webMethods EntireX

Software AG IDL Extractor for COBOL

Version 9.7

October 2014

Document ID: EXX-EEXXCOBEXTRACTOR-97-20160805

Table of Contents

I Introduction to the IDL Extractor for COBOL 1
1 Introduction to the IDL Extractor for COBOL 3
Introduction 4
Extractor Wizard 5
Mapping Editor 6
Supported COBOL Interface Types 7
Usage of Server Mapping Files 17
II Using the IDL Extractor for COBOL - Overview 19
2 Scenario I: Create New IDL and Server Mapping Files 23
Step 1: Start the IDL Extractor for COBOL Wizard 24
Step 2: Select a COBOL Extractor Environment or Create a New One 25
Step 3: Select the COBOL Source 28
Step 4: Define the Extraction Settings and Start Extraction 36
Step 5: Select the COBOL Interface and Map to IDL Interface 45
Step 6: Finishing the Mapping Editor 47
Step 7: Validate the Extraction and Test the IDL File 49
3 Scenario II: Append to Existing IDL and Server Mapping Files 51
4 Scenario III: Modify Existing IDL and Server Mapping Files 53
III COBOL Mapping Editor 57
5 CICS with DFHCOMMAREA Calling Convention - In same as Out 59
Introduction 60
Extracting from a CICS DFHCOMMAREA Program 60
Mapping Editor User Interface 61
Mapping Editor IDL Interface Mapping Functions 68
Programming Techniques 72
6 CICS with DFHCOMMAREA Large Buffer Interface - In same as Out 77
Introduction 78
Extracting from a CICS DFHCOMMAREA Large Buffer Program 79
Mapping Editor User Interface 80
Mapping Editor IDL Interface Mapping Functions 87
Programming Techniques 92
7 Batch with Standard Linkage Calling Convention 95
Introduction 96
Extracting from a Standard Call Interface 96
Mapping Editor User Interface 97
Mapping Editor IDL Interface Mapping Functions 104
Programming Techniques 108
8 Micro Focus with Standard Linkage Calling Convention 111
Introduction 112
Extracting from a Standard Call Interface 112
Mapping Editor User Interface 113
Mapping Editor IDL Interface Mapping Functions 120
Programming Techniques 124

9 IMS BMP with Standard Linkage Calling Convention 127
Introduction 128
Extracting from an IMS BMP Standard Call Interface 128
Mapping Editor User Interface 130
Mapping Editor IDL Interface Mapping Functions 137
Programming Techniques 141
10 CICS with DFHCOMMAREA Calling Convention - In different to Out 143
Introduction 144
Extracting from a CICS DFHCOMMAREA Program 144
Mapping Editor User Interface 145
Mapping Editor IDL Interface Mapping Functions 152
Programming Techniques 156
11 CICS with DFHCOMMAREA Large Buffer Interface - In different to Out 161
Introduction 162
Extracting from a CICS DFHCOMMAREA Large Buffer Program 163
Mapping Editor User Interface 164
Mapping Editor IDL Interface Mapping Functions 171
Programming Techniques 176
12 CICS with Channel Container Calling Convention 179
Introduction 180
Extracting from a CICS Channel Container Program 180
Mapping Editor User Interface 182
Mapping Editor IDL Interface Mapping Functions 189
Programming Techniques 194
13 IMS MPP Message Interface (IMS Connect) 197
Introduction 198
Extracting from an IMS MPP Message Interface Program 199
Mapping Editor User Interface 202
Mapping Editor IDL Interface Mapping Functions 210
Programming Techniques 215
14 IDL Extractor for COBOL Preferences 219
Create New Local Extractor Environment (z/OS, z/VSE, BS2000/OSD and IBM i) 222
Create New Local Extractor Environment for Micro Focus (UNIX and Windows) 226
Create New Remote Extractor Environment (z/OS) 230
Create New Remote Extractor Environment (BS2000/OSD) 234
15 COBOL to IDL Mapping 239
COBOL Data Type to Software AG IDL Mapping 241
DATA DIVISION Mapping 244
PROCEDURE DIVISION Mapping 254
Copybooks 255

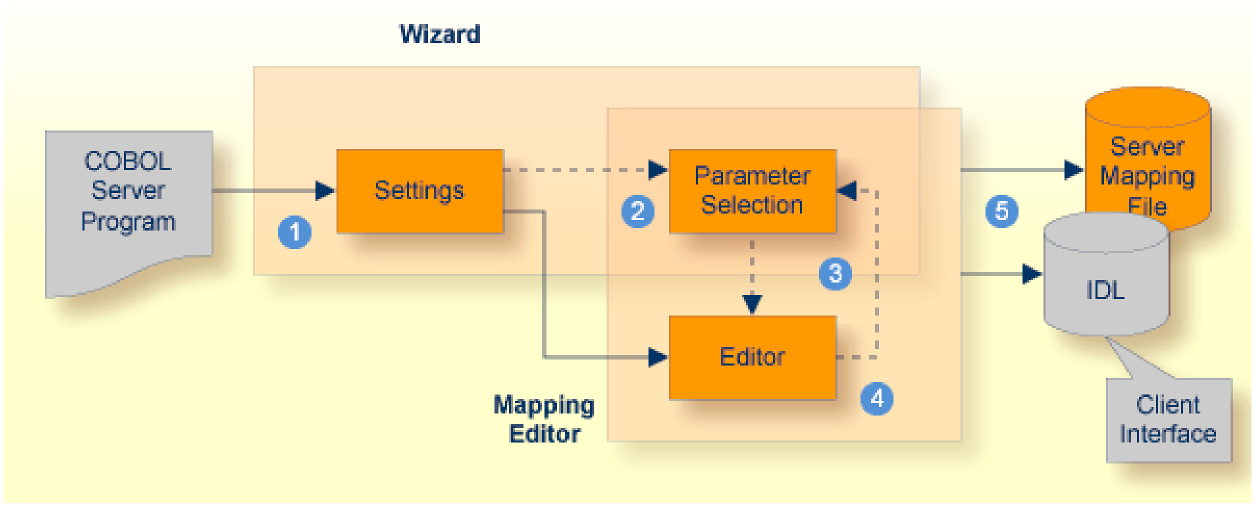
I Introduction to the IDL Extractor for COBOL

1 Introduction to the IDL Extractor for COBOL

- Introduction 4
- Extractor Wizard 5
- Mapping Editor 6
- Supported COBOL Interface Types 7
- Usage of Server Mapping Files 17

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*.
- **Server Mapping File**
  A server mapping file to complete the mapping is generated only if it is required by the RPC server during runtime to call the COBOL server. See *Usage of Server Mapping Files*.

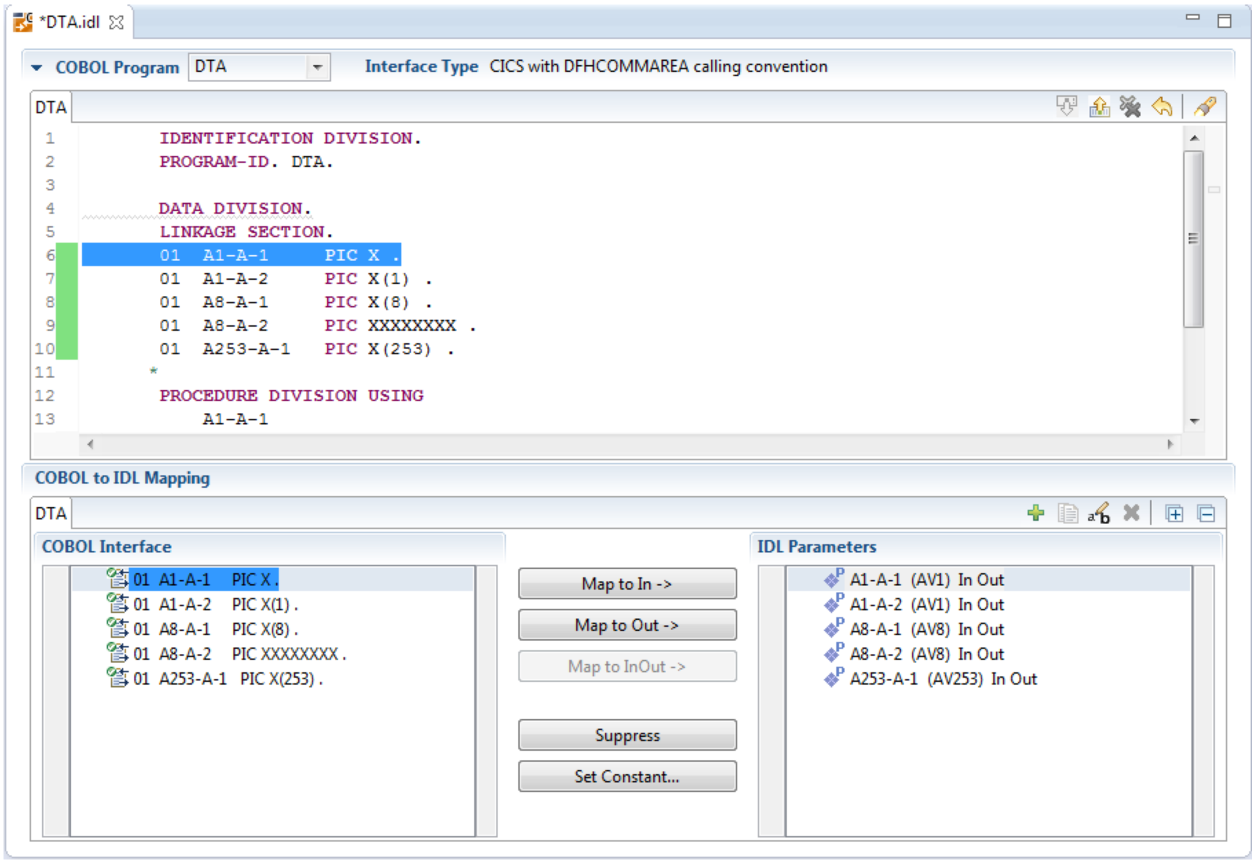
Extractor Wizard

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 EntireX Workbench 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/OSD LMS libraries. See *Extractor Service* in the z/OS administration and BS2000/OSD Batch RPC Server documentation. For this purpose, define a local or remote COBOL extractor environment. See ***IDL Extractor for 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 ***IDL Extractor for 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, Micro Focus standard calling conventions, 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 ***IDL Extractor for 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 CO-BOL - Overview***.

