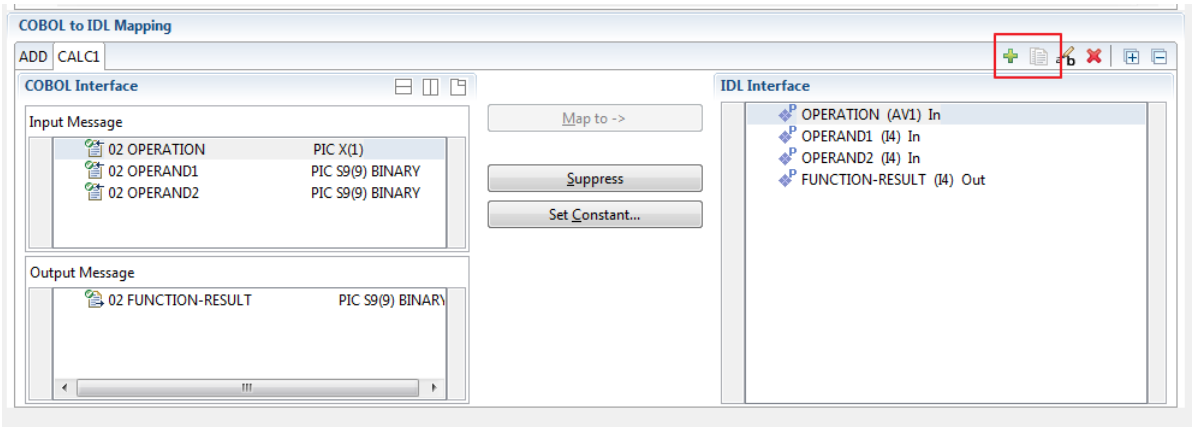
```
           MULTIPLY OPERAND1 BY OPERAND2
              GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

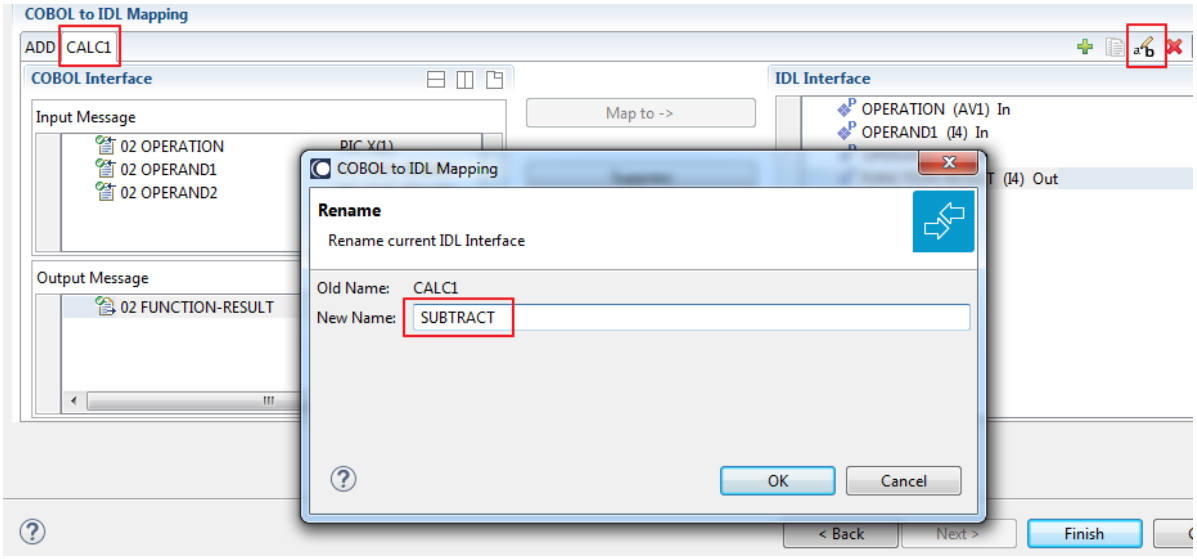
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

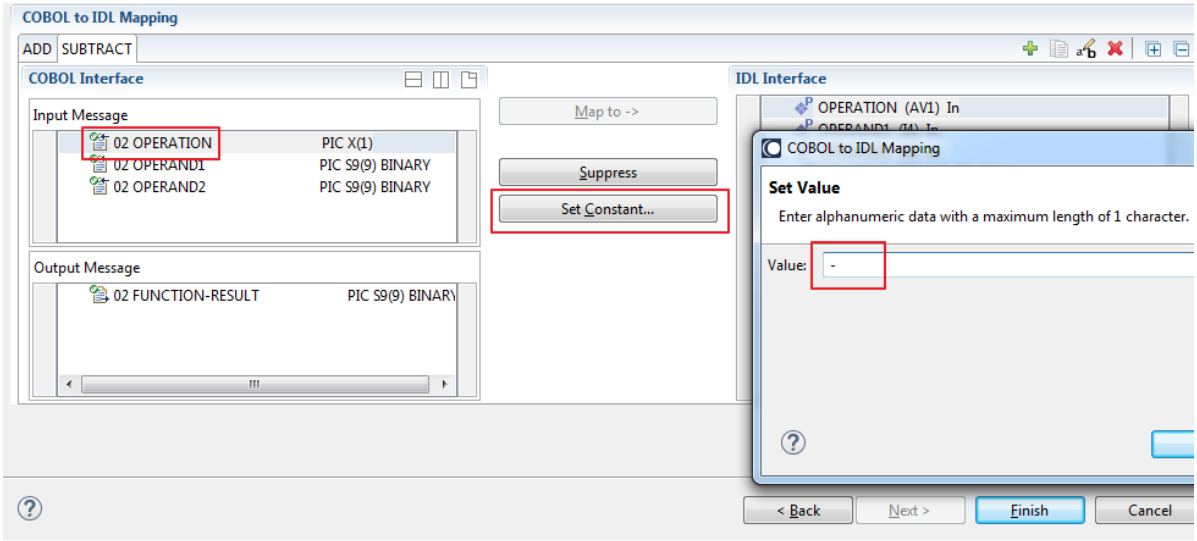
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.
- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

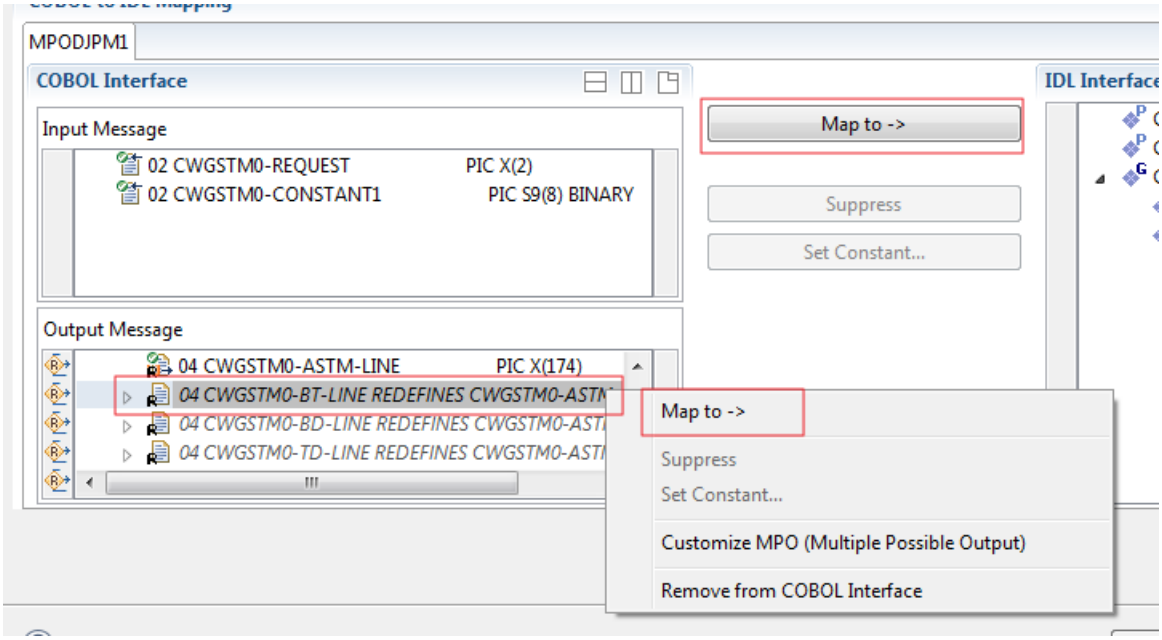
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

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 of the COBOL interface 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:

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

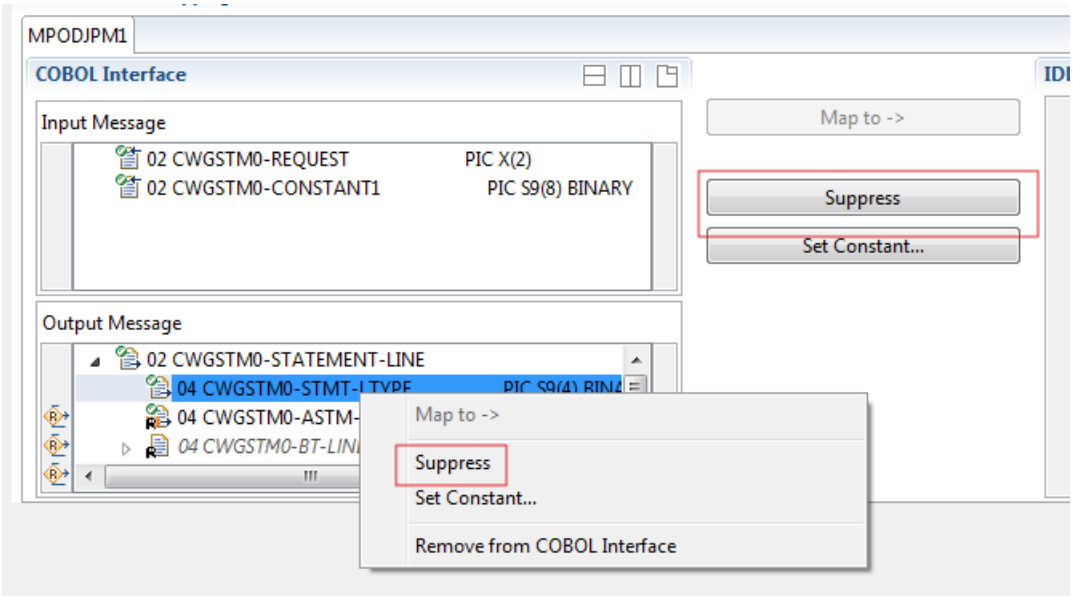
Suppress Unneeded COBOL Data Items

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 consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

To suppress unneeded COBOL data items

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



Notes:

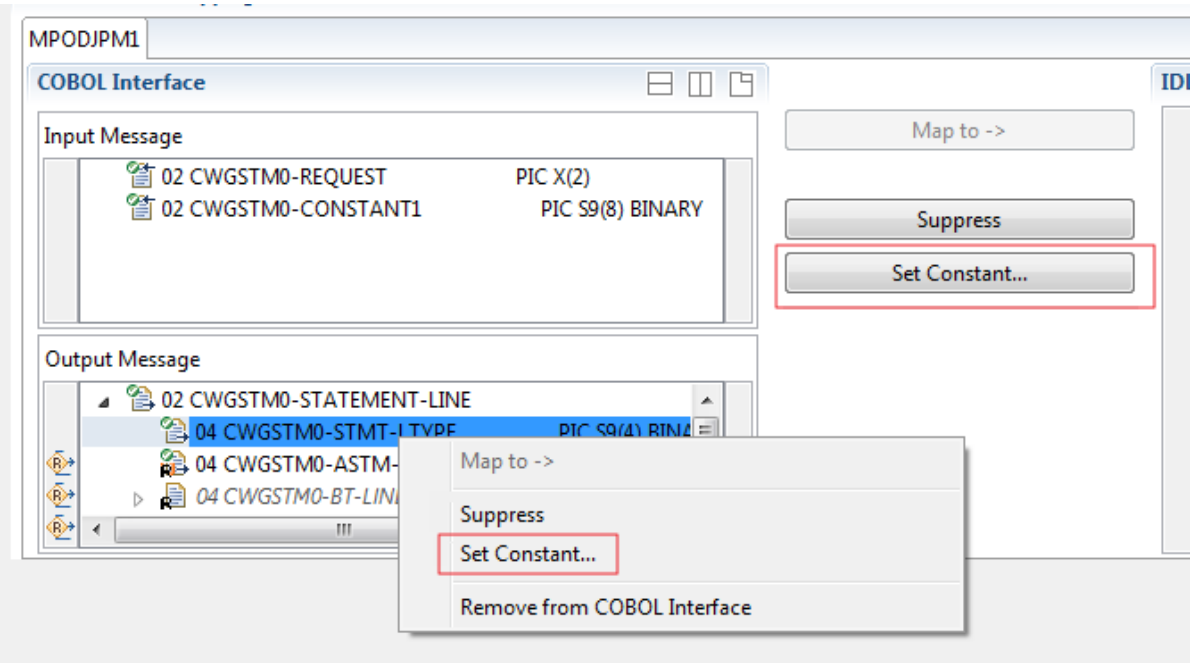
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed,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

Set COBOL Data Items to Constants

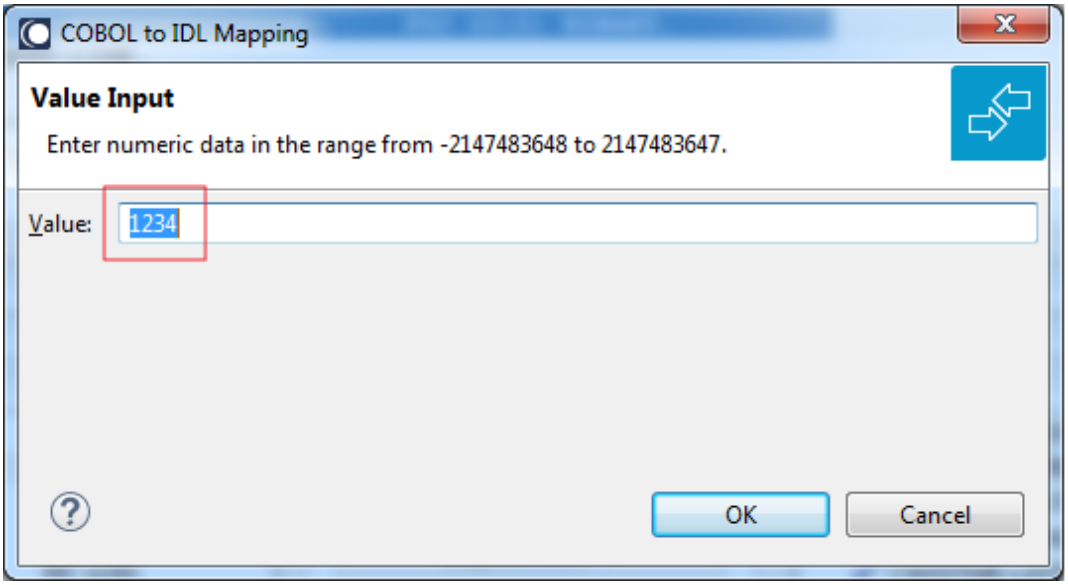
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter 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 of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

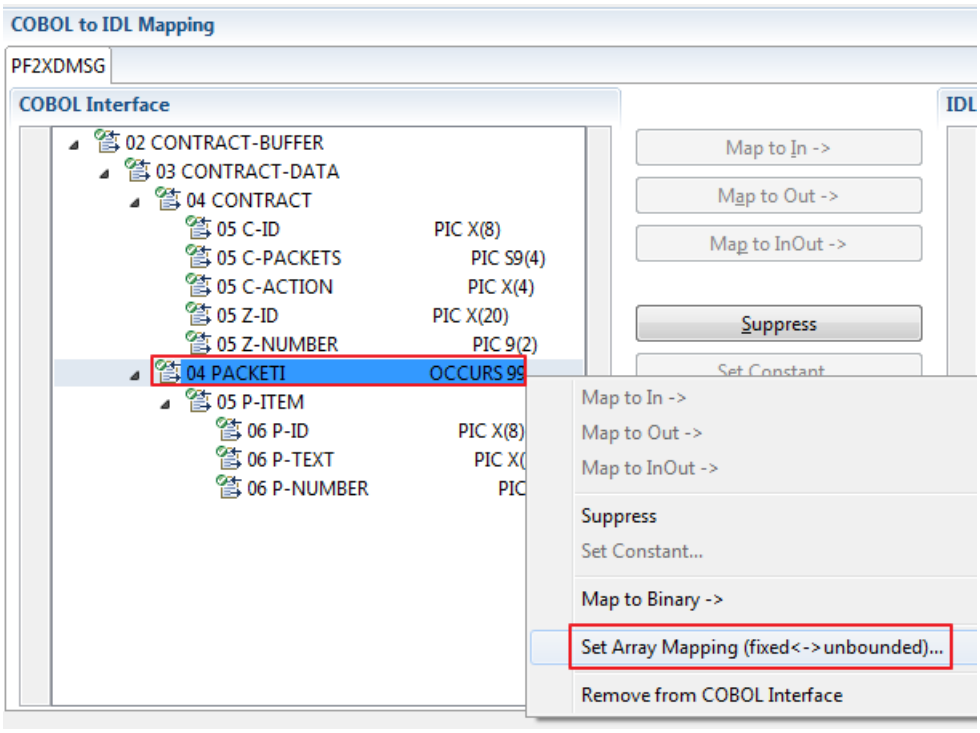
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

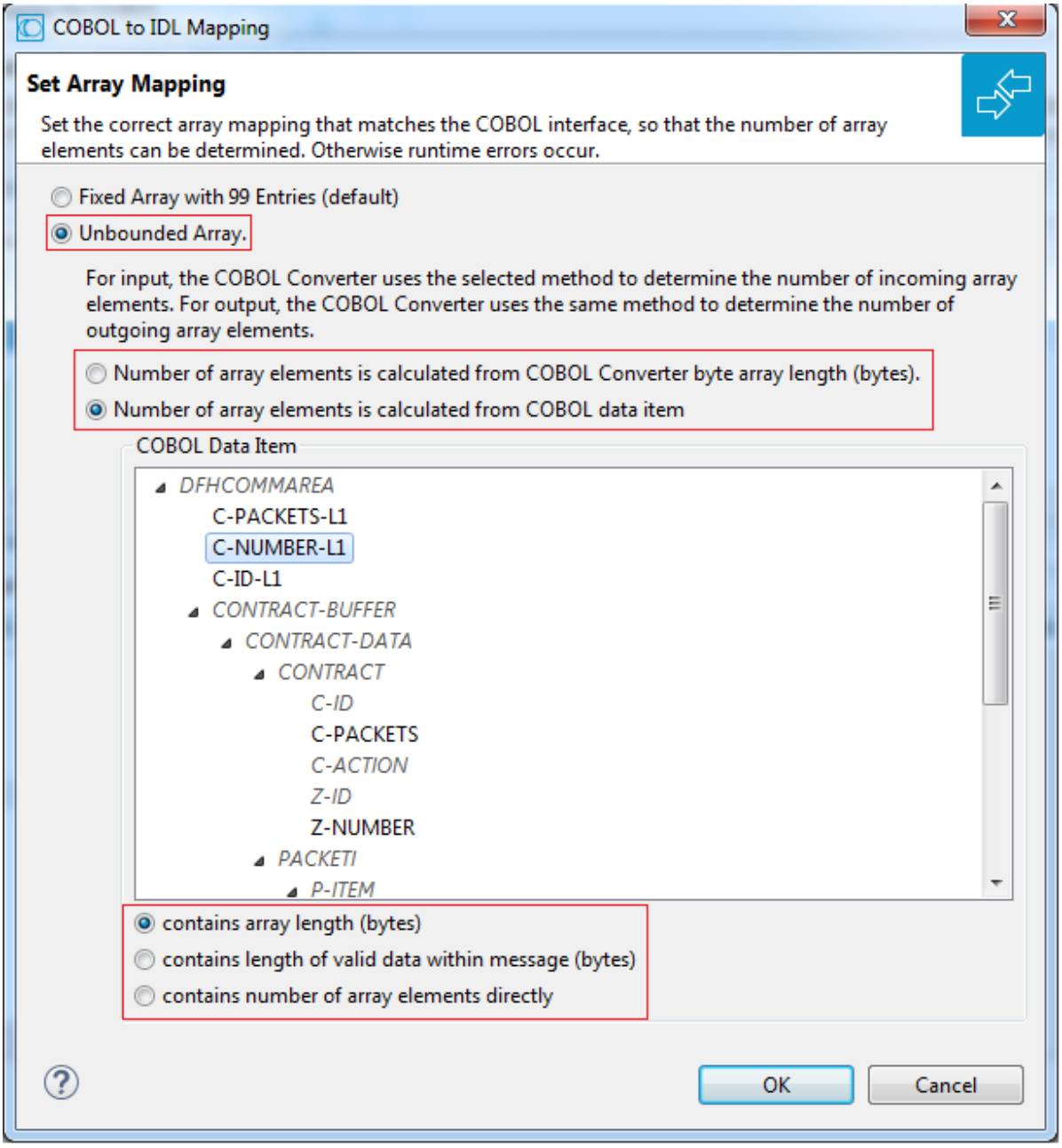
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **COBOL Converter length (bytes)**
  For input, the COBOL Converter uses the length of byte array `cobolInput` as COBOL Converter length. For output, the length of byte array `cobolOutput` has to be used accordingly. To determine the number of array elements, the length of the byte array is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` used in this manner:

```
          01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-BYTES                           PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
```

Assume the array `PACKETI` is filled with 15 occurrences. The length of byte array `cobolInput` is calculated then as follows: (`LENGTH OF ZONE`) + (`LENGTH OF CONTRACT`) + (`LENGTH OF P-ITEM`) * 15.
The number of array elements of the fixed-size array `PACKETI` is implicitly contained in the COBOL converter length.

- **COBOL data item contains array length (bytes)**
  For input, The COBOL Converter inspects the COBOL data item in byte array `cobolInput` and sets it accordingly for output in byte array `cobolOutput`. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` and `C-BYTES` as the COBOL data item used in this manner:

```
          01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-BYTES                           PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).
```

Assume the array `PACKETI` is filled with 7 occurrences. The contents of COBOL data item `C-BYTES` are calculated as follows: `(LENGTH OF P-ITEM) * 7`.
The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-BYTES`.

- **COBOL data item contains length of valid data within messages (bytes)**
  For input, The COBOL converter inspects the COBOL data item in byte array `cobolInput` and sets it accordingly for output in byte array `cobolOutput`. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than COBOL converter length. All lengths are in bytes. The following COBOL snippet shows the layout of a COBOL interface with fixed-size array `PACKETI` and `C-APPDATA` as the COBOL data item used in this manner:

```
        01 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                                PIC X(8).
            05 C-APPDATA                           PIC S9(4).
            05 C-ACTION                            PIC X(4).
            05 Z-ID                                PIC X(20).
            05 Z-NUMBER                            PIC 9(2).
           04 PACKETI                              OCCURS 99.
            05 P-ITEM.
             06 P-ID                               PIC X(8).
             06 P-TEXT                             PIC X(30).
             06 P-NUMBER                           PIC 9(2).
```

  Assume the array `PACKETI` is filled with 31 occurrences. The contents of COBOL data item `C-APPDATA` are calculated as follows: `(LENGTH OF CONTRACT) + (LENGTH OF P-ITEM) * 31`. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

- **COBOL data item contains number of array elements directly**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                   PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-NUM                             PIC S9(4).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-NUMBER                          PIC 9(2).
            05 Z-ID                              PIC X(20).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
```

```
           06 P-ID                                 PIC X(8).
           06 P-TEXT                               PIC X(30).
           06 P-NUMBER                             PIC 9(2).
     . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set occurrences
        MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

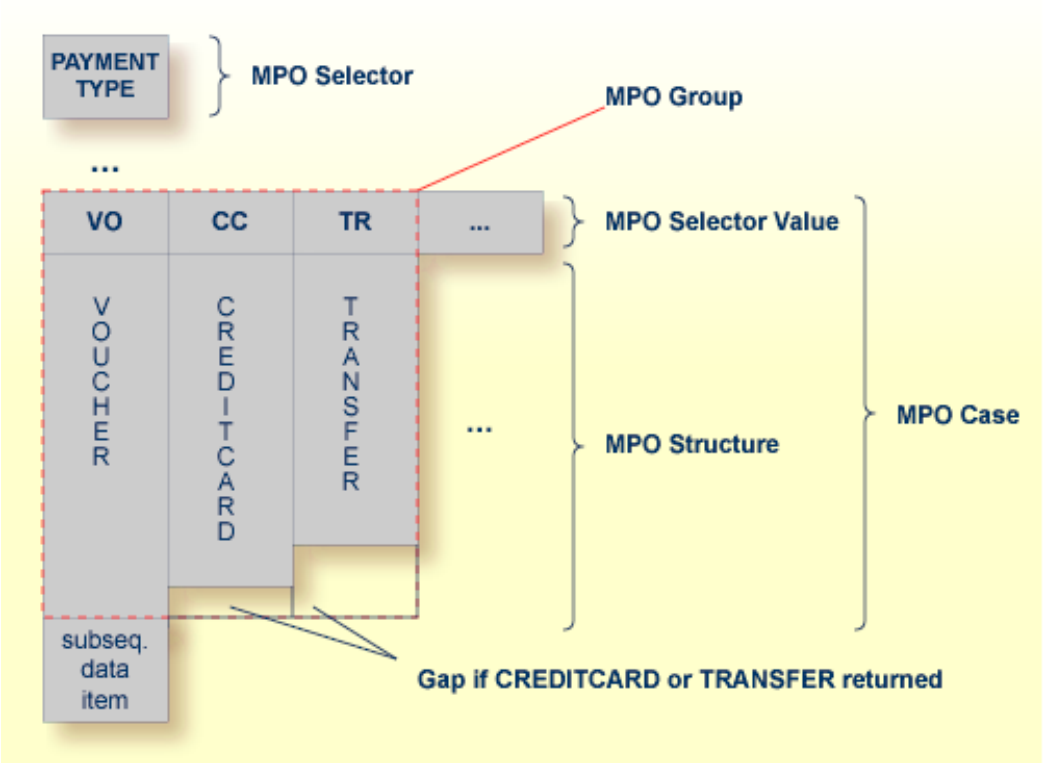
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>                PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>               PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

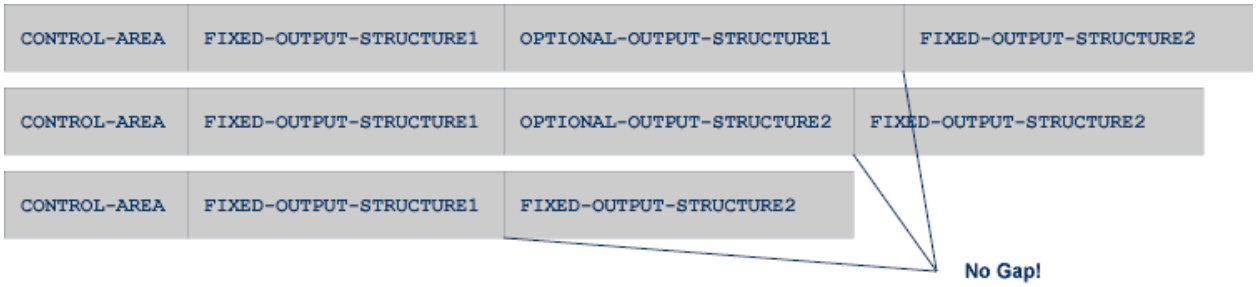
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

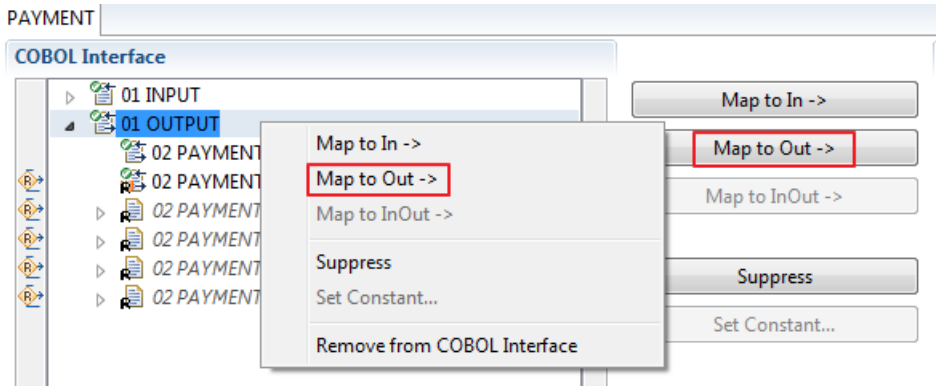
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

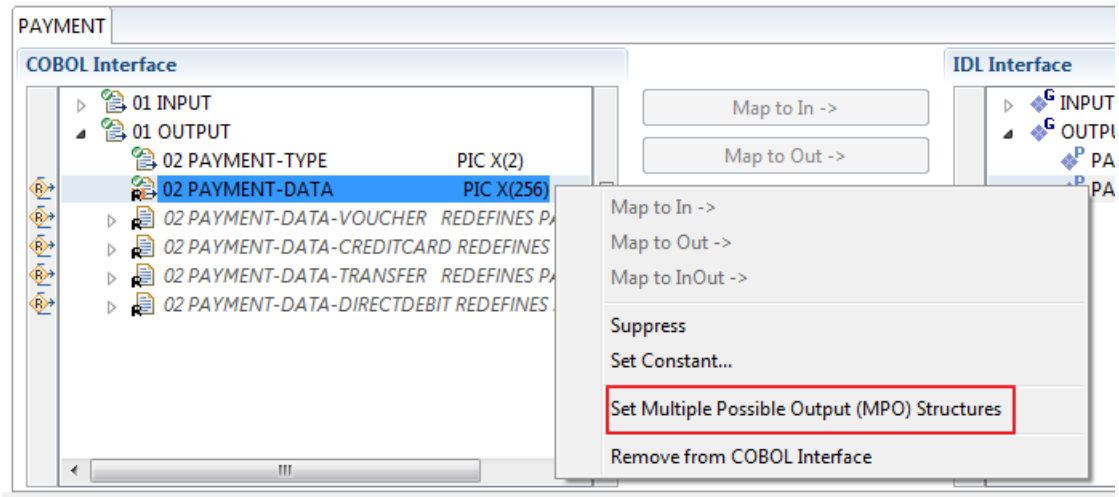
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

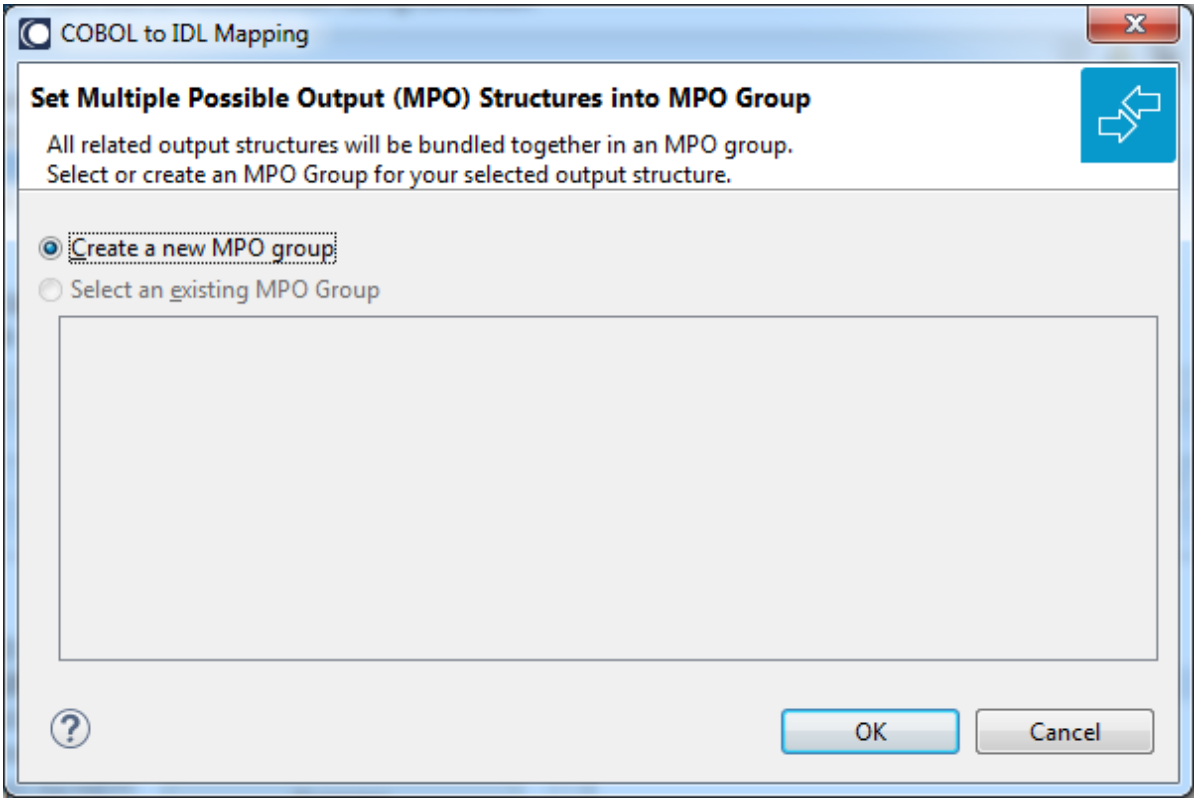
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



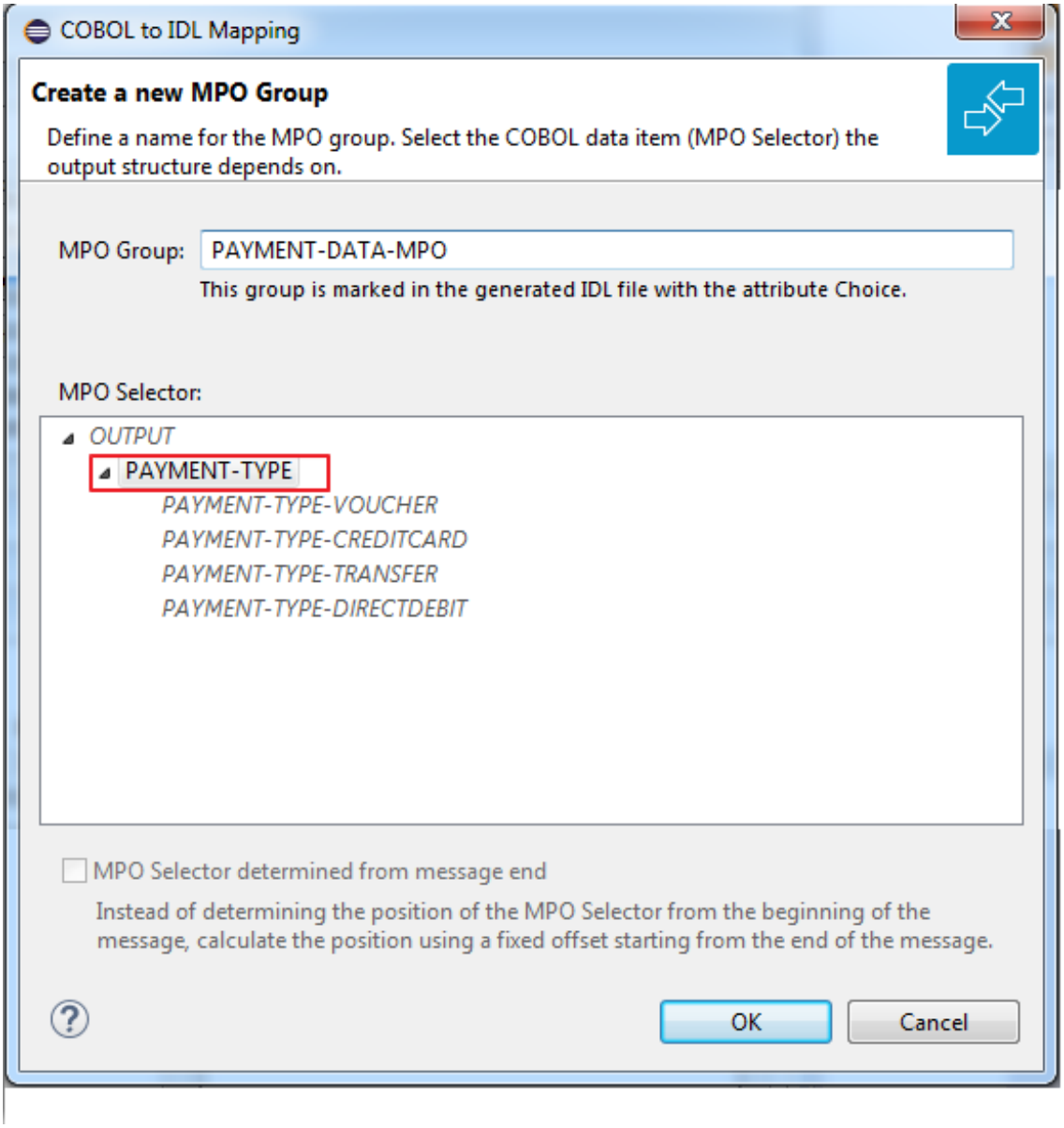
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



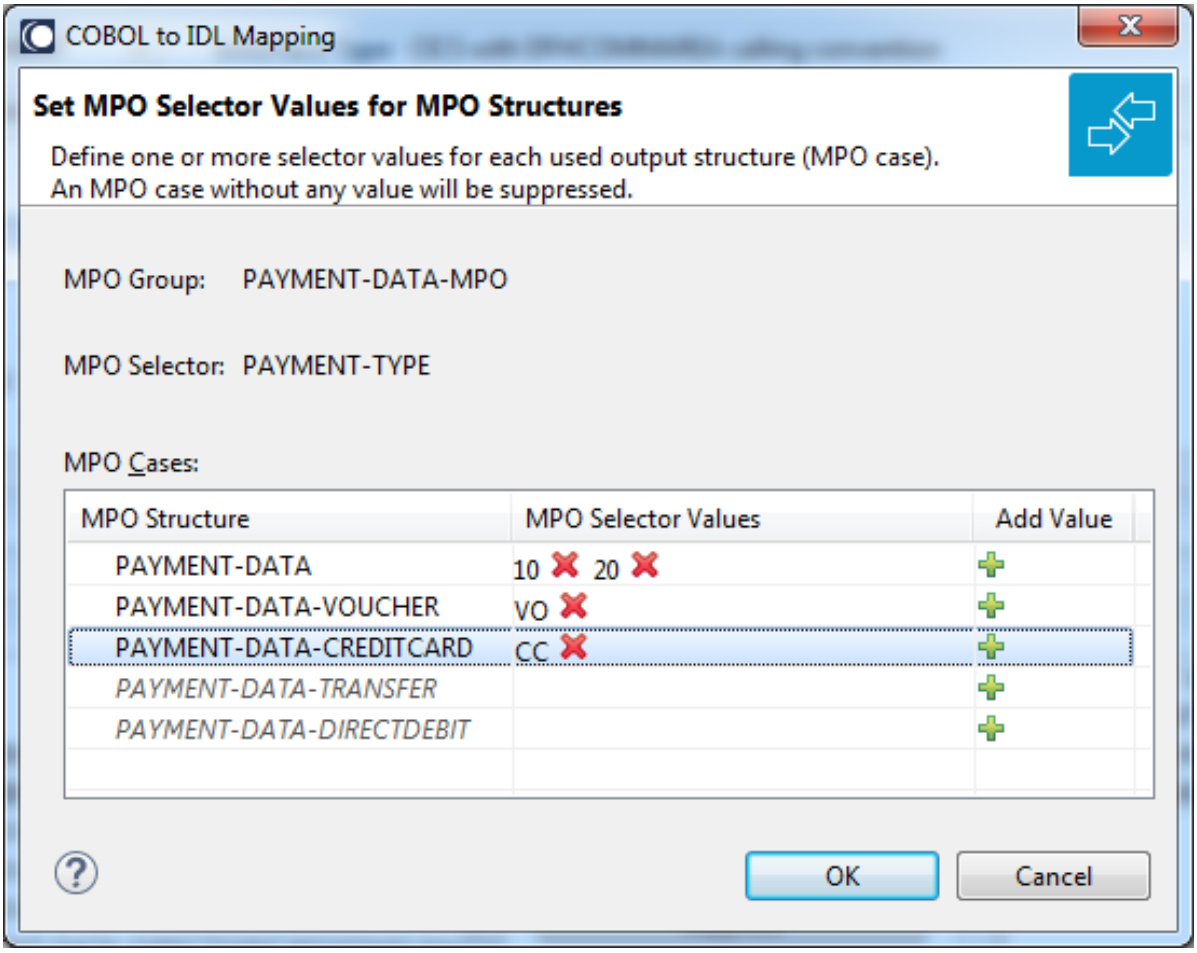
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



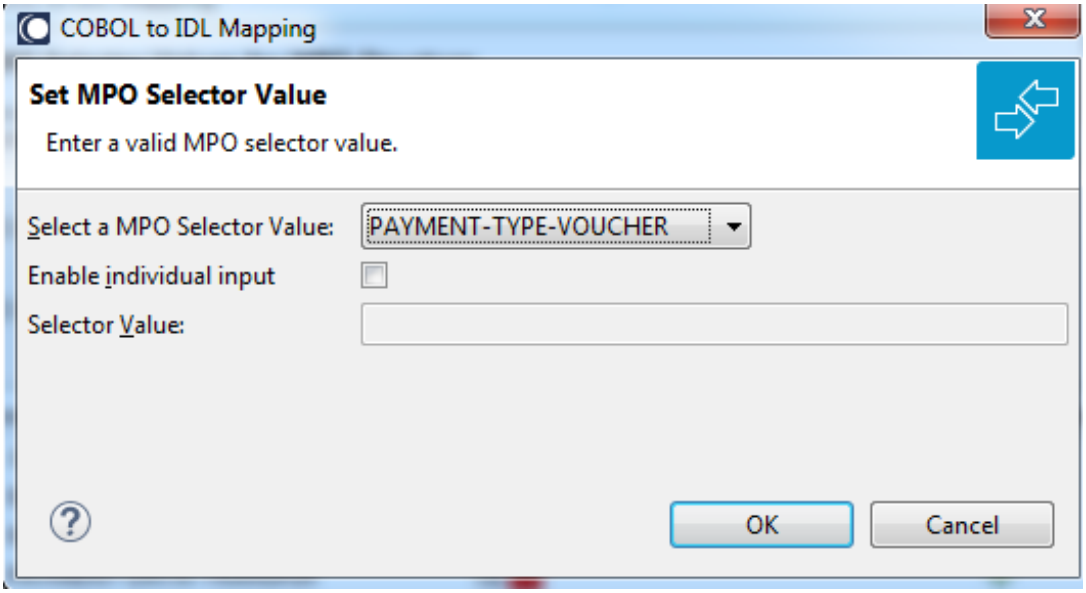
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

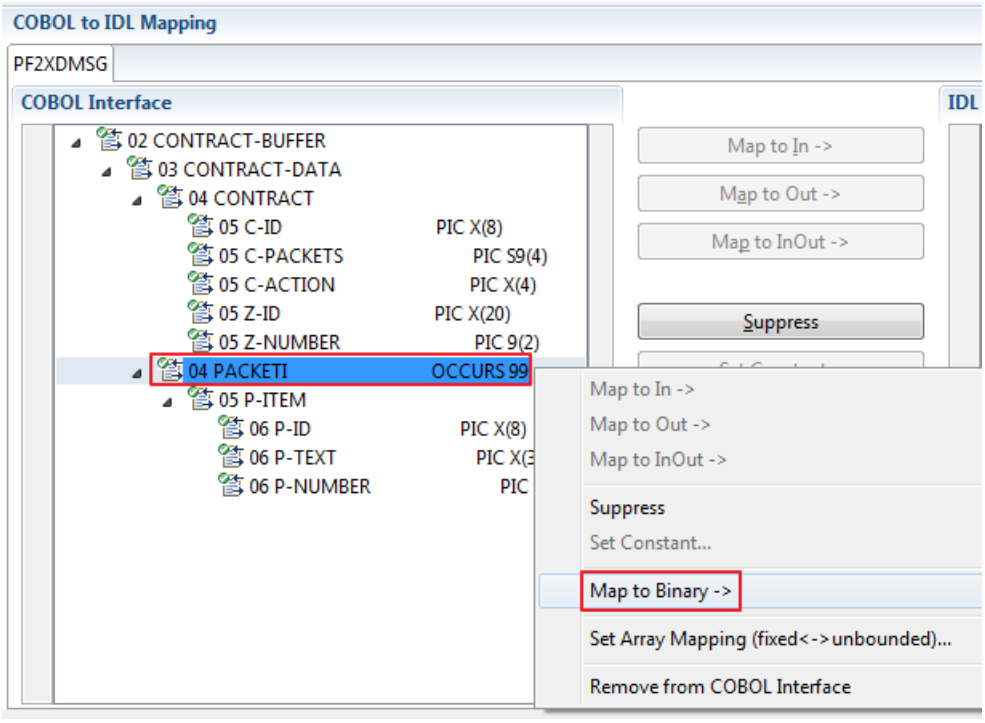
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

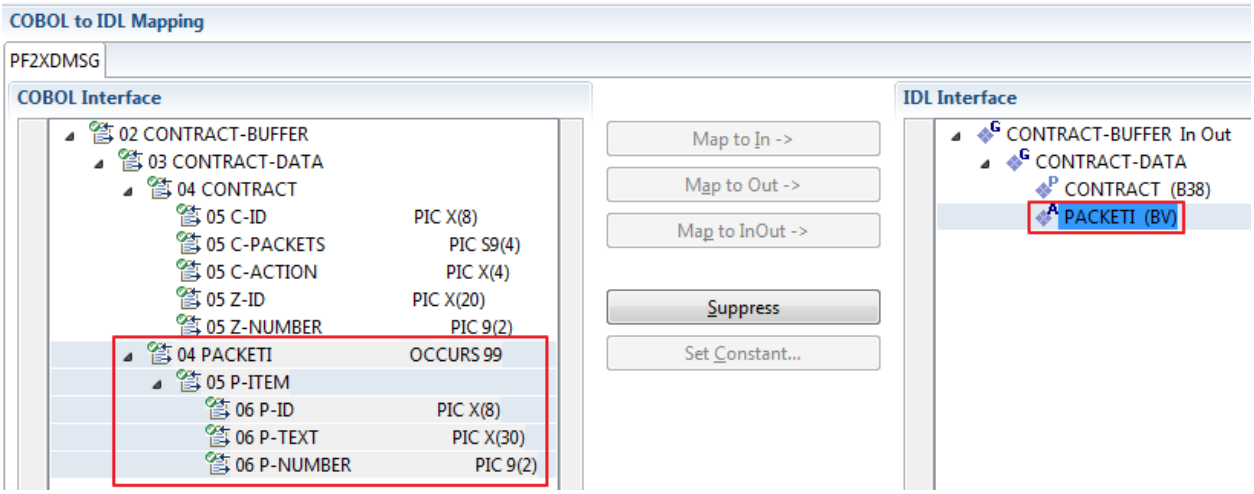
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.



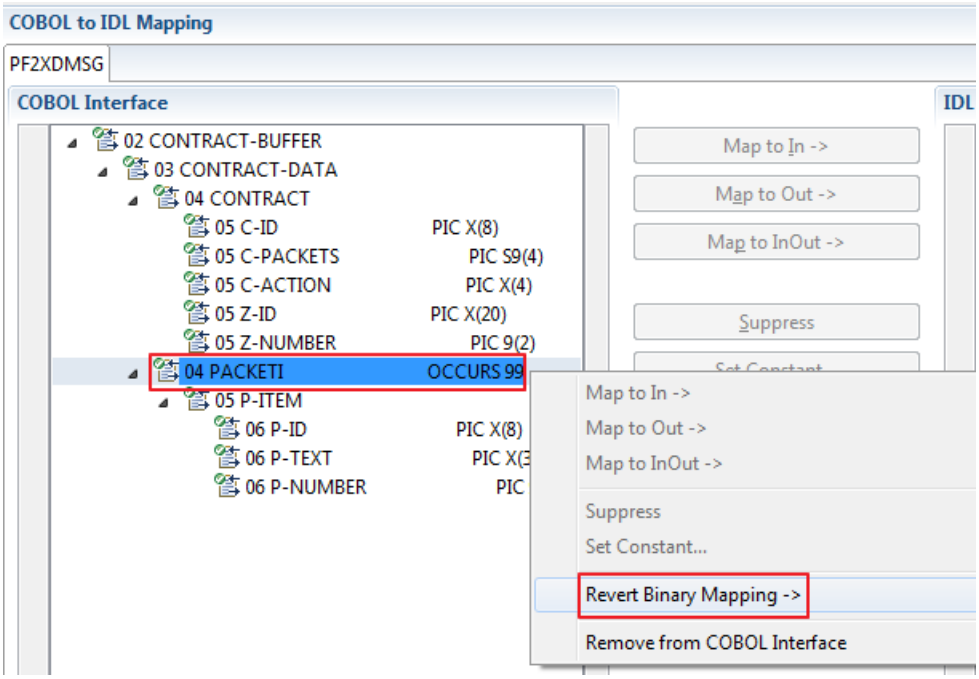
The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.

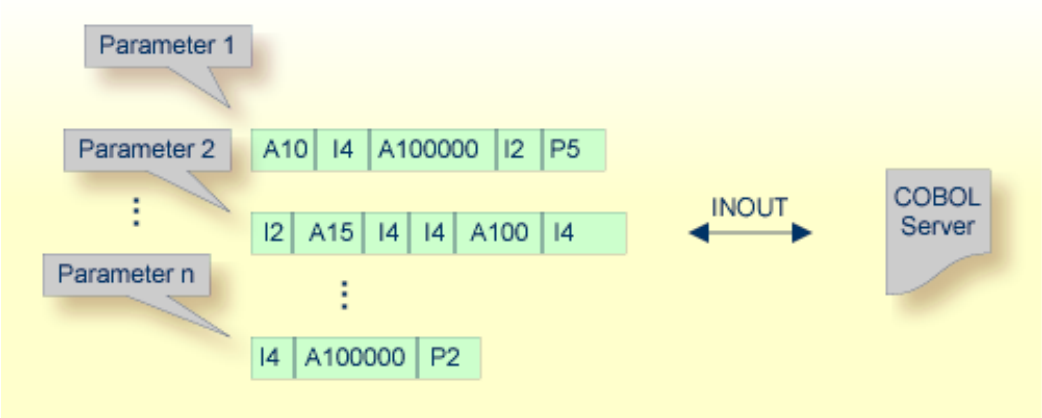


Note: The last COBOL data items are mapped to IDL data type BV instead of B*n* (`PACKETI (BV)` in this example).

To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.



13 Batch with Standard Linkage Calling Convention



Introduction

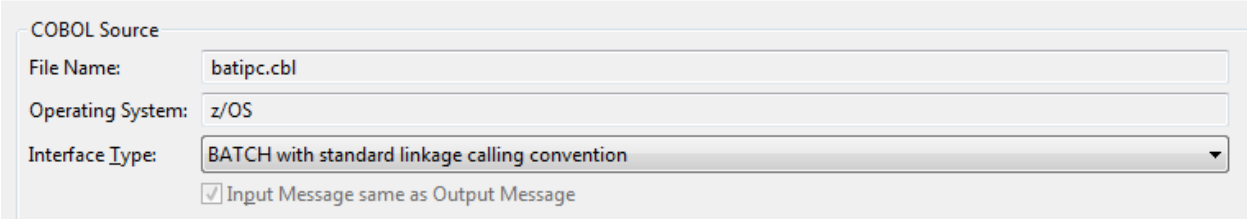
Because COBOL servers with a standard call interface always contain a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters, the COBOL data items of the interface can be evaluated by the IDL Extractor for COBOL and are already offered by the wizard. In most cases the offered COBOL data items will be correct, but you should always check them manually.

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 a Standard Call Interface

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

Make sure the interface type is correct.



Press **Next** to open the COBOL Mapping Editor.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items to the **COBOL Interface**, using the context menu or toolbar available in the *COBOL Source View* and *COBOL Interface*. See **Notes**.

2 Continue with *COBOL to IDL Mapping*.

Notes:

1. If there is a `PROCEDURE DIVISION` header available, the parameters listed define exactly the COBOL interface. These COBOL data items are within the `LINKAGE SECTION` and are already selected to the COBOL interface in initial state when you enter the COBOL Mapping Editor. The `PROCEDURE DIVISION` header might not be available if you are extracting from a copybook or part of the COBOL source.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL interface contains `REDEFINE`s, 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.

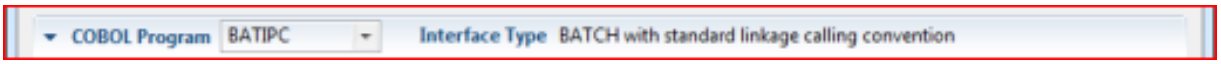
The user interface of the COBOL Mapping Editor is described below.

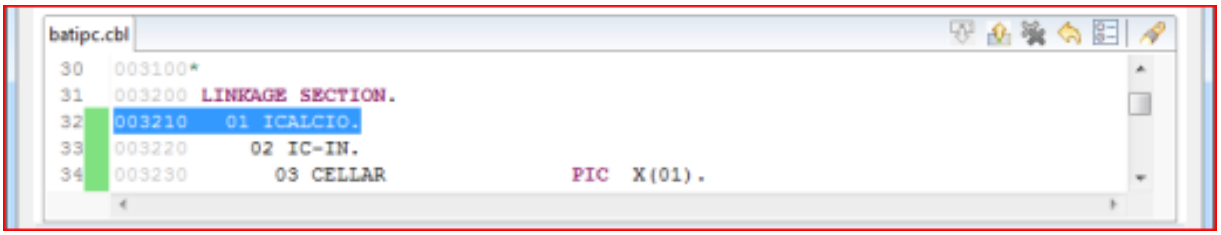
Mapping Editor User Interface

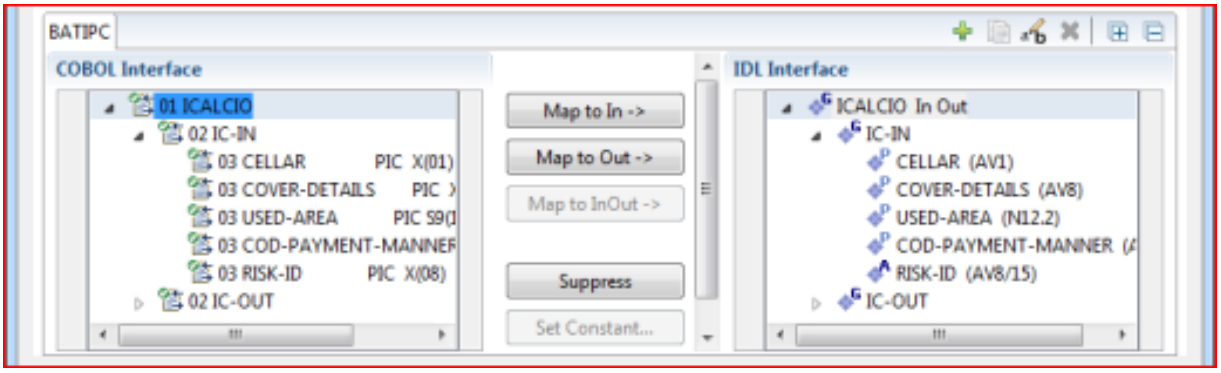
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL server programs with standard call interface types, the user interface of the COBOL Mapping Editor looks like this:

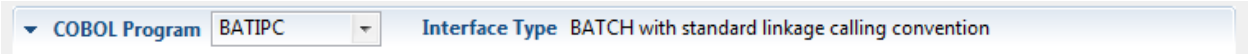
1. 

2. 

3. 

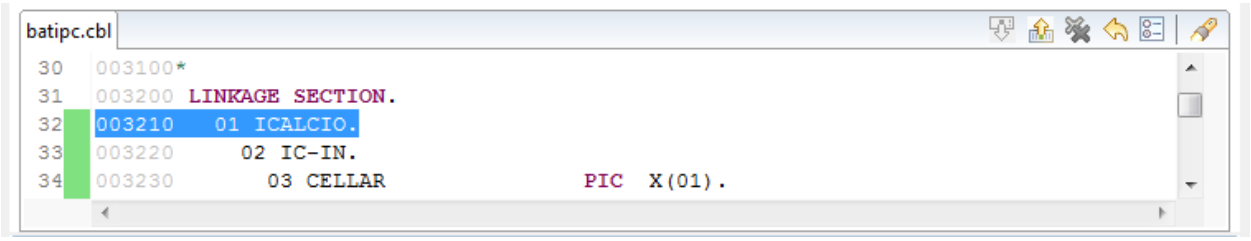
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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.
- 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

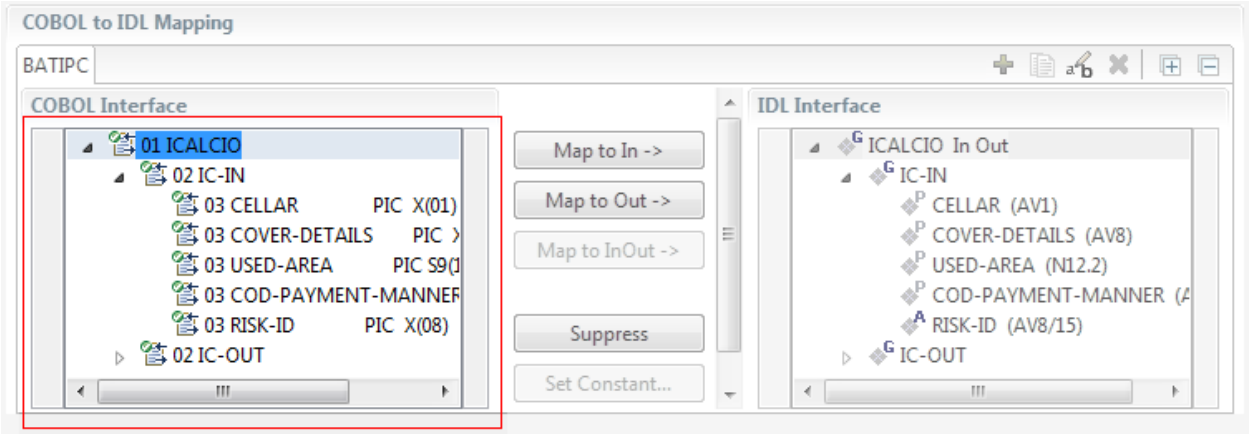
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

Map to In | Out | InOut	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 Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

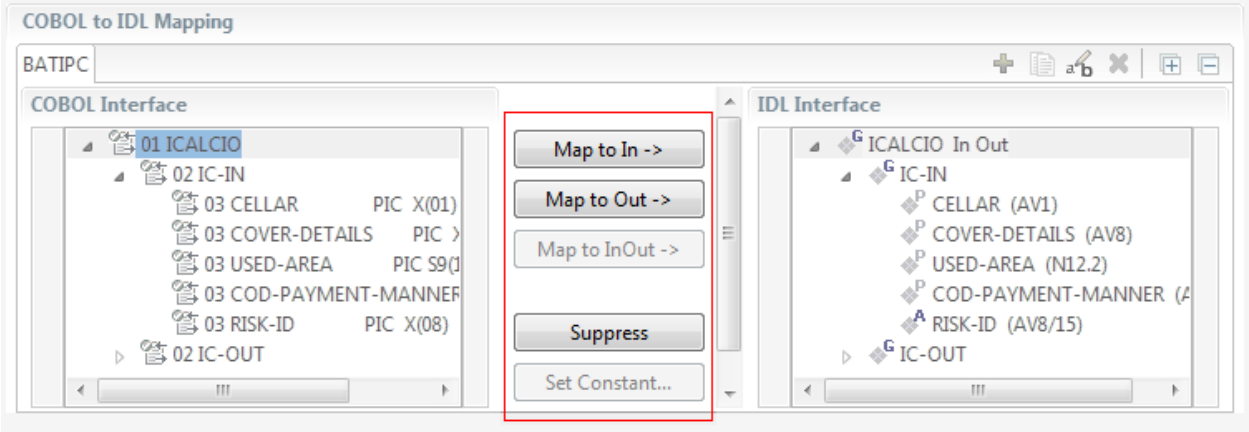
Mapping Icons

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

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

Mapping Buttons

The following buttons are available:



Map to In | Out | InOut ->
See *Map to In, Out, InOut*. A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.

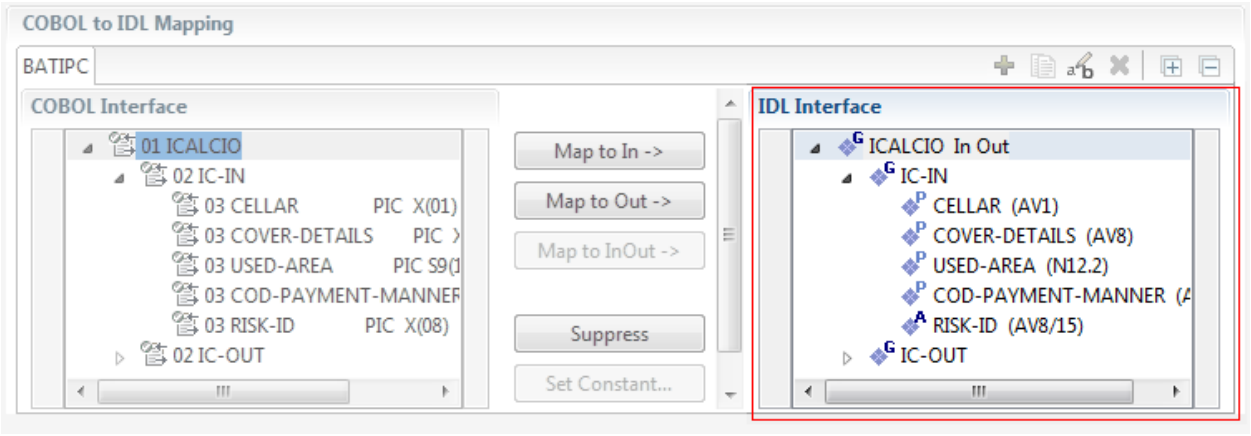
Suppress
See *Suppress Unneeded COBOL Data Items*.

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

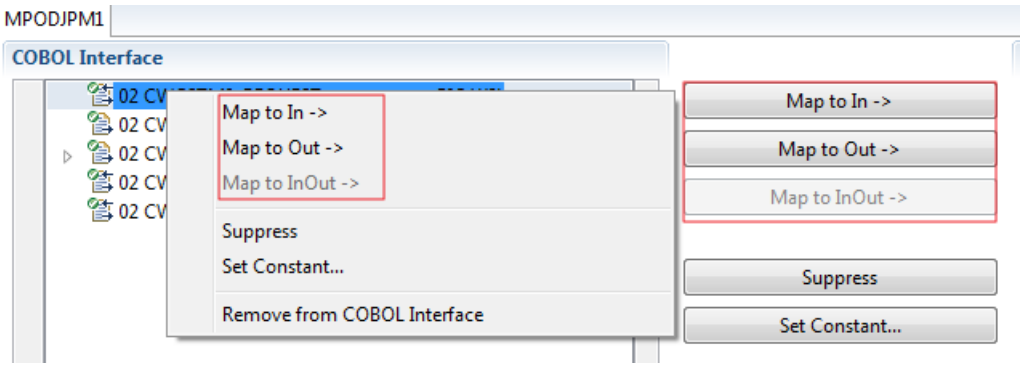
This section covers the following topics:

Map to In, Out, InOut

With the **Map to In**, **Out**, **InOut** functions you make a COBOL data item visible as an IDL parameter in the IDL interface. With correct IDL directions you design the IDL interface by defining input and output parameters. COBOL programs have no parameter directions, so you need to set IDL directions manually.

To provide IDL directions

- Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons to make the COBOL data items visible and provide IDL directions in the IDL interface:



Notes:

1. If a *top-level* COBOL *group* is mapped, the IDL direction is inherited by all subordinate COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subordinate COBOL data items can only be mapped to the same IDL direction as their *top-level* COBOL *group* data item.
3. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.
4. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Map OCCURS DEPENDING ON

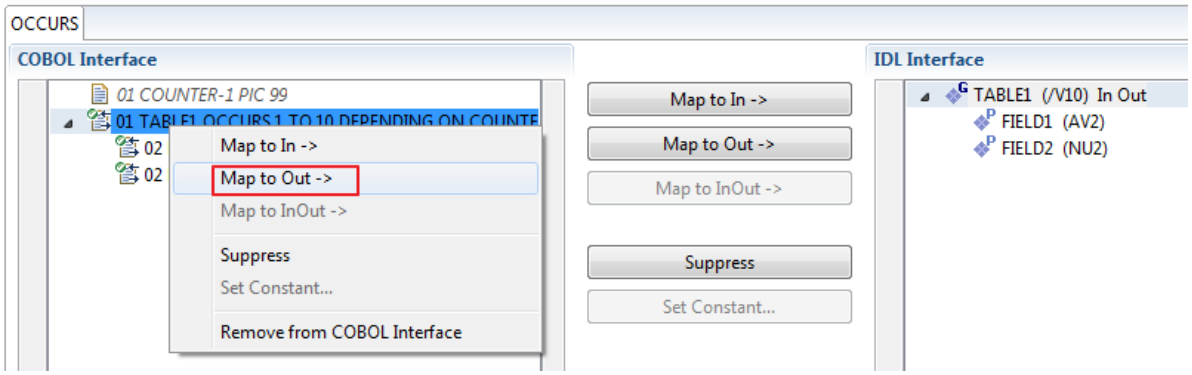
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):

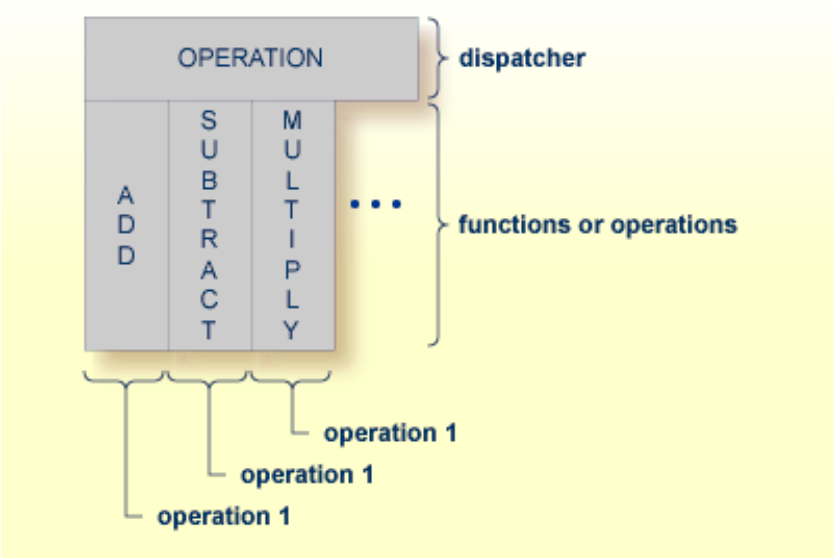


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "-"
```