Mapping Editor



```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DTA.

       DATA DIVISION.
       LINKAGE SECTION.
       01  A1-A-1      PIC X .
       01  A1-A-2      PIC X(1) .
       01  A8-A-1      PIC X(8) .
       01  A8-A-2      PIC XXXXXXXX .
       01  A253-A-1    PIC X(253) .
      *
       PROCEDURE DIVISION USING
           A1-A-1
```

COBOL Interface:

```
01 A1-A-1   PIC X.
01 A1-A-2   PIC X(1) .
01 A8-A-1   PIC X(8) .
01 A8-A-2   PIC XXXXXXXX .
01 A253-A-1  PIC X(253) .
```

IDL Parameters:

```
A1-A-1  (AV1)  In Out
A1-A-2  (AV1)  In Out
A8-A-1  (AV8)  In Out
A8-A-2  (AV8)  In Out
A253-A-1  (AV253)  In Out
```

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 EntireX Workbench, 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:

- Supported CICS COBOL Interface Types
- Micro Focus with Standard Linkage Calling Convention
- Batch with Standard Linkage Calling Convention
- IMS MPP Message Interface (IMS Connect)
- IMS BMP with Standard Linkage Calling Convention
- What to do with other Interface Types?
- Compatibility between COBOL Interface Types and RPC Server

The interface type you are mostly working with can be set in the preferences. See ***IDL Extractor for 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.

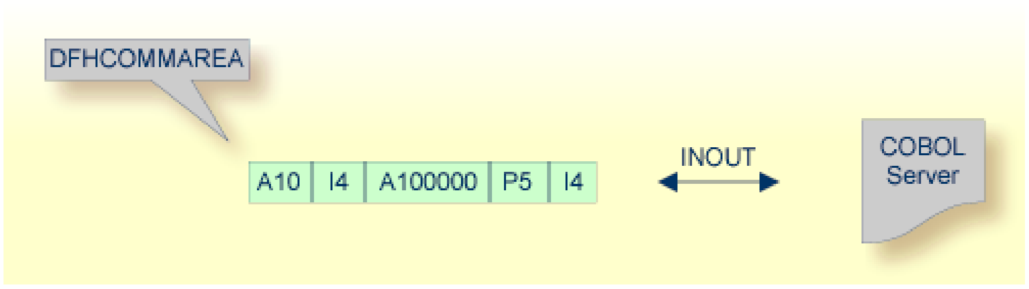
- The payload size of the CICS COBOL server is greater than 32 KB:

  - 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 (designed to assist with webMethods mainframe migration), because large buffer programs must reside on the same CICS region as the caller, that is, the CICS RPC Server (z/OS | z/VSE).
  - 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.
   . . .
```

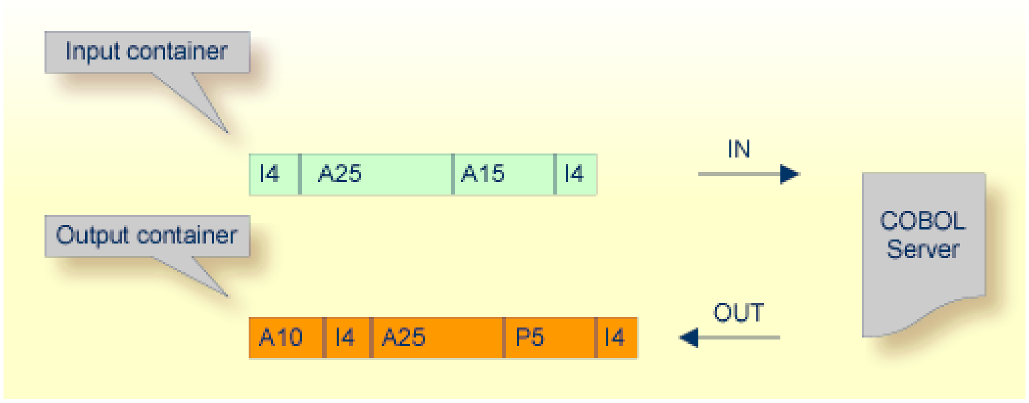
Most `DFHCOMMAREA` programs have a `DFHCOMMAREA` data item in their `LINKAGE SECTION` and may address this item in the `PROCEDURE DIVISION` header. If you find this in your COBOL source it's a clear indication it is a `DFHCOMMAREA` server program. But even if this is missing, it can be a

`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* 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.

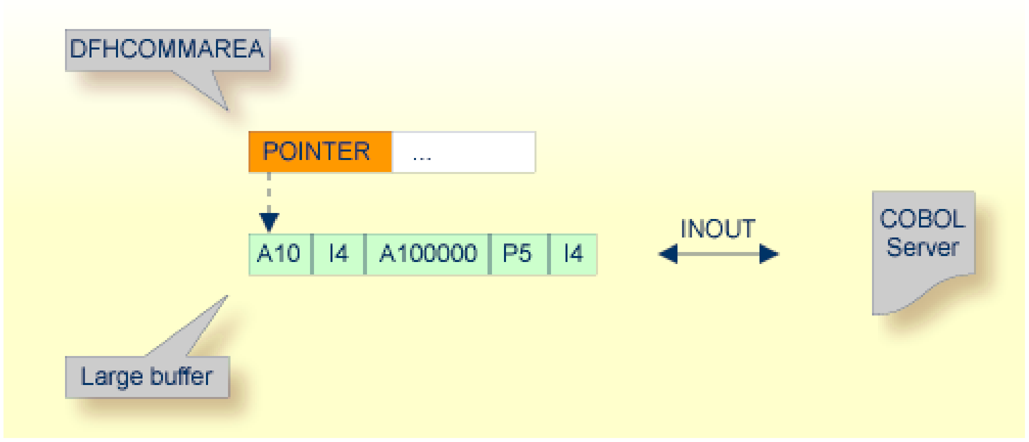
```
WORKING-STORAGE SECTION.
01 WS-CONTAINER-IN-NAME                PIC X(16) VALUE "CALC-IN".
01 WS-CONTAINER-OUT-NAME               PIC X(16) VALUE "CALC-OUT".
. . .
LINKAGE SECTION.
01 LS-CONTAINER-IN-LAYOUT.
   02 OPERATION                        PIC X(1).
   02 OPERAND1                         PIC 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* 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.
   . . .
   SET ADDRESS OF INOUT-BUFFER TO WM-LCB-OUTPUT-BUFFER.
```

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.

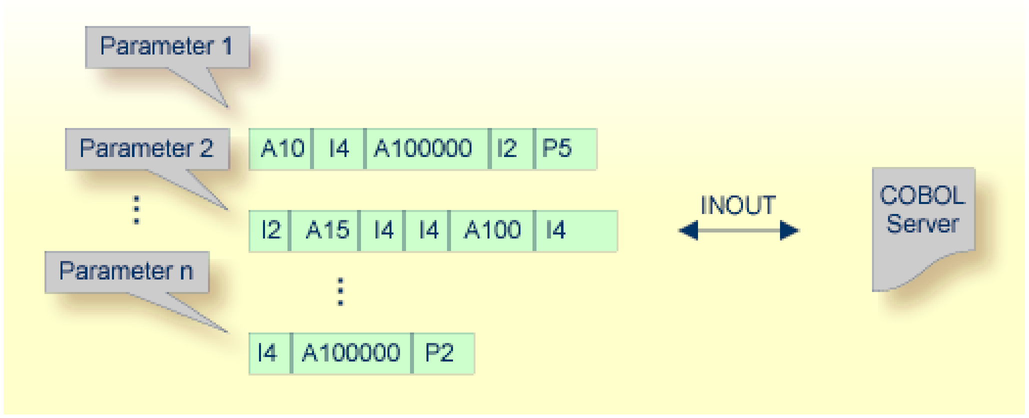
- data is described by separate structures, here `INOUT-BUFFER` in the linkage section.

  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*** for more information on extracting COBOL servers with this interface type.

Micro Focus with Standard Linkage Calling Convention

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



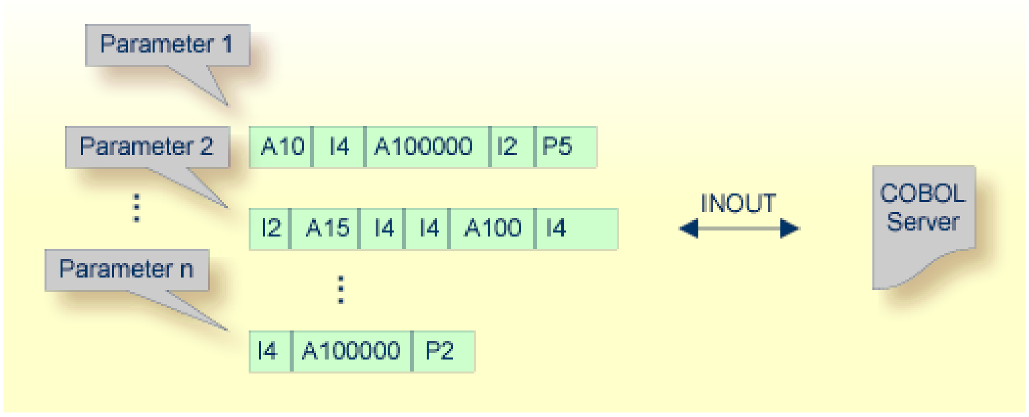
Technically, the generated COBOL server skeleton contains

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

See *Step 4: Define the Extraction Settings and Start Extraction* and *Micro Focus with Standard Linkage Calling Convention* 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.

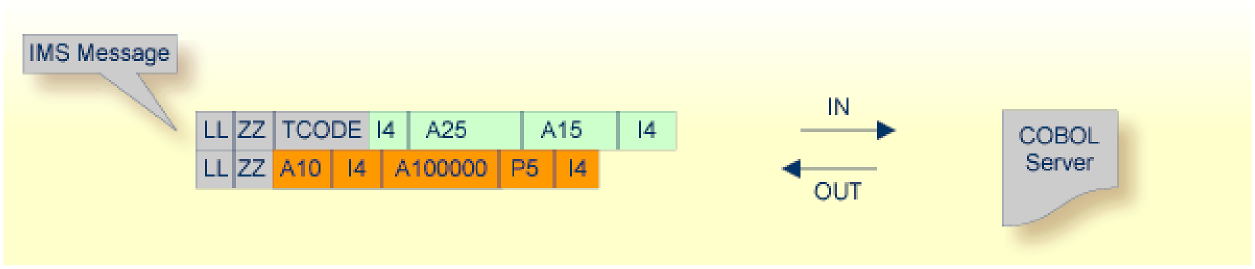


Technically, the COBOL server contains

- a parameter list: `PROCEDURE DIVISION USING PARM1 PARM2 ... PARMn`
- 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.

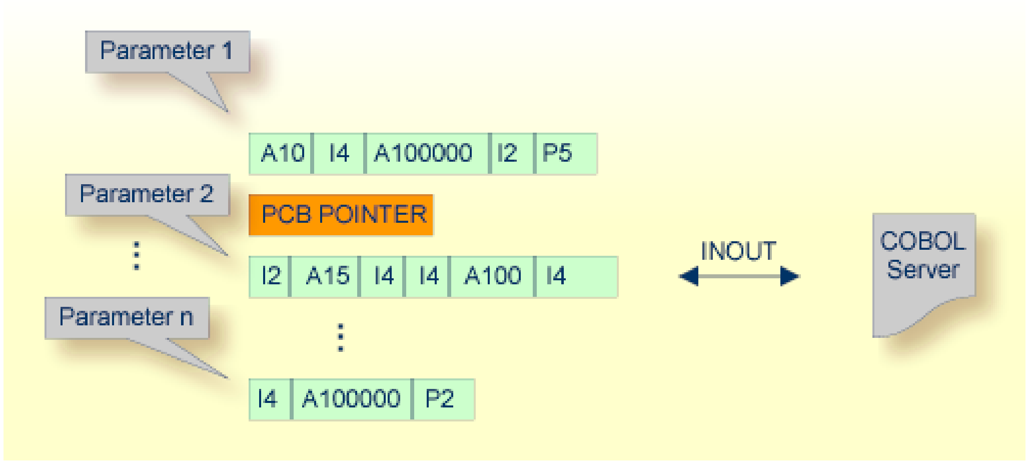
The COBOL server contains:

- 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.

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

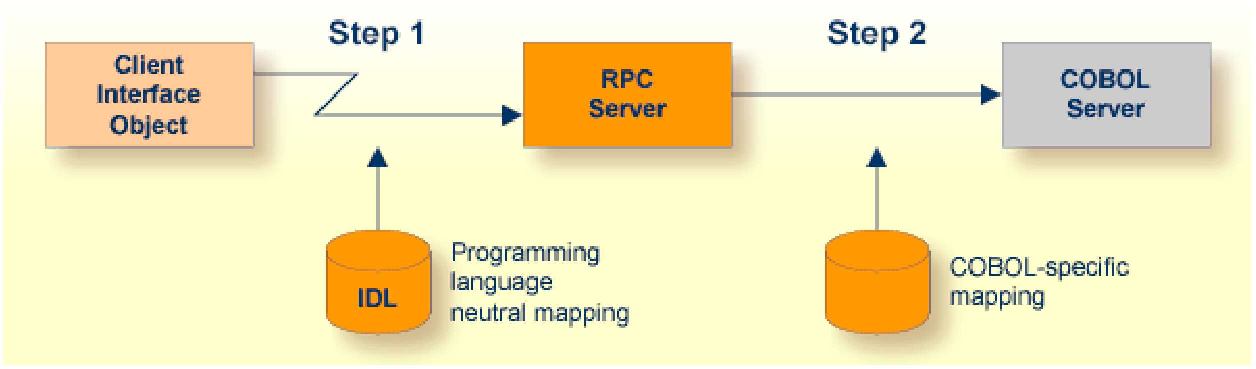
To call a server successfully, the RPC server used must support the interface type of the COBOL server. The table below gives an overview of possible combinations of an interface type and a supporting RPC server:

Interface Type	Supported by EntireX Adapter	Supported by RPC Server								
		z/OS			UNIX/Windows			BS2000/OSD	z/VSE	
		CICS	Batch	IMS	CICS ECI	Micro Focus	IMS Connect	Batch	CICS	Batch
CICS with `DFHCOMMAREA` Calling Convention (Extractor | Wrapper)	x	x			x				x	
CICS with `DFHCOMMAREA` Large Buffer Interface (Extractor | Wrapper)		x							x	
CICS with Channel Container Calling Convention (Extractor | Wrapper)		x								
Batch with Standard Linkage Calling Convention (Extractor | Wrapper)			x	x				x		x
Micro Focus with Standard Linkage Calling Convention (Extractor | Wrapper)						x				
IMS BMP with Standard Linkage Calling Convention (Extractor | Wrapper)				x						
IMS MPP Message Interface (IMS Connect) (Extractor)	x						x			

Usage of Server Mapping Files

There are many situations where the RPC server requires a server mapping file to correctly support special COBOL syntax such as `REDEFINES`, `SIGN LEADING` and `OCCURS DEPENDING ON` clauses, `LEVEL-88` fields, etc.

Server mapping files contain COBOL-specific mapping information that is not included in the IDL file, but is needed to successfully call the COBOL server program.



The RPC server marshals the data in a two-step process: the RPC request coming from the RPC client (Step 1) is completed with COBOL-specific mapping information taken from the server mapping file (Step 2). In this way the COBOL server can be called as expected.

The server mapping files are retrieved as a result of the *IDL Extractor for COBOL* extraction process and the *COBOL Wrapper* if a COBOL server is generated. See *When is a Server Mapping File Required?*.

There are *server*-side mapping files (*EntireX Workbench* files with extension .svm) and *client*-side mapping files (Workbench files with extension .cvm). See *Server Mapping Files for COBOL* and *How to Set the Type of Server Mapping Files*.

If you are using server-side mapping files, perform the following tasks:

- Customize the server-side mapping container. See *Server-side Mapping Files in the RPC Server* in the respective sections of the documentation.
- Deploy the files to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* in the respective sections of the documentation.

Note: For IMS Connect and CICS ECI connections with the webMethods EntireX Adapter, server-side mapping files are not deployed. They are wrapped into the Integration Server connection - the same as client-side mapping files. For RPC connections, deployment to the target RPC server is mandatory. See the EntireX Adapter documentation under **http://documentation.softwareag.com** > *webMethods Product Line*.