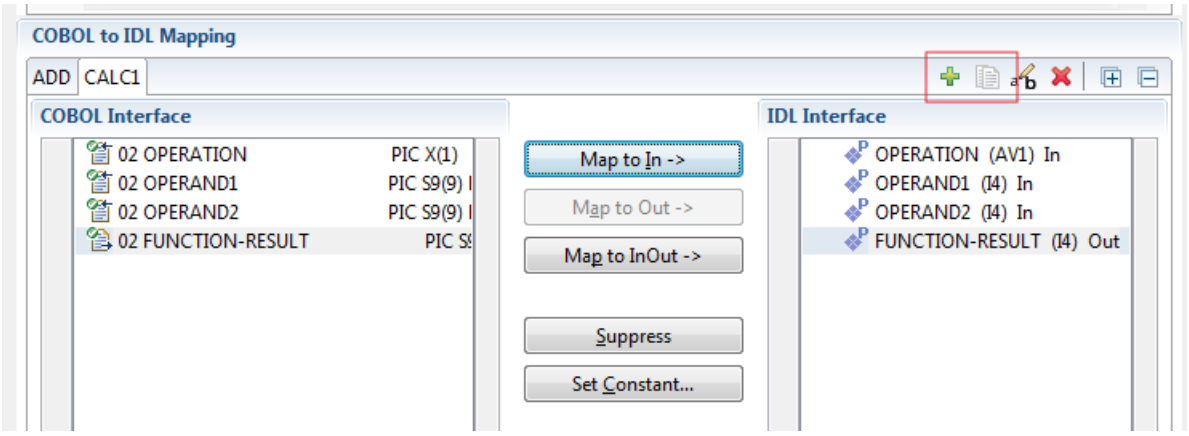
```
            SUBTRACT OPERAND2 FROM OPERAND1
            GIVING FUNCTION-RESULT
         WHEN "*"
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

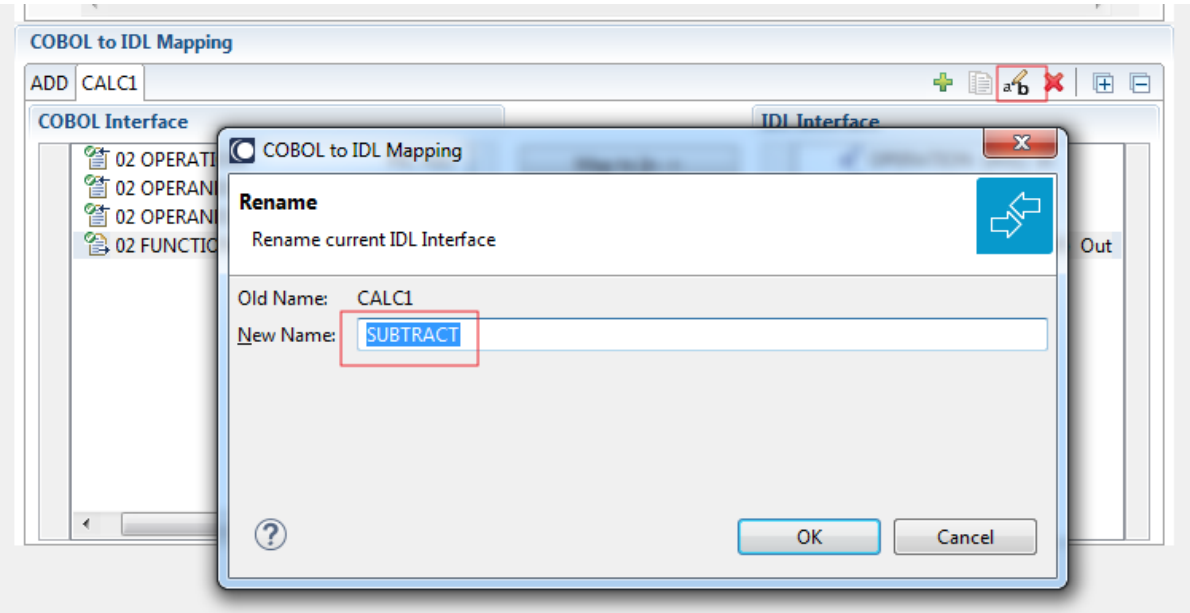
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

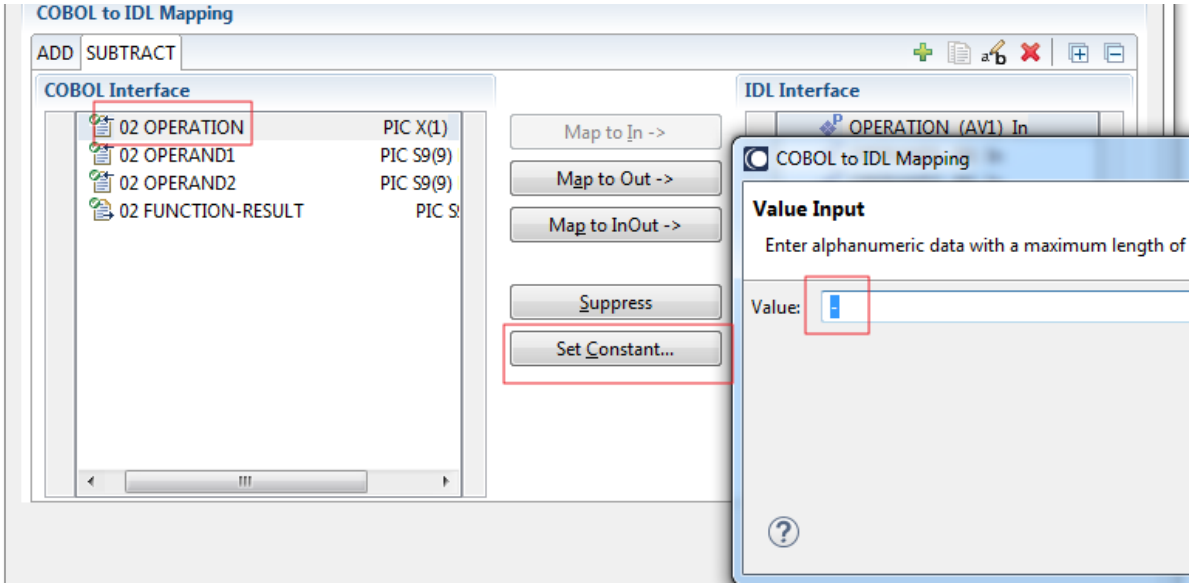
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

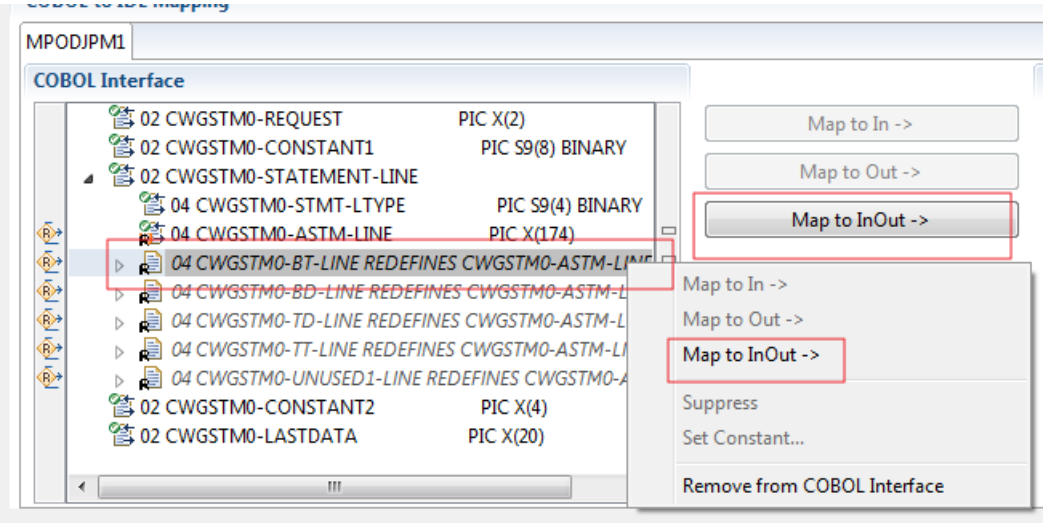
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

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

To select redefine paths

- Use the **Map to In**, **Out** or **InOut** functions available in the context menu of the COBOL interface and as mapping buttons 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:

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

Suppress Unneeded COBOL Data Items

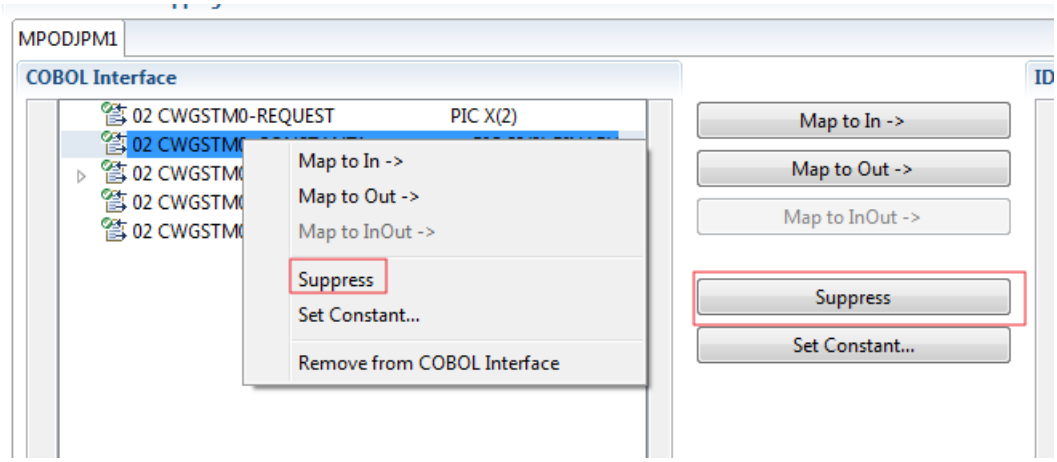
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface and as mapping button to make the COBOL data item invisible in the IDL interface:



Notes:

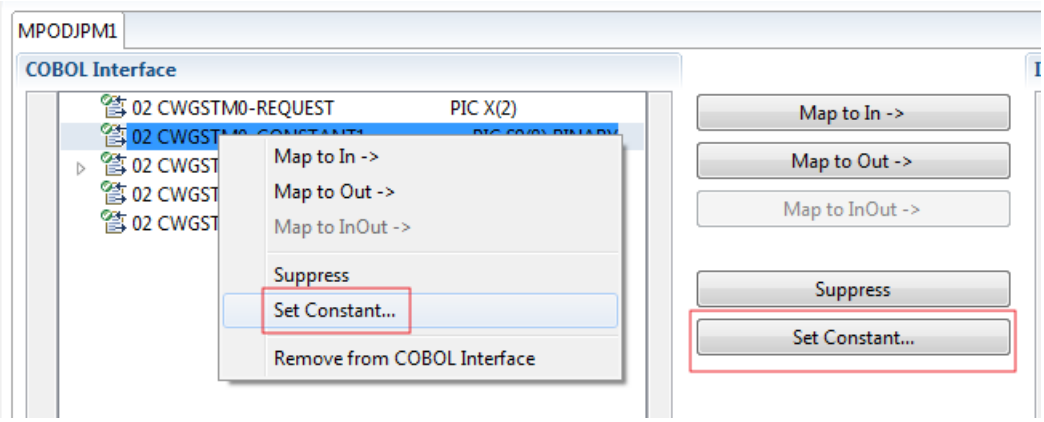
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subordinate COBOL data items are suppressed as well.
4. With the inverse functions **Map to In**, **Out** or **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item is made visible in the IDL interface again.

Set COBOL Data Items to Constants

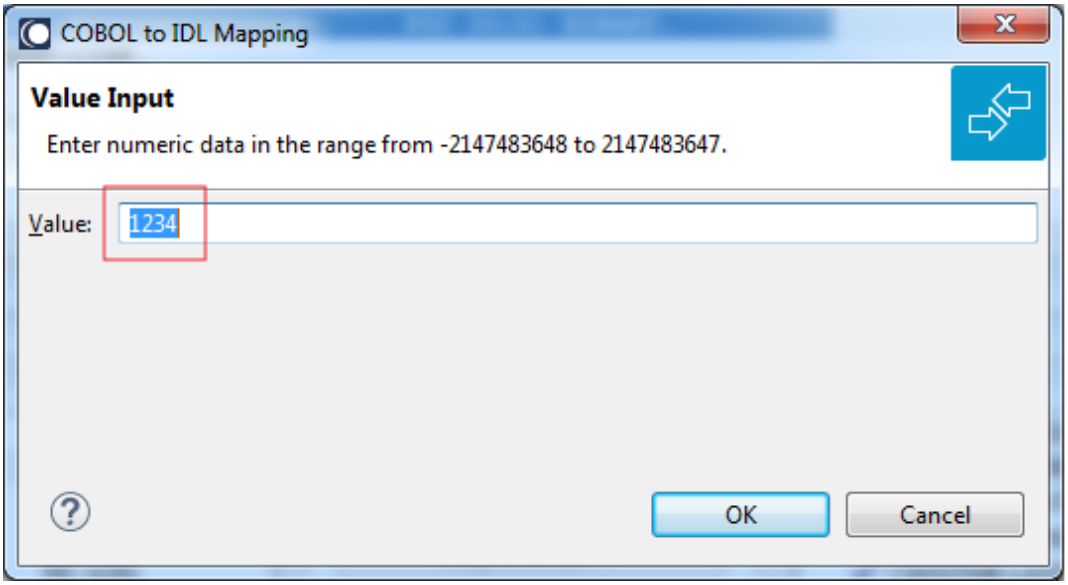
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter provides the defined constant in the COBOL data item to your COBOL server.
3. With the functions **Map to In**, **Out**, **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item can be made visible in the IDL interface again.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology
- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

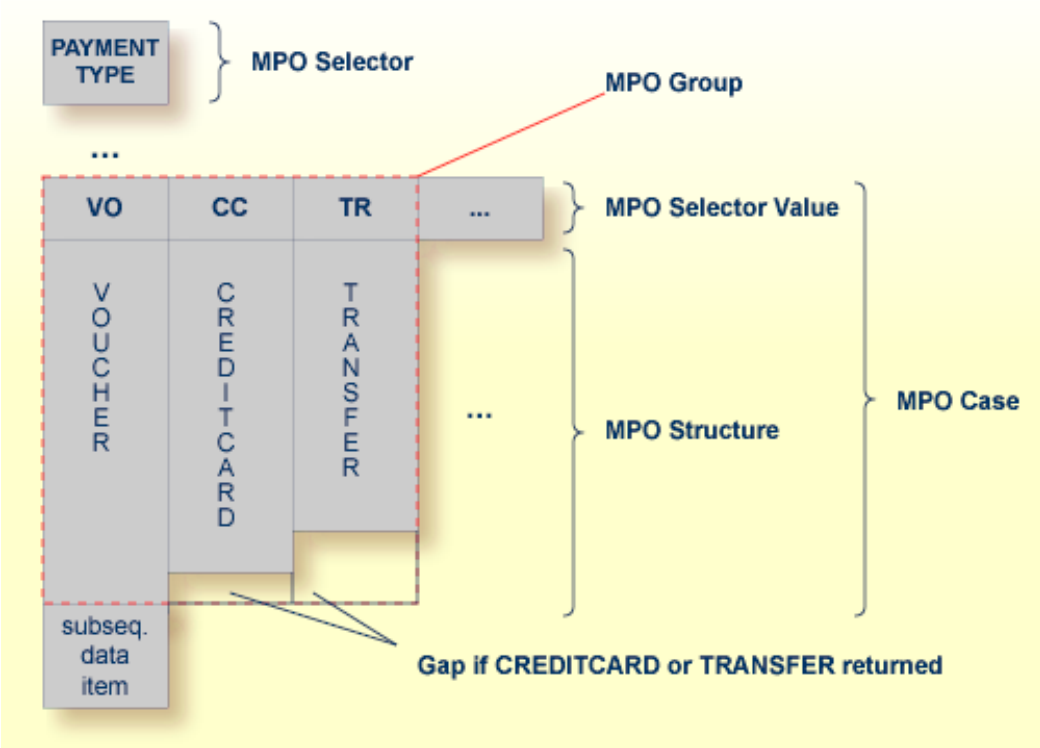
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
```

```
          88 PAYMENT-TYPE-DIRECTDEBIT           VALUE "DB".
         . . .
       02 <preceding data items>                PIC <clause>.
      . . .
       02 PAYMENT-DATA                          PIC X(256).
       02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
          04 VOUCHER-ORIGIN                     PIC X(128).
          04 VOUCHER-SERIES                     PIC X(128).
       02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
          04 CREDITCARD-NUMBER                  PIC 9(18).
          04 CREDITCARD-COMPANY                 PIC X(128).
          04 CREDITCARD-CODE                    PIC 9(12).
          04 CREDITCARD-VALIDITY                PIC X(8).
       02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
          04 TRANSFER-NAME                      PIC X(128).
          04 TRANSFER-IBAN                      PIC X(34).
          04 TRANSFER-BIC                       PIC X(11).
       02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
          04 DIRECTDEBIT-IBAN                   PIC X(34).
          04 DIRECTDEBIT-NAME                   PIC X(128).
          04 DIRECTDEBIT-EXPIRES                PIC 9(8).
         . . .
       02 <subsequent data items>               PIC <clause>.
      . . .

   . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
```

```
        WHEN PAYMENT-TYPE-TRANSFER
           MOVE . . . TO TRANSFER-NAME
           MOVE . . . TO TRANSFER-IBAN
           MOVE . . . TO TRANSFER-BIC
        WHEN PAYMENT-TYPE-DIRECTDEBIT
           MOVE . . . TO DIRECTDEBIT-IBAN
           MOVE . . . TO DIRECTDEBIT-NAME
           MOVE . . . TO DIRECTDEBIT-EXPIRES
        WHEN
           . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.
- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

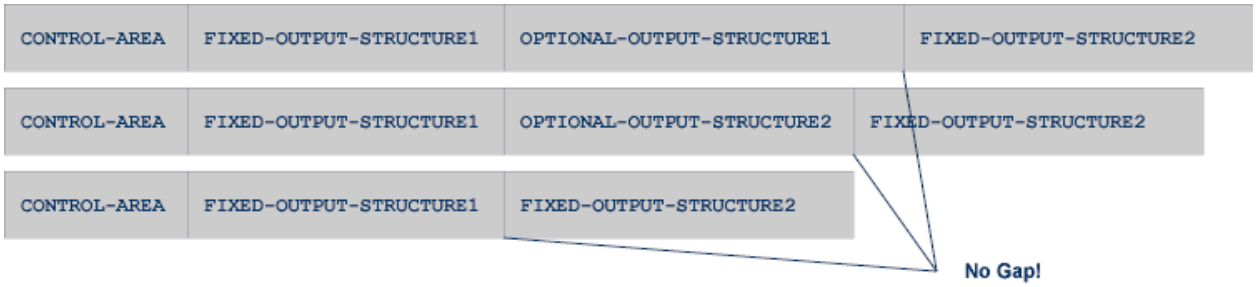
Optional Output with Groups

COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.
- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
      02 FIX-OUTPUT-ITEM13                  PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
```

```
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

Note: If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

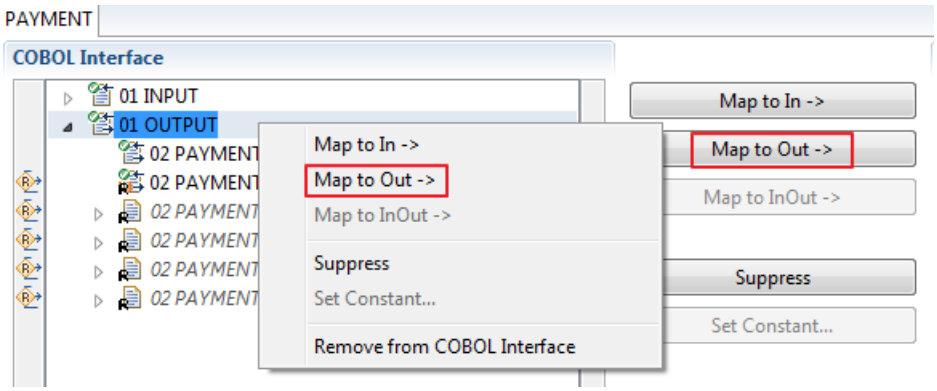
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

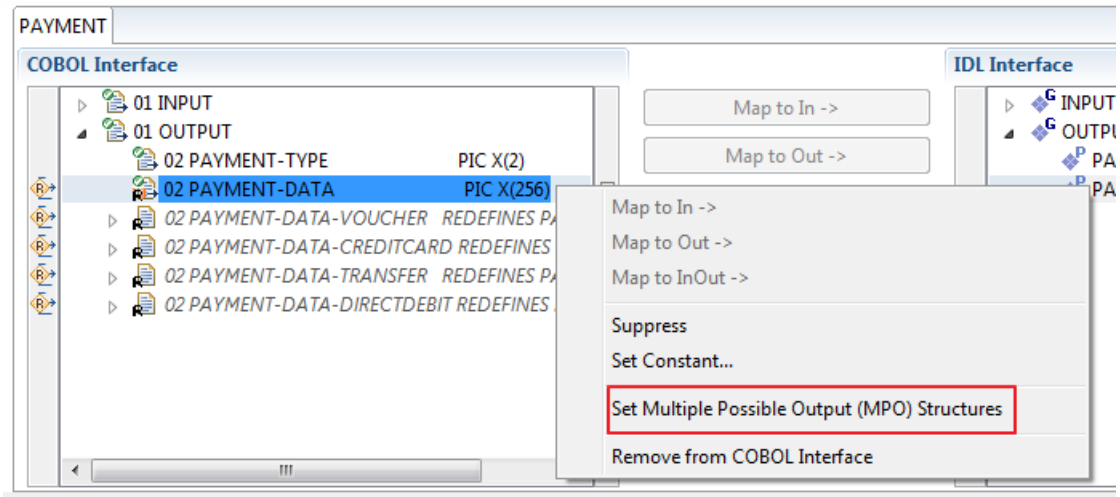
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

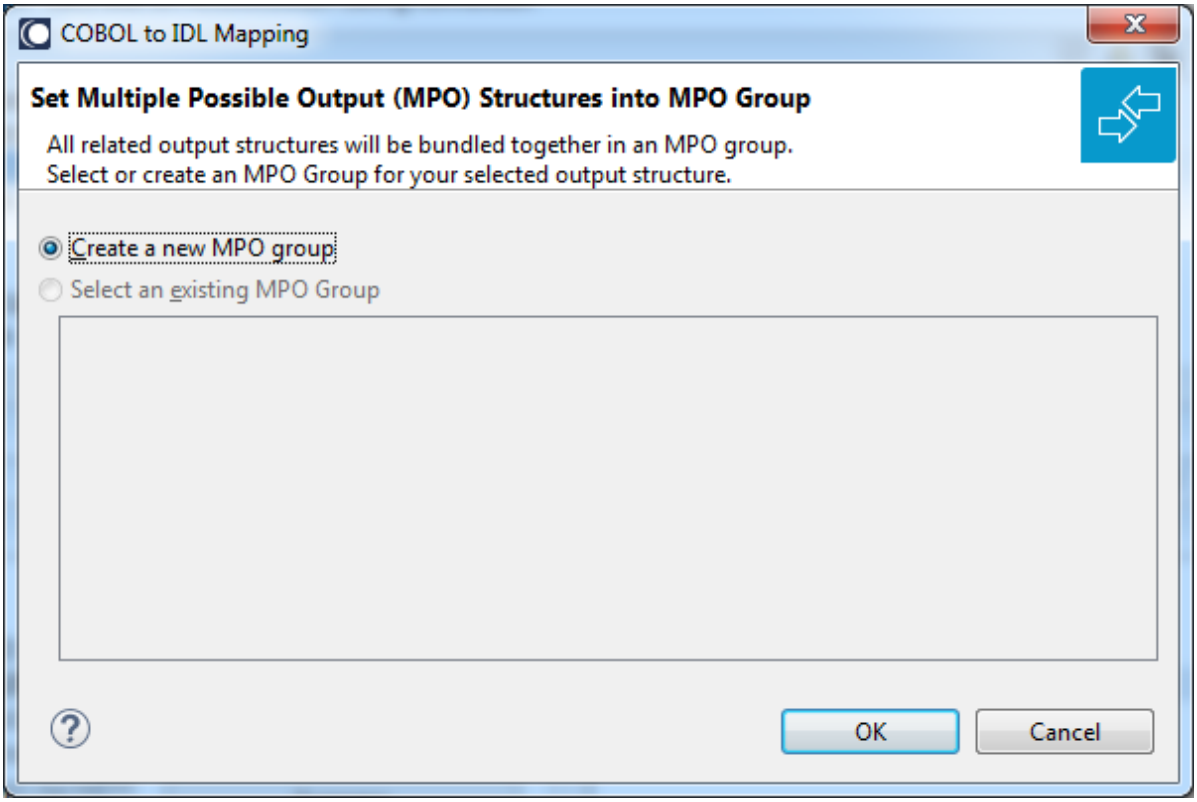
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



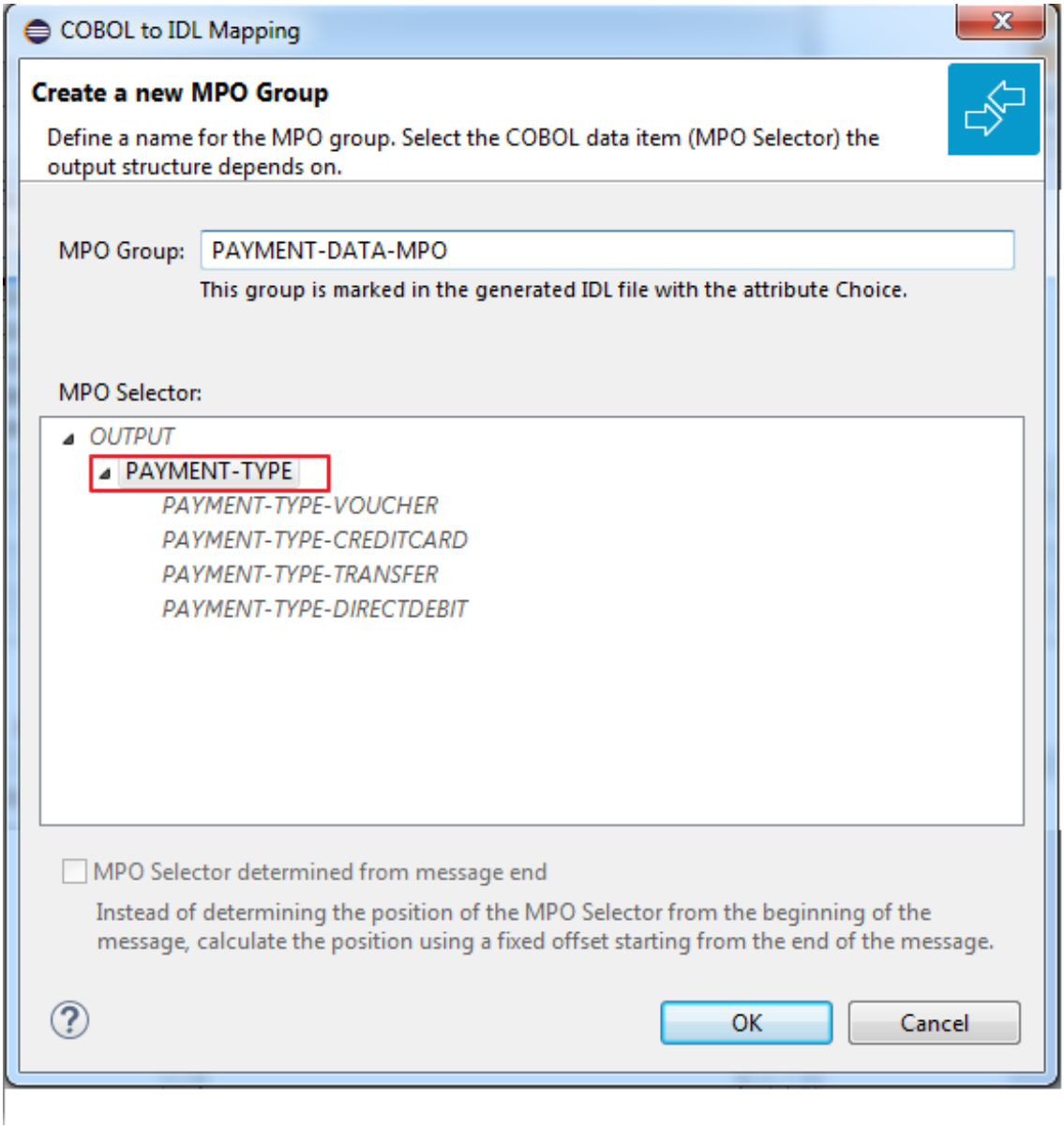
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



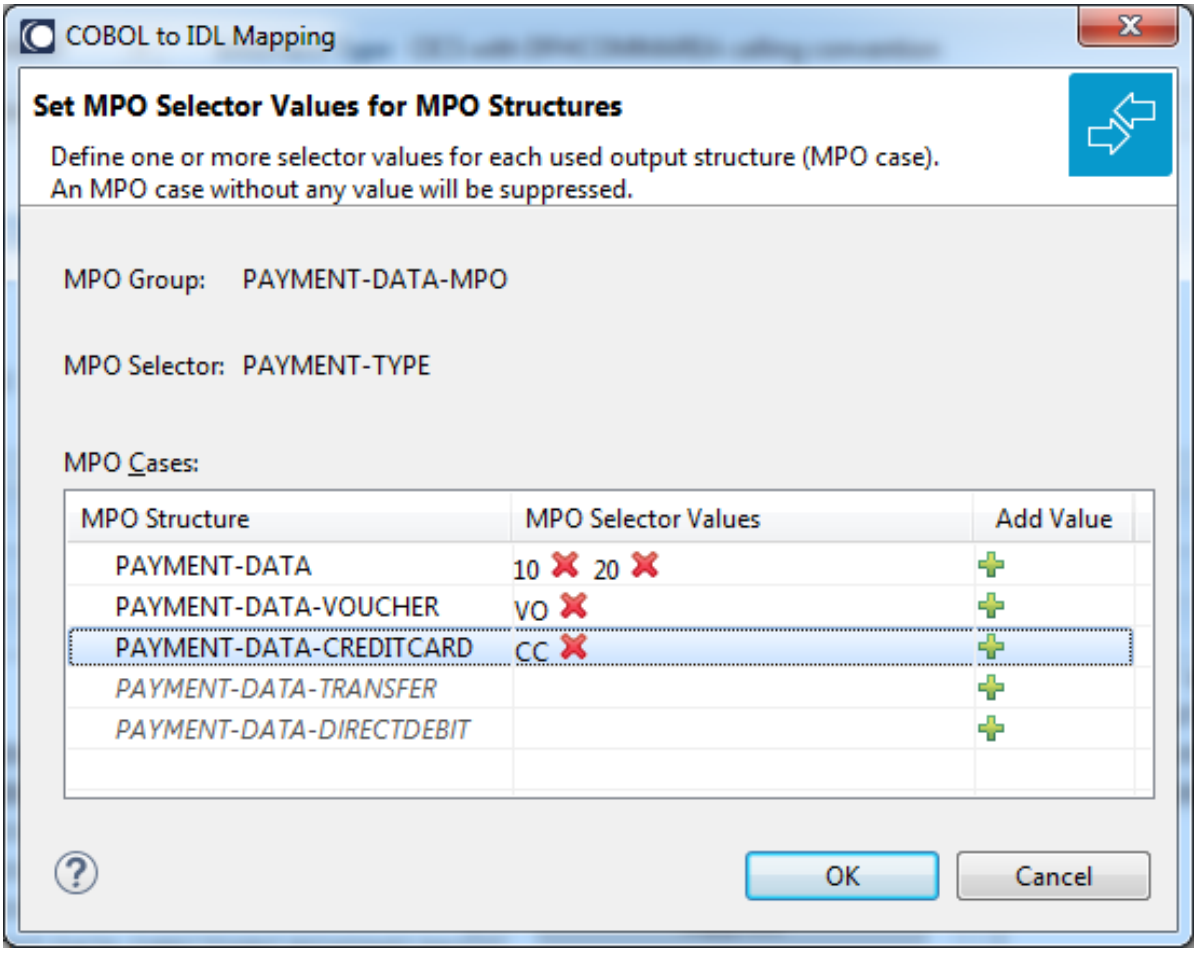
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



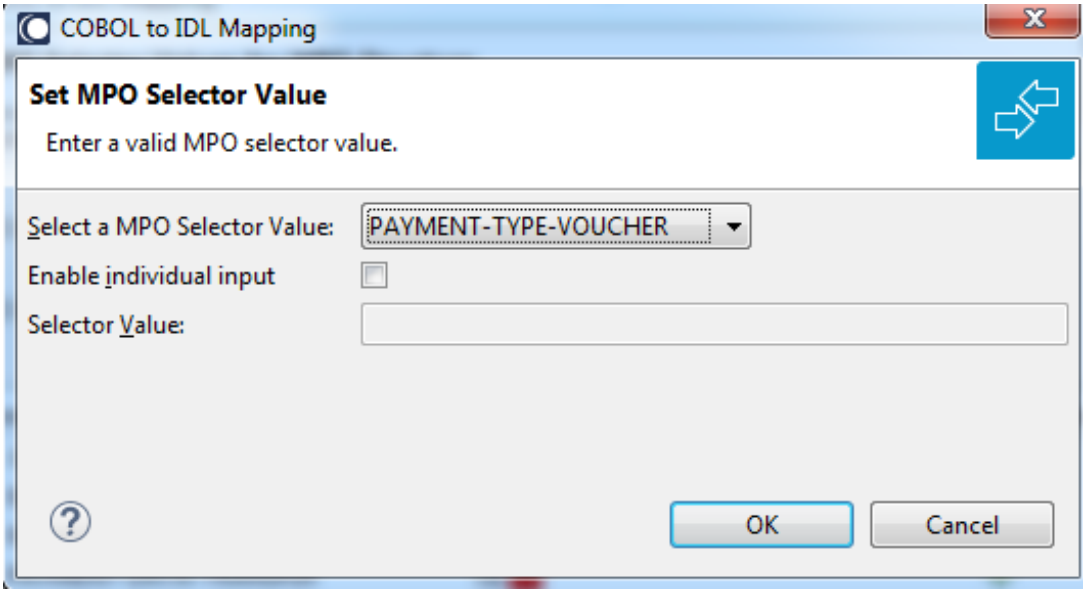
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

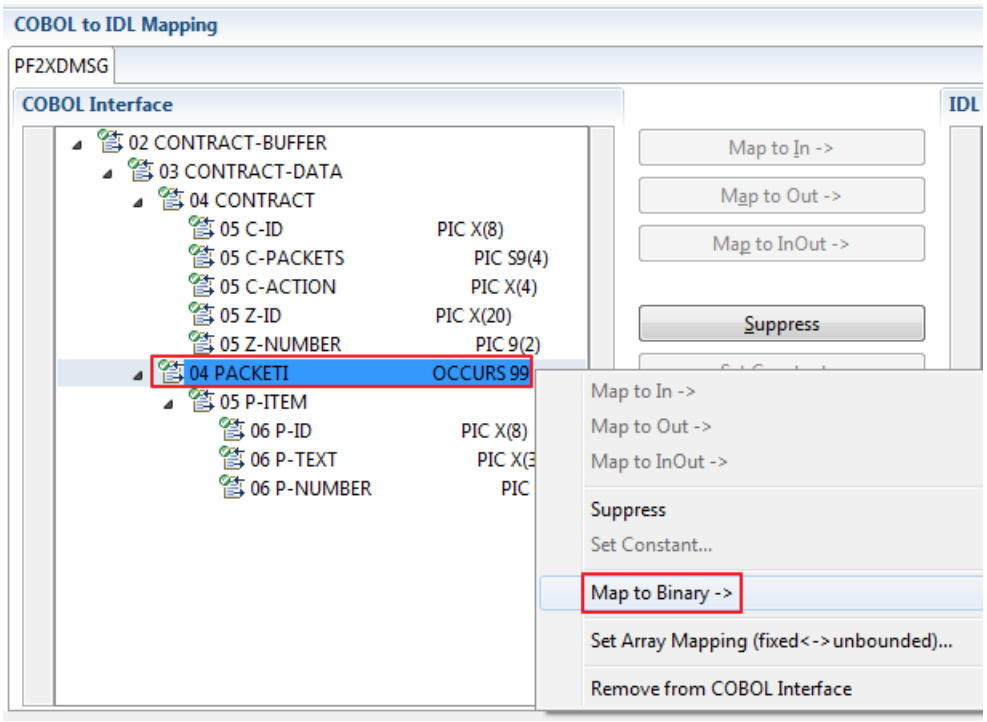
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

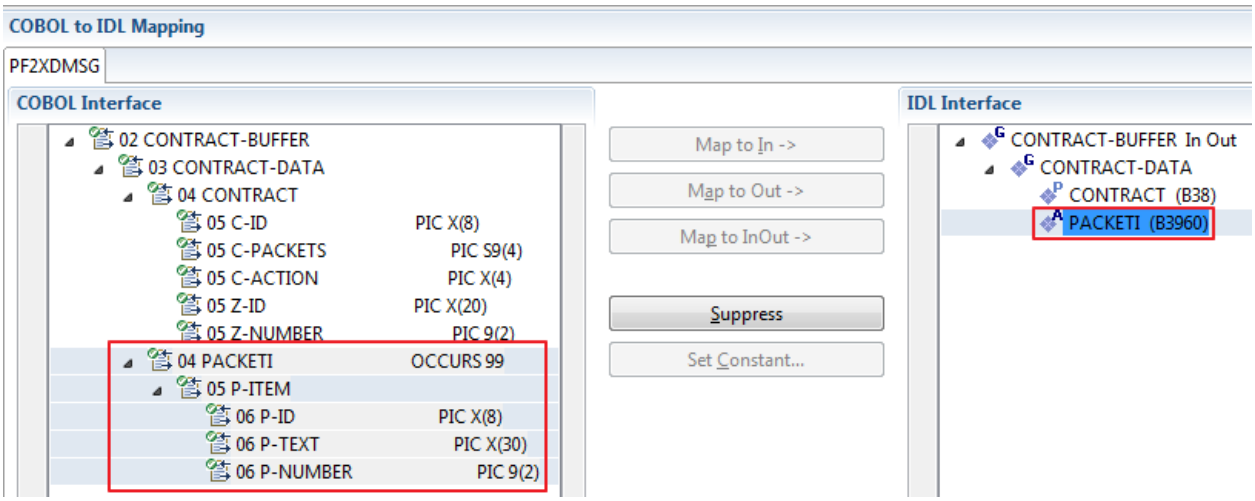
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images).

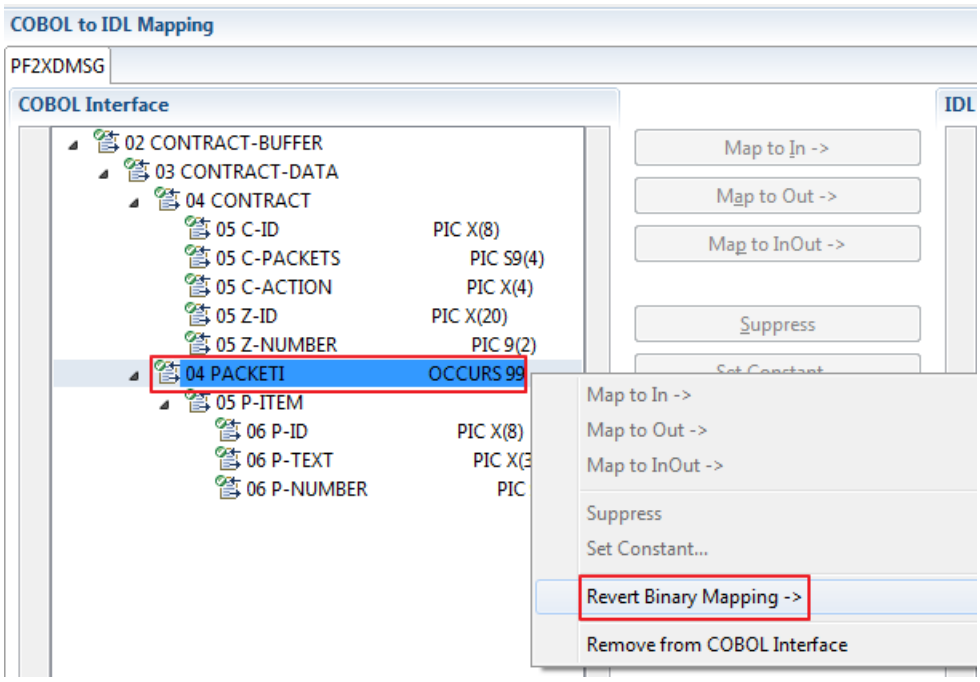


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

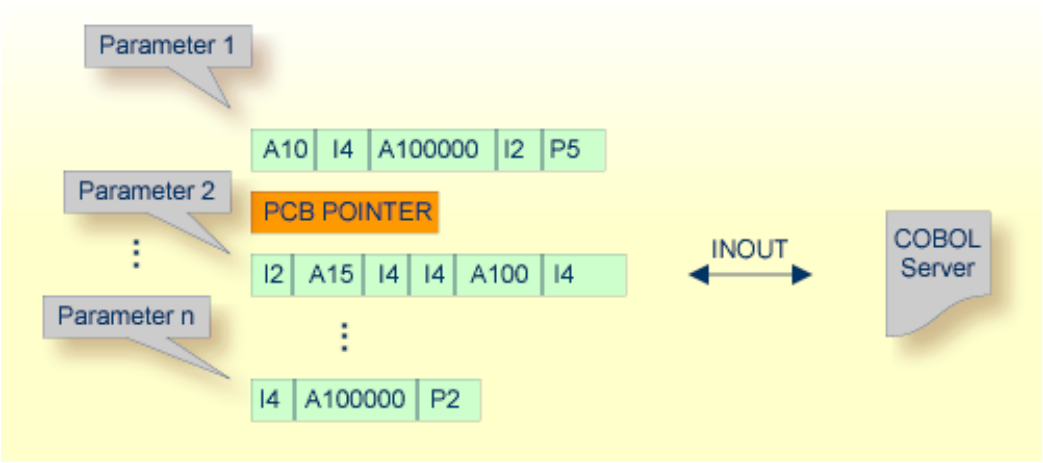
When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

14 IMS BMP with Standard Linkage Calling Convention



Introduction

If your IMS BMP program contains PCB pointers, you have assigned the IMS PSB list in the previous step *Step 4: Define the Extraction Settings and Start Extraction*. If a required IMS PSB list is not assigned, the PCB pointers are not detected; go back to *Step 4: Define the Extraction Settings and Start Extraction* and assign the IMS PSB list first.

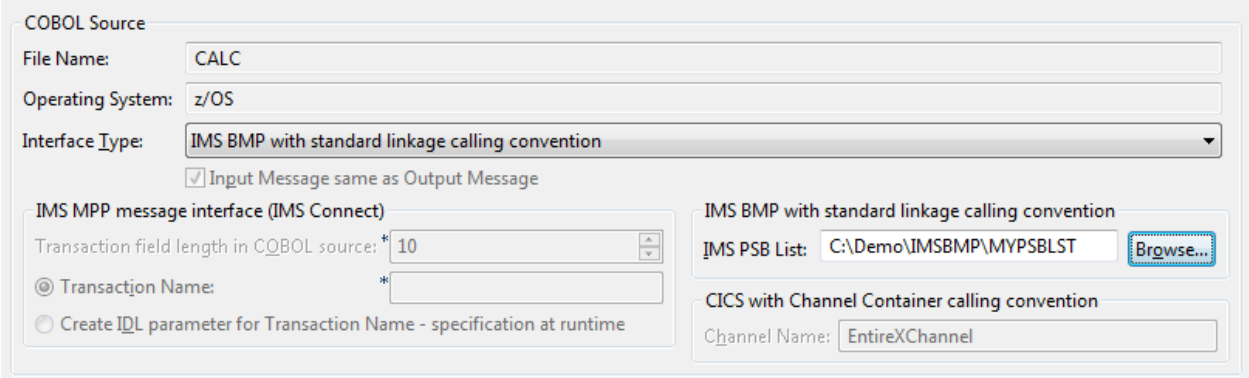
If the IMS PSB list is correctly assigned, the COBOL data items (including the PCB pointers) can be evaluated by the extractor because this type of COBOL server contains a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters. In most cases the offered COBOL data items will be correct, but you should always check them manually.

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 BMP Standard Call Interface

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

Make sure the interface type is correct.



You can set optionally the IMS PSB List. If your COBOL server contains PCB pointers, choose **Browse**. Otherwise, the PCB pointers are not detected and cannot be provided by the RPC Server for IMS to your COBOL server at runtime, and unexpected behavior may occur. For the contents of the IMS PSB list, see *IMS PCB Pointer IDL Rules*.

To select the COBOL interface data items of your COBOL server

1 Add the COBOL data items to the **COBOL Interface** using the context menu or toolbar available in the ***COBOL Source View*** and ***COBOL Interface***. See **Notes**.

2 Continue with ***COBOL to IDL Mapping***.

Notes:

1. If there is a `PROCEDURE DIVISION` header available, the parameters listed define exactly the COBOL interface. These COBOL data items are within the `LINKAGE SECTION` and are already selected to the COBOL interface in initial state when you enter the COBOL Mapping Editor. The `PROCEDURE DIVISION` header might not be available if you are extracting from a copybook or part of the COBOL source.
2. It is very important to select the right COBOL data items describing the COBOL interface correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL interface contains `REDEFINE`s, 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.
4. Make sure the PCB pointers are also selected at the correct position.

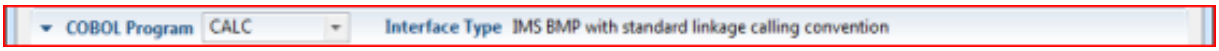
The user interface of the COBOL Mapping Editor is described below.

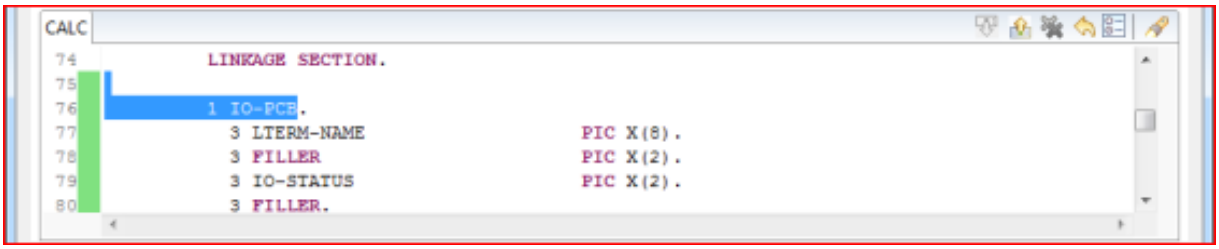
Mapping Editor User Interface

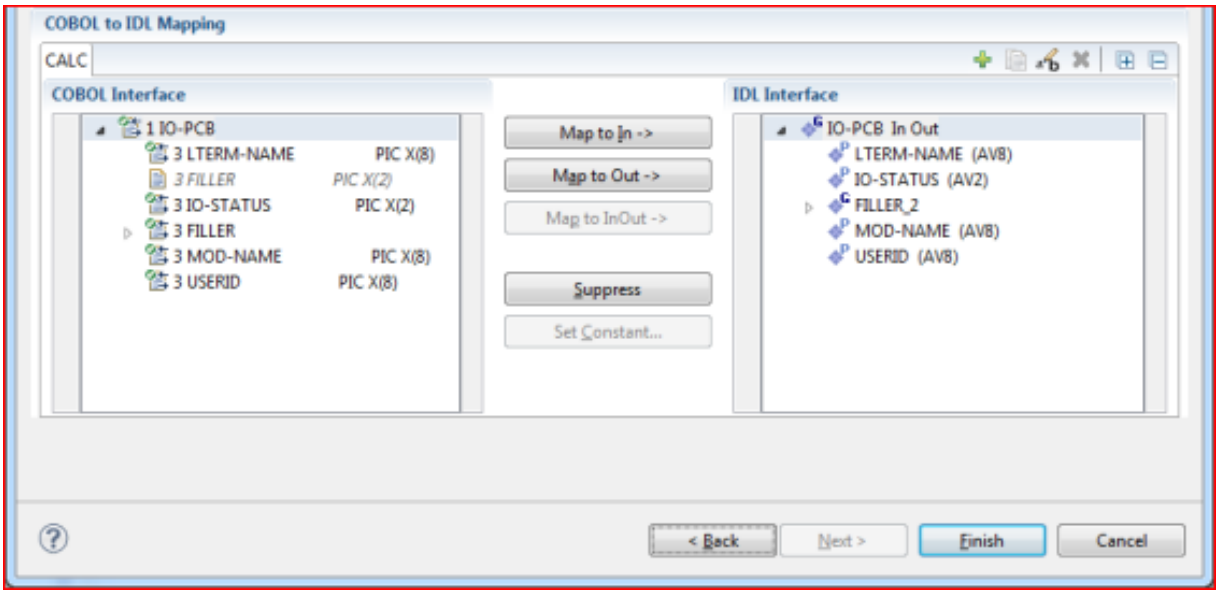
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

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

For COBOL server programs with standard call interface types, the user interface of the COBOL Mapping Editor looks like this:

1. 

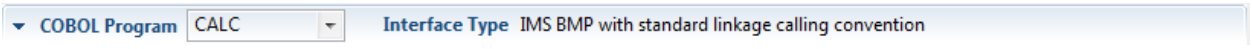
2. 

3. 

1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program

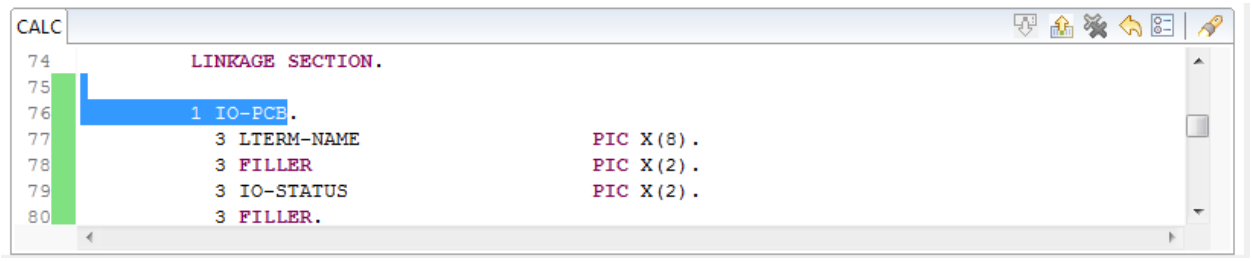
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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.
- 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

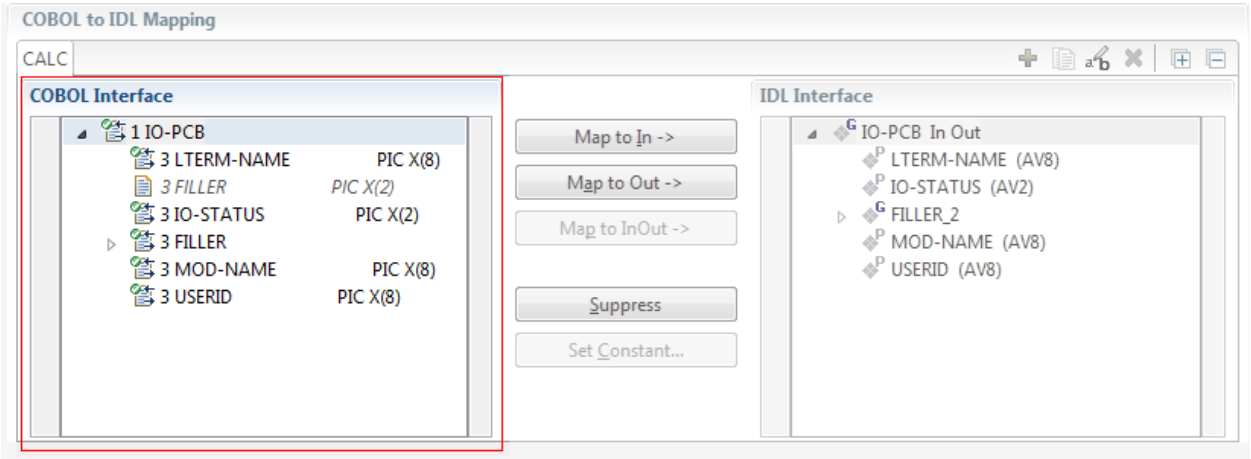
This section covers the following topics:

- 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 (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.



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.

Map to In | Out | InOut	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 Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

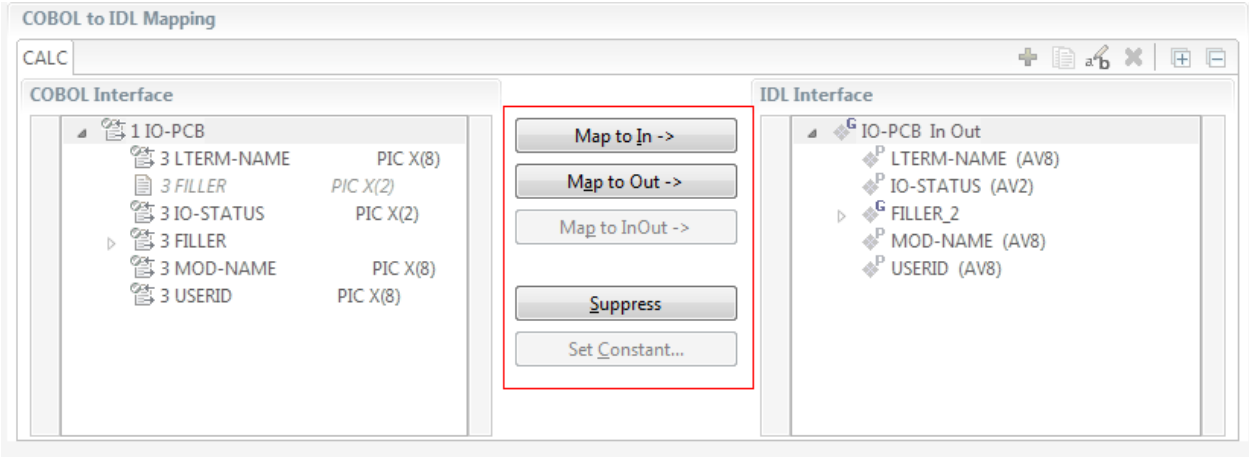
Mapping Icons

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

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

Mapping Buttons

The following buttons are available:



Map to In | Out | InOut ->
See *Map to In, Out, InOut*. A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.

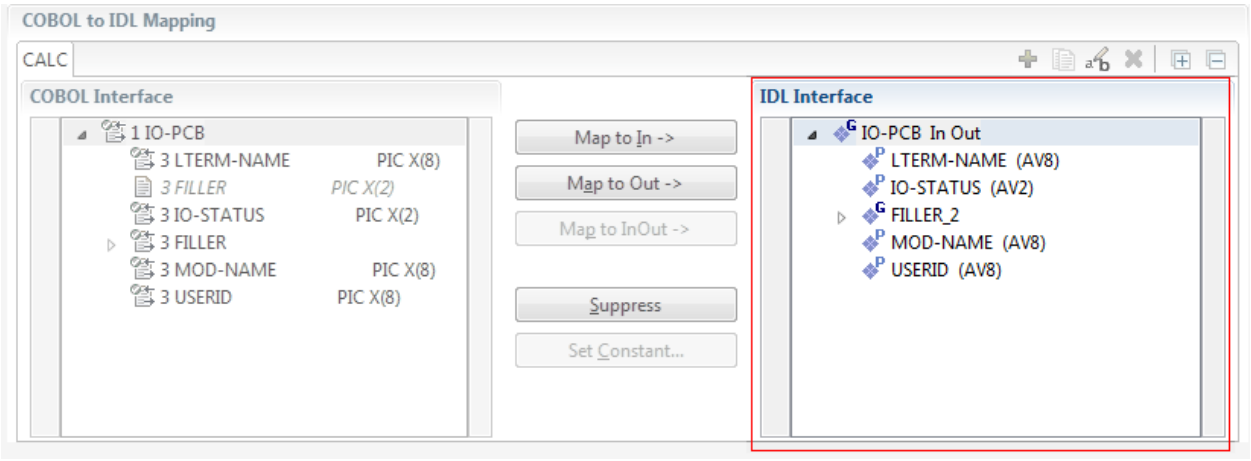
Suppress
See *Suppress Unneeded COBOL Data Items*.

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 the IDL parameter.
- 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.



Mapping Editor IDL Interface Mapping Functions

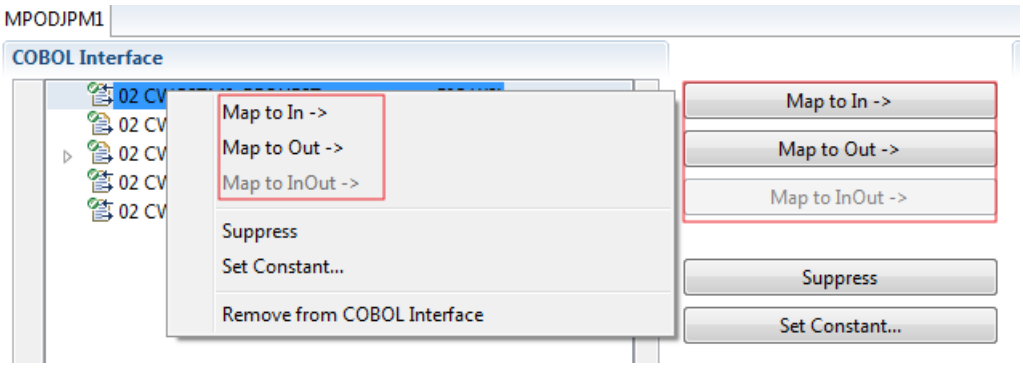
This section covers the following topics:

Map to In, Out, InOut

With the **Map to In**, **Out**, **InOut** functions you make a COBOL data item visible as an IDL parameter in the IDL interface. With correct IDL directions you design the IDL interface by defining input and output parameters. COBOL programs have no parameter directions, so you need to set IDL directions manually.

To provide IDL directions

- Go *step-by-step* through all *top-level* COBOL data items in the COBOL interface and use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons to make the COBOL data items visible and provide IDL directions in the IDL interface:



Notes:

1. If a *top-level* COBOL *group* is mapped, the IDL direction is inherited by all subordinate COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subordinate COBOL data items can only be mapped to the same IDL direction as their *top-level* COBOL *group* data item.
3. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.
4. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Map OCCURS DEPENDING ON

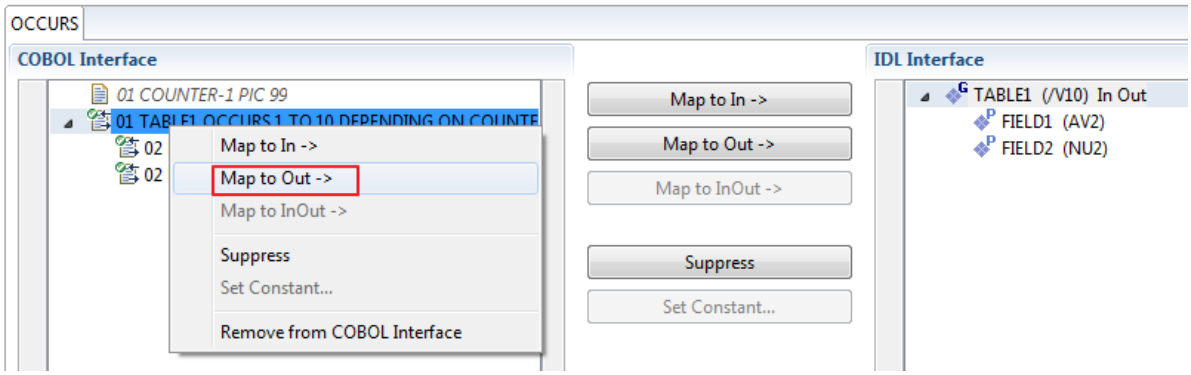
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):

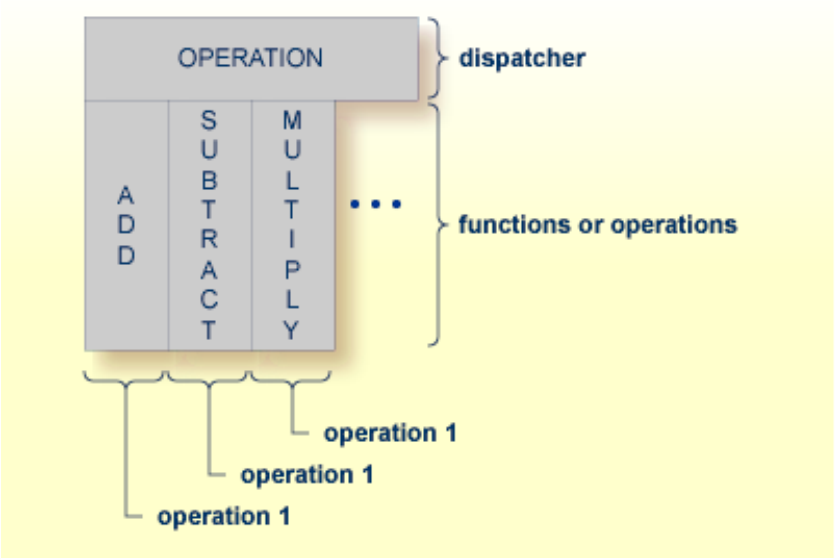


Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.
3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 "-"
```