II Using the IDL Extractor for COBOL - Overview

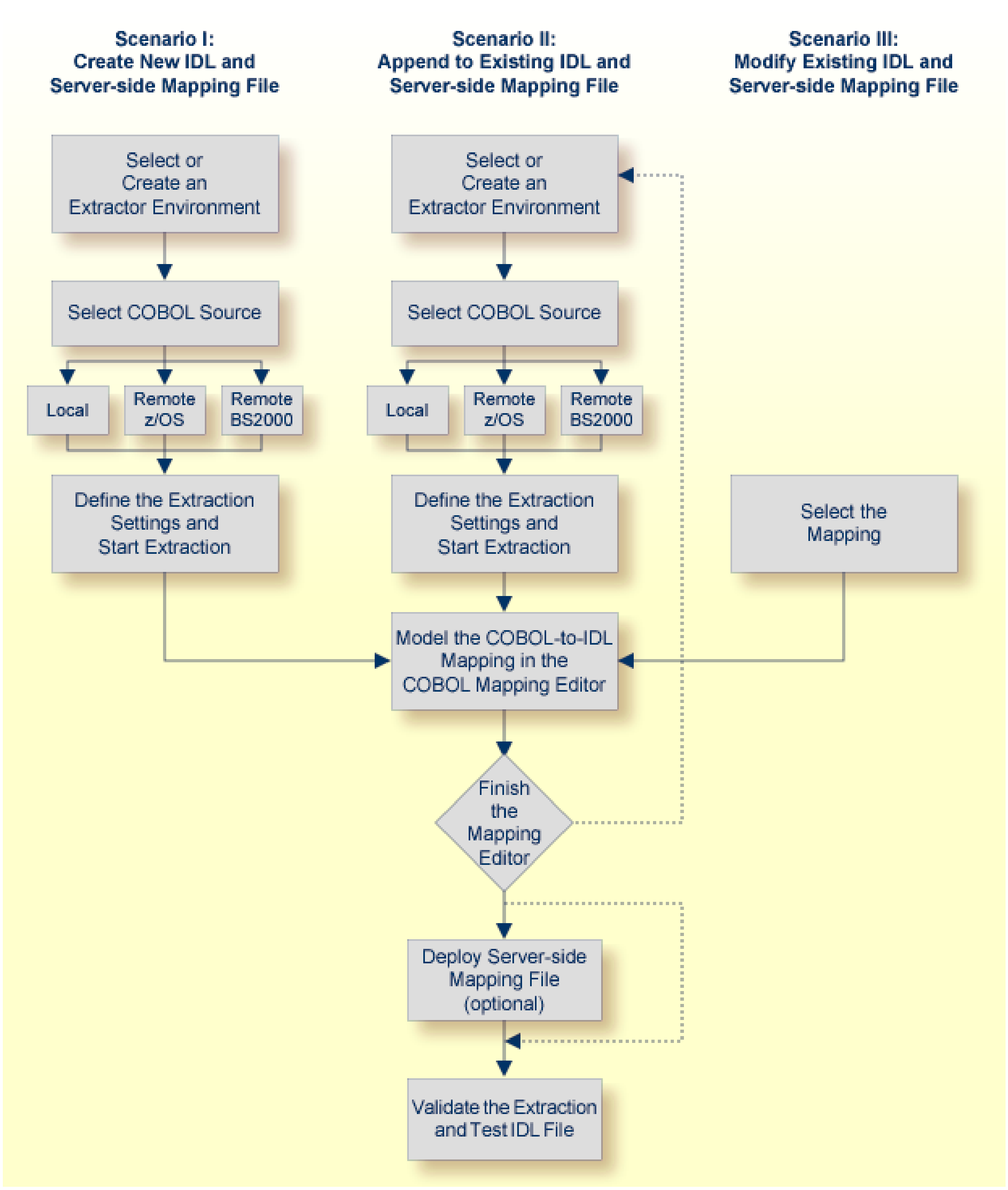
This chapter describes how to extract IDL from a COBOL source, using the IDL Extractor for COBOL, deploy, validate and test the extraction results. IDL extraction is supported by wizards, editors and generators.

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 COBOL Source
Local
Remote z/OS
Remote BS2000
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.

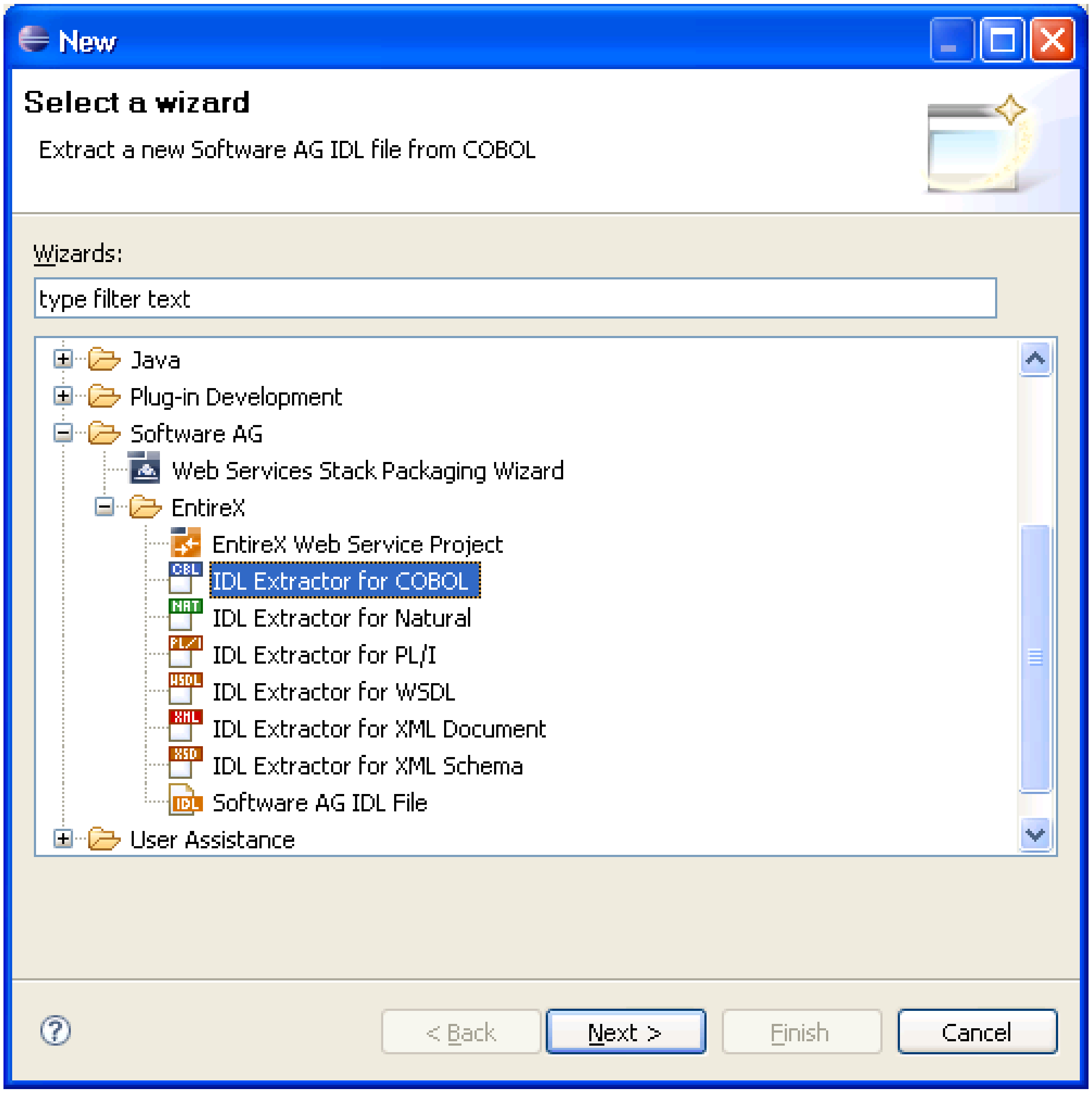
Make sure the COBOL source

- 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, z/VSE, BS2000/OSD and IBM i extractions
- is either written in COBOL fixed or variable format for Micro Focus UNIX or Windows extractions and your preferences are adjusted accordingly; see *Step 2: Define the Default Settings* under *Create New Local Extractor Environment for Micro Focus (UNIX and Windows)*.

2 Scenario I: Create New IDL and Server Mapping Files

- Step 1: Start the IDL Extractor for COBOL Wizard 24
- Step 2: Select a COBOL Extractor Environment or Create a New One 25
- Step 3: Select the COBOL Source 28
- Step 4: Define the Extraction Settings and Start Extraction 36
- Step 5: Select the COBOL Interface and Map to IDL Interface 45
- Step 6: Finishing the Mapping Editor 47
- Step 7: Validate the Extraction and Test the IDL File 49

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*.

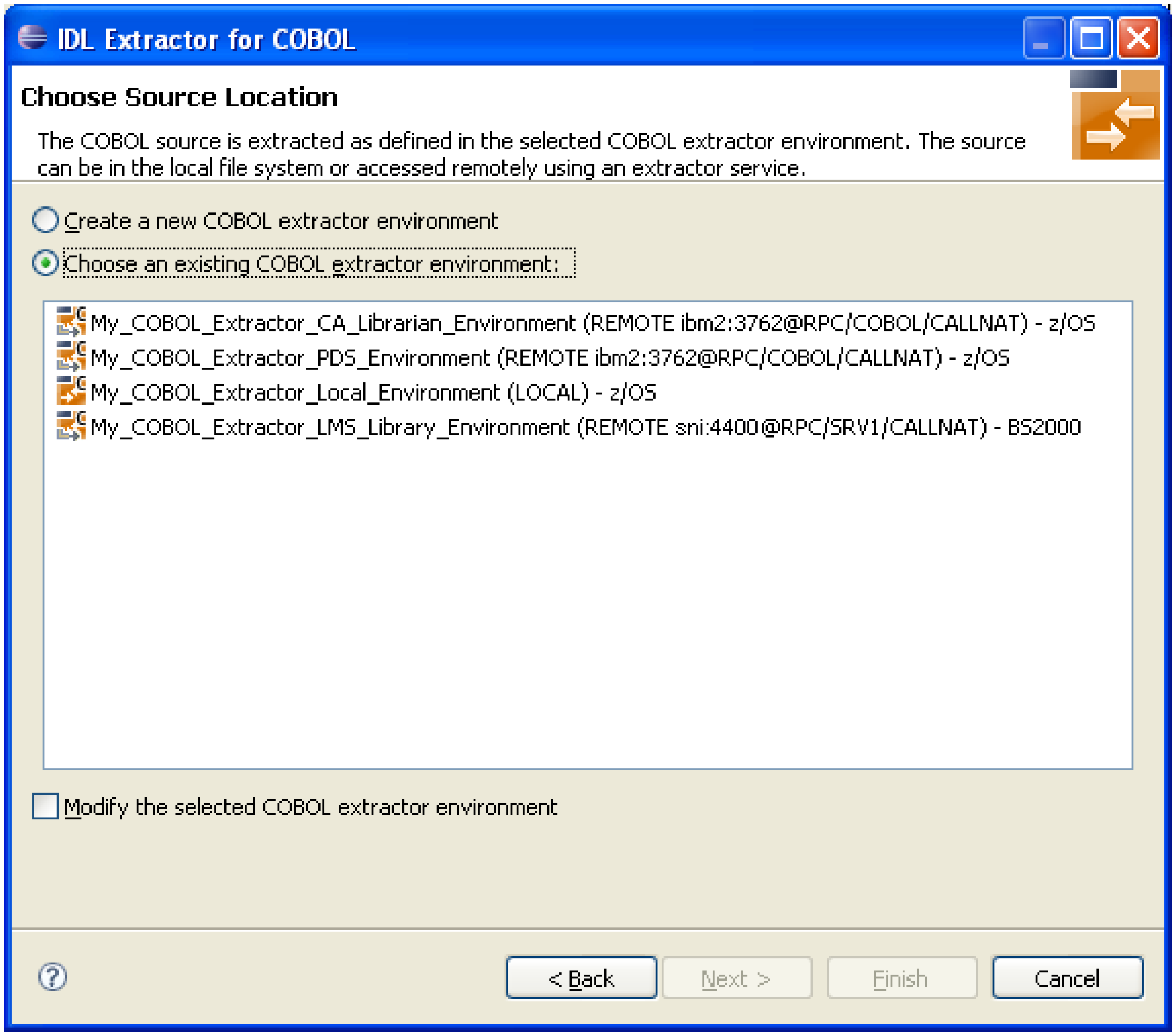
Step 2: Select a COBOL Extractor Environment or Create a New One

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 EntireX Workbench 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/OSD. See *Extractor Service* in the z/OS administration and BS2000/OSD Batch RPC Server documentation.


IDL Extractor for COBOL

Choose Source Location

The COBOL source is extracted as defined in the selected COBOL extractor environment. The source can be in the local file system or accessed remotely using an extractor service.

( ) Create a new COBOL extractor environment

(●) Choose an existing COBOL extractor environment:

- My_COBOL_Extractor_CA_Librarian_Environment (REMOTE ibm2:3762@RPC/COBOL/CALLNAT) - z/OS
- My_COBOL_Extractor_PDS_Environment (REMOTE ibm2:3762@RPC/COBOL/CALLNAT) - z/OS
- My_COBOL_Extractor_Local_Environment (LOCAL) - z/OS
- My_COBOL_Extractor_LMS_Library_Environment (REMOTE sni:4400@RPC/SRV1/CALLNAT) - BS2000

[ ] Modify the selected COBOL extractor environment

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, z/VSE, BS2000/OSD and IBM i) | Micro Focus (UNIX and Windows) in the IDL Extractor for COBOL documentation.
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/OSD in the IDL Extractor for COBOL documentation.
3. Continue with *Step 3: Select the COBOL Source* below.

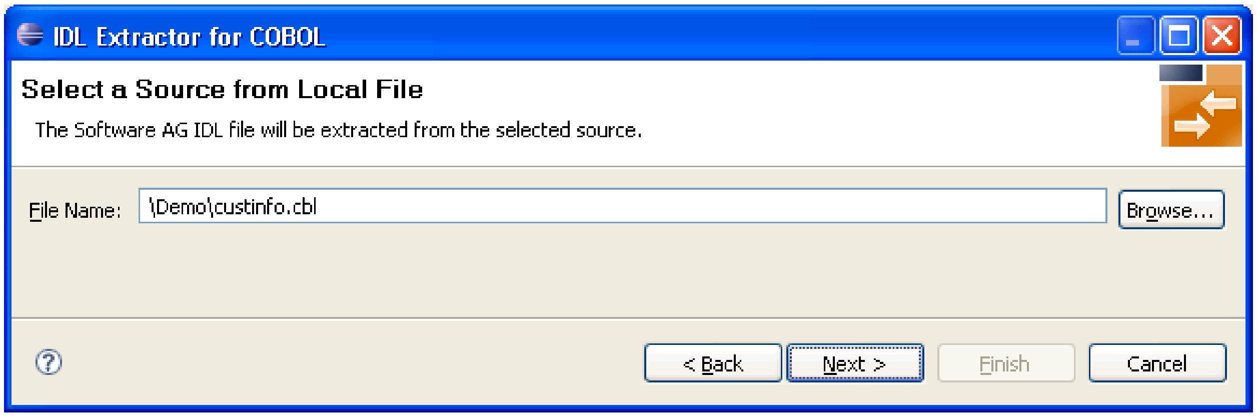
Step 3: Select the COBOL Source

Selecting the COBOL source is different depending on whether the COBOL source is stored locally on the same machine where the *EntireX Workbench* is running, or on a remote host computer.

- Selecting a COBOL Source Stored Locally
- Selecting a Member from a Partitioned Data Set on Remote Host (z/OS)
- Selecting a Member 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)
- Selecting an Element (S) from an LMS Library on Remote Host (BS2000/OSD)

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, z/VSE, BS2000/OSD and IBM i) | Micro Focus (UNIX and Windows) in the IDL Extractor for COBOL documentation. To browse for the COBOL source file, choose **Browse**.



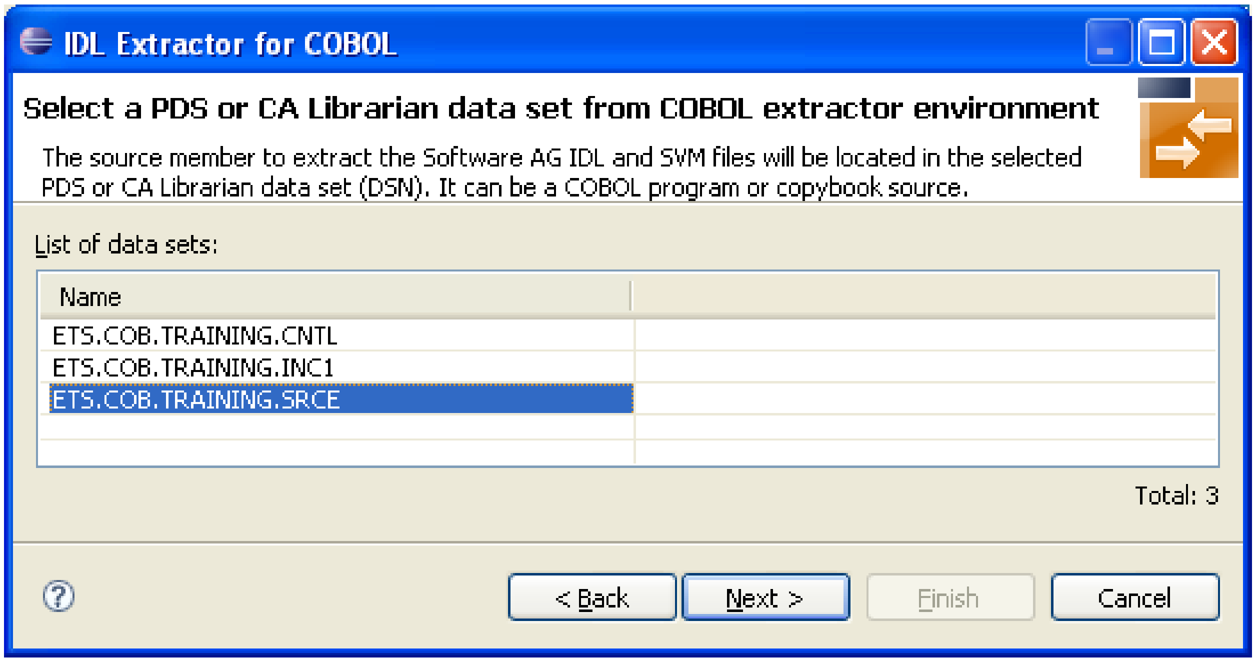
Select a Source from Local File

The Software AG IDL file will be extracted from the selected source.

File Name: \Demo\custinfo.cbl

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 *Create New Remote Extractor Environment (z/OS)* under *IDL Extractor for COBOL Preferences*.


IDL Extractor for COBOL

Select a PDS or CA Librarian data set from COBOL extractor environment

The source member to extract the Software AG IDL and SVM files will be located in the selected PDS or CA Librarian data set (DSN). It can be a COBOL program or copybook source.

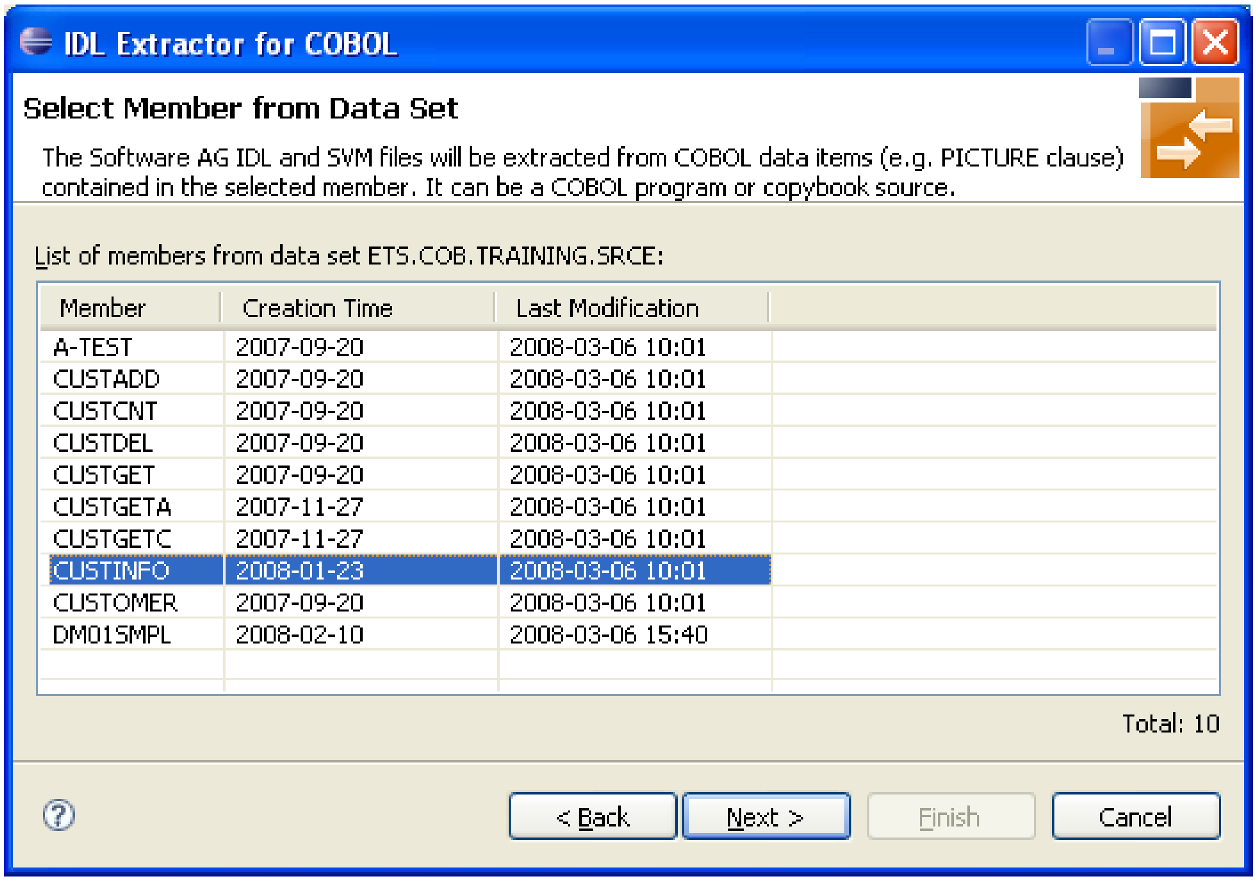
List of data sets:

Name	
ETS.COB.TRAINING.CNTL	
ETS.COB.TRAINING.INC1	
ETS.COB.TRAINING.SRCE	

Total: 3

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 *Step 3: Define the Remote Extractor Environment* under *IDL Extractor for COBOL Preferences*.



Select Member from Data Set

The Software AG IDL and SVM files will be extracted from COBOL data items (e.g. PICTURE clause) contained in the selected member. It can be a COBOL program or copybook source.

List of members from data set ETS.COB.TRAINING.SRCE:

Member	Creation Time	Last Modification
A-TEST	2007-09-20	2008-03-06 10:01
CUSTADD	2007-09-20	2008-03-06 10:01
CUSTCNT	2007-09-20	2008-03-06 10:01
CUSTDEL	2007-09-20	2008-03-06 10:01
CUSTGET	2007-09-20	2008-03-06 10:01
CUSTGETA	2007-11-27	2008-03-06 10:01
CUSTGETC	2007-11-27	2008-03-06 10:01
CUSTINFO	2008-01-23	2008-03-06 10:01
CUSTOMER	2007-09-20	2008-03-06 10:01
DM01SMPL	2008-02-10	2008-03-06 15:40

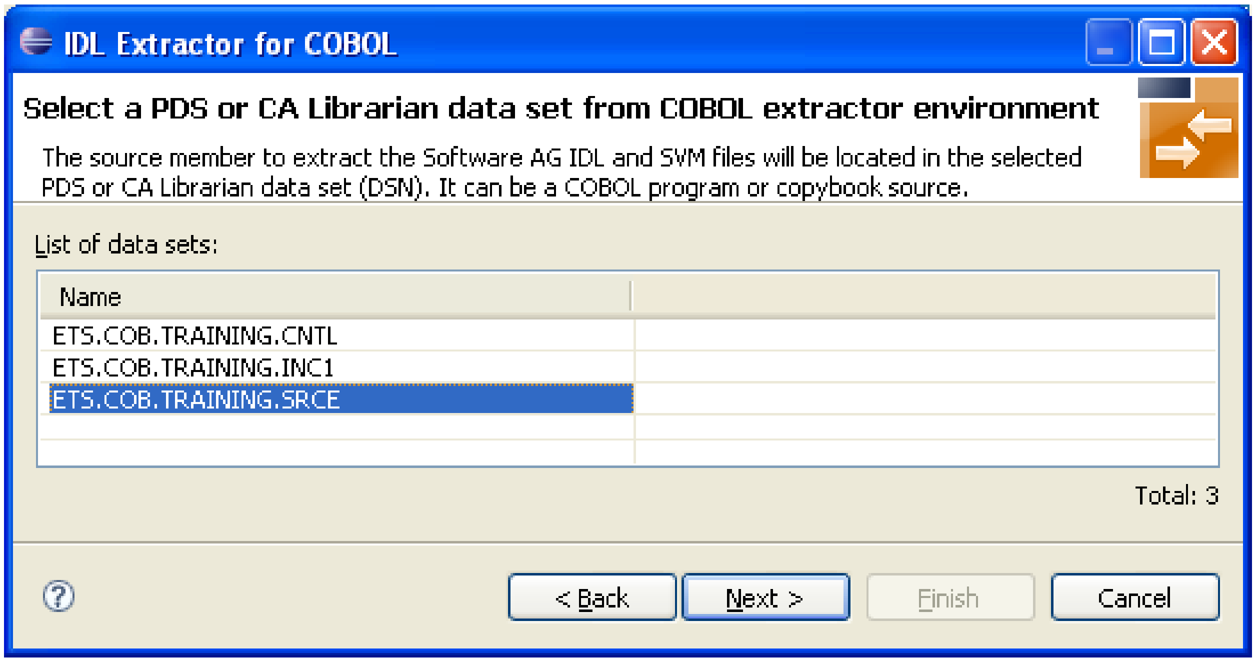
Total: 10

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 *Create New Remote Extractor Environment (z/OS)* under *IDL Extractor for COBOL Preferences*.