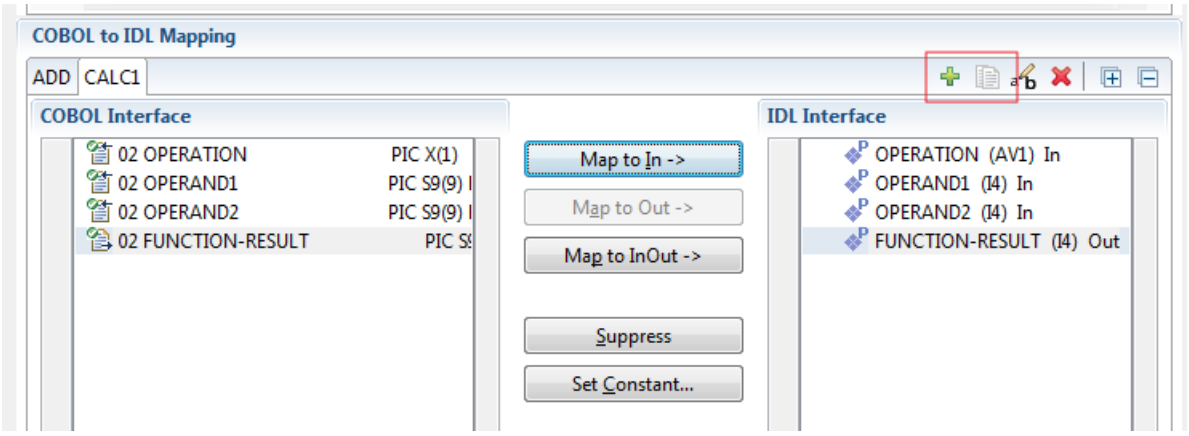
```
            SUBTRACT OPERAND2 FROM OPERAND1
            GIVING FUNCTION-RESULT
         WHEN "*"
            MULTIPLY OPERAND1 BY OPERAND2
            GIVING FUNCTION-RESULT
         WHEN . . .

     END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

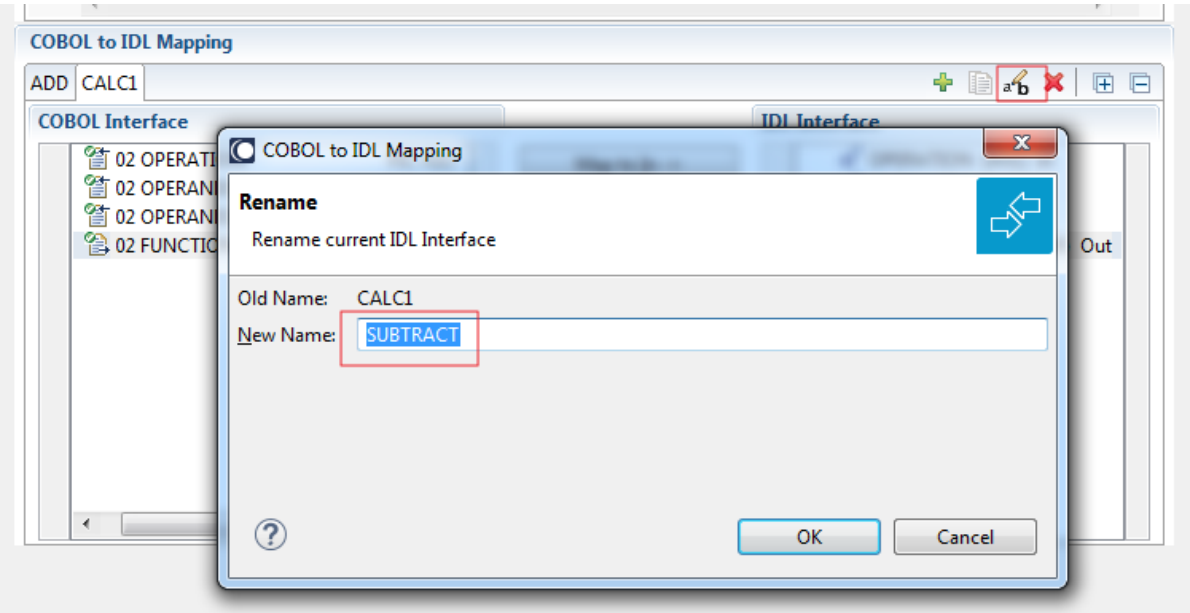
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

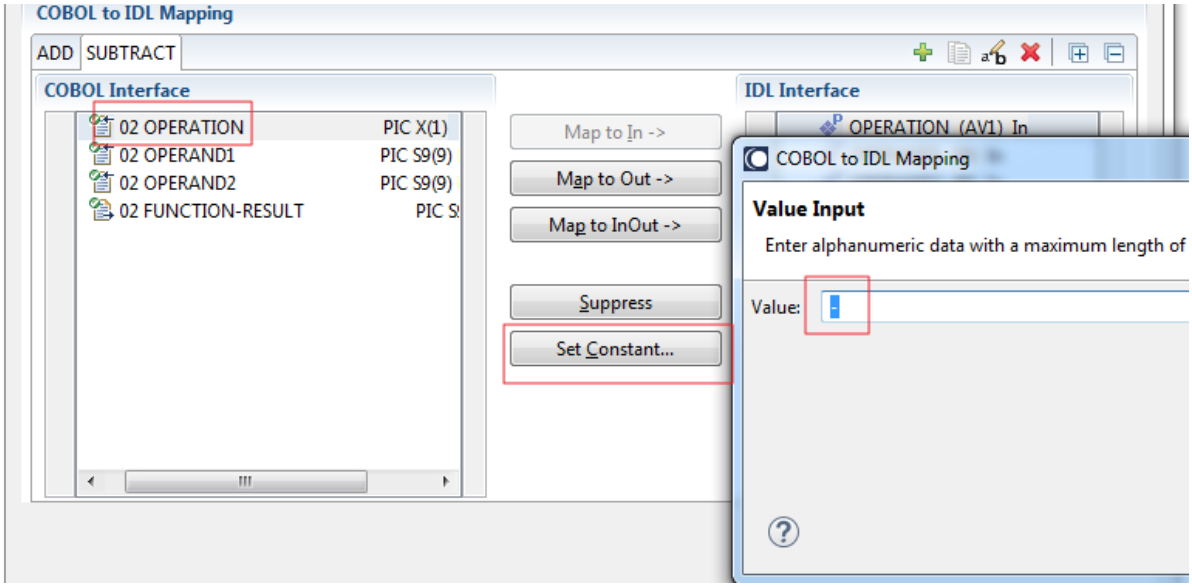
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.

- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

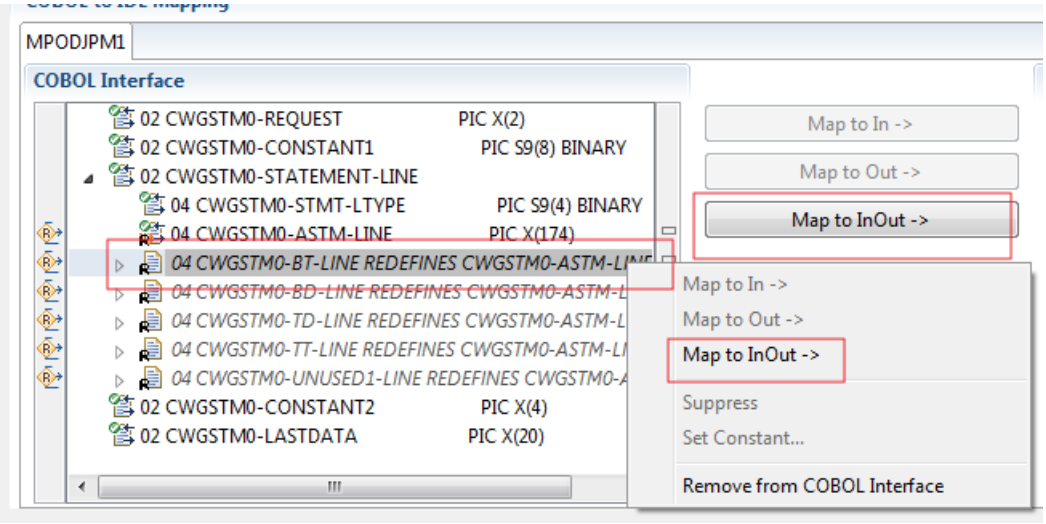
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

Select REDEFINE Paths

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

To select redefine paths

- Use the **Map to In**, **Out** or **InOut** functions available in the context menu of the COBOL interface and as mapping buttons 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:

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

Suppress Unneeded COBOL Data Items

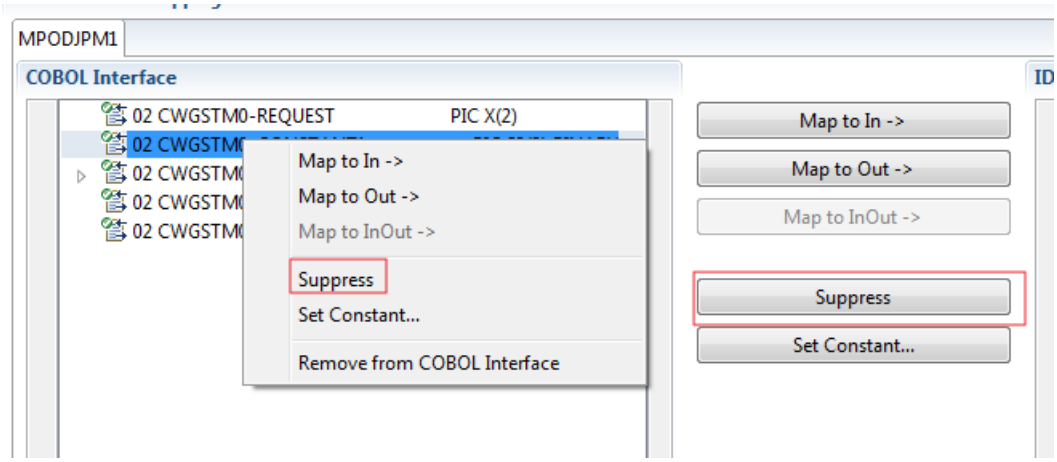
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface and as mapping button to make the COBOL data item invisible in the IDL interface:



Notes:

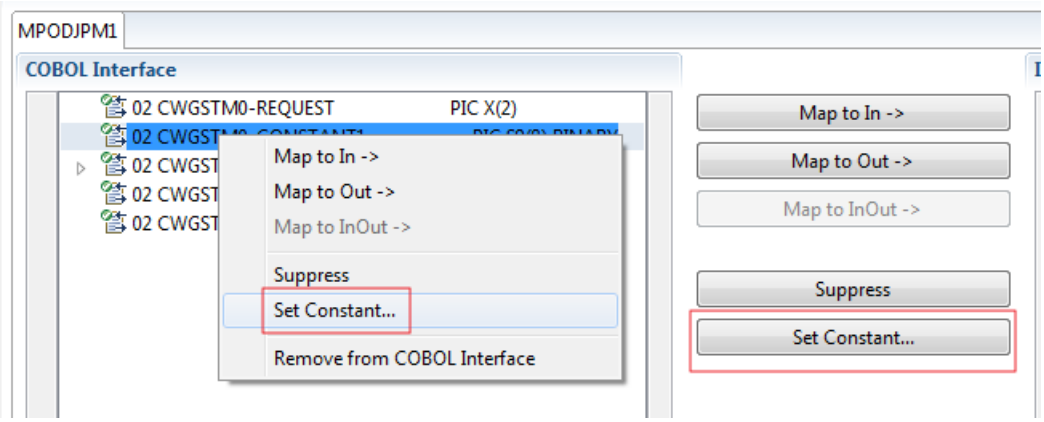
1. The COBOL data item is not part of the IDL interface. It is invisible for consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subordinate COBOL data items are suppressed as well.
4. With the inverse functions **Map to In**, **Out** or **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item is made visible in the IDL interface again.

Set COBOL Data Items to Constants

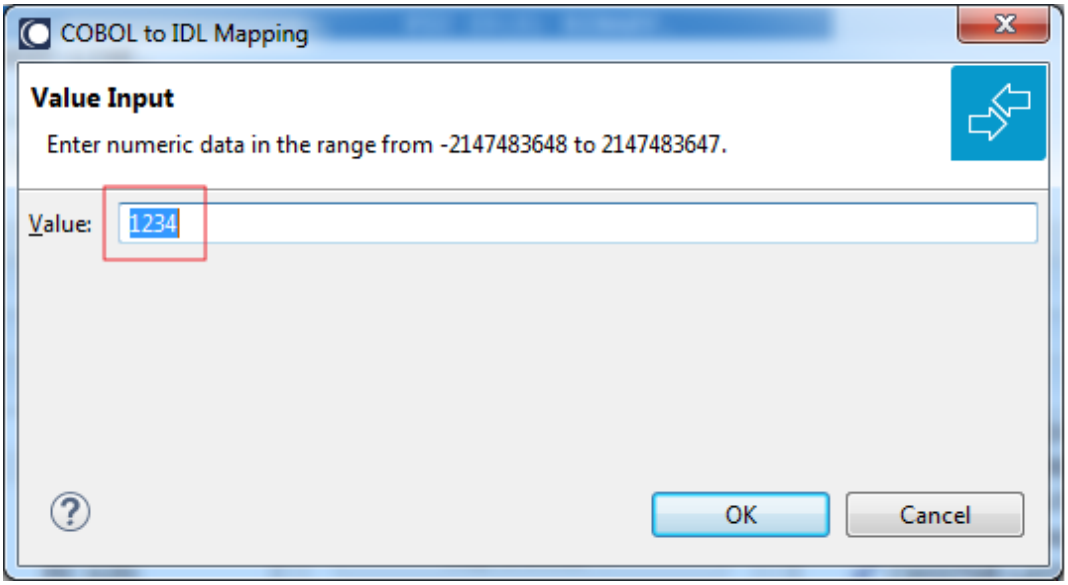
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter provides the defined constant in the COBOL data item to your COBOL server.
3. With the functions **Map to In**, **Out**, **InOut** (see above) available in the context menu of the COBOL interface and as mapping buttons, a COBOL data item can be made visible in the IDL interface again.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology
- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

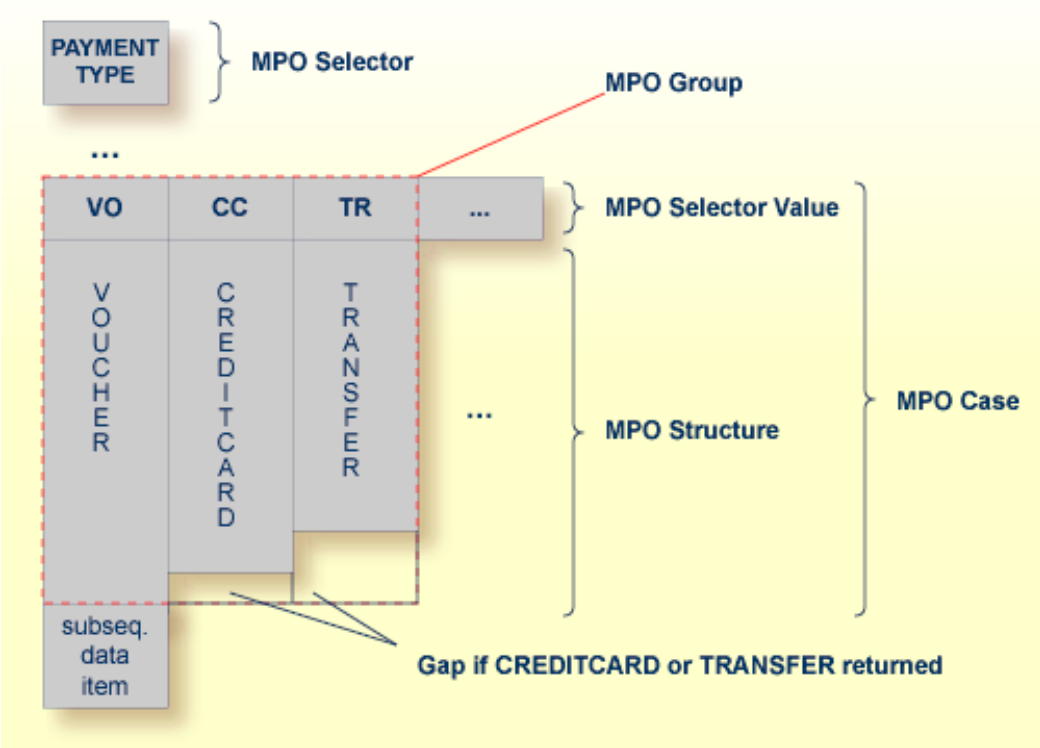
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
```

```
           88 PAYMENT-TYPE-DIRECTDEBIT            VALUE "DB".
          . . .
        02 <preceding data items>                 PIC <clause>.
       . . .
        02 PAYMENT-DATA                           PIC X(256).
        02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
           04 VOUCHER-ORIGIN                      PIC X(128).
           04 VOUCHER-SERIES                      PIC X(128).
        02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
           04 CREDITCARD-NUMBER                   PIC 9(18).
           04 CREDITCARD-COMPANY                  PIC X(128).
           04 CREDITCARD-CODE                     PIC 9(12).
           04 CREDITCARD-VALIDITY                 PIC X(8).
        02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
           04 TRANSFER-NAME                       PIC X(128).
           04 TRANSFER-IBAN                       PIC X(34).
           04 TRANSFER-BIC                        PIC X(11).
        02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
           04 DIRECTDEBIT-IBAN                    PIC X(34).
           04 DIRECTDEBIT-NAME                    PIC X(128).
           04 DIRECTDEBIT-EXPIRES                 PIC 9(8).
          . . .
        02 <subsequent data items>                 PIC <clause>.
       . . .

    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
```

```
        WHEN PAYMENT-TYPE-TRANSFER
           MOVE . . . TO TRANSFER-NAME
           MOVE . . . TO TRANSFER-IBAN
           MOVE . . . TO TRANSFER-BIC
        WHEN PAYMENT-TYPE-DIRECTDEBIT
           MOVE . . . TO DIRECTDEBIT-IBAN
           MOVE . . . TO DIRECTDEBIT-NAME
           MOVE . . . TO DIRECTDEBIT-EXPIRES
        WHEN
           . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.
- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

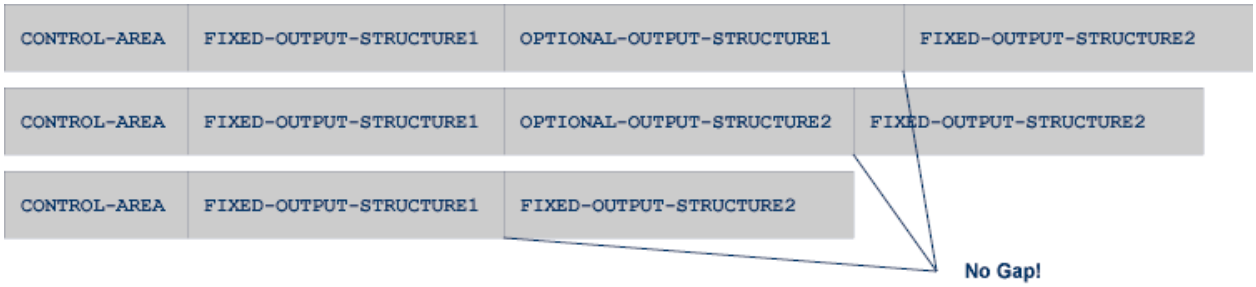
Optional Output with Groups

COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.
- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
      02 FIX-OUTPUT-ITEM13                  PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
```

```
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

Note: If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

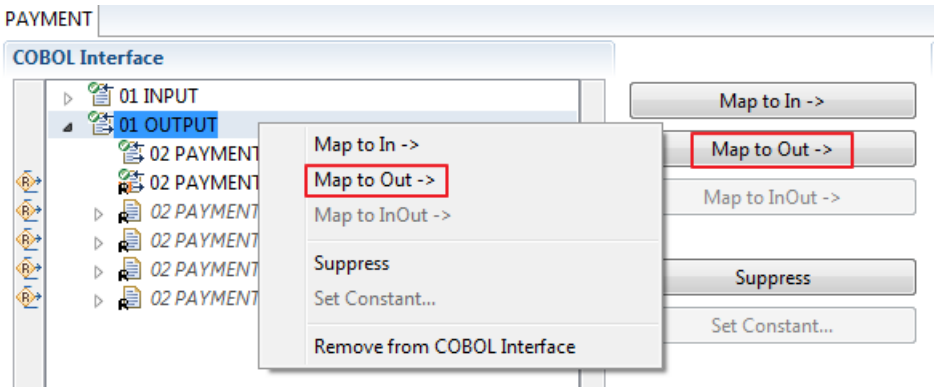
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

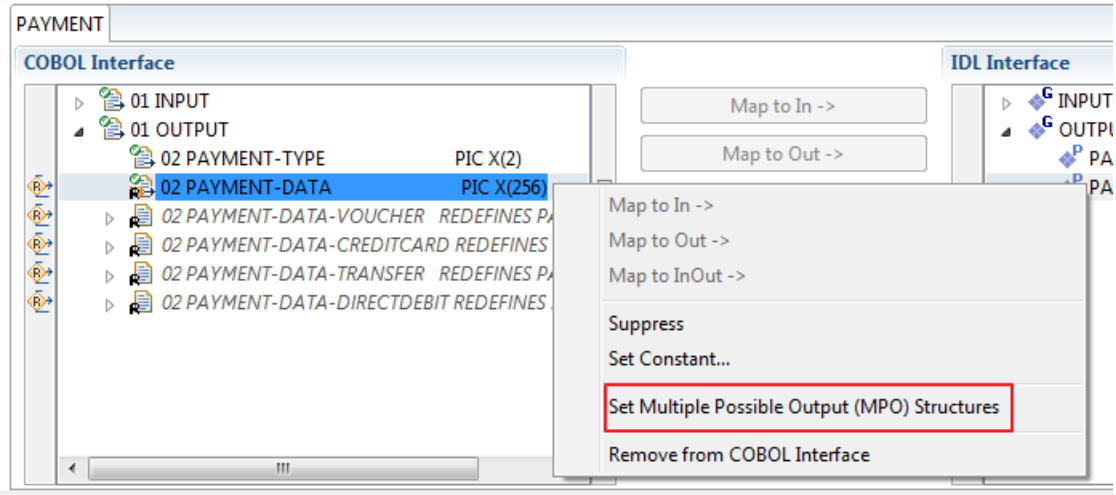
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

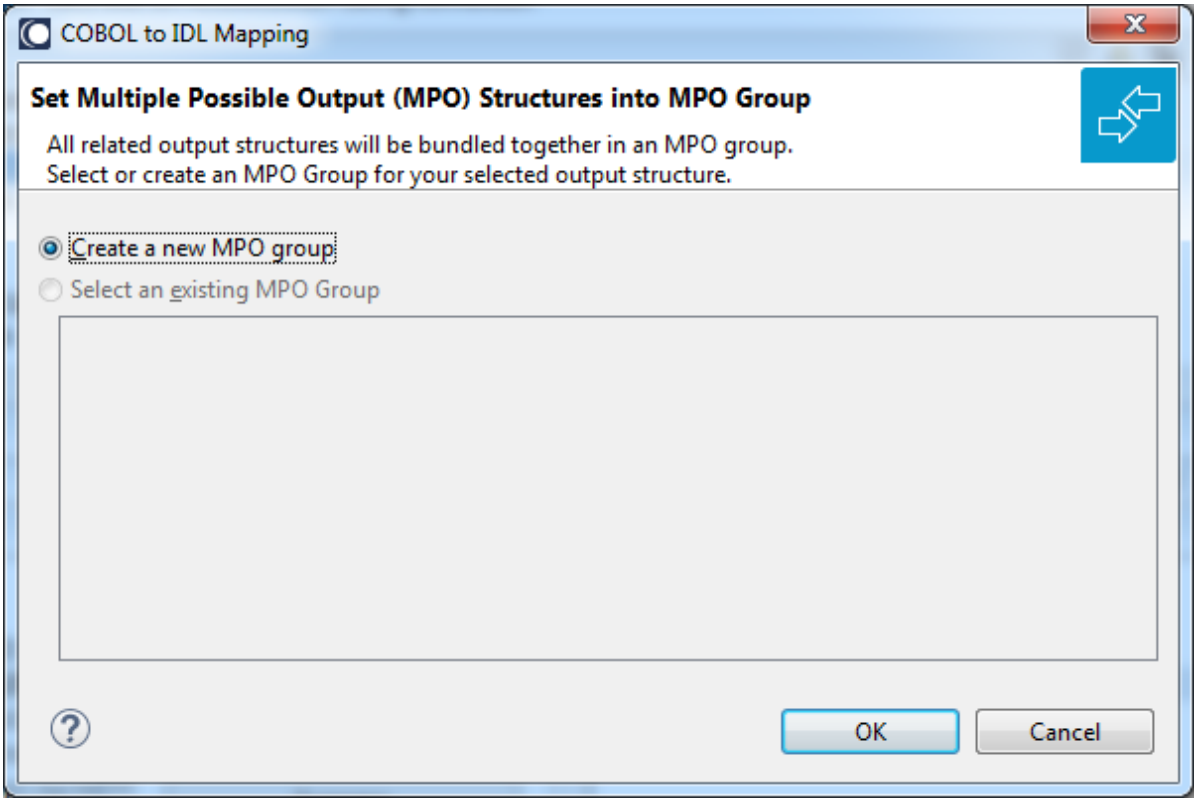
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



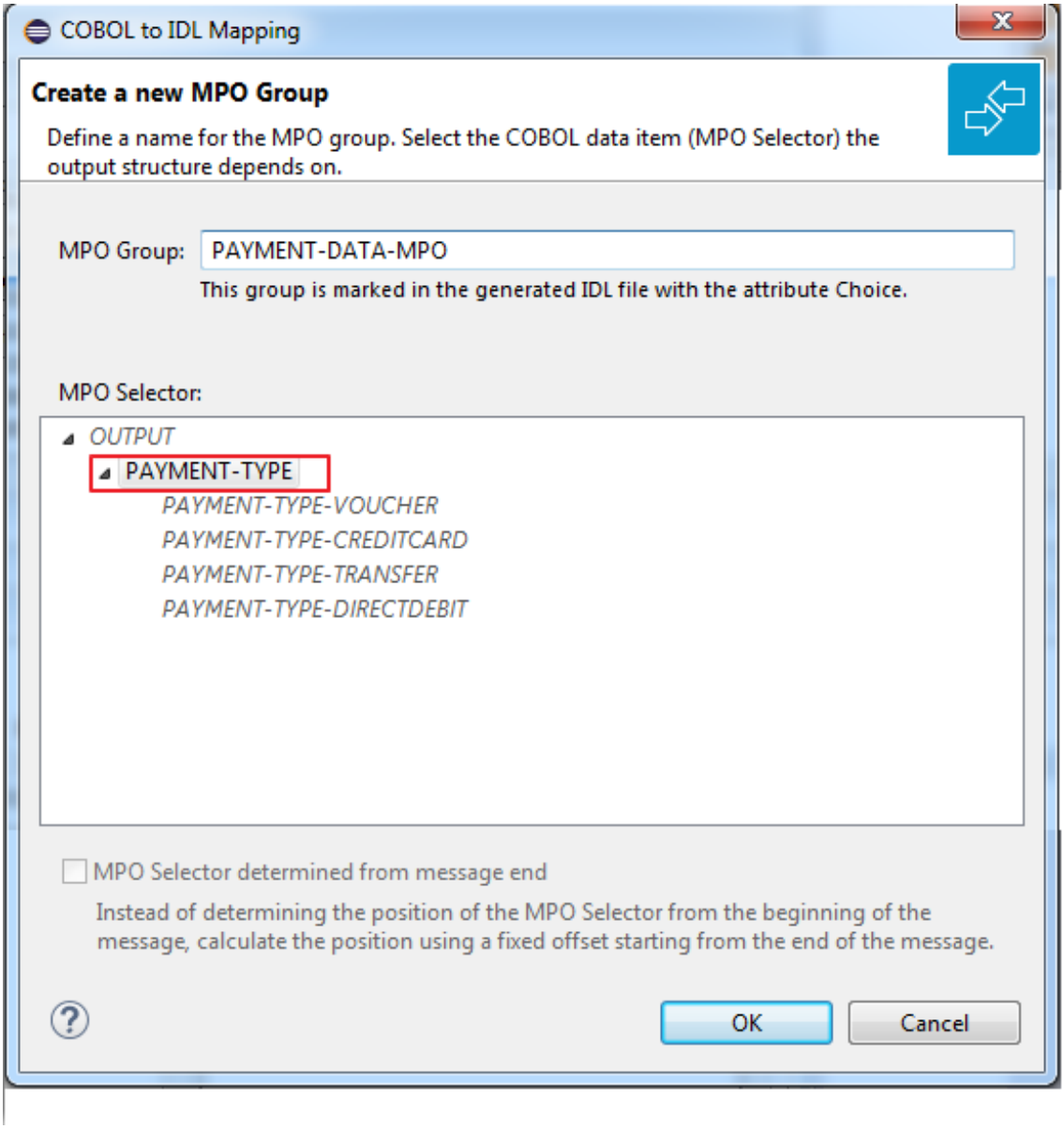
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



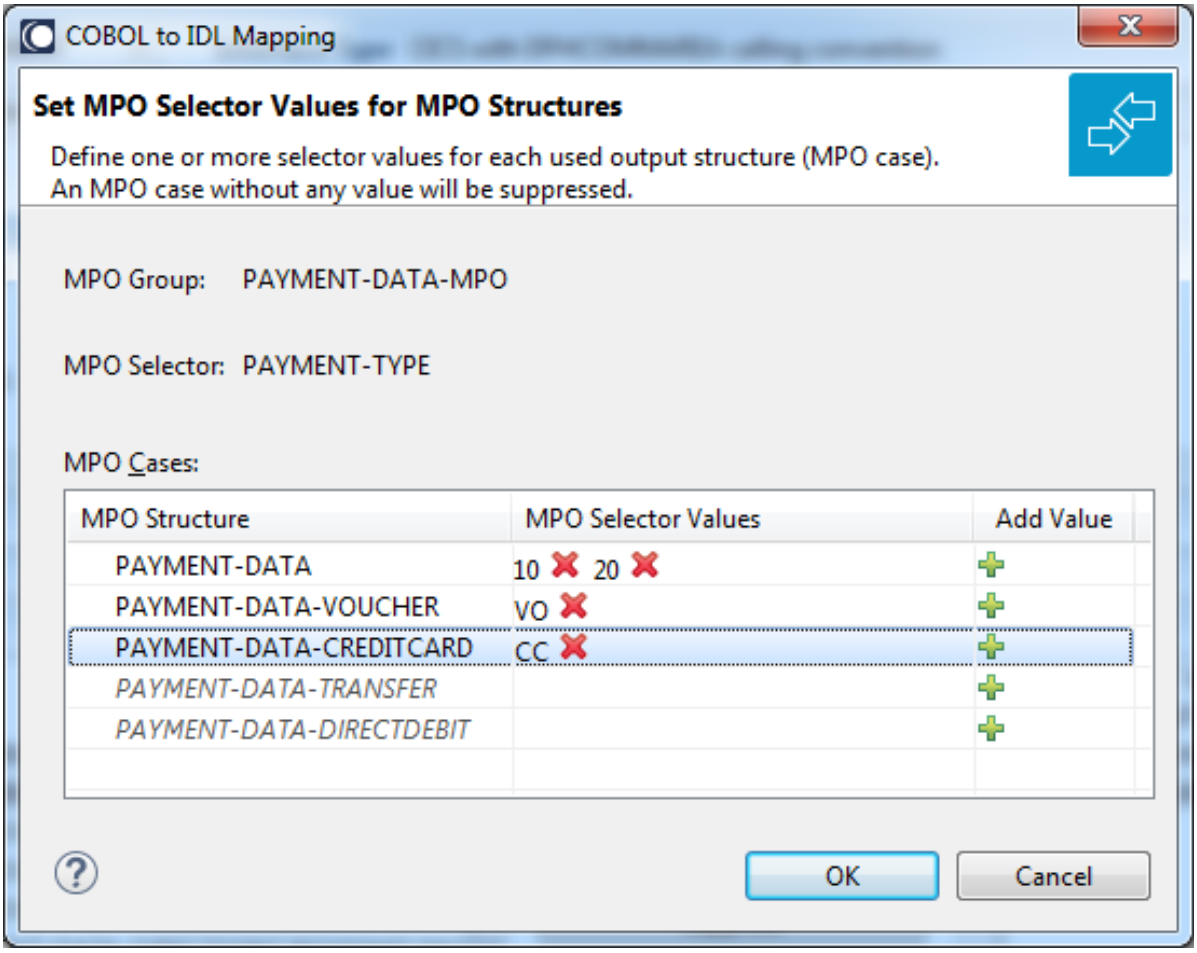
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



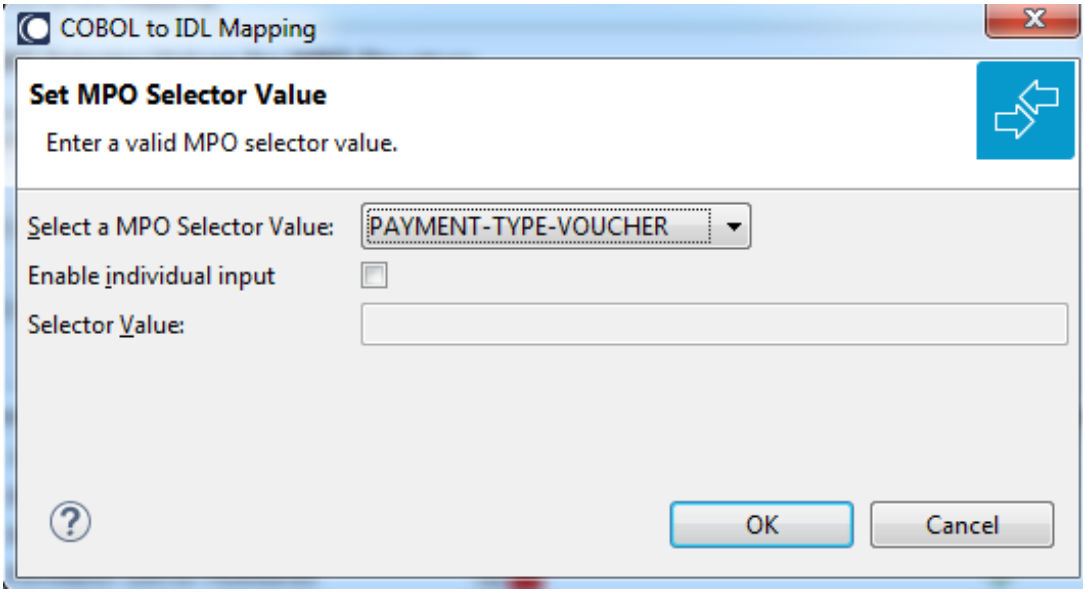
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

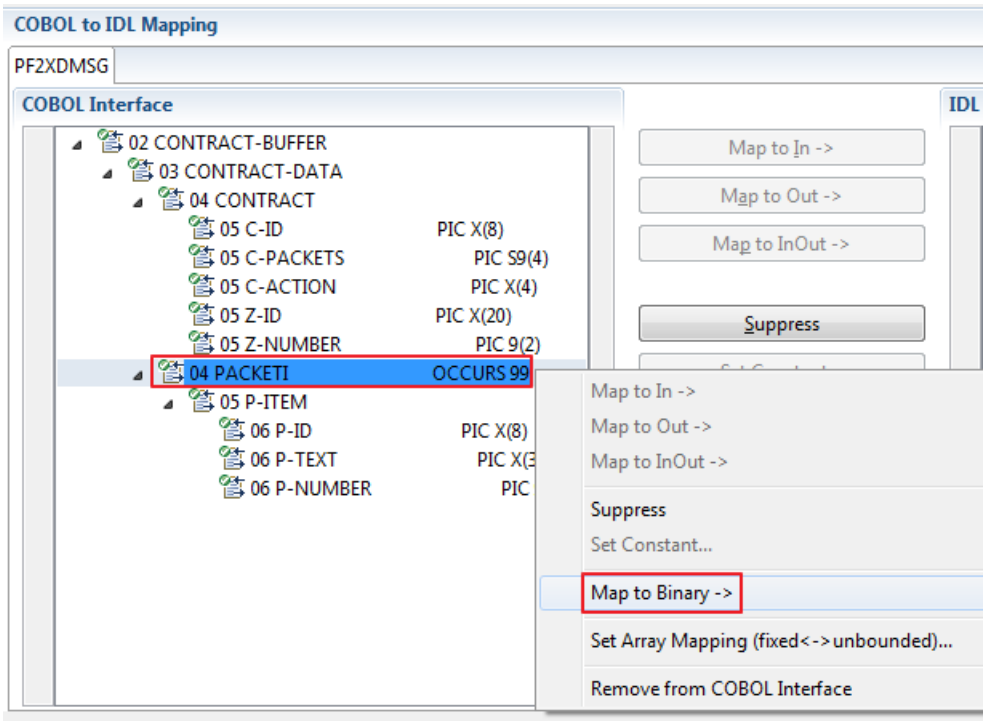
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

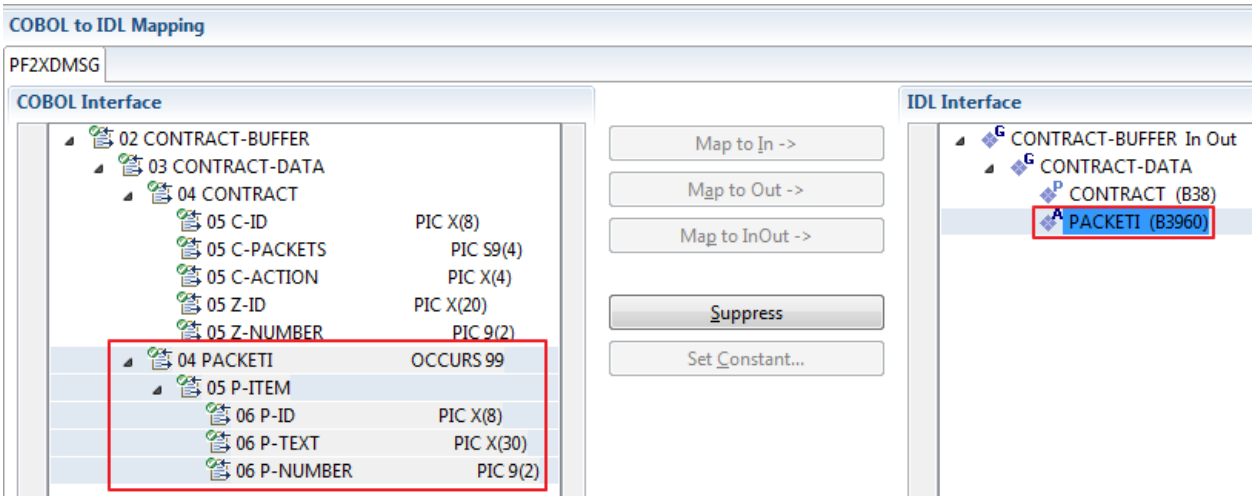
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images).

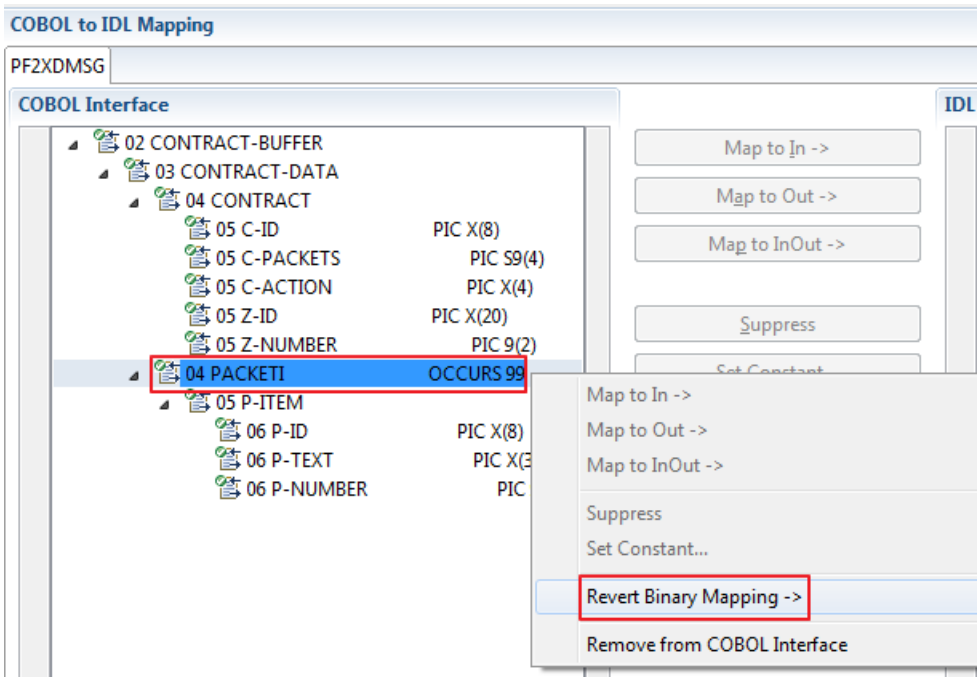


The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

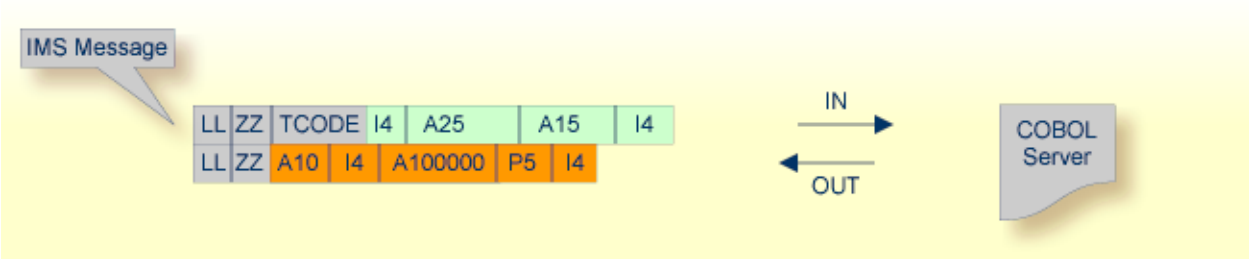
When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.



To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.


COBOL to IDL Mapping
PF2XDMSG
COBOL Interface
IDL
02 CONTRACT-BUFFER
03 CONTRACT-DATA
04 CONTRACT
05 C-ID PIC X(8)
05 C-PACKETS PIC S9(4)
05 C-ACTION PIC X(4)
05 Z-ID PIC X(20)
05 Z-NUMBER PIC 9(2)
04 PACKETI OCCURS 99
05 P-ITEM
06 P-ID PIC X(8)
06 P-TEXT
06 P-NUMBER
Map to In ->
Map to Out ->
Map to InOut ->
Suppress
Set Constant...
Revert Binary Mapping ->
Remove from COBOL Interface

15 IMS MPP Message Interface (IMS Connect)



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:

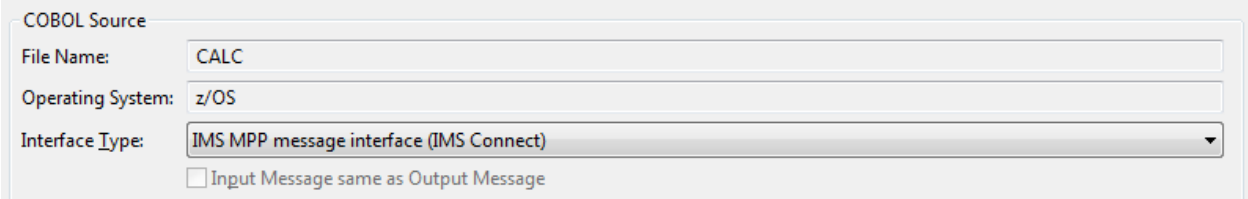
```
WORKING-STORAGE SECTION.
    . . .
 *  DLI Function Codes
    77 FCT-GU                          PIC X(4) VALUE 'GU  '.
    77 FCT-ISRT                        PIC X(4) VALUE 'ISRT'.
    . . .
    LINKAGE SECTION.
    . . .
    1 IO-PCB.
      3 LTERM-NAME                     PIC X(8).
      3 FILLER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