Select a PDS or CA Librarian data set from COBOL extractor environment

The source member to extract the Software AG IDL and SVM files will be located in the selected PDS or CA Librarian data set (DSN). It can be a COBOL program or copybook source.

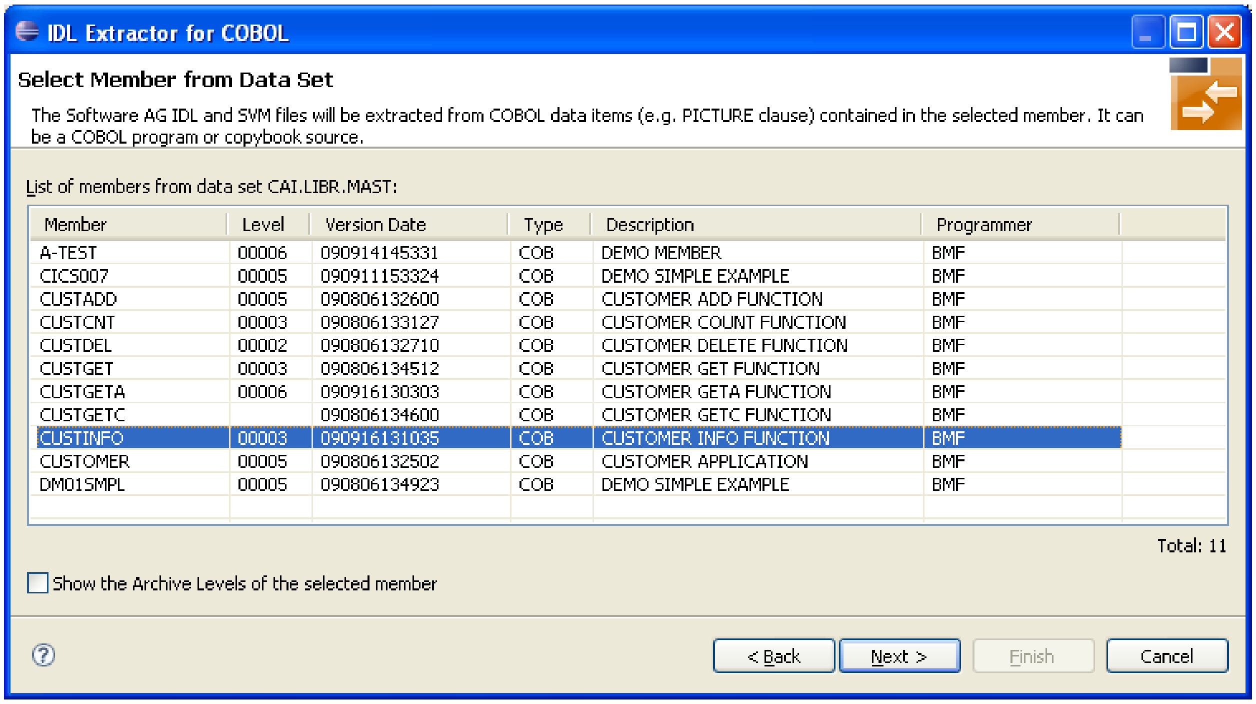
List of data sets:

Name	
ETS.COB.TRAINING.CNTL	
ETS.COB.TRAINING.INC1	
ETS.COB.TRAINING.SRCE	

Total: 3

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 *Step 3: Define the Remote Extractor Environment* under *IDL Extractor for COBOL Preferences*.



Select Member from Data Set

The Software AG IDL and SVM files will be extracted from COBOL data items (e.g. PICTURE clause) contained in the selected member. It can be a COBOL program or copybook source.

List of members from data set CAI.LIBR.MAST:

Member	Level	Version Date	Type	Description	Programmer
A-TEST	00006	090914145331	COB	DEMO MEMBER	BMF
CICS007	00005	090911153324	COB	DEMO SIMPLE EXAMPLE	BMF
CUSTADD	00005	090806132600	COB	CUSTOMER ADD FUNCTION	BMF
CUSTCNT	00003	090806133127	COB	CUSTOMER COUNT FUNCTION	BMF
CUSTDEL	00002	090806132710	COB	CUSTOMER DELETE FUNCTION	BMF
CUSTGET	00003	090806134512	COB	CUSTOMER GET FUNCTION	BMF
CUSTGETA	00006	090916130303	COB	CUSTOMER GETA FUNCTION	BMF
CUSTGETC		090806134600	COB	CUSTOMER GETC FUNCTION	BMF
CUSTINFO	00003	090916131035	COB	CUSTOMER INFO FUNCTION	BMF
CUSTOMER	00005	090806132502	COB	CUSTOMER APPLICATION	BMF
DM01SMPL	00005	090806134923	COB	DEMO SIMPLE EXAMPLE	BMF

Total: 11

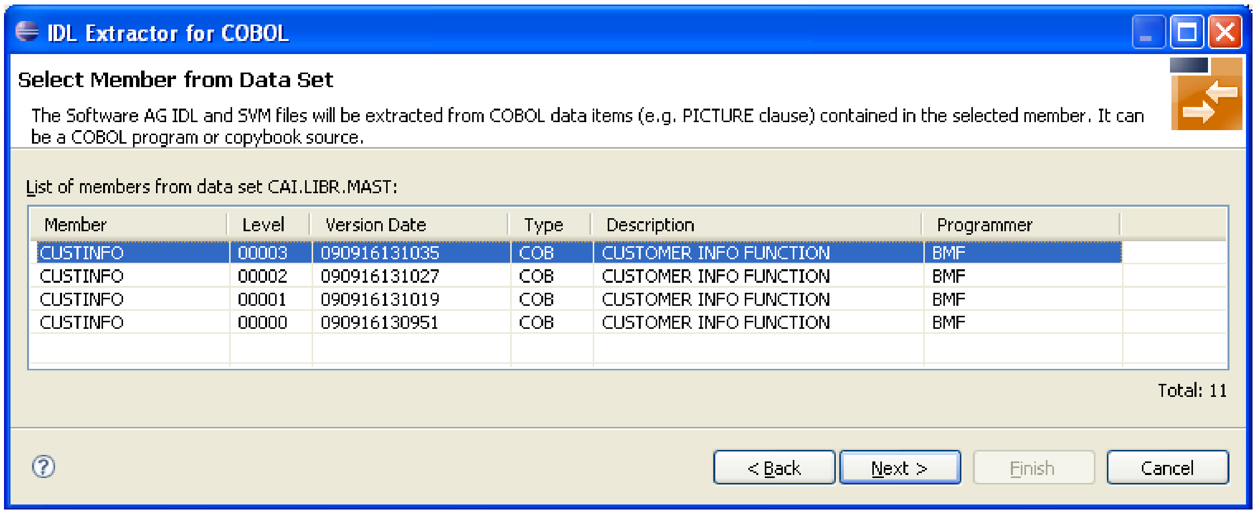
☐ Show the Archive Levels of the selected member

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 Member from Data Set

The Software AG IDL and SVM files will be extracted from COBOL data items (e.g. PICTURE clause) contained in the selected member. It can be a COBOL program or copybook source.

List of members from data set CAI.LIBR.MAST:

Member	Level	Version Date	Type	Description	Programmer
CUSTINFO	00003	090916131035	COB	CUSTOMER INFO FUNCTION	BMF
CUSTINFO	00002	090916131027	COB	CUSTOMER INFO FUNCTION	BMF
CUSTINFO	00001	090916131019	COB	CUSTOMER INFO FUNCTION	BMF
CUSTINFO	00000	090916130951	COB	CUSTOMER INFO FUNCTION	BMF

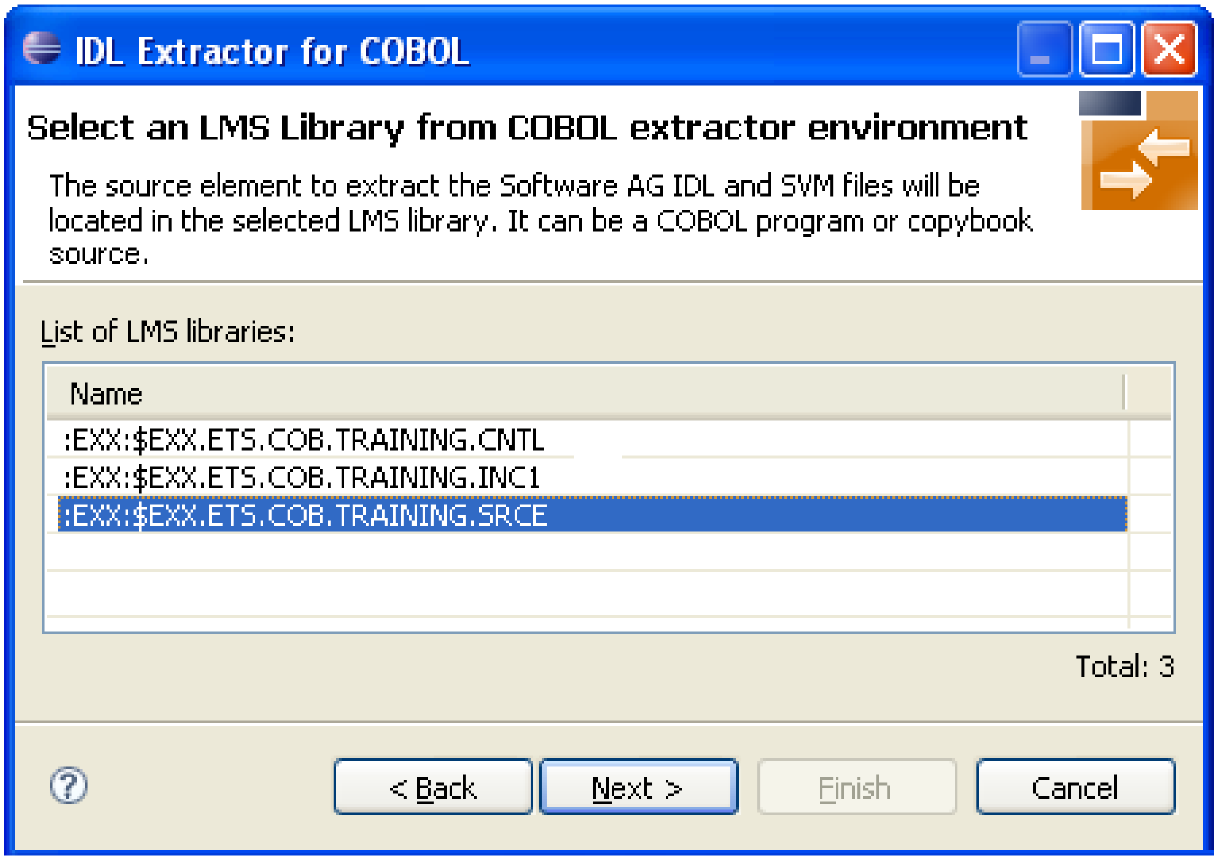
Total: 11

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/OSD)

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 *Create New Remote Extractor Environment (BS2000/OSD)* under *IDL Extractor for COBOL Preferences* .



Select an LMS Library from COBOL extractor environment

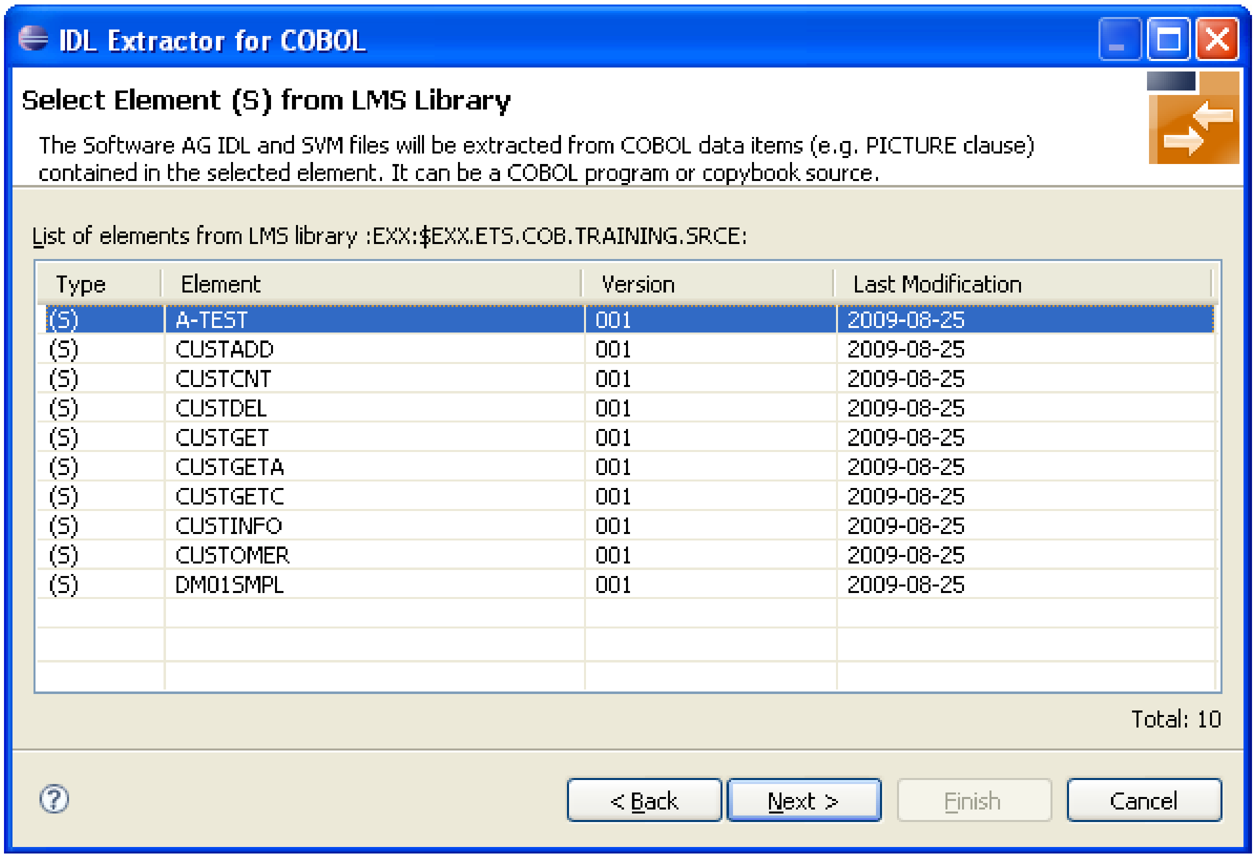
The source element to extract the Software AG IDL and SVM files will be located in the selected LMS library. It can be a COBOL program or copybook source.