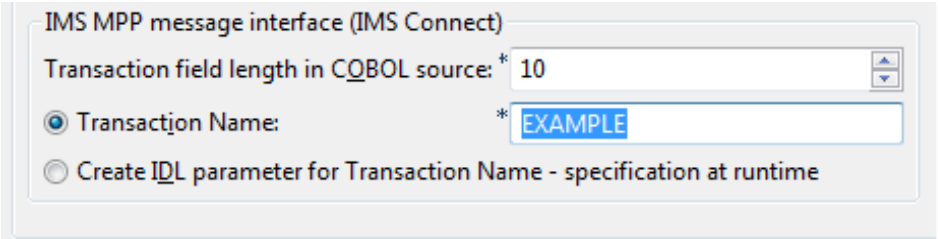
If you are extracting IDL from a COBOL source or extending the IDL file by extracting an additional COBOL source with interface type IMS MPP message interface (IMS Connect), the **Extractor Settings** dialog appears (see also *Step 4: Define the Extraction Settings and Start Extraction*).

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



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.

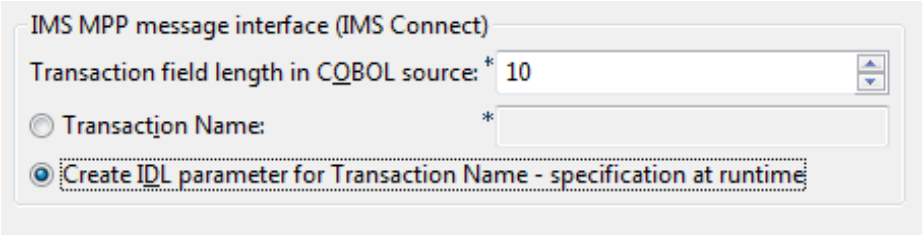


  Specify the length of the transaction field, which is usually the third physical field starting from offset 5 (bytes) declared in the input message layout within the server program. Example:

```
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.



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 COBOL interface 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 interface contains 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.

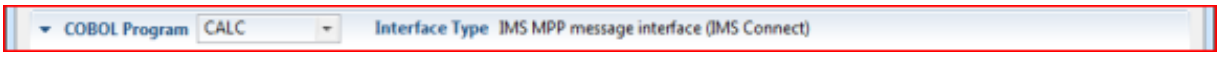
The user interface of the COBOL Mapping Editor is described below.

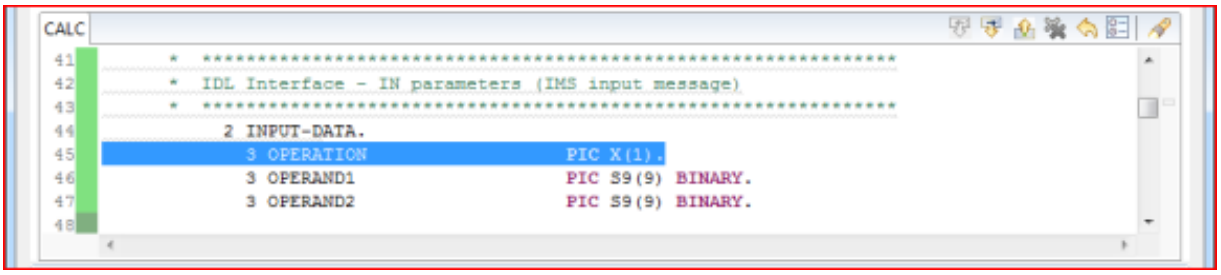
Mapping Editor User Interface

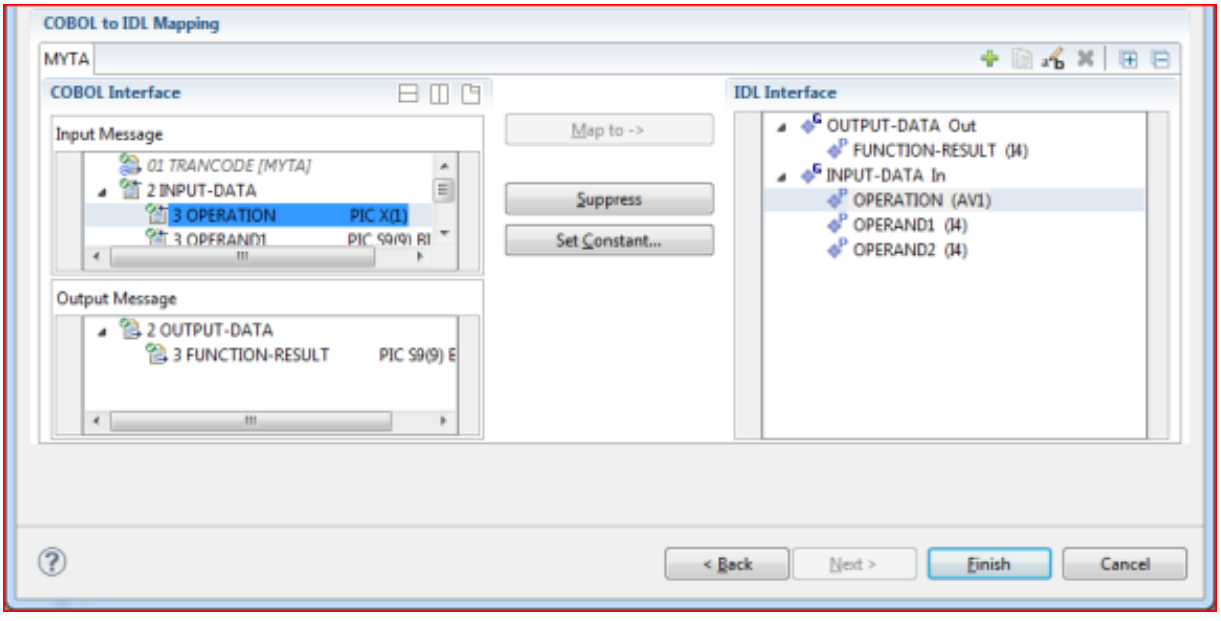
This section assumes you have set the extraction settings as described above. The following areas of the COBOL Mapping Editor user interface are described here:

- 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:

1. 

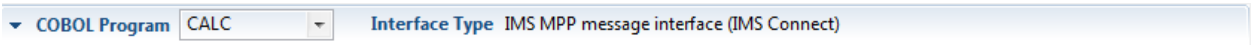
2. 

3. 

1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program

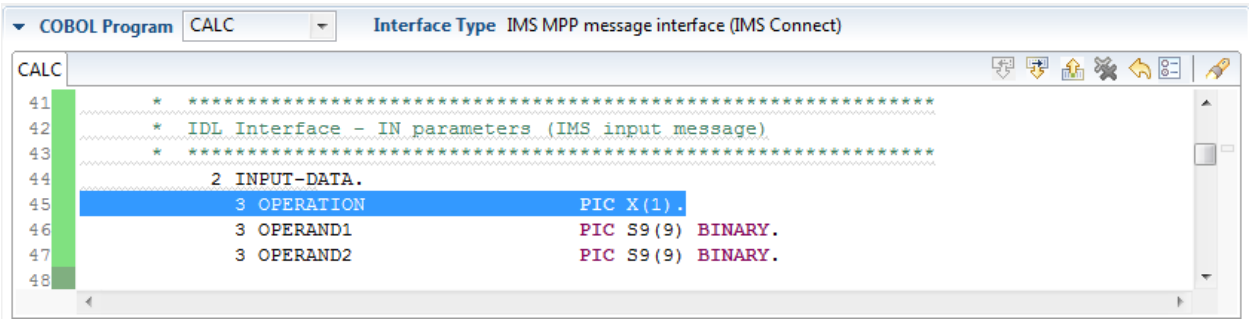
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 the 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 for selection. 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 modify COBOL Source Characteristics. Not available for interface type *COBOL Converter*.
- 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

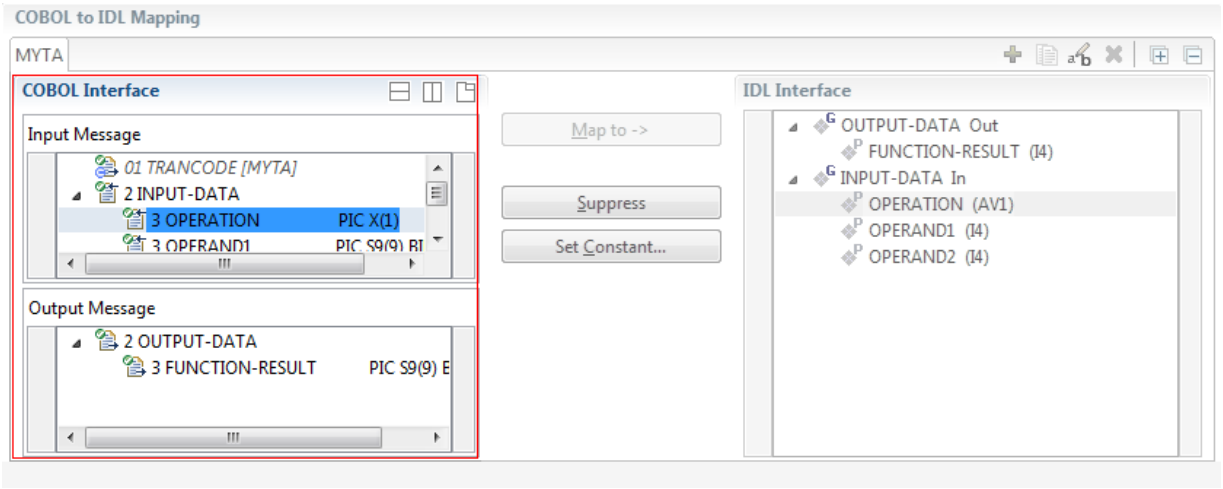
COBOL Interface

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.

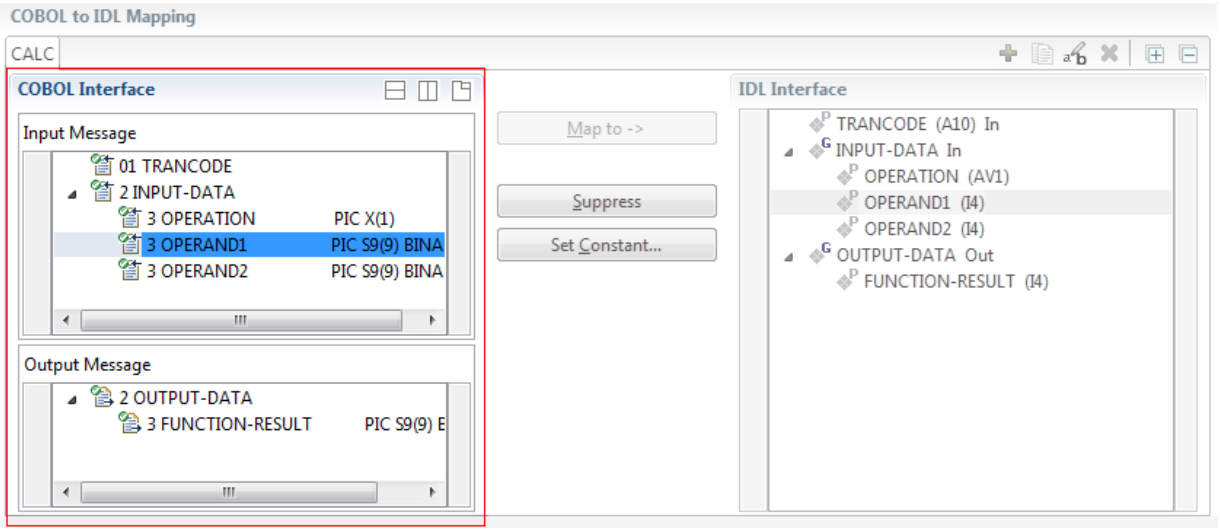
The COBOL data item names are derived from the COBOL source from which they were extracted. If your COBOL interface contains parameters without a name (for example, the keyword `FILLER` is used) those COBOL data items are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.

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.

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.
Set Multiple Possible Output (MPO) Structures	Set COBOL data items where the server program decides the output structure used on return. Specify the set of multiple possible output (MPO) structures and the criteria when a structure is used.
Map to Binary	Map a COBOL data item as IDL parameter of type binary (B*n*, BV) to exchange binary data (for example images). See ***Map to Binary and Revert Binary Mapping*** under *Mapping Editor IDL Interface Mapping Functions*.
Revert Binary Mapping	Undo the **Map to Binary** operation and use the standard mapping.
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 ***COBOL Program Selection***.

See also ***Mapping Editor IDL Interface Mapping Functions***.

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; `FILLER`s 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".

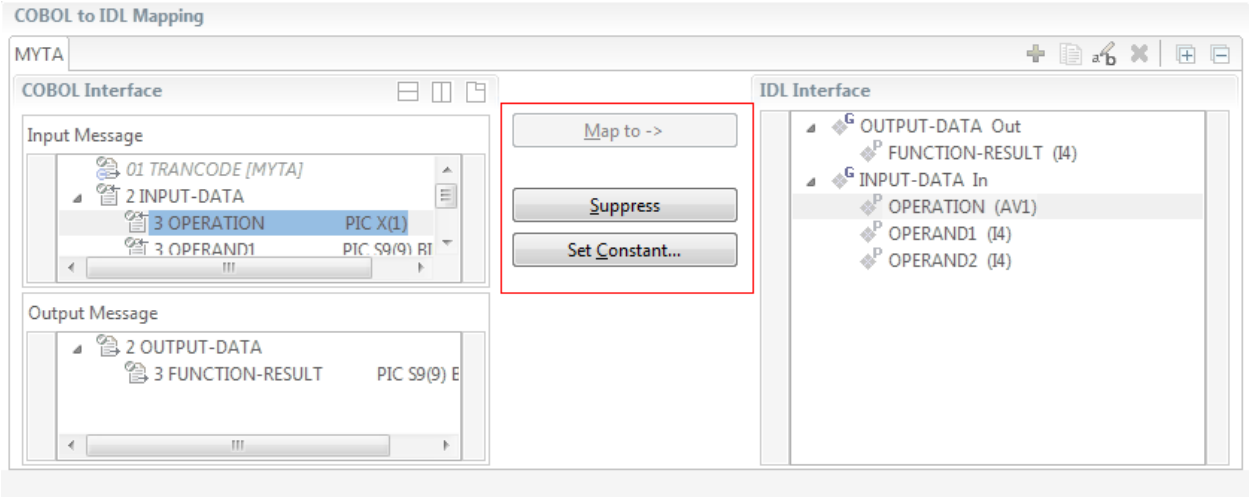
Mapping Icons

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

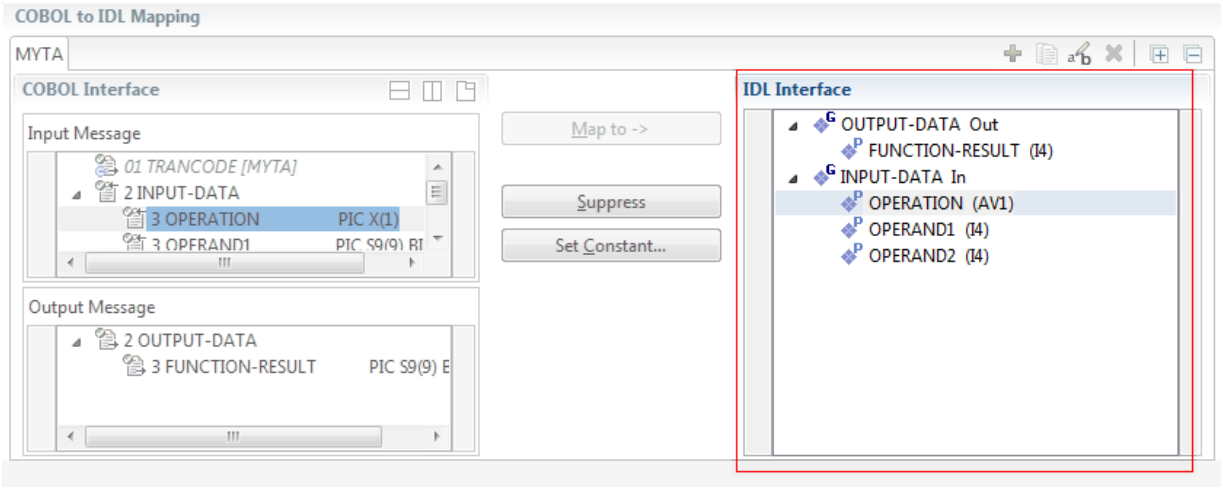
If you have mapped the COBOL interface to multiple IDL interfaces, select the IDL interface by choosing the tabs. In the **IDL Interface** tree view, a context menu is also available with the following possibilities:

- Rename the IDL parameter.
- 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*.

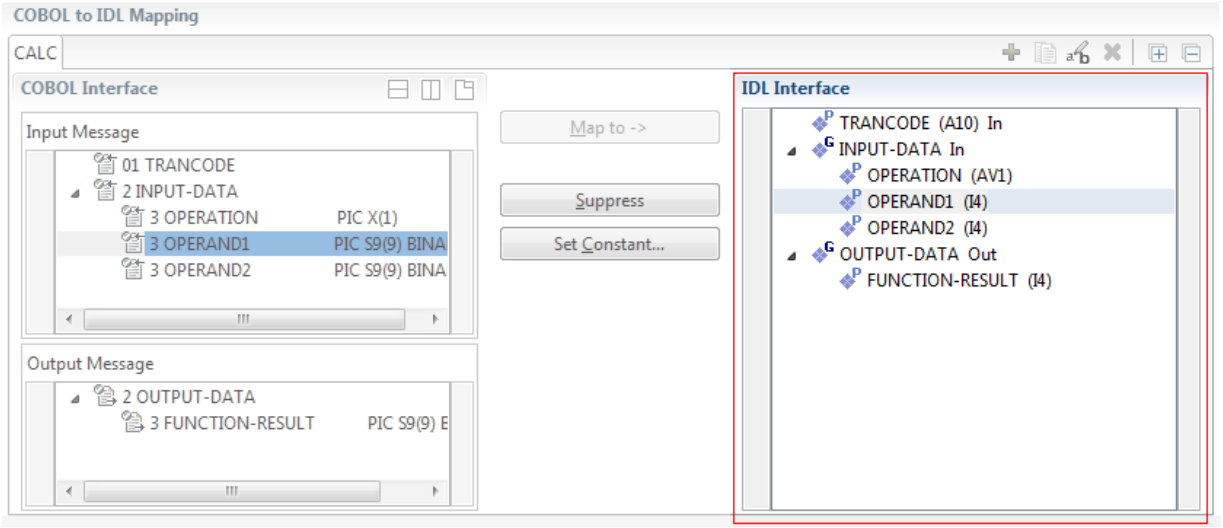
- **Fixed Value**

  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

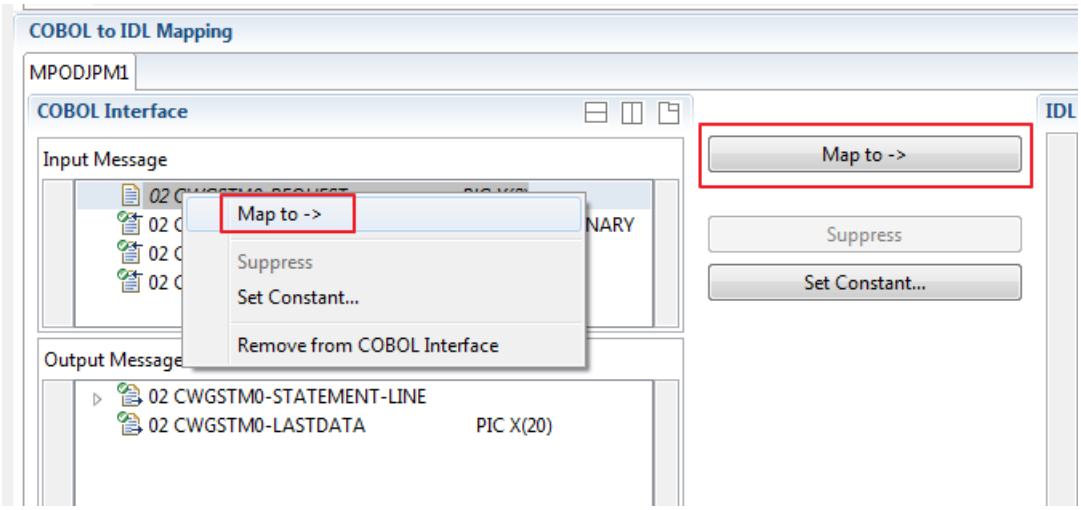
This section covers the following topics:

Map to

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 COBOL data items 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 of the COBOL interface and as mapping button to make a COBOL data item visible as an IDL parameter in the IDL interface:



2 Do the same for the output message of the COBOL interface.

Notes:

1. If a COBOL group is mapped, all subordinate 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 of the COBOL interface and as mapping button, a COBOL data item can be removed from the IDL interface.

Map OCCURS DEPENDING ON

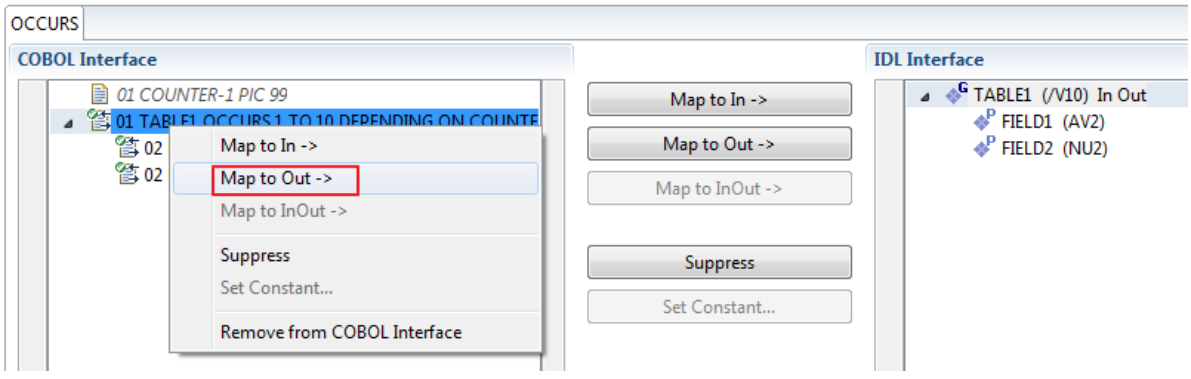
With the **Map to In**, **Out**, **InOut** functions you can make the COBOL ODO subject (here COBOL data item `TABLE`) of a variable-sized COBOL table (see *COBOL Tables with Variable Size - DEPENDING ON Clause*) visible as an IDL unbounded group (with maximum). The ODO object (here COBOL data item `COUNTER-1`) is suppressed and therefore not part of the IDL interface. This is because the number of elements of the IDL unbounded group is already implicitly available. See the following example:

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

To map `OCCURS DEPENDING ON`

1 Add the COBOL ODO subject (here data item `TABLE`) and ODO object (here data item `COUNTER-1`) to the COBOL interface. It is important both data items are in the COBOL interface.

2 Use the **Map to In**, **Out** and **InOut** functions available in the context menu of the COBOL interface and as mapping buttons and apply IDL directions for the ODO subject (data item `TABLE`):



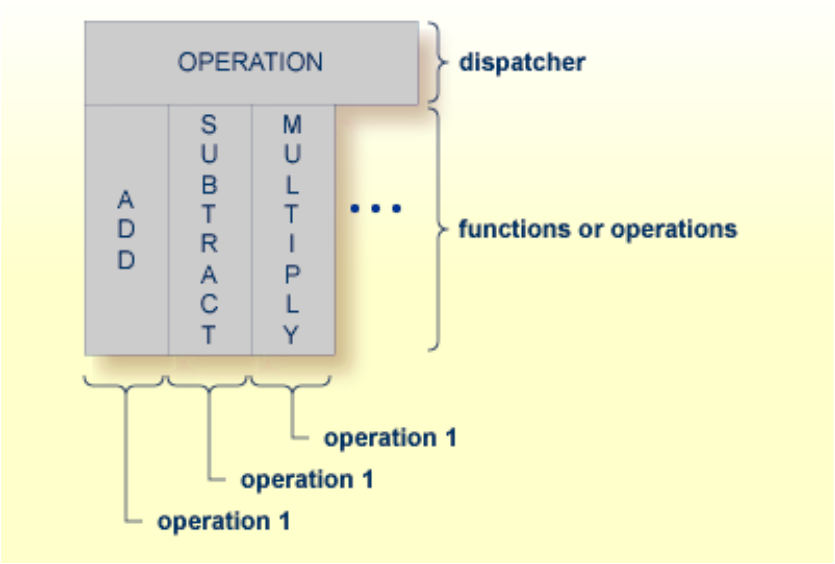
Notes:

1. The ODO subject can be mapped to the IDL interface.
2. The ODO object is always suppressed, but is required to be part of the COBOL interface.

3. IDL directions are described in the direction-attribute in `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.

Map to Multiple IDL Interfaces

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



COBOL snippet: The execution of the different functions `ADD`, `SUBTRACT`, `MULTIPLY` is controlled by the COBOL data item `OPERATION`. The contents of this decide on the function executed:

```
. . .

    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 . . .
```

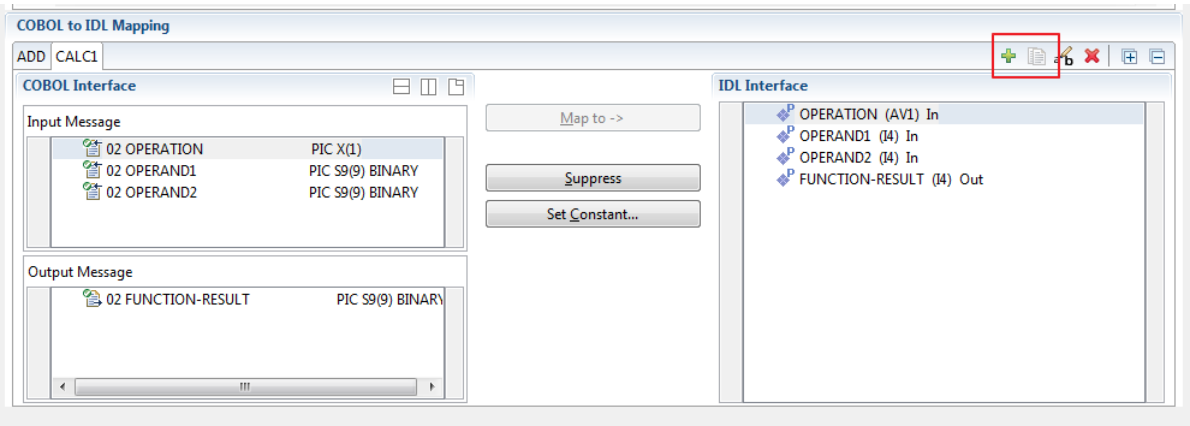
```
    END-EVALUATE.
. . .
```

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. See the following examples, depending on your target endpoint:

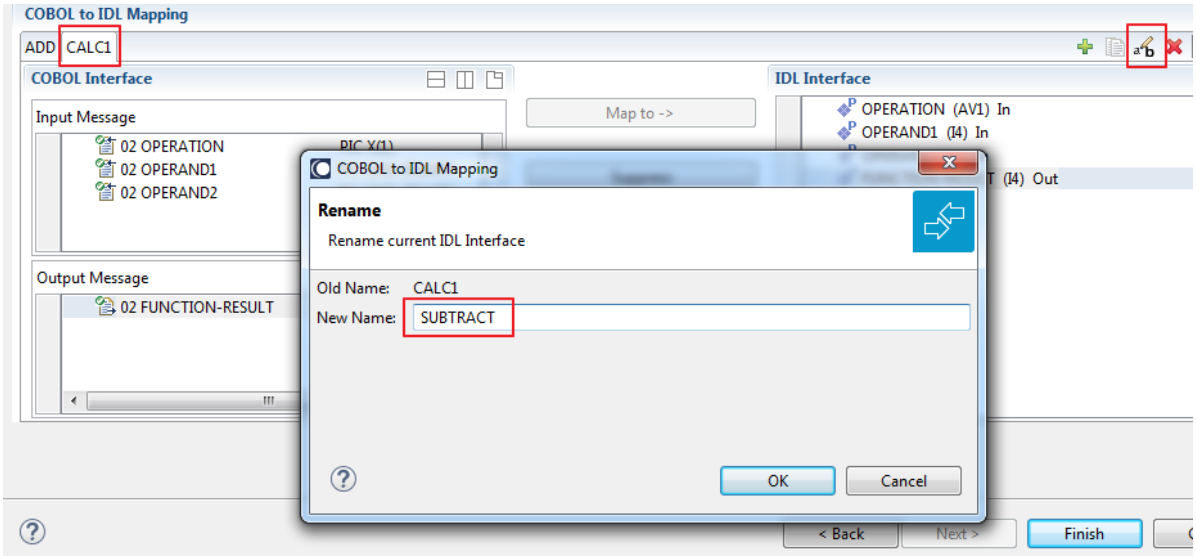
- **Integration Server**
  Instead of having a single adapter service for the *EntireX Adapter* generated with the *Integration Server Wrapper*, you have separate adapter services, one for each COBOL function.
- **Web service**
  Instead of having a Web service with a single operation generated with the *Web Services Wrapper*, you get a web service with multiple operations, one operation for each COBOL function.
- **DCOM, Java or .NET**
  Instead having a class with a single method generated with the respective wrapper (DCOM | Java | .NET) you get a class with multiple methods, one method for each COBOL function.

To map a COBOL interface to multiple IDL interfaces

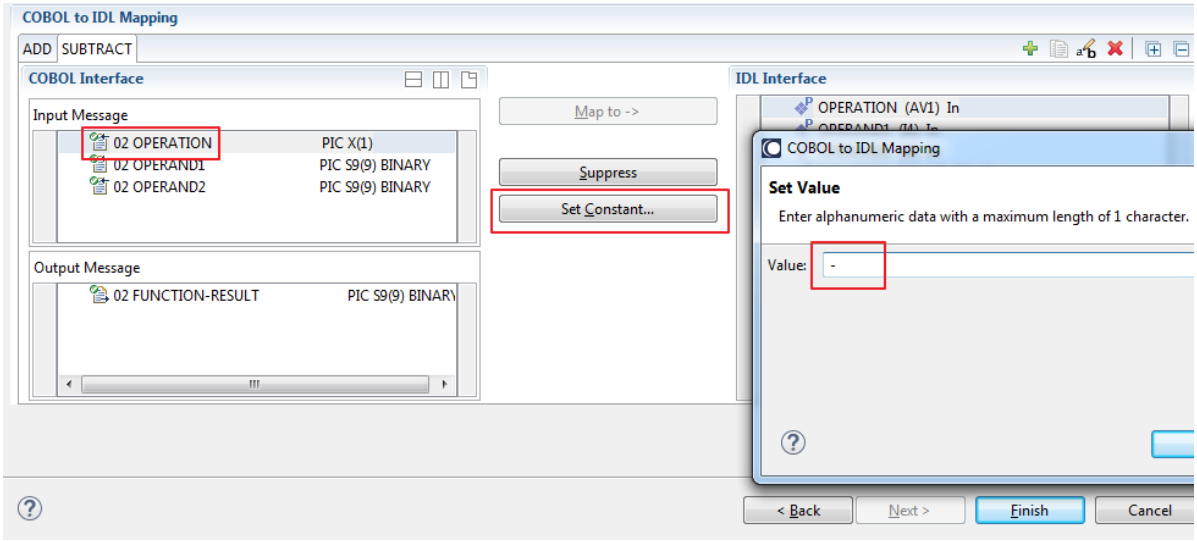
1 Select the tab with COBOL to IDL Mapping. For each function, define a separate IDL interface with the toolbar functions or :



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 suitabable names, e.g. '`ADD`', '`SUBTRACT`', '`MULTIPLY`'.
- Third, for step 3 above: Define the constants '+', '-' and '*' to the parameter `OPERATION` respectively.
- Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'EXAMPLE' is

 program 'ADD' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'SUBTRACT' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define

 program 'MULTIPLY' is
  define data parameter
   1 OPERAND1 (I4) In
   1 OPERAND2 (I4) In
   1 FUNCTION-RESULT (I4) Out
  end-define
```

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; `FILLER`s are suppressed according to your selection, see *Step 4: Define the Extraction Settings and Start Extraction*.
	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.

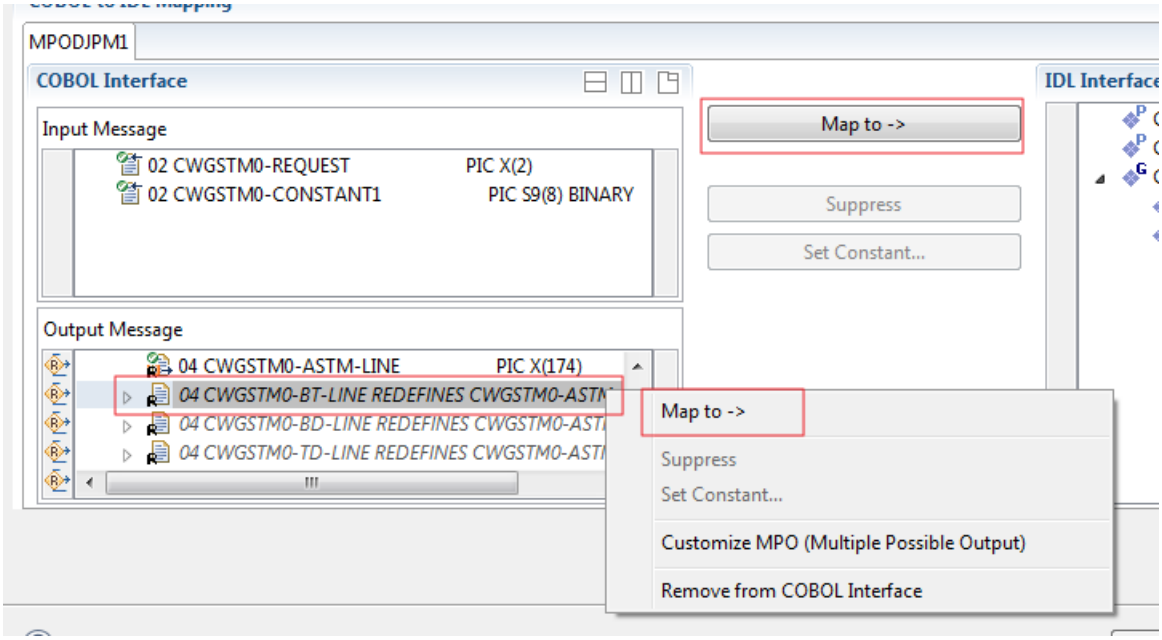
2. With the steps 1 thru 3 described here you can emulate the behavior of function Map to Operation of EntireX version 9.6 and earlier.

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 of the COBOL interface 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:

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

Suppress Unneeded COBOL Data Items

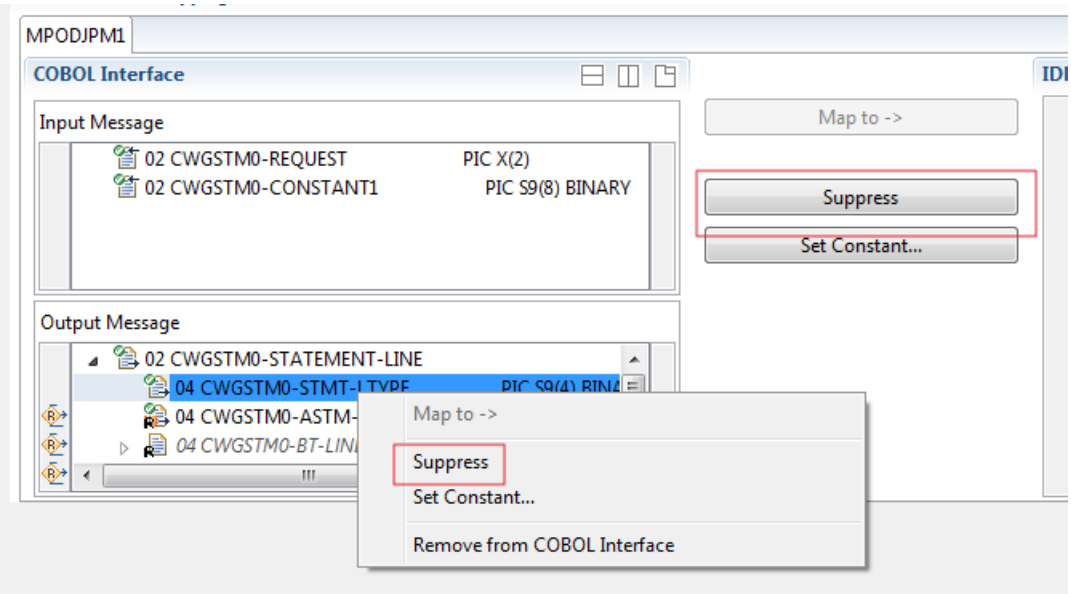
COBOL data items without any relevant information can be made invisible in the IDL interface. The IDL interface is simplified - it becomes shorter and tidier. This is useful, for example

- for `FILLER` data items
- if the consuming RPC client or IS service does not need an Out parameter
- if the COBOL data item is an In parameter and a low value can be provided

If you are using an RPC server such as the z/OS (CICS | Batch) or BS2000 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 of the COBOL interface 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 consuming RPC clients or IS services.
2. The RPC server or *EntireX Adapter* 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 subordinate COBOL data items are suppressed as well.

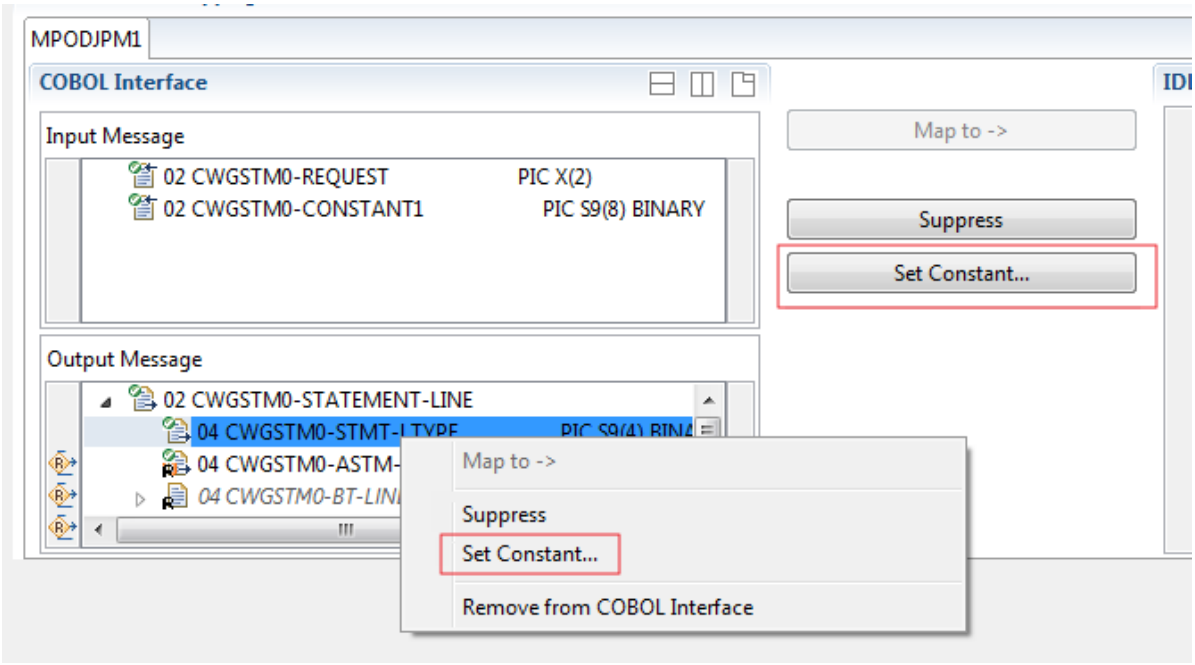
4. With the inverse function **Map to** (see above) available in the context menu of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

Set COBOL Data Items to Constants

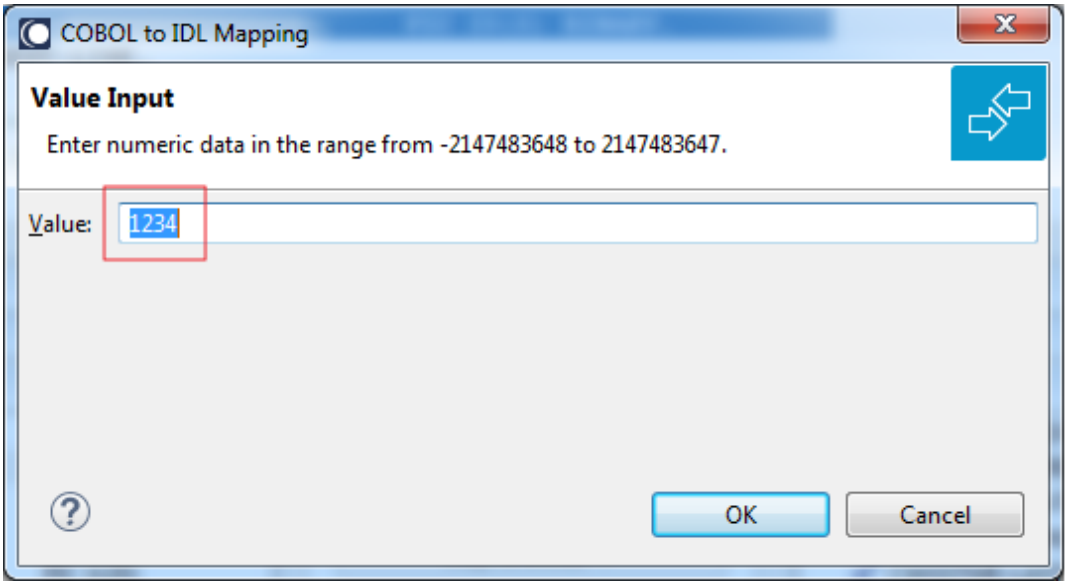
COBOL data items that always require fixed constant values on input to the COBOL interface can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. Consuming RPC clients or IS services are not bothered with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

To set COBOL data items to constants

1 Use the **Set Constant** function available in the context menu of the COBOL interface and as mapping button to define a constant value for a COBOL data item:



2 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 consuming RPC clients or IS services.
2. The RPC server or EntireX Adapter 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 of the COBOL interface and as a mapping button, a COBOL data item can be made visible in the IDL interface again.

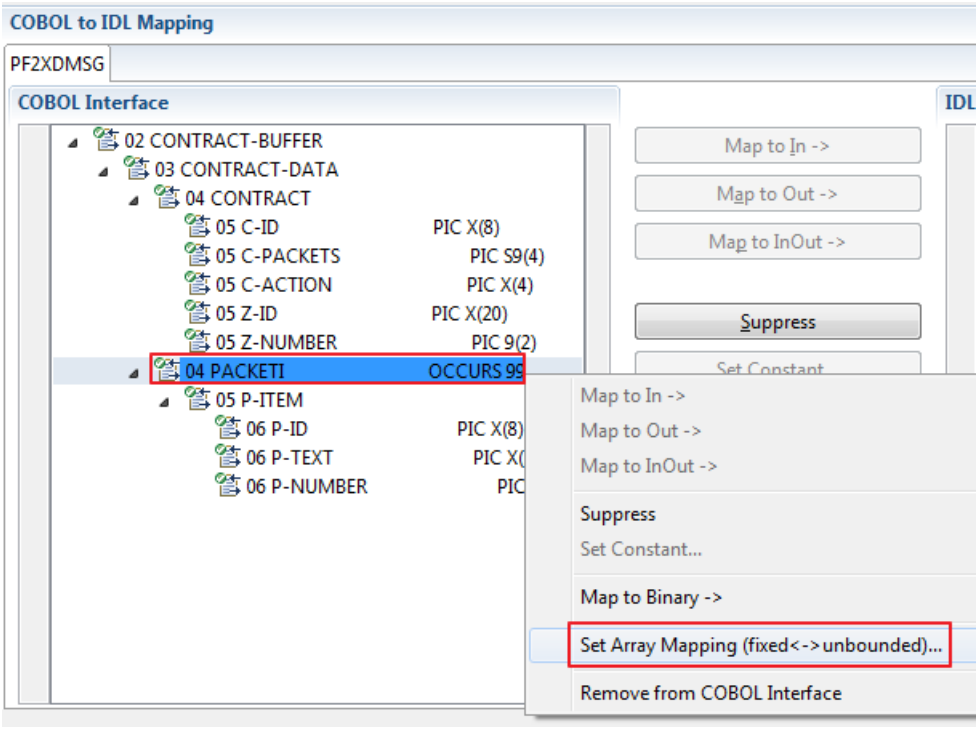
Set Arrays (Fixed <-> Unbounded)

A COBOL server defines in its interface as the last parameter a *COBOL Tables with Fixed Size* (fixed-size array). In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. Their content is undefined. The current number of elements is transferred directly or implicitly outside the array. There are multiple options to specify how the receiver calculates the number of array elements.

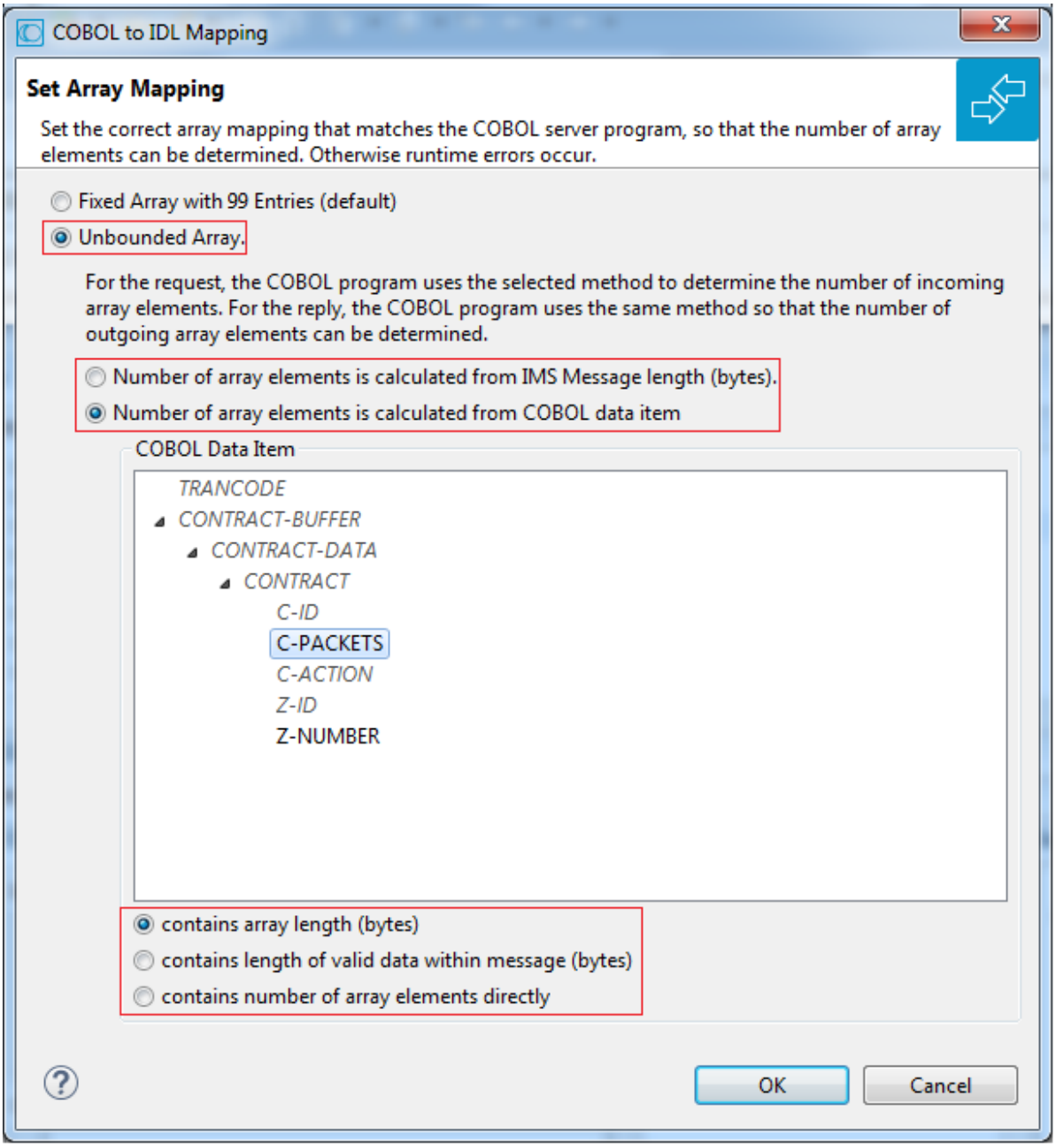
With this mapping you map the fixed-size array of the COBOL interface with the usage described above to an IDL unbounded array in the IDL interface. A consuming RPC client or IS service can use it then as any other IDL unbounded array.

To set arrays from fixed to unbounded or vice versa

1 Select the COBOL table and use the function **Set Array Mapping (fixed<->unbounded)** available in the context menu. The following window is displayed:



2 Select **Unbounded Array** and the technique for determining the number of elements.



The number of array elements is calculated using one of the following options:

- **IMS Message Length (bytes)**
  The COBOL server program inspects IMS field `LL` of the input message for the request and sets IMS field `LL` of the output message for the reply. To determine the number of array elements, the IMS message length is subtracted first to calculate the array length. The result is then divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows the reply of an IMS message. It assumes `OUTPUT-CONTRACT` with fixed array `PACKETI` is the IMS output message.

```
       WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
        01 OUTPUT-MESSAGE.
          02 OUTPUT-IMS-META.
            03 OUTPUT-LL                         PIC S9(3) BINARY.
            03 OUTPUT-ZZ                         PIC S9(3) BINARY.
         02 OUTPUT-CONTRACT.
           04 CONTRACT.
            05 C-ID                              PIC X(8).
            05 C-ACTION                          PIC X(4).
           04 ZONE.
            05 Z-ID                              PIC X(20).
            05 Z-NUMBER                          PIC 9(2).
           04 PACKETI                            OCCURS 99.
            05 P-ITEM.
             06 P-ID                             PIC X(8).
             06 P-TEXT                           PIC X(30).
             06 P-NUMBER                         PIC 9(2).


      . . .
     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set IMS output message length depending on number of elements
           COMPUTE OUTPUT-LL =
                  (LENGTH OF P-ID IN OUTPUT-MESSAGE +
                   LENGTH OF P-TEXT IN OUTPUT-MESSAGE +
                   LENGTH OF P-NUMBER IN OUTPUT-MESSAGE) * II.

           ADD LENGTH OF CONTRACT IN OUTPUT-MESSAGE TO OUTPUT-LL.
           ADD LENGTH OF ZONE     IN OUTPUT-MESSAGE TO OUTPUT-LL.

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

- **COBOL data item contains array length (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. This COBOL data item contains the array length. To determine the number of array elements, the contents of the COBOL data item are divided by the length of one array element. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The length of the fixed-size array `PACKETI` is contained in COBOL data item `C-BYTES`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-BYTES                            PIC S9(4).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
            05 Z-ID                               PIC X(20).
           04 PACKETI                             OCCURS 99.
            05 P-ITEM.
             06 P-ID                              PIC X(8).
             06 P-TEXT                            PIC X(30).
             06 P-NUMBER                          PIC 9(2).
      . . .

     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set table length
        COMPUTE C-BYTES = (LENGTH OF P-ITEM) * II.
```

- **COBOL data item contains length of valid data within messages (bytes)**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. To determine the number of array elements, the contents of the COBOL data item are subtracted first to calculate the array length. The result is then divided by the length of one array element. The length of the transferred application data within the message can be shorter than the respective message length. All lengths are in bytes. The following COBOL snippet shows how the COBOL interface `CONTRACT` is filled by the COBOL server on reply. COBOL data item `C-APPDATA` contains the length of the valid data of the reply message. The number of array elements of the fixed-size array `PACKETI` is implicitly contained in COBOL data item `C-APPDATA`.

```
       WORKING-STORAGE SECTION.
         77 II                                    PIC S9(4).
         77 EPARM                                 PIC  9(2).
         77 EPARM2                                PIC  9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-APPDATA                          PIC S9(4).
            05 C-ACTION                           PIC X(4).
            05 Z-ID                               PIC X(20).
            05 Z-NUMBER                           PIC 9(2).
           04 PACKETI                             OCCURS 99.
            05 P-ITEM.
             06 P-ID                              PIC X(8).
             06 P-TEXT                            PIC X(30).
             06 P-NUMBER                          PIC 9(2).
       . . .
     *  Fill variable output array
        MOVE 0 TO II.
        PERFORM RANDOMNUM TIMES
           ADD 1 TO II
           MOVE ... TO P-ID    (II)
           MOVE ... TO P-TEXT  (II)
           MOVE ... TO P-NUMBER(II)
        END-PERFORM.
     *  Set length
        COMPUTE C-APPDATA = (LENGTH OF P-ITEM) * II
                          +  LENGTH OF CONTRACT.
```

- **COBOL data item contains number of array elements directly**
  The COBOL server program inspects a numeric COBOL data item (`ZONED`, `PACKED` or `BINARY COBOL` type) for the request and sets it accordingly for the reply. The content of the COBOL data item is the number of array elements. The following COBOL snippet shows how the COBOL interface `CONTRACT-DATA` is filled by the COBOL server on reply. The number of array elements of the fixed-size array `PACKETI` is directly contained in COBOL data item `C-NUM`.

```
      WORKING-STORAGE SECTION.
        77 II                                    PIC S9(4).
      . . .
      LINKAGE SECTION.
        01 DFHCOMMAREA.
          03 CONTRACT-DATA.
           04 CONTRACT.
            05 C-ID                               PIC X(8).
            05 C-NUM                              PIC S9(4).
            05 C-ACTION                           PIC X(4).
           04 ZONE.
            05 Z-NUMBER                           PIC 9(2).
```

```
          05 Z-ID                                 PIC X(20).
         04 PACKETI                               OCCURS 99.
          05 P-ITEM.
           06 P-ID                                PIC X(8).
           06 P-TEXT                              PIC X(30).
           06 P-NUMBER                            PIC 9(2).
    . . .

*   Fill variable output array
    MOVE 0 TO II.
    PERFORM RANDOMNUM TIMES
       ADD 1 TO II
       MOVE ... TO P-ID    (II)
       MOVE ... TO P-TEXT  (II)
       MOVE ... TO P-NUMBER(II)
    END-PERFORM.
*   Set occurrences
    MOVE II TO C-NUM.
```

Press **OK** to change the IDL array parameter from fixed array /number to an unbounded array /Vnumber. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. If a COBOL data item is used, it will be set to suppressed because it is superfluous for RPC clients.

See *Suppress Unneeded COBOL Data Items*.

Notes:

1. This option should be used carefully and requires knowledge of the COBOL interface. Be aware that an incorrect mapping results in runtime errors.
2. The COBOL table 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 ***COBOL Tables with Variable Size - `DEPENDING ON` Clause***).
3. If a COBOL data item is used, it must be physically located before the COBOL table. The IDL directions must also match.

Set Multiple Possible Output (MPO) Structures

A COBOL server program produces more than one type of output. The layout of the output can therefore take two or more dissimilar shapes. The COBOL server program decides at runtime the output structure returned, that is, the COBOL layout on output varies.

- Multiple Possible Output with REDEFINES
- Optional Output with Groups
- Complex MPO Selections
- MPO Terminology

- Steps

Multiple Possible Output with REDEFINES

A COBOL *REDEFINES Clause* is often used to describe the possible output structures. In COBOL this is the standard way to describe multiple possible output:

Similar to COBOL data item PAYMENT-DATA in the example below; for this purpose, PAYMENT-DATA is redefined; each redefinition represents an output structure (MPO case); on return exactly one output structure is used; by inspecting COBOL data item PAYMENT-TYPE (MPO selector) first, a caller can determine the returned output structure; the caller then uses the correct redefinition to access the data.

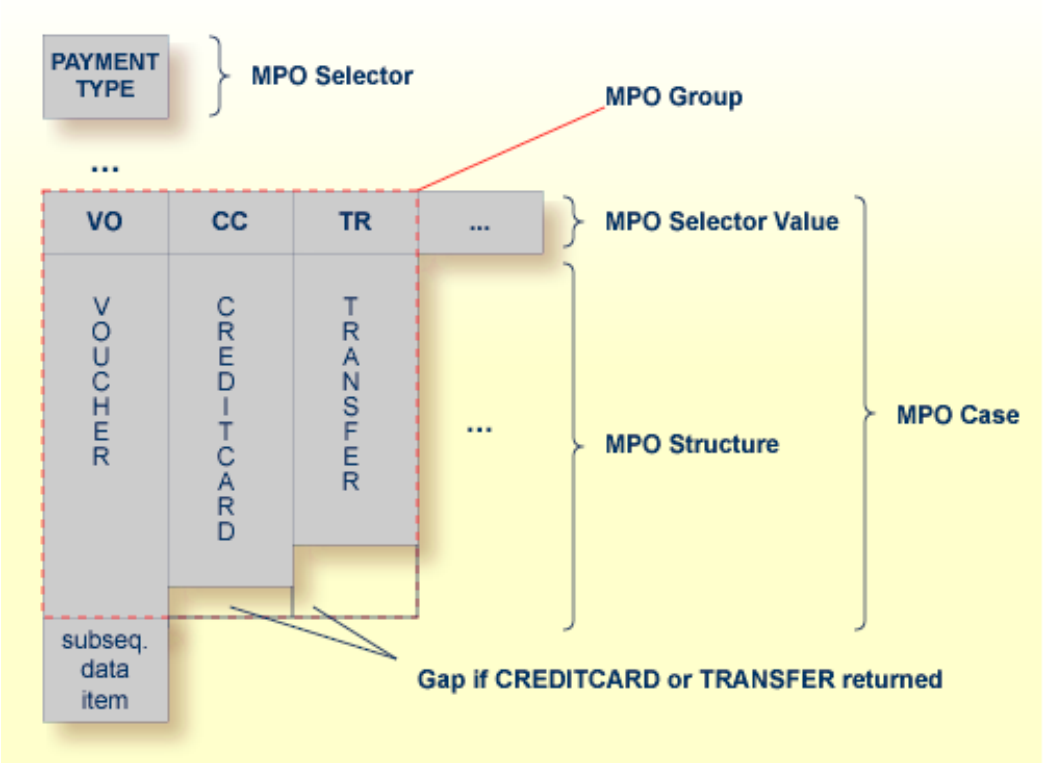
```
. . .
  01 INPUT-DATA.
     02 ORDER-NUMBER                          PIC 9(10).

  . . .
  01 OUTPUT-DATA.
      02 <some fields>                        PIC <clause>.
     . . .

      02 PAYMENT-TYPE                         PIC X(2).
         88 PAYMENT-TYPE-VOUCHER              VALUE "VO".
         88 PAYMENT-TYPE-CREDITCARD           VALUE "CC".
         88 PAYMENT-TYPE-TRANSFER             VALUE "TR".
         88 PAYMENT-TYPE-DIRECTDEBIT          VALUE "DB".
       . . .
      02 <preceding data items>               PIC <clause>.
     . . .
      02 PAYMENT-DATA                         PIC X(256).
      02 PAYMENT-DATA-VOUCHER     REDEFINES PAYMENT-DATA.
         04 VOUCHER-ORIGIN                    PIC X(128).
         04 VOUCHER-SERIES                    PIC X(128).
      02 PAYMENT-DATA-CREDITCARD  REDEFINES PAYMENT-DATA.
         04 CREDITCARD-NUMBER                 PIC 9(18).
         04 CREDITCARD-COMPANY                PIC X(128).
         04 CREDITCARD-CODE                   PIC 9(12).
         04 CREDITCARD-VALIDITY               PIC X(8).
      02 PAYMENT-DATA-TRANSFER    REDEFINES PAYMENT-DATA.
         04 TRANSFER-NAME                     PIC X(128).
         04 TRANSFER-IBAN                     PIC X(34).
         04 TRANSFER-BIC                      PIC X(11).
      02 PAYMENT-DATA-DIRECTDEBIT REDEFINES PAYMENT-DATA.
         04 DIRECTDEBIT-IBAN                  PIC X(34).
         04 DIRECTDEBIT-NAME                  PIC X(128).
         04 DIRECTDEBIT-EXPIRES               PIC 9(8).
       . . .
      02 <subsequent data items>              PIC <clause>.
     . . .
```

```
    . . .

*   read order record using ORDER-NUMBER
    . . .

*  set value indicating type of reply (MPO selector)
    IF <some-condition> THEN
      SET PAYMENT-TYPE-VOUCHER TO TRUE
    ELSE IF <some-other-condition> THEN
      SET PAYMENT-TYPE-CREDITCARD TO TRUE
    ELSE IF <some-further-condition> THEN
      SET PAYMENT-TYPE-TRANSFER TO TRUE
    ELSE
      SET PAYMENT-TYPE-DIRECTDEBIT TO TRUE
    END-IF.
   . . .

*   set fields (MPO case) depending on type of reply
    INITIALIZE PAYMENT-DATA.
    EVALUATE TRUE
         WHEN PAYMENT-TYPE-VOUCHER
            MOVE . . . TO VOUCHER-ORIGIN
            MOVE . . . TO VOUCHER-SERIES
         WHEN PAYMENT-TYPE-CREDITCARD
            MOVE . . . TO CREDITCARD-NUMBER
            MOVE . . . TO CREDITCARD-CODE
            MOVE . . . TO CREDITCARD-VALIDITY
         WHEN PAYMENT-TYPE-TRANSFER
            MOVE . . . TO TRANSFER-NAME
            MOVE . . . TO TRANSFER-IBAN
            MOVE . . . TO TRANSFER-BIC
         WHEN PAYMENT-TYPE-DIRECTDEBIT
            MOVE . . . TO DIRECTDEBIT-IBAN
            MOVE . . . TO DIRECTDEBIT-NAME
            MOVE . . . TO DIRECTDEBIT-EXPIRES
         WHEN
            . . .
    END-EVALUATE.
    . . .
```

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all possible output structures, that is, they are known during extraction. In the example these are the structures `PAYMENT-DATA-VOUCHER`, `PAYMENT-DATA-CREDITCARD` and `PAYMENT-DATA-TRANSFER`. These are the MPO structures.
- contains an additional COBOL data item carrying a value related to the returned output structure. By inspecting this data item first, the appropriate output structure can be selected to address the data correctly. In the example it is `PAYMENT-TYPE`. This item is the MPO selector.

- always occupies memory to be able to transfer the longest output structure. If the actual returned output structure is shorter than the longest possible output structure, there is a gap (space) between the multiple possible output and the subsequent data item.



This abstract concept is known as *multiple possible output* (MPO) EntireX bundles all MPO structures into an MPO group. See *MPO Terminology* below.

Optional Output with Groups

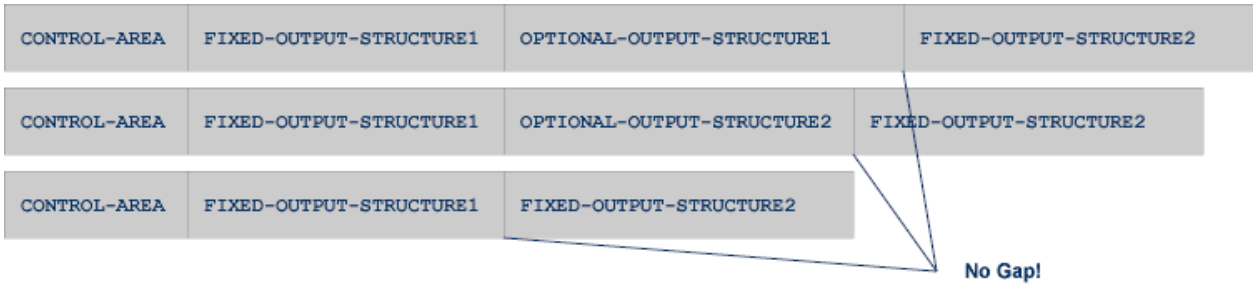
COBOL group data items can be used to describe optional output structures. The contents of a COBOL data item define under which circumstances COBOL groups are part of the returned data or not. Optional output with group data items are a variant of multiple possible output (MPO).

In addition, the COBOL interface

- limits the number of possible output structures returned
- defines all optional output structures, that is, they are known during extraction. In the COBOL snippet below these are the structures `OPTIONAL-OUTPUT-STRUCTURE1` and `OPTIONAL-OUTPUT-STRUCTURE2`. These are the MPO structures.
- contains an additional COBOL data item carrying an indication which optional output is present. By inspecting this data item first, the appropriate optional output structure can be selected to address the data correctly. If its value does not match, the optional output is not present. In the COBOL snippet it is COBOL data item `OPTIONAL-OUTPUT`. This item is the MPO selector.

- If the optional output is not present no memory is occupied. There is no gap between the optional output and the subsequent data item, as opposed to *Multiple Possible Output with* `REDEFINES` above.

In the COBOL snippet below there are three different shapes of output:



COBOL snippet:

```
WORKING-STORAGE SECTION.

   01 INPUT-AREA.
      02 FIX-INPUT-ITEM1                    PIC X(4).
      02 <some fields>                      PIC <clause>.
      . . .

   01 OUTPUT-OFFSET                         PIC  S9(9) BINARY.
   01 OUTPUT-AREA                           PIC X(32000).
      . . .

   01 CONTROL-AREA.
      02 OPTIONAL-OUTPUT                    PIC X(1).
         88 OPTIONAL-OUTPUT-1               VALUE "1".
         88 OPTIONAL-OUTPUT-2               VALUE "2".
         88 OPTIONAL-OUTPUT-NONE            VALUE "N".
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE1.
      02 OPTIONAL-OUTPUT-ITEM11             PIC X(10).
      02 OPTIONAL-OUTPUT-ITEM12             PIC X(100).
      02 OPTIONAL-OUTPUT-ITEM13             PIC X(20).
      . . .

   01 OPTIONAL-OUTPUT-STRUCTURE2.
      02 OPTIONAL-OUTPUT-ITEM21             PIC X(4).
      02 OPTIONAL-OUTPUT-ITEM22             PIC X(50).
      02 OPTIONAL-OUTPUT-ITEM23             PIC X(50).
      . . .

   01 FIX-OUTPUT-STRUCTURE1.
      02 FIX-OUTPUT-ITEM11                  PIC X(4).
      02 FIX-OUTPUT-ITEM12                  PIC X(20).
```

```
      02 FIX-OUTPUT-ITEM13                    PIC X(8).
      . . .

   01 FIX-OUTPUT-STRUCTURE2.
      02 FIX-OUTPUT-ITEM21                    PIC X(2).
      02 FIX-OUTPUT-ITEM22                    PIC X(10).
      02 FIX-OUTPUT-ITEM23                    PIC X(10).
      . . .


      IF <some-condition> THEN
         SET OPTIONAL-OUTPUT-1 TO TRUE
      ELSE IF <some-other-condition> THEN
         SET OPTIONAL-OUTPUT-2 TO TRUE
      ELSE
         SET OPTIONAL-OUTPUT-NONE TO TRUE
      END-IF.

      . . .

*     provide control area for optional output
      MOVE 1 TO OUTPUT-OFFSET.
      STRING CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide data items before optional output
      STRING FIX CONTROL-AREA DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.

*     provide optional output
      EVALUATE TRUE
          WHEN OPTIONAL-OUTPUT-1
             STRING OPTIONAL-OUTPUT-STRUCTURE1 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
          WHEN OPTIONAL-OUTPUT-2
             STRING OPTIONAL-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
             INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET
      END-EVALUATE.

*     provide data items after optional output
      STRING FIX-OUTPUT-STRUCTURE2 DELIMITED BY SIZE
      INTO OUTPUT-AREA WITH POINTER OUTPUT-OFFSET.
      . . .
```

The returned data is built by copying the necessary COBOL structures into an output area. The optional output is one of `OPTIONAL-OUTPUT-STRUCTURE1`, `OPTIONAL-OUTPUT-STRUCTURE2` or nothing. The presence of the optional output is controlled by a structure named `CONTROL-AREA`.

Complex MPO Selections

If the MPO case detection is complicated and cannot be defined by available Extractor features (for example the MPO selector and its values), perform the following steps:

To map a complex MPO selection

1 Map the complete MPO group to binary. See *Map to Binary and Revert Binary Mapping*.

> **Note:** If an MPO group is already defined, you cannot map it to binary. Decide first whether MPO case detection is covered by available extractor features.

2 Implement MPO case detection in your RPC client, using the binary mapping from step 1.

3 Implement MPO case parsing in your RPC client, using the binary mapping from step 1. For the *EntireX Adapter*, use the COBOL Converter for this purpose. See *Converting IS Data Structures with the COBOL Converter* in the EntireX Adapter documentation.

MPO Terminology

The following terminology is used with MPOs:

MPO structure
A COBOL group describing the output layout used in an MPO case. All alternative layouts in an MPO group are often described with COBOL `REDEFINEs`.

MPO group
Bundles together all MPO structures that can be used alternatively. A COBOL interface can contain more than one MPO group.

MPO case
An MPO structure together with its MPO selector values (one or more).

MPO selector
A COBOL data item containing a specific value (MPO selector value) where the actual MPO case can be determined.

- For MPOs based on `REDEFINEs`, the MPO selector can be placed before, inside or after the MPO group.
- For optional output with groups, the MPO selector precedes the MPO group and is located outside the MPO group.
- Only for MPP Message Interface (IMS Connect): Instead of determining the position of the MPO selector from beginning of the message, you can calculate the position using a *fixed offset starting from the end of the message*. This alternative is limited to one MPO group per program. See check box **MPO Selector determined from message end** in step *Create a new MPO group* below.

MPO selector value

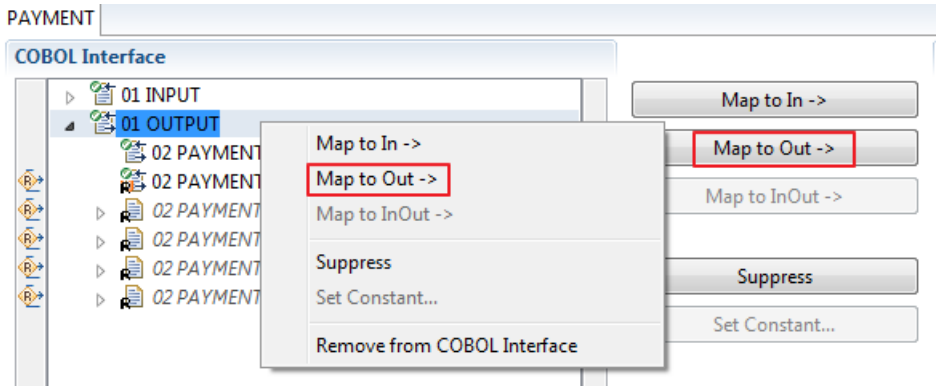
Each value indicates exactly one output structure. An output structure can be indicated by further values.

Steps

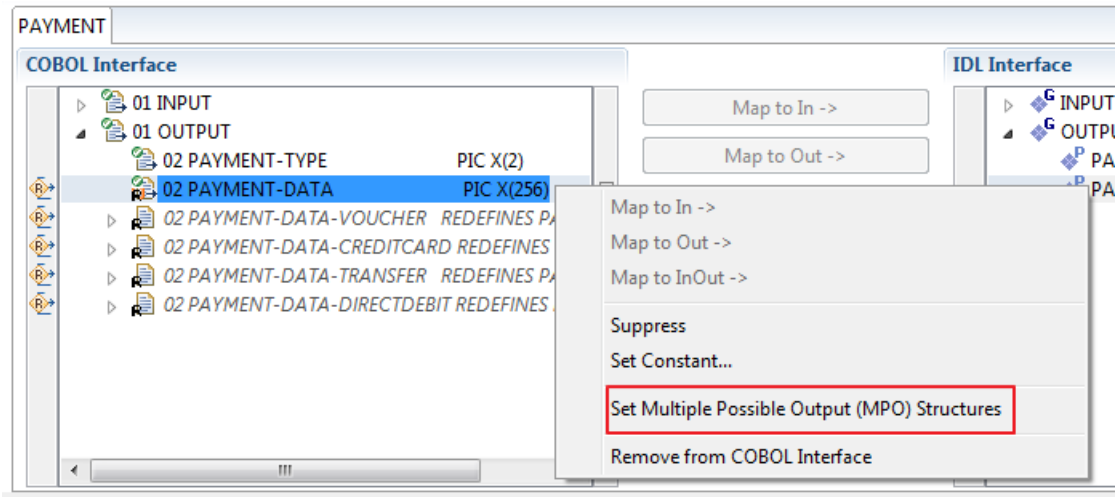
To set multiple possible output (MPO) structures with `REDEFINES` or groups

Use the **Set Multiple Possible Output (MPO) Structures** function available in the context menu of the COBOL interface to create new or modify existing MPO groups.

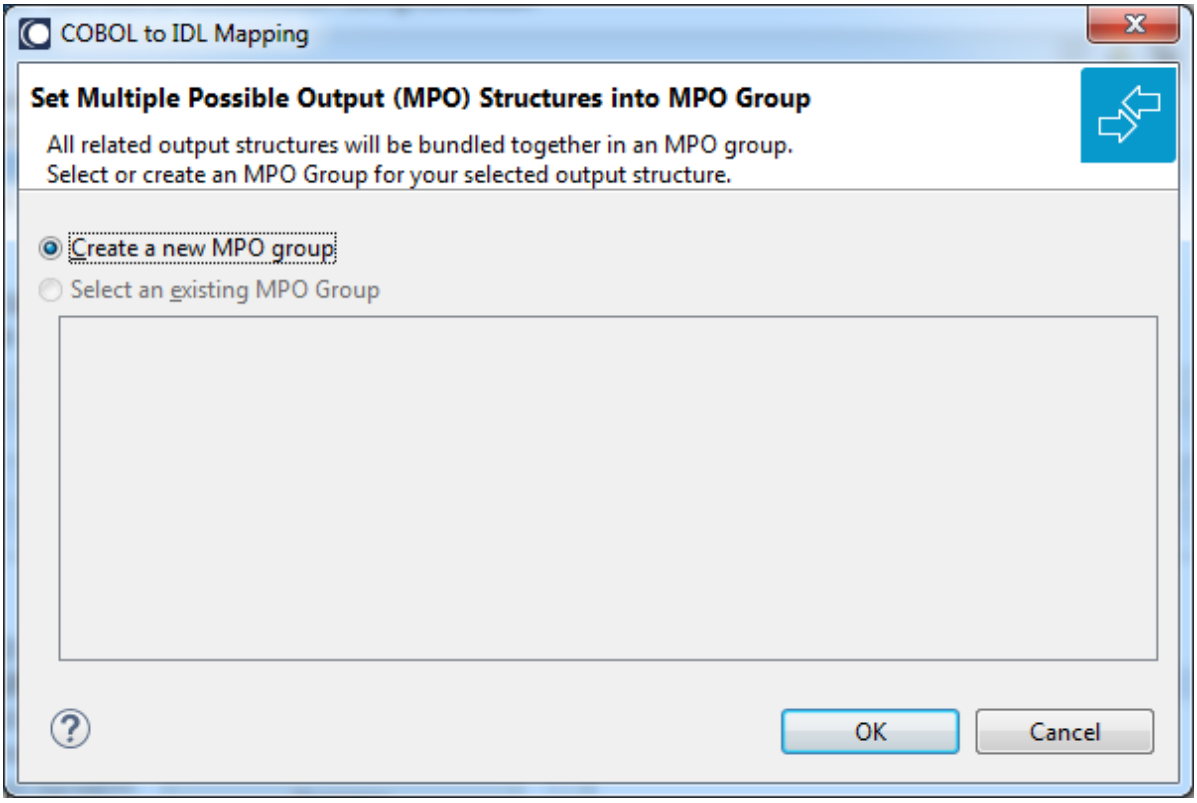
1 Set the top-level COBOL data item where the MPO structures are contained to IDL direction Out. Use the **Map to Out** function for this purpose:



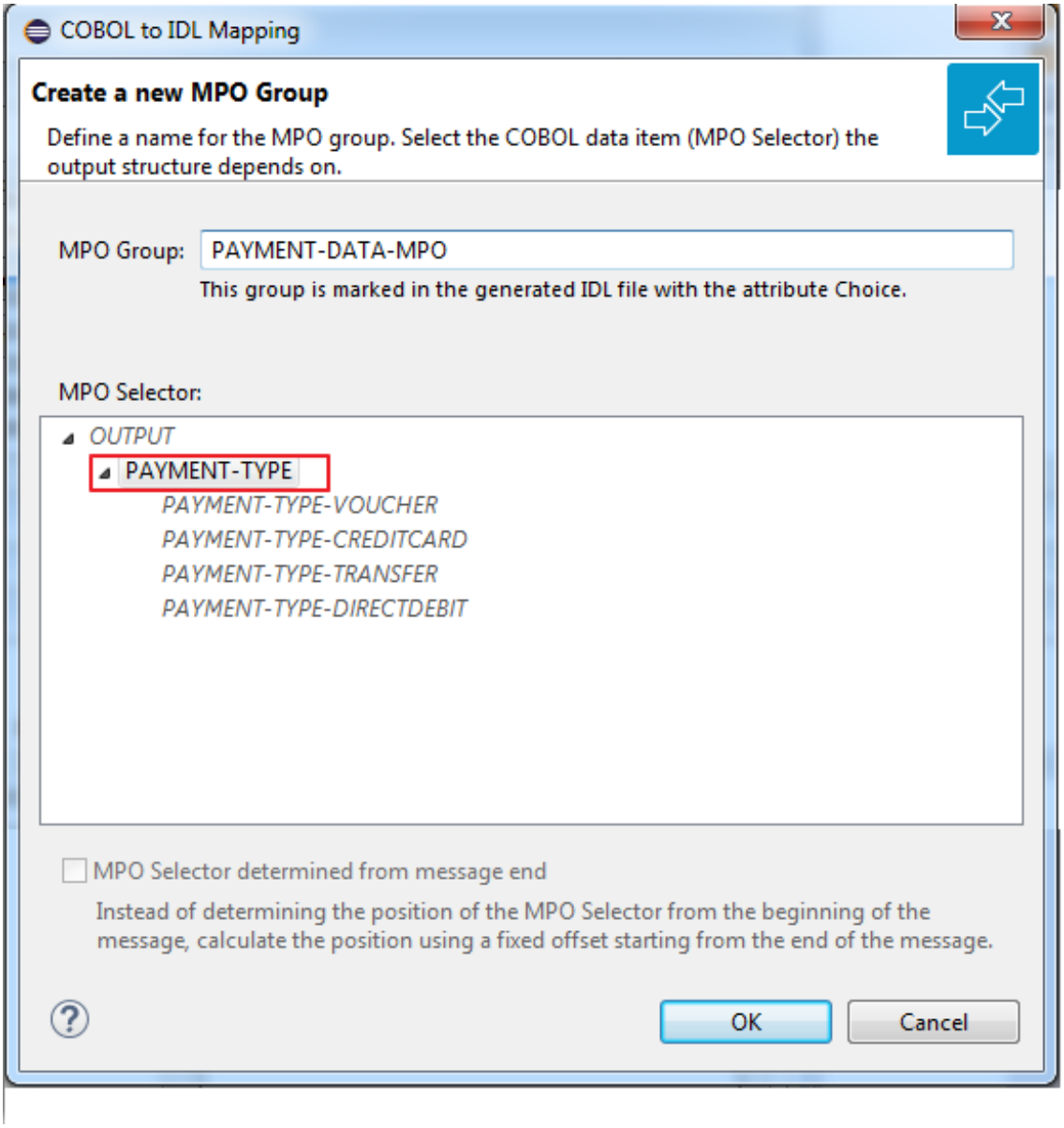
2 From the context menu of the COBOL interface of the COBOL `REDEFINE`, choose **Set Multiple Possible Output (MPO) Structures**.



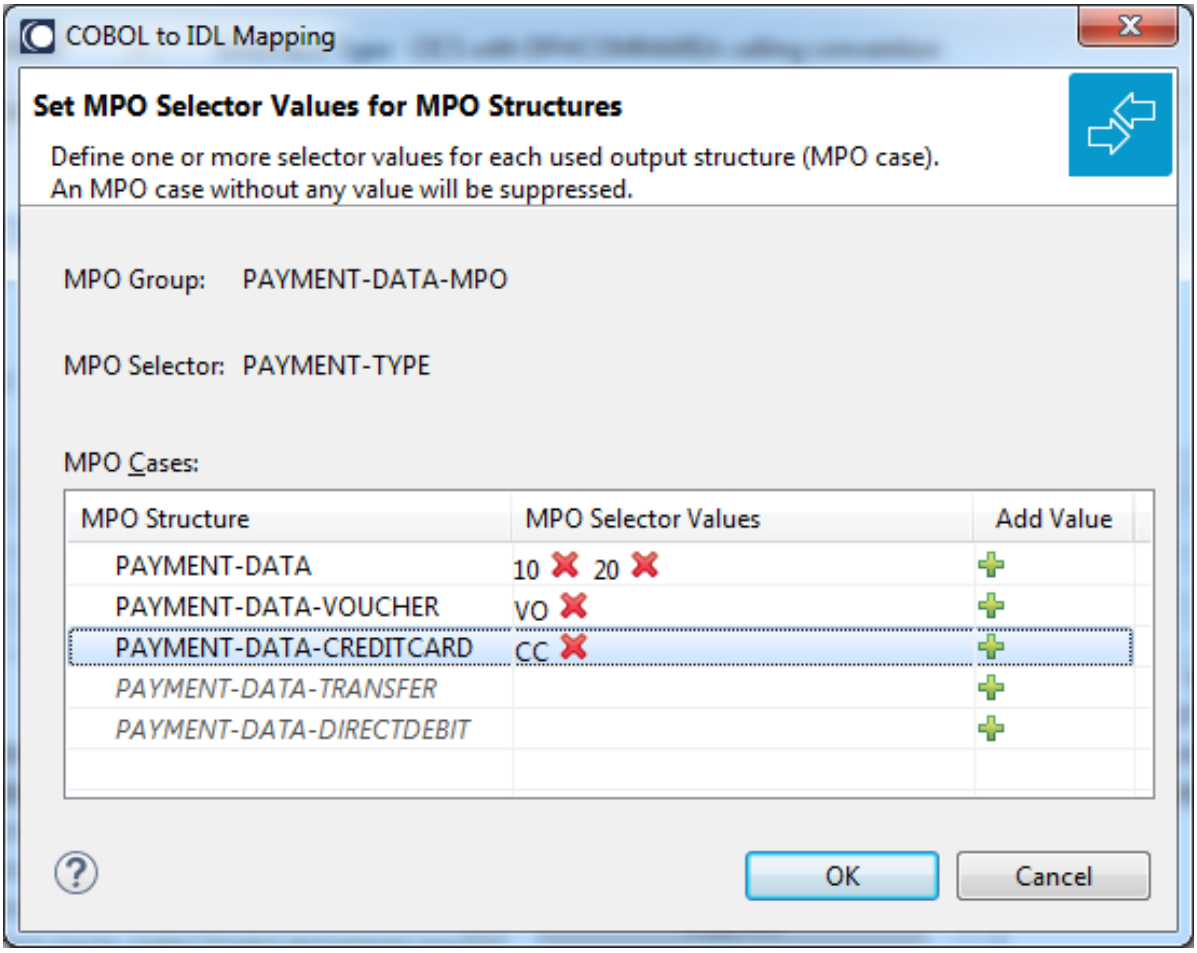
3 Set Multiple Possible Output (MPO) Structures into MPO Group.



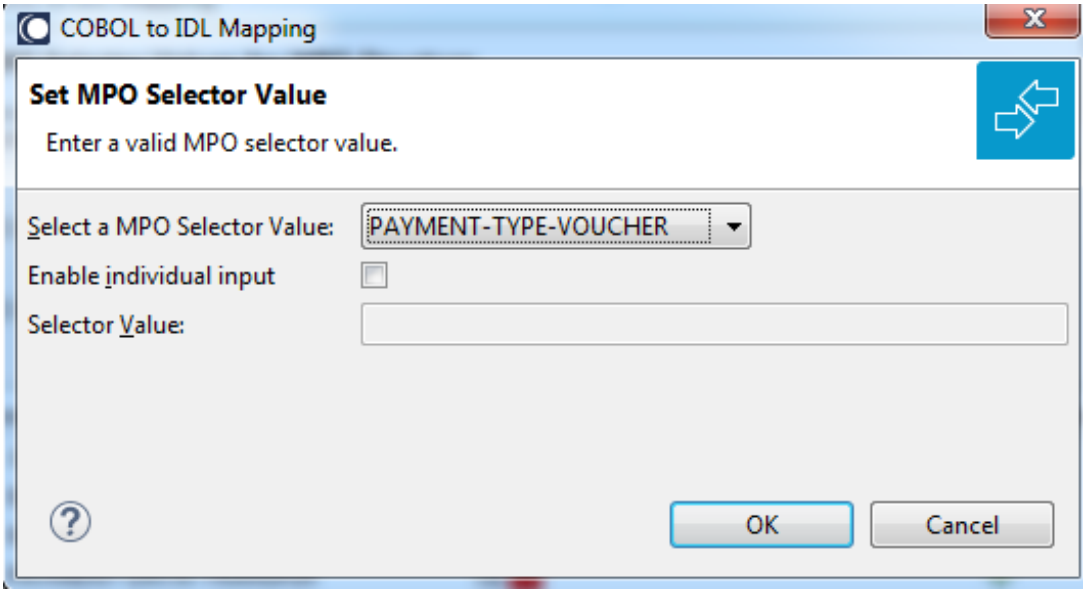
4 Create a new MPO group.



5 Set MPO selector values for MPO Structures.



Use the functions ✖ to delete and ✚ to add MPO selector values:



Notes:

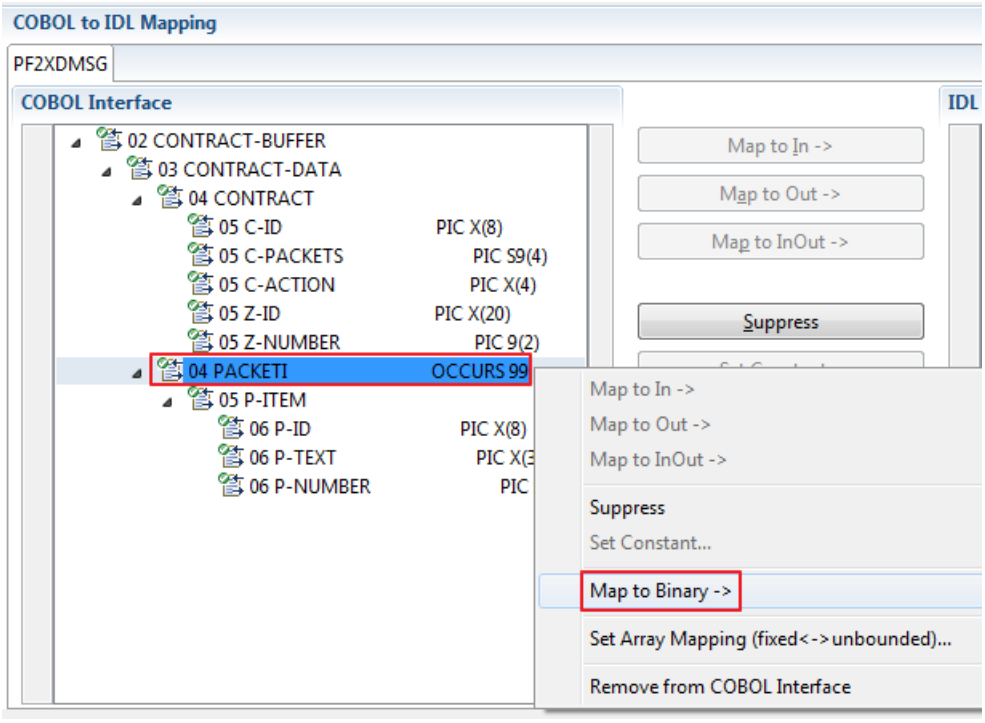
1. To add multiple MPO selector values per MPO structure, use the function + multiple times for the same MPO structure (see value 10 and 20 for structure `PAYMENT-DATA`).
2. MPO structures without any MPO selector value are unused and suppressed in the IDL interface value (e.g. see structure `PAYMENT-DATA-TRANSFER`).
3. Each MPO selector value must uniquely identify an MPO structure. The same value cannot be used more than once for different MPO structures.
4. If no defined MPO selector value matches at runtime, an empty MPO group is delivered to the RPC client, that is, none of the MPO cases contain any data. No runtime error is produced.

6 Press **Finish** to create the following IDL together with a server mapping file. See *Server Mapping Files for COBOL* in the Designer documentation.

```
library 'PAYMENT' is

 program 'PAYMENT' is
  define data parameter
   1 INPUT              In
    2 ORDER-NUMBER             (NU10)
   1 OUTPUT             Out
    2 PAYMENT-TYPE             (A2)
    2 PAYMENT-DATA-MPO  Choice
     3 PAYMENT-DATA            (/V1)
      4 PAYMENT-DATA           (AV256)
     3 PAYMENT-DATA-VOUCHER    (/V1)
      4 VOUCHER-ORIGIN         (AV128)
      4 VOUCHER-SERIES         (AV128)
     3 PAYMENT-DATA-CREDITCARD (/V1)
      4 CREDITCARD-NUMBER      (NU18)
      4 CREDITCARD-CODE        (NU12)
      4 CREDITCARD-VALIDITY    (AV8)
  end-define
```

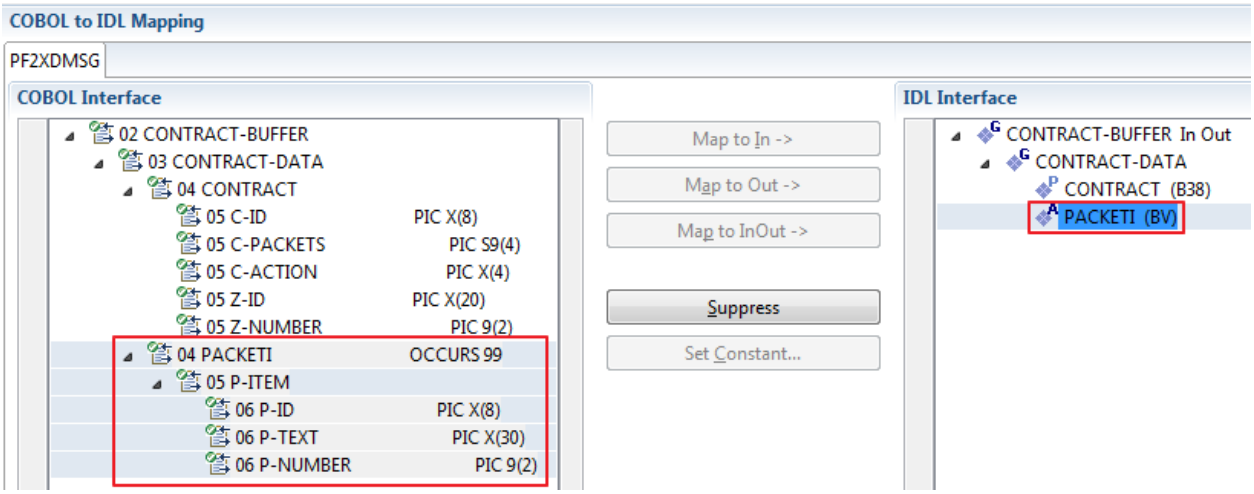
Map to Binary and Revert Binary Mapping

With such a mapping you allow the COBOL server to deal with binary data (for example images). You can also manage ***Complex MPO Selections***.



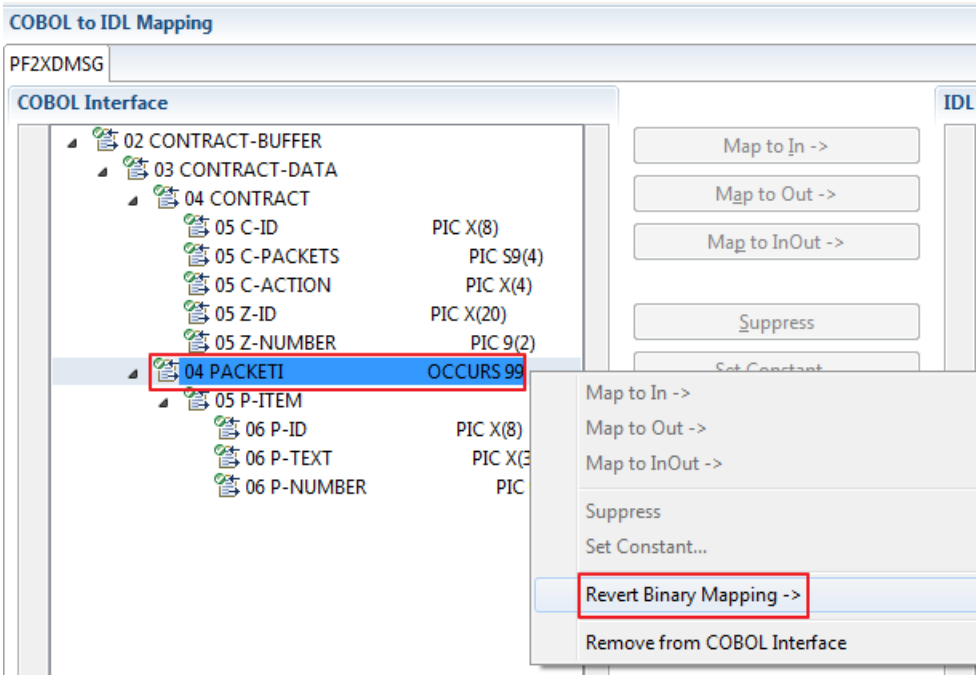
The menu entry **Map to Binary** appears only on COBOL data items were it makes sense, for example in Channel Container interface types it is not allowed to map the container reference itself as binary, but inner items can be mapped as binary. Redefine groups will be handled as a block, that means the largest redefine path or redefine base defines the binary length.

When the binary IDL parameter is selected, all corresponding COBOL data items are selected as well.

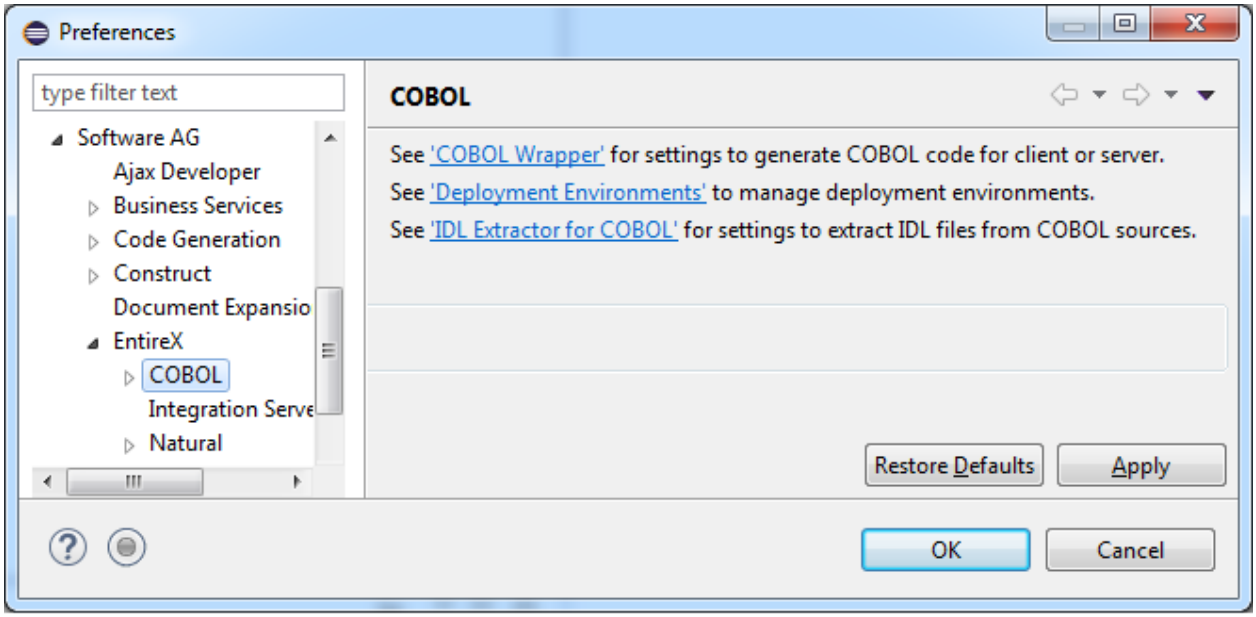


Note: The last COBOL data items are mapped to IDL data type BV instead of B*n* (`PACKETI (BV)` in this example).

To undo the binary mapping, select the root COBOL data item (the first of the selection group) and from the context menu choose **Revert Binary Mapping**.



16 COBOL Preferences



This page provides links to the following:

COBOL Wrapper Preferences

Use the preferences page **COBOL > COBOL Wrapper** to set the workspace defaults for target operating system, interface types etc. The settings are used as the defaults for the IDL properties when a new IDL file is created. See *Generation Settings - Preferences* in the COBOL Wrapper documentation.

Deployment Environments Preferences

Use the preferences page **COBOL > Deployment Environments** to define a connection to the Deployment Service of the RPC server. See *Preferences* under *Server Mapping Deployment Wizard* in the Designer documentation.

IDL Extractor for COBOL Preferences

Use the preferences page **COBOL > IDL Extractor for COBOL** to perform the following tasks:

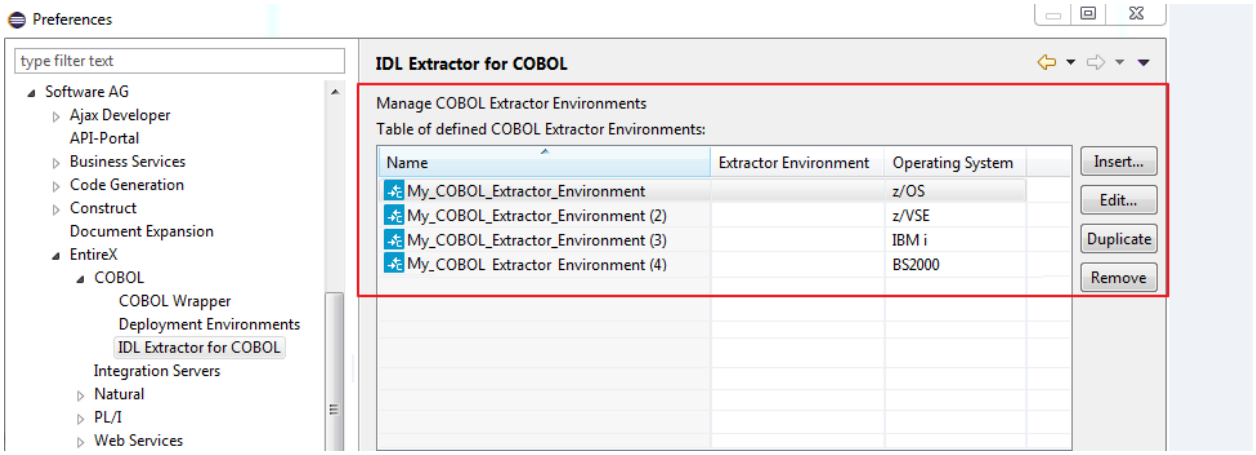
- Manage COBOL Extractor Environments
- Define Prefixes for IDL Parameter Names

Manage COBOL Extractor Environments

A COBOL extractor environment provides defaults for the extraction and refers to COBOL programs and copybooks

- stored locally on the same machine where the Designer is running, a so-called local COBOL extractor environment, or
- stored remotely on a host computer, a so-called remote COBOL extractor environment. The Extractor Service is required to access COBOL programs and copybooks remotely with a remote COBOL extractor environment. The Extractor Service is supported on operating systems z/OS and BS2000. See *Extractor Service* in the RPC Server documentation for Batch | IMS | BS2000.

COBOL extractor environments are offered in the IDL Extractor for COBOL wizard to reference the COBOL programs and copybooks and retrieve defaults for the IDL extraction. To create, edit, duplicate and remove COBOL extractor environments, open the preferences page **COBOL > IDL Extractor for COBOL** and use the buttons on the right.



This section describes how to create the following extractor environments:

- Creating a New Local Extractor Environment
- Creating a New Remote Extractor Environment (z/OS)

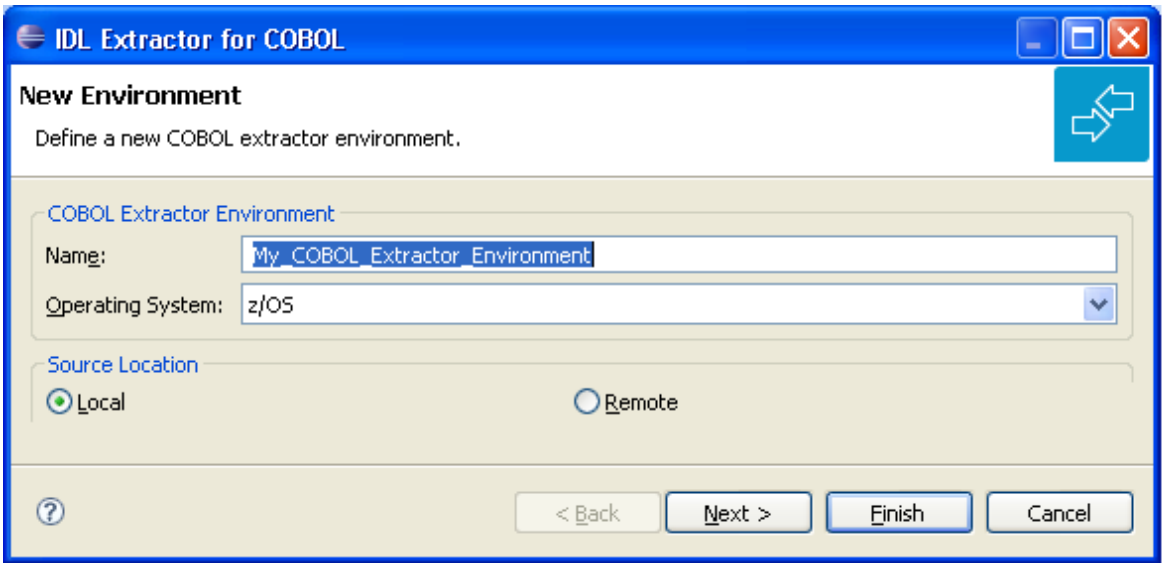
- Creating a New Remote Extractor Environment (BS2000)

Creating a New Local Extractor Environment

This section describes the four steps for creating a new local COBOL extractor environment to extract COBOL programs stored inside Software AG Designer or locally on your PC.

To create a new local extractor environment

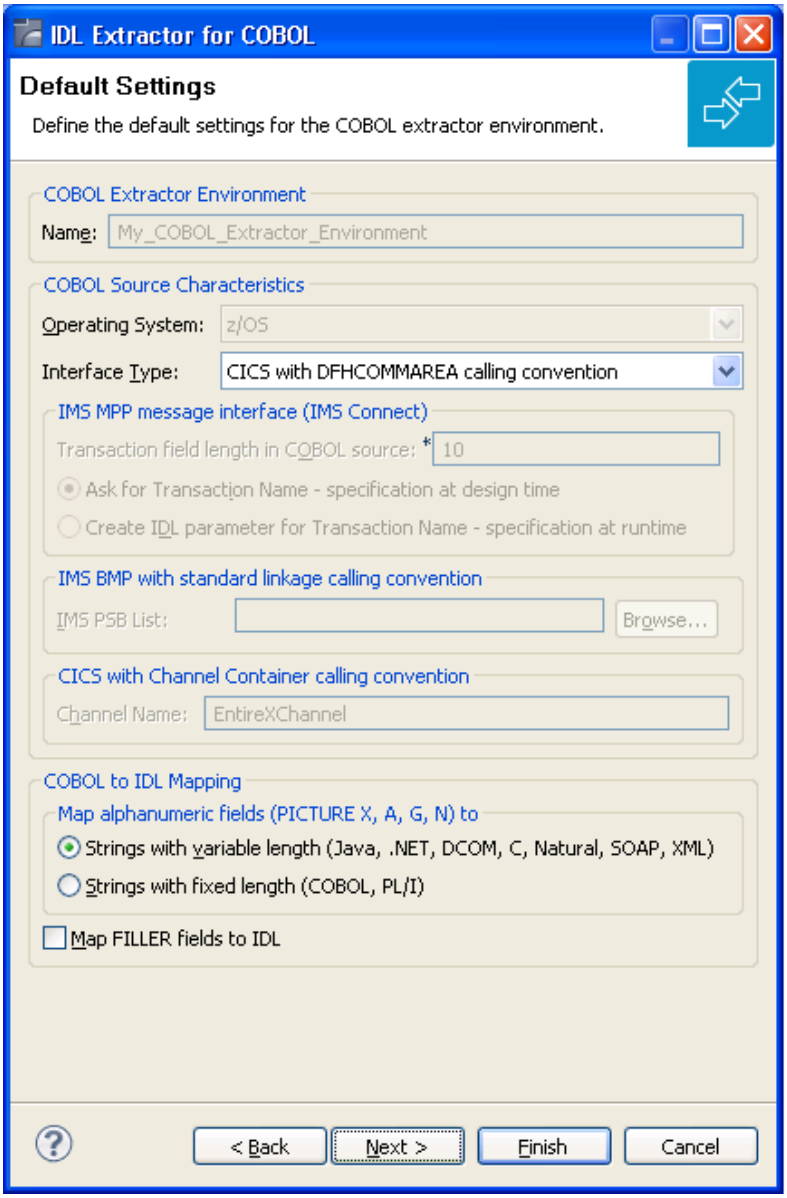
1 Define the new local environment. On the **New Environment** page you can specify **Name** and **Operating system**.



Define the new environment settings:

- Enter a unique **Name** for the COBOL extractor environment.
- Select the **Operating system** where the COBOL source originates from.
- Select "Local" for **Source Location**.

2 Define the default settings. The **Default Settings** page provides defaults for *Step 4: Define the Extraction Settings and Start Extraction* in *Using the IDL Extractor for COBOL - Overview*. You can set defaults for interface type and COBOL to IDL mapping.



Define the default extraction settings:

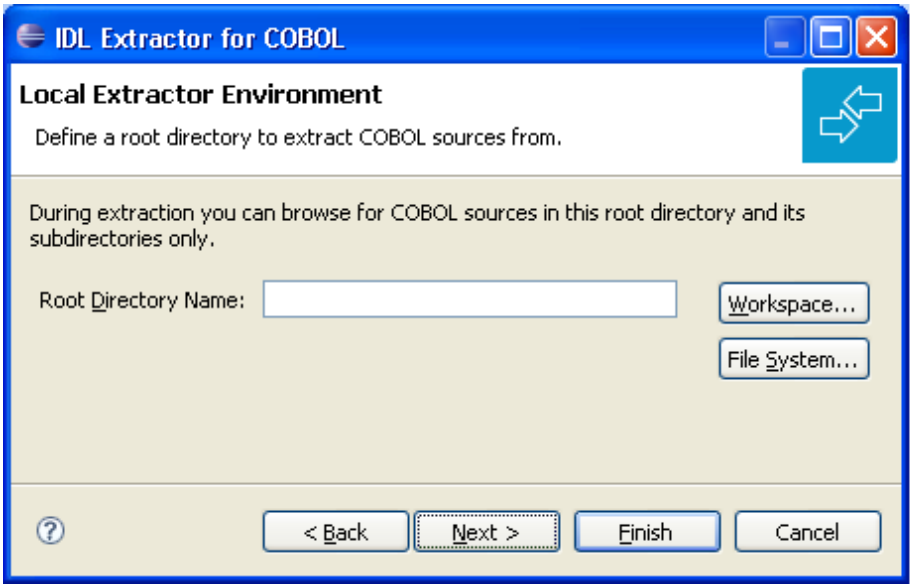
- Select the default **Interface Type**. See *Supported COBOL Interface Types*.
- Depending on the interface type, additional information can be set. For interface type
  - *CICS with Channel Container Calling Convention*, you can set the channel name.
  - *IMS MPP Message Interface (IMS Connect)*, you can set defaults for the transaction name. Possible options are a constant transaction name defined during extraction process or an IDL parameter to be specified at runtime.
  - *IMS BMP with Standard Linkage Calling Convention*, you can set the default for **IMS PSB List**.

For more information refer to *Step 4: Define the Extraction Settings and Start Extraction*.

- Specify a default value for **COBOL to IDL Mapping**. See *COBOL to IDL Mapping*.

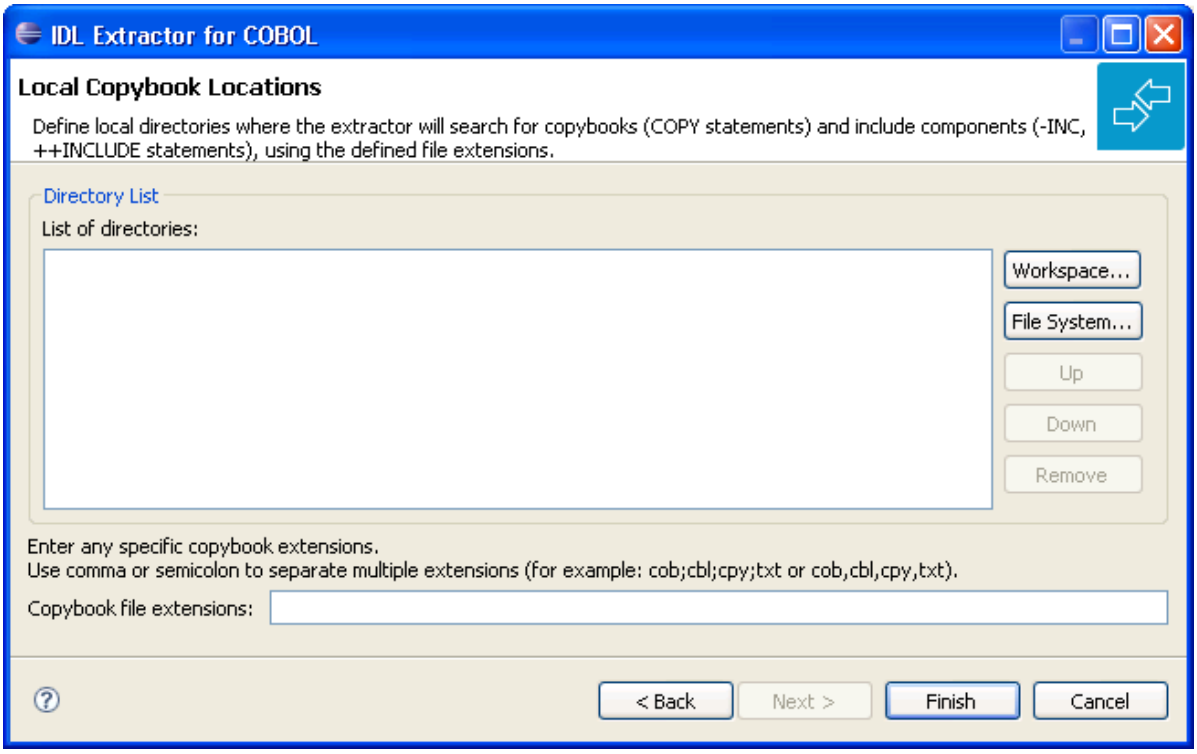
Press **Next**.

3 Define the local extractor environment. On the **Local Extractor Environment** page you can provide a default directory name for the COBOL programs:



Choose **Workspace...** or **File System...** to browse for a folder. Continue with **Next**.

4 Define the local copybook locations. On the **Local Copybook Location** page you can add directories that will be used to resolve copybooks. Copybooks and members referenced with COPY statements, CA Librarian `-INC` statements and CA Panvalet `++INCLUDE` statements will be searched for in the defined local directories:



The file extensions for copybooks can also be entered. If no extensions are specified, the IDL Extractor for COBOL wizard will try to locate copybooks without any file extensions.

Press **Workspace...** or **File System...** to browse for a folder.

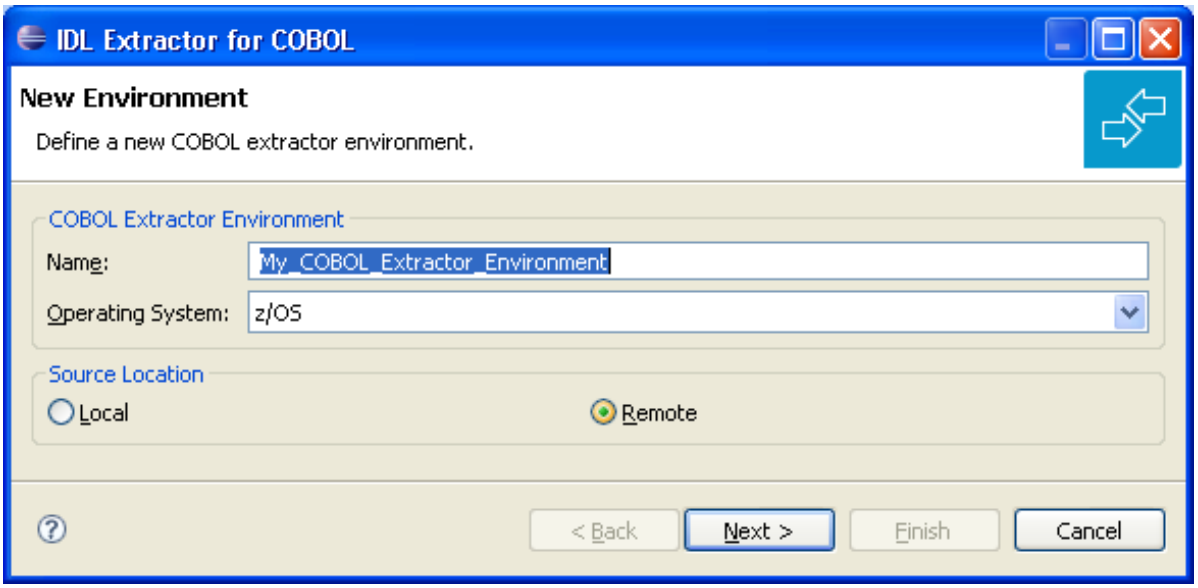
Press **Finish**.

Creating a New Remote Extractor Environment (z/OS)

This section describes the four steps for creating a new remote COBOL extractor environment to extract remotely z/OS COBOL programs stored in partitioned data sets or CA Librarian data sets.

To create a new remote extractor environment

1 Define the new remote environment. >On the **New Environment** page you can specify **Name**, **Operating system** and the **Remote Source Location**.

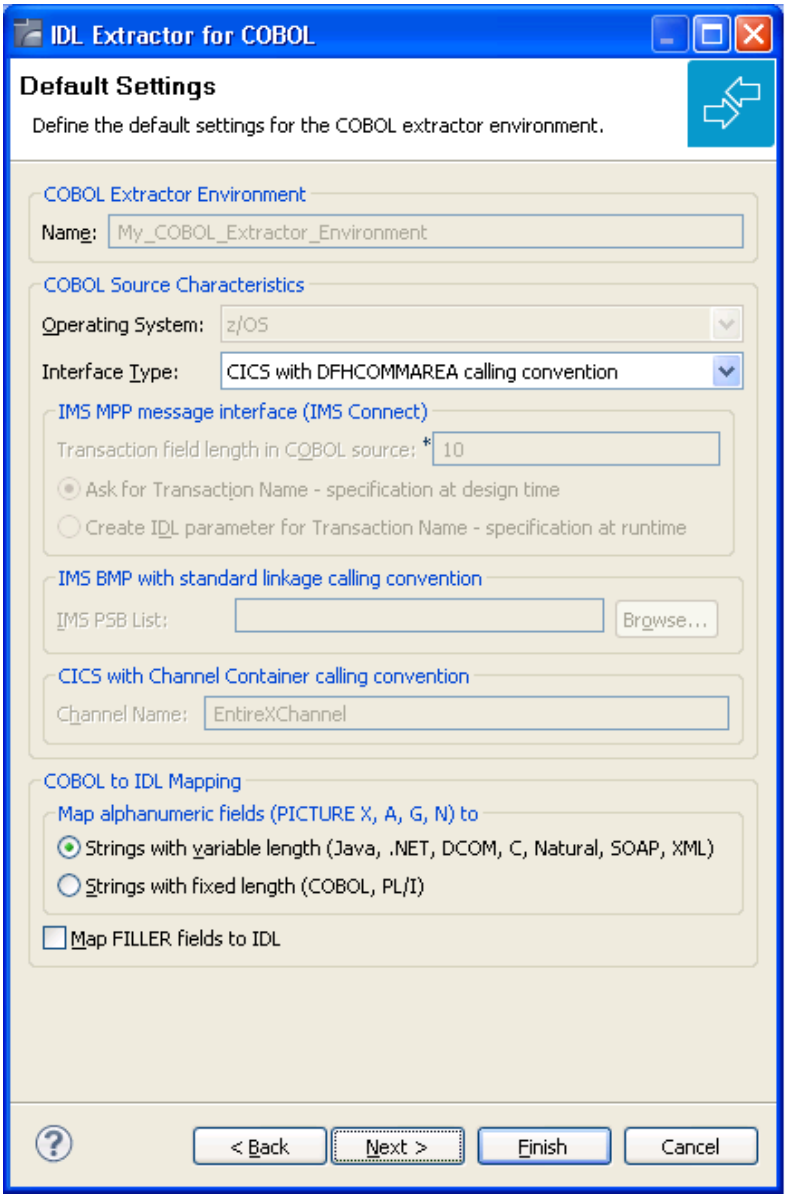


Define the new environment settings:

- Enter a unique name for the COBOL extractor environment.
- Select the **Operating system**.
- Select "Remote" for **Source location**.

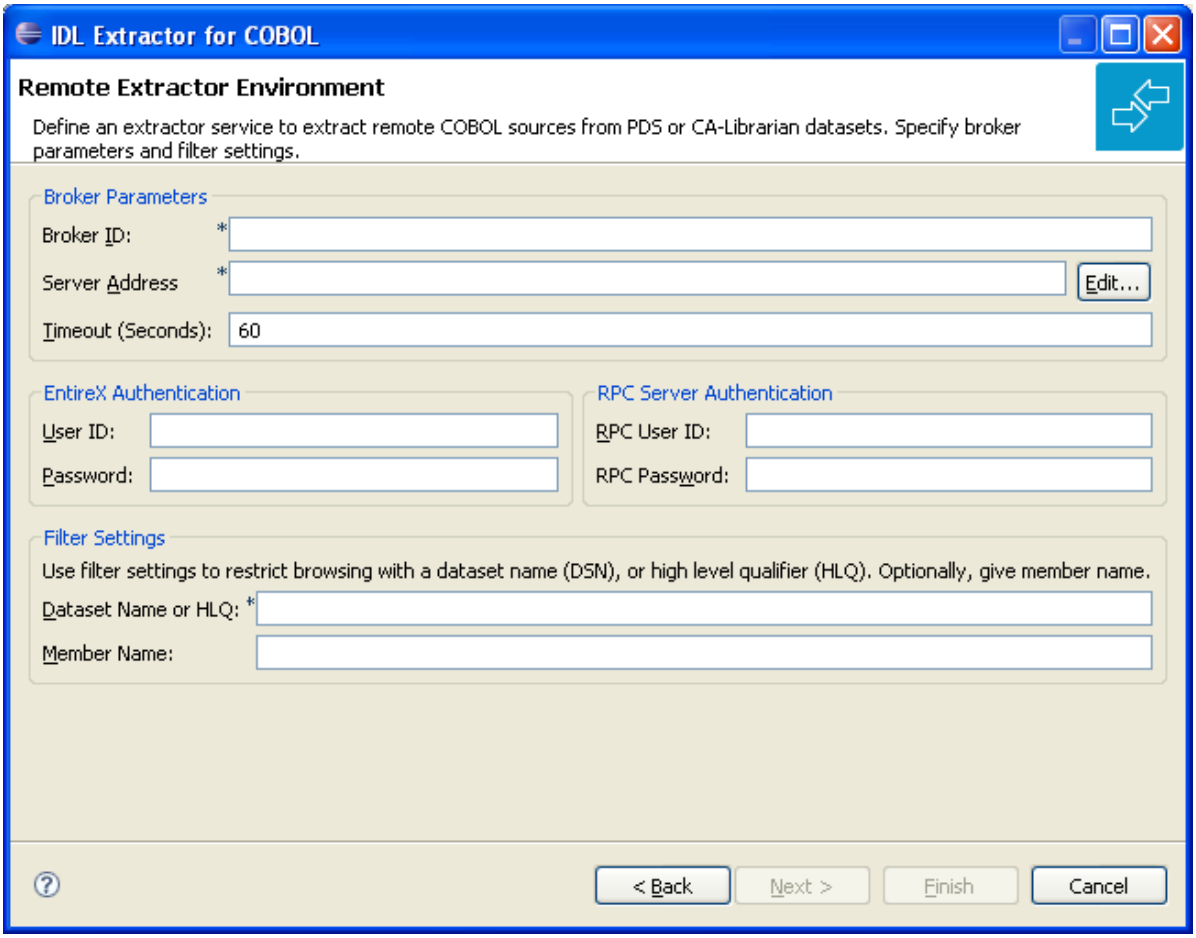
2 Define the default settings.The **Default Settings** page provides defaults for *Step 4: Define the Extraction Settings and Start Extraction* in *Using the IDL Extractor for COBOL - Overview*.

You can set defaults for **Interface Type** and **COBOL to IDL Mapping**.



Define the default settings. See *Define the default settings* in section *Creating a New Local Extractor Environment*. Continue with **Next**.

3 Define the remote extractor environment. The connection to the Extractor Service to browse for COBOL programs is defined on the **Remote Extractor Environment** page. See *Extractor Service*.



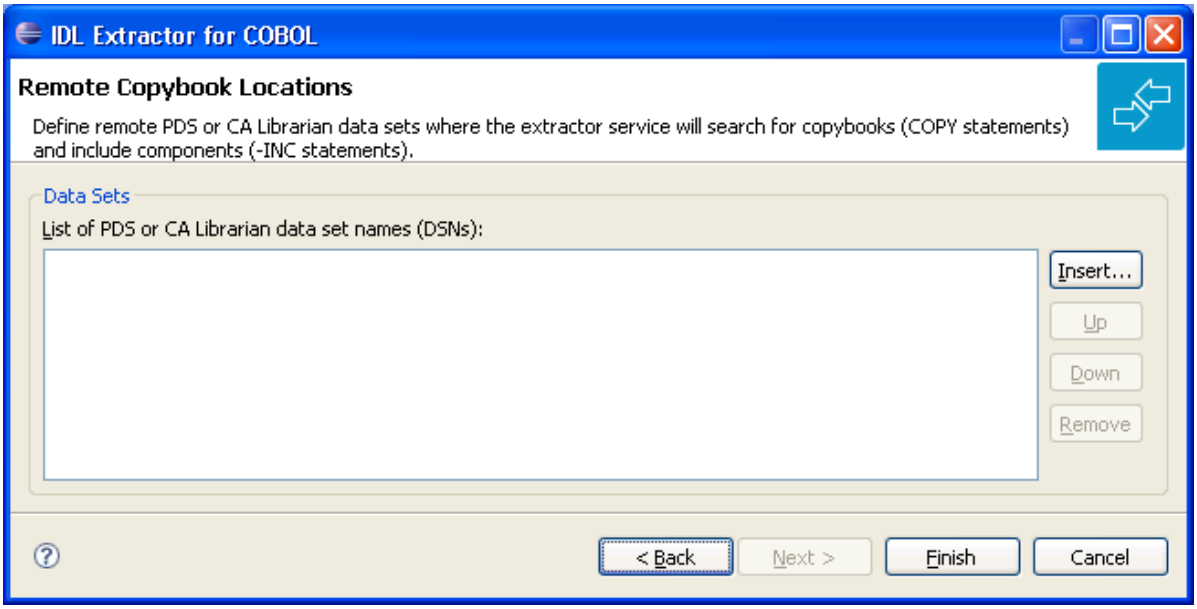
Define the remote extractor environment:

- Under **Broker Parameters**, enter the required fields **Broker ID** and **Server Address**, usually "RPC/<*servername*>/CALLNAT". The timeout value must be in the range 1-9999 seconds (default is 60).
- The **EntireX Authentication** parameters describe the settings for the broker. See *Authentication of User*.
- The **RPC Server Authentication** parameters describe the settings for the RPC server. See *Administering the RPC server* | *Administering the RPC Server for IMS*.
- A high-level qualifier is required in the **Data Set Name or HLQ** field. The extractor service will then offer only data sets with this high-level qualifier.
- In the **Member Name** field you can provide a prefix for the partitioned data set or CA Librarian members. The extractor service will then offer only members beginning with this prefix.

Continue with **Next**.

4 Define the remote copybook locations. On the **Remote Copybook Location** page you can add PDS or CA Librarian data sets that will be used to resolve copybooks. Copybooks and members

referenced with COPY statements and CA Librarian `-INC` statements will be searched for in the defined remote data sets:



Press **Insert...** to add a new data set entry in the table. Use **Remove**, **Up** and **Down** to manage the data set list.

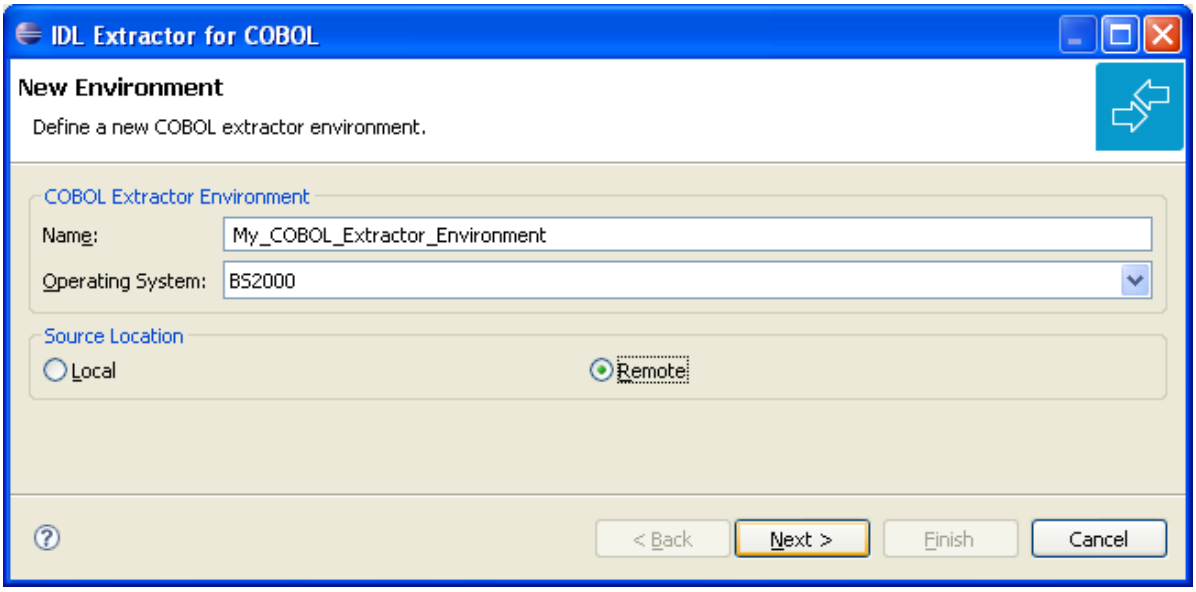
Press **Finish**.

Creating a New Remote Extractor Environment (BS2000)

This section describes the four steps for creating a new remote COBOL extractor environment to extract remotely BS2000 COBOL programs stored in LMS libraries.

To create a new remote extractor environment

1 Define the new remote environment. On the **New Environment** page you can specify **Name**, **Operating system** and the **Remote Source Location**.

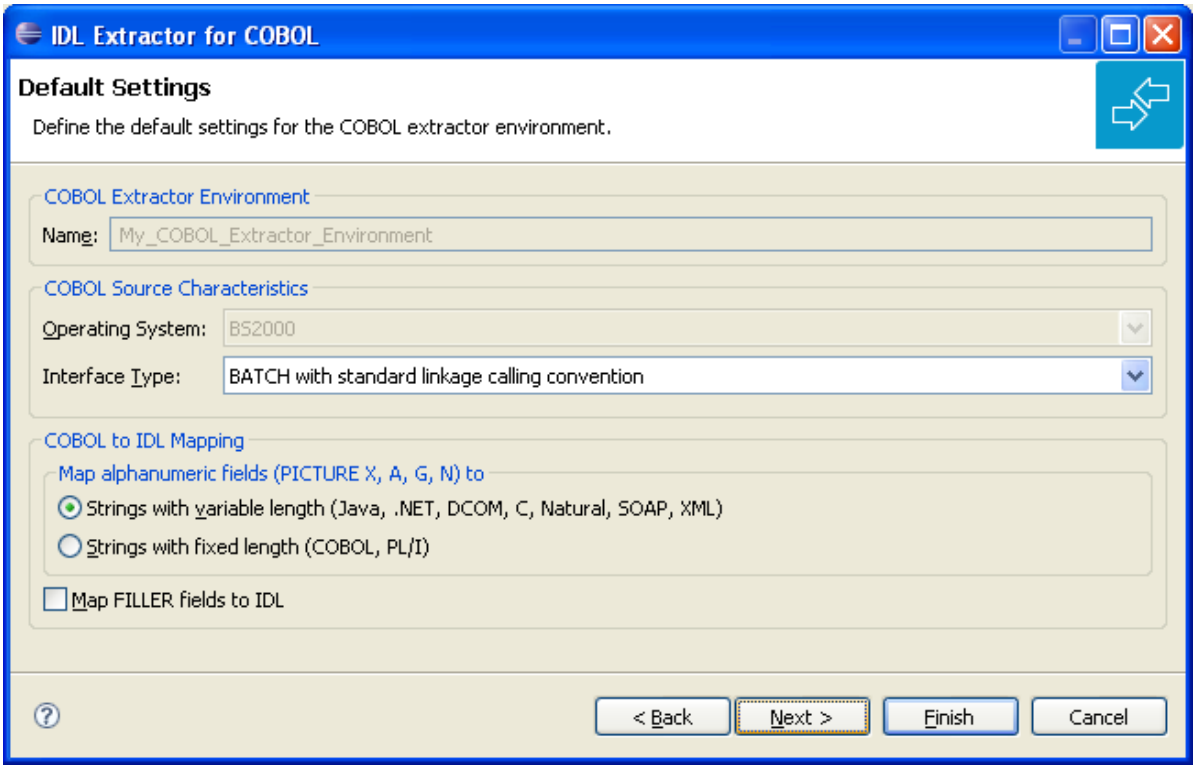


Define the new environment settings:

- Enter a unique name for the COBOL extractor environment.
- Select the **Operating system**
- Select "Remote" for **Source location**

2 Define the default settings. The **Default Settings** page provides defaults for *Step 4: Define the Extraction Settings and Start Extraction* in *Using the IDL Extractor for COBOL - Overview*.

You can set defaults for **Interface Type** and **COBOL to IDL Mapping**.

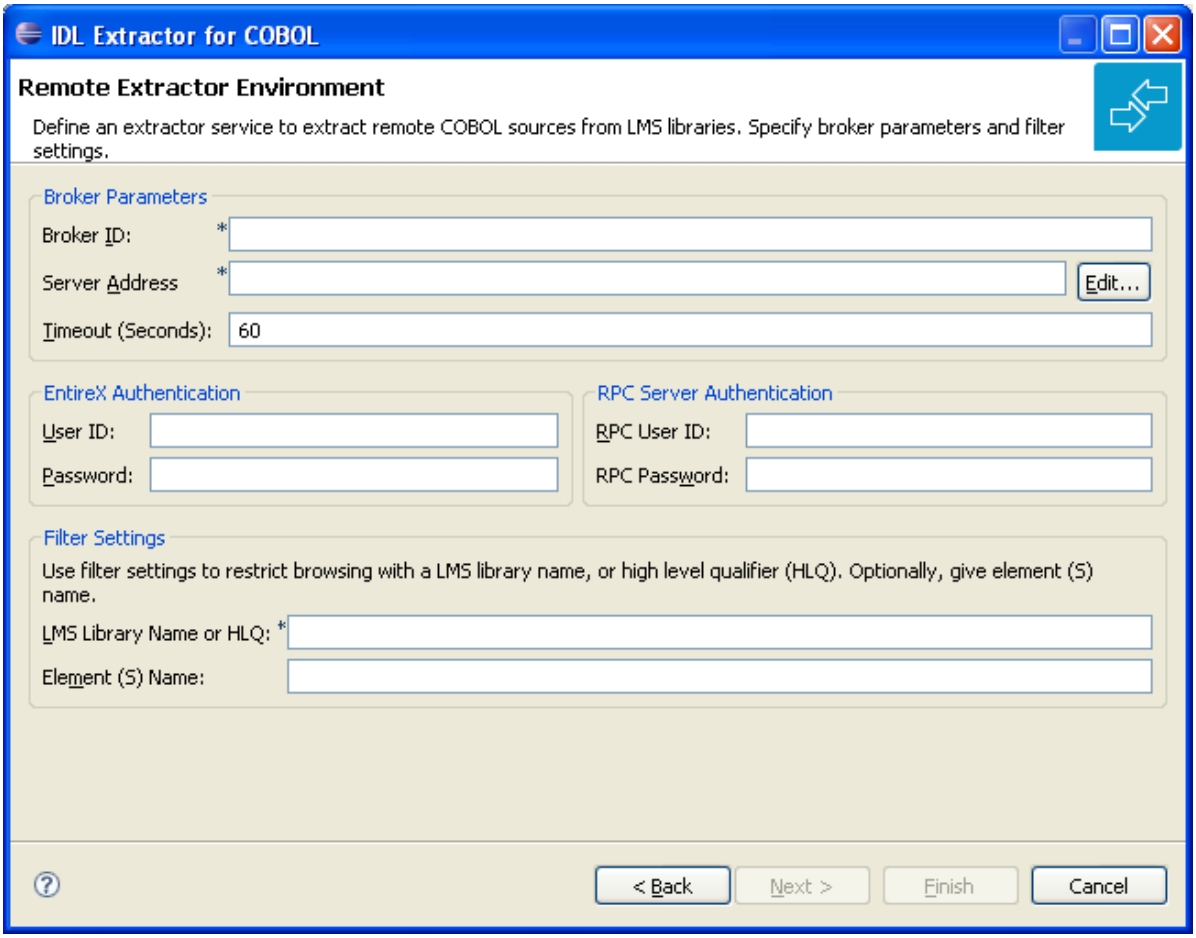


Define the default extraction settings:

- Select the default **Interface Type**. See *Supported COBOL Interface Types*.
- Specify the default **COBOL to IDL Mapping**. See *COBOL to IDL Mapping*.

Continue with **Next**.

3 Define the remote extractor environment. The connection to the Extractor Service to browse for COBOL programs is defined on the **Remote Extractor Environment** page. See *Extractor Service*.

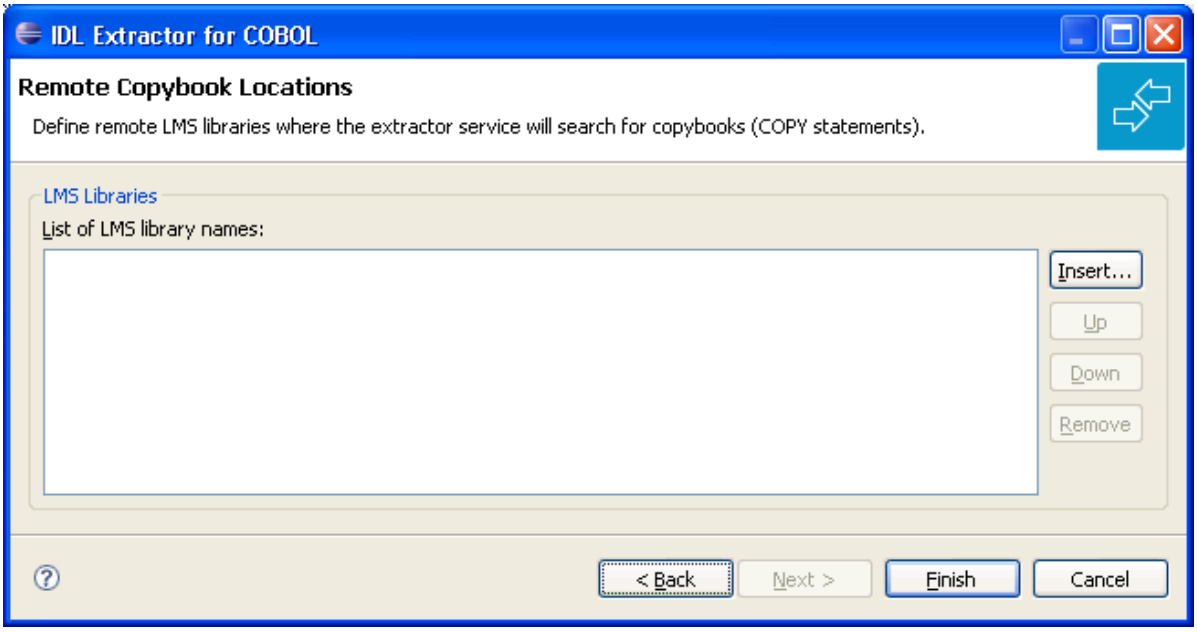


Define the remote extractor environment:

- Under **Broker Parameters**, enter the required fields **Broker ID** and **Server Address**, usually "RPC/<*servername*>/CALLNAT". The timeout value must be in the range 1-9999 seconds (default is 60).
- The **EntireX Authentication** parameters describe the settings for the broker. See *Authentication of User*.
- The **RPC Server Authentication** parameters describe the settings for the RPC server. See *Configuring the RPC Server*.
- A high-level qualifier can be entered in the **LMS Library Name or HLQ** field. The extractor service will then offer only LMS libraries with this high-level qualifier. You can use wildcard notation with asterisk to specify a range of values.
- In the **Element Name** field you can provide a prefix for LMS library source elements. The extractor service will then offer only COBOL programs beginning with this prefix.

Continue with **Next**.

4 Define the remote copybook locations. On the **Remote Copybook Location** page you can add directories that will be used to resolve copybooks. Copybooks referenced with `COPY` statements will be searched for in the defined remote LMS libraries:

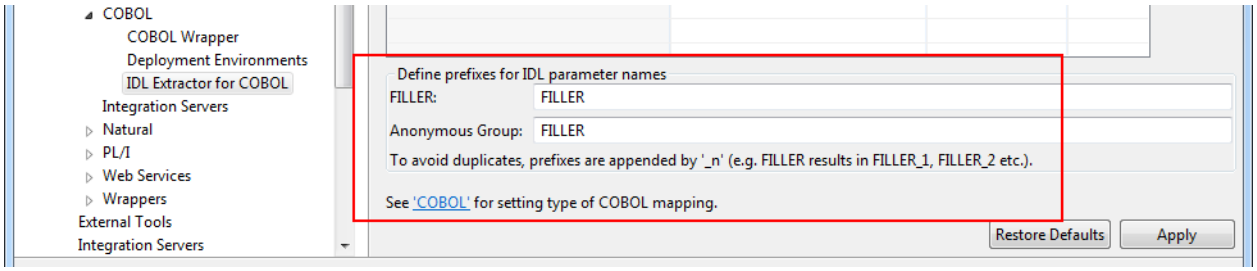


Press **Insert...** to add a new data set entry in the table. Use **Remove**, **Up** and **Down** to manage the list of LMS libraries.

Press **Finish**.

Define Prefixes for IDL Parameter Names

You can also use the preferences page **COBOL > IDL Extractor for COBOL** to define prefixes for IDL parameter names.



The defined prefixes are used for *FILLER Pseudo-Parameter* and are valid for all COBOL extractor environments.

17 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***.

See also ***IDL Extraction per Interface Type*** under ***COBOL Mapping Editor*** for guidelines on IDL extraction per interface type.

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.

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols "[" and "]" enclose optional lexical entities.
- The informal term *number* (or in some cases *number1*.*number2*) is a sequence 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	1,2,3,9
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
	Zoned decimal	PIC S9(*n*)[V9(*m*)]	N*n*[,*m*]	Unpacked decimal	2,4
	Packed decimal	PIC 9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	PU*n*[,*m*]	Packed decimal unsigned	2,4
	Packed decimal	PIC S9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	P*n*[,*m*]	Packed decimal	2,4
	Packed decimal	PIC 9(*n*) [V9(*m*)] PACKED-DECIMAL	PU*n*[,*m*]	Packed decimal unsigned	2,4
	Packed decimal	PIC S9(*n*) [V9(*m*)] PACKED-DECIMAL	P*n*[,*m*]	Packed decimal	2,4
	Binary	PIC [S]9(*n*) BINARY (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6

COBOL Data Type			Software AG IDL Data Type		Notes
	Binary	PIC [S]9(*n*) BINARY (5<=*n*<=9)	I4	Integer (large)	2,4,5,6,7
	Computational	PIC 9(*n*) COMP[UTATIONAL]	PU*n*	Packed decimal unsigned	2,4,11
	Computational	PIC 9(*n*) COMP[UTATIONAL] (1<=*n*=4)	I2	Integer (medium)	2,4,5,6,12
	Computational	PIC 9(*n*) COMP[UTATIONAL] (5<=*n*=9)	I4	Integer (large)	2,4,5,6,7,12
	Computational	PIC S9(*n*) COMP[UTATIONAL]	P*n*	Packed decimal	2,4,11
	Computational	PIC S9(*n*) COMP[UTATIONAL] (1<=*n*=4)	I2	Integer (medium)	2,4,5,6,12
	Computational	PIC S9(*n*) COMP[UTATIONAL] (5<=*n*=9)	I4	Integer (large)	2,4,5,6,7,12
	Binary	PIC [S]9(*n*) COMP[UTATIONAL][-4] (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6
	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
	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)	8
	Floating point	COMP[UTATIONAL]-2	F8	Floating point (large)	8
Alphanumeric-edited		Alphanumeric item containing "0" or "/"	A(*length of PIC*)	Alphanumeric	10
Numeric-edited		Numeric item containing "DB", "CR", "Z", "$", ".", ",", "+", "-", "*", "B", "0" or "/"	A(*length of PIC*)	Alphanumeric	10

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* in the *IDL Editor* documentation.
   - 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
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. COBOL binary or computational items with more than 9 digits in the PICTURE clause cannot be mapped to IDL type I. 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. 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.

9. 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.
10. On platform IBM i, COBOL computational items are mapped by default to packed decimal.
11. On all platform except IBM i, COBOL computational items are mapped by default to IDL type I.

User-defined Mapping

Depending on the COBOL syntax and the COBOL server implementation, user interaction may be required to get correct extraction results. User interaction can also simplify or modernize the extracted IDL. As a result, the user-defined mapping is contained in a Designer file with extension .cvm that contains COBOL-specific mapping information. See *Server Mapping Files for COBOL* in the Designer documentation. The following is covered:

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 or InOut. RPC clients have to pass correct values, the 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. For more information and COBOL examples, see Mapping Editor IDL Interface mapping function *Set COBOL Data Items to Constants* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).
- **Function or Operation Codes**
  If your COBOL server implements verious functions or operations and the level-88 values are function or operation codes, map the COBOL interface to multiple IDL interfaces. For more information and COBOL examples see the Mapping Editor IDL Interface mapping function *Map to Multiple IDL Interfaces* for interface type DFHCOMMAREA (In same as Out, In differ-

ent to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

If the mapping functions **Set COBOL Data Items to Constants** or **Map to Multiple IDL Interfaces** are used, a server mapping file is required to provide additional information. See *Server Mapping Files for COBOL*.

COBOL Data Items

This section covers the following topics:

- IDL Directions for COBOL Data Items
- IDL Parameter Names derived from COBOL Names
- COBOL Data Items Expecting Single Constant Values
- COBOL Data Items used as Function or Operation Codes
- Optional COBOL Group Data Items
- Unneeded COBOL Data Items

IDL Directions for COBOL Data Items

COBOL server programs do not contain parameter direction information (input, output). Therefore IDL directions (see `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation) need to be added manually in the COBOL Mapping Editor. See *Map to In, Out, InOut* for interface type DFHCOMMAREA (In same as Out) | Large Buffer (In same as Out) | Batch | IMS BMP | In same as Out).

IDL Parameter Names derived from COBOL Names

Numbers in the first position of the parameter name are not allowed in Software AG IDL syntax (see *Software AG IDL Grammar* in the *IDL Editor* documentation). Thus COBOL names starting with a number are prefixed with the character "#" by default. 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.

You can rename all IDL parameters in the ***COBOL Mapping Editor***. See *IDL Interface* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

COBOL Data Items Expecting Single Constant Values

If your COBOL server interface expects for your purpose always a constant value, use *Set COBOL Data Items to Constants* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

COBOL Data Items used as Function or Operation Codes

If your COBOL server implements various functions or operations and the data items represent function or operation codes, map the COBOL interface to multiple IDL interfaces. For more information and COBOL examples see the Mapping Editor IDL Interface mapping function *Map to Multiple IDL Interfaces* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

Optional COBOL Group Data Items

If your COBOL server interface produces dissimilar shapes of optional output, COBOL group data items can be mapped to multiple possible output (MPO). Criteria can be added under which circumstances COBOL groups are part of the returned data or not. This is done with Mapping Editor IDL Interface mapping function *Set Multiple Possible Output (MPO) Structures* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | IMS Connect | COBOL Converter (In same as Out, In different to Out).

Unneeded COBOL Data Items

COBOL data items that are not needed in the IDL Interface but are required by the COBOL server can be suppressed. See *Suppress Unneeded COBOL Data Items* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out)

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 ***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)
```

REDEFINES 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 directly supported. The following possibilities are available to map COBOL `REDEFINE`s:

1. You can select a single redefine path for IDL usage. In this case, the COBOL server requires predictable input and output structures. The redefine path can be determined at design time (extraction time). This is supported for all IDL directions that is, In, Out and InOut. For more information and COBOL examples, see Mapping Editor IDL Interface mapping function *Select REDEFINE Paths* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

   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 REDEFINES *name_2*	1 FILLER_*n*
1 FILLER REDEFINES *name_2*	1 FILLER_*n*

2. If the COBOL server supports more than one type of input (redefine paths) but uses predictable output structures, you can map the COBOL interface to multiple IDL interfaces. This is supported for IDL direction In only. In this case, the redefine path used is selected as described under 1 above. For more information and COBOL examples, see Mapping Editor IDL Interface mapping function *Map to Multiple IDL Interfaces* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).
3. If the COBOL server produces more than one type of output (redefine paths) and implements the multiple possible output (MPO) concept, you can map the redefine to MPO. In this case the redefine path used is determined at runtime from a set of predefined redefine paths. The set of alternate redefine paths is determined during design time (extraction time). This is supported for IDL direction Out only. For more information and COBOL examples of the MPO concept, see Mapping Editor IDL Interface mapping function *Set Multiple Possible Output (MPO) Structures*

for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | IMS Connect.

If a `REDEFINE` is mapped to multiple possible output (MPO), the mapping is as follows:

COBOL Syntax	Software AG IDL Syntax
1 *name_1* 2 *name_1_1*	1 *name_1* (/*V1*) 2 *name_1_1* . . . . .
1 *name_2* REDEFINES *name_1* 2 *name_2_1*	1 *name_2* (/*V1*) 2 *name_2_1*. . . . .
1 *name_3* REDEFINES *name_1* 2 *name_3_1*	1 *name_3* (/*V1*) 2 *name_3_1* . . . . .

4. If the COBOL server supports more than one type of input (redefine paths) and implements the multiple possible output (MPO) concept as well, you can combine extraction as described under 2 and 3 above.

In all cases the, COBOL `REDEFINE` requires a server mapping file to provide additional information. See *Server Mapping Files for COBOL*.

COBOL Tables with Fixed Size

The following possibilities are available to map COBOL tables with fixed size:

- By default, fixed-size COBOL tables are converted automatically to fixed-size IDL groups (see `group-parameter-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) with fixed-bound-array (see `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation). This is the usual way and is suitable for most situations. See the following syntax:

COBOL Syntax	Software AG IDL Syntax
1 name OCCURS *n* [TIMES] 2 *name_1*	1 *name* (/*n*) 2 *name_1* . . . . .

COBOL Syntax	Software AG IDL Syntax
1 *name* OCCURS *n* [TIMES] [ASCENDING | DESCENDING [KEY] [IS] ↩ key_name] 2 *name_1*	1 *name* (/*n*) 2 *name_1* . . . . .
1 *name* OCCURS *n* [TIMES] [[INDEXED [BY] *index_name*] 2 *name_1*	1 *name* (/*n*) 2 *name_1* . . . . .

- In very rare situations, the COBOL server uses a fixed-size COBOL table in a variable-size manner. In contrast - as the syntax implies - a variable number of elements is transferred in this fixed-size array (input only, output only or both directions are possible). Array elements at the end of the array are unused. The current number of elements can be calculated using different approaches by the receiver of such an array. This is possible for message-oriented interface types: DFHCOMMAREA, Large Buffer, Channel Container, IMS Connect. The fixed-sized COBOL table must be the last parameter in the interface. For more information and COBOL examples see the Mapping Editor IDL Interface mapping function *Set Arrays (Fixed <-> Unbounded)* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | IMS Connect | COBOL Converter (In same as Out, In different to Out).

The following rules apply:

- The combination of phrases `ASCENDING` and `INDEXED BY` and phrases `DESCENDING` and `INDEXED BY` is meaningless for EntireX and therefore ignored by the IDL Extractor for COBOL.
- If the mapping function *Set Arrays (Fixed <-> Unbounded)* is used, a server mapping file is required to provide additional information. See *Server Mapping Files for COBOL*.

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. If you require the value on input to the COBOL server you specify to be a constant, see *Set COBOL Data Items to Constants* for interface type DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel

Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

DATA DIVISION Mapping

This section describes the COBOL syntax relevant for extracting the `DATA DIVISION`. No user decisions in the COBOL Mapping Editor are required or possible here.

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.

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 DATE FORMAT clause. The semantic given by the DATE FORMAT clause has to be considered by RPC clients.

GLOBAL and EXTERNAL Clause

The GLOBAL clause

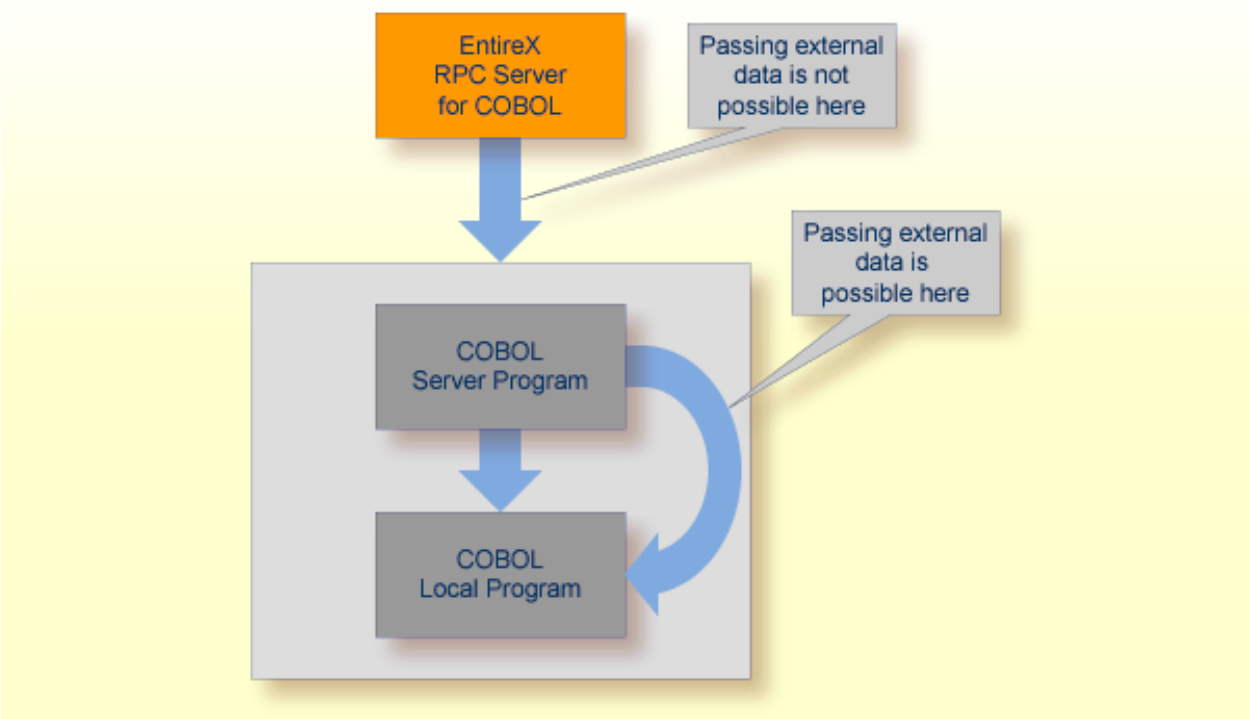
- specifies that a data-name is available to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name.
- is not considered by the IDL Extractor for COBOL. The DATA DIVISION is parsed as without the GLOBAL clause.

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.

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

The following rules apply:

- 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 a server mapping file. See *Server Mapping Files for COBOL*. 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.

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 Clauses

The `SIGN LEADING` and `TRAILING SEPARATE` clauses are supported. Both require a server mapping file. See *Server Mapping Files for COBOL*.

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. See *Server Mapping Files for COBOL*. At runtime, the RPC server aligns the data items accordingly.

COBOL Tables with Variable Size - DEPENDING ON Clause

Variable size COBOL tables are converted to IDL unbounded groups (see `group-parameter-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) with an unbounded array (see `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) and maximum upper bound set. The lower-bound is always set to 1 in the IDL. See the following example:

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

The ODO subject (data item `TABLE`) will be mapped in the IDL to an unbounded group (with maximum). The ODO object (data item `COUNTER-1`) is not part of the IDL because the number of elements is implicitly available with the IDL unbounded group. See *Map `OCCURS DEPENDING ON` for interface type* DFHCOMMAREA (In same as Out, In different to Out) | Large Buffer (In same as Out, In different to Out) | Channel Container | Batch | IMS BMP | IMS Connect | COBOL Converter (In same as Out, In different to Out).

```
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*
  2 *name_1* . .
  . . .
	1 *name* (/Vm)
	  2 *name_1* . .
	  . . .

1 *name* OCCURS *n* TO *m* [TIMES] DEPENDING [ON] *index* [ASCENDING | ↩ DESCENDING [KEY] [IS] *key_name*]
  2 *name_1* . .
  . . .
	1 *name* (/Vm)
	  2 *name_1* . .
	  . . .

1 *name* OCCURS *n* TO *m* [TIMES] DEPENDING [ON] *index* [INDEXED ↩ [BY] *index_name*]
  2 *name_1* . .
  . . .
	1 *name* (/Vm)
	  2 *name_1* . .
	  . . .

The following rules apply:

- 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 phrases `ASCENDING` and `INDEXED BY` and phrases `DESCENDING` and `INDEXED BY` is meaningless for EntireX and therefore ignored by the IDL Extractor for COBOL.
- The COBOL clause `OCCURS DEPENDING ON` requires a server mapping file to provide additional information. See *Server Mapping Files for COBOL*.

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.

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 BMP 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 DFHCOMMAREA programs, 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`, no suggestion is made.

For CICS Large Buffer, Channel Container and IMS MPP (IMS Connect) programs, parameters are not suggested; you select the parameters in the Mapping Editor manually.

However, you can always add, change and correct the suggested parameters if they are not the correct ones in the extraction wizard. See ***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 *Define the remote copybook locations* or *Define the local copybook locations* in *COBOL Preferences*.
- the copybook extensions for local extractions, see *Define the local copybook locations* in *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)==.
```