List of LMS libraries:

Name
:EXX:$EXX.ETS.COB.TRAINING.CNTL
:EXX:$EXX.ETS.COB.TRAINING.INC1
:EXX:$EXX.ETS.COB.TRAINING.SRCE

Total: 3

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 *Step 3: Define the Remote Extractor Environment* under *IDL Extractor for COBOL Preferences*.



Select Element (S) from LMS Library

The Software AG IDL and SVM files will be extracted from COBOL data items (e.g. PICTURE clause) contained in the selected element. It can be a COBOL program or copybook source.

List of elements from LMS library :EXX:$EXX.ETS.COB.TRAINING.SRCE:

Type	Element	Version	Last Modification
(S)	A-TEST	001	2009-08-25
(S)	CUSTADD	001	2009-08-25
(S)	CUSTCNT	001	2009-08-25
(S)	CUSTDEL	001	2009-08-25
(S)	CUSTGET	001	2009-08-25
(S)	CUSTGETA	001	2009-08-25
(S)	CUSTGETC	001	2009-08-25
(S)	CUSTINFO	001	2009-08-25
(S)	CUSTOMER	001	2009-08-25
(S)	DM01SMPL	001	2009-08-25

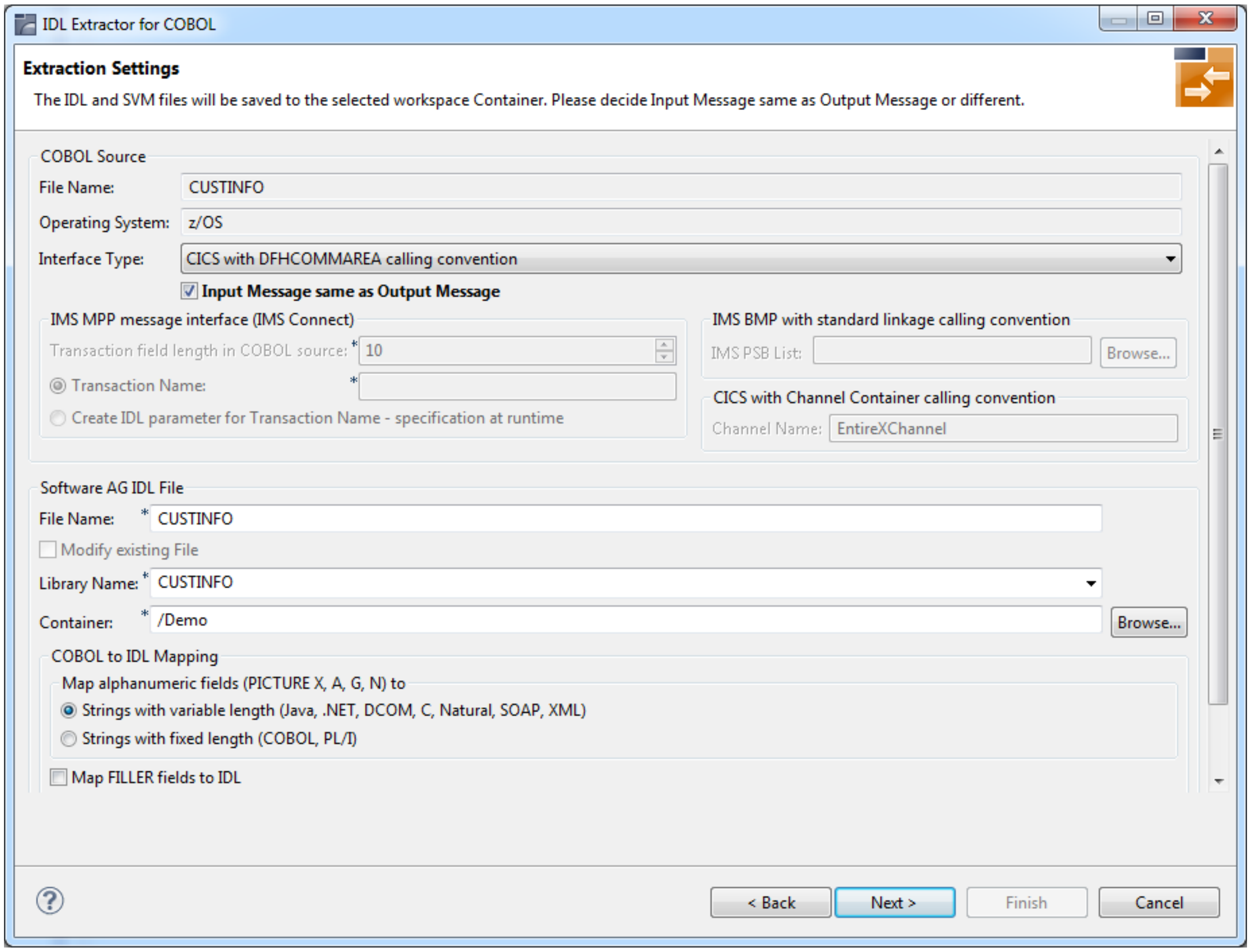
Total: 10

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.



IDL Extractor for COBOL

Extraction Settings

The IDL and SVM files will be saved to the selected workspace Container. Please decide Input Message same as Output Message or different.

COBOL Source

File Name: CUSTINFO

Operating System: z/OS

Interface Type: CICS with DFHCOMMAREA calling convention

[✓] Input Message same as Output Message

IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: * 10

● Transaction Name: *

○ Create IDL parameter for Transaction Name - specification at runtime

IMS BMP with standard linkage calling convention

IMS PSB List:

CICS with Channel Container calling convention

Channel Name: EntireXChannel

Software AG IDL File

File Name: * CUSTINFO

[ ] Modify existing File

Library Name: * CUSTINFO

Container: * /Demo

COBOL to IDL Mapping

Map alphanumeric fields (PICTURE X, A, G, N) to

● Strings with variable length (Java, .NET, DCOM, C, Natural, SOAP, XML)

○ Strings with fixed length (COBOL, PL/I)

[ ] Map FILLER fields to IDL

Operating System

The operating system is already defined in the extractor environment in the IDL Extractor for COBOL preferences, see *IDL Extractor for 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 RPC server 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 2: Redefines*
  - *Example 3: Buffer Technique*
  - *Example 4: 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 2: Redefines*
  - *Example 3: Buffer Technique*
  - *Example 4: 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**.
- **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.
- **MicroFocus 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.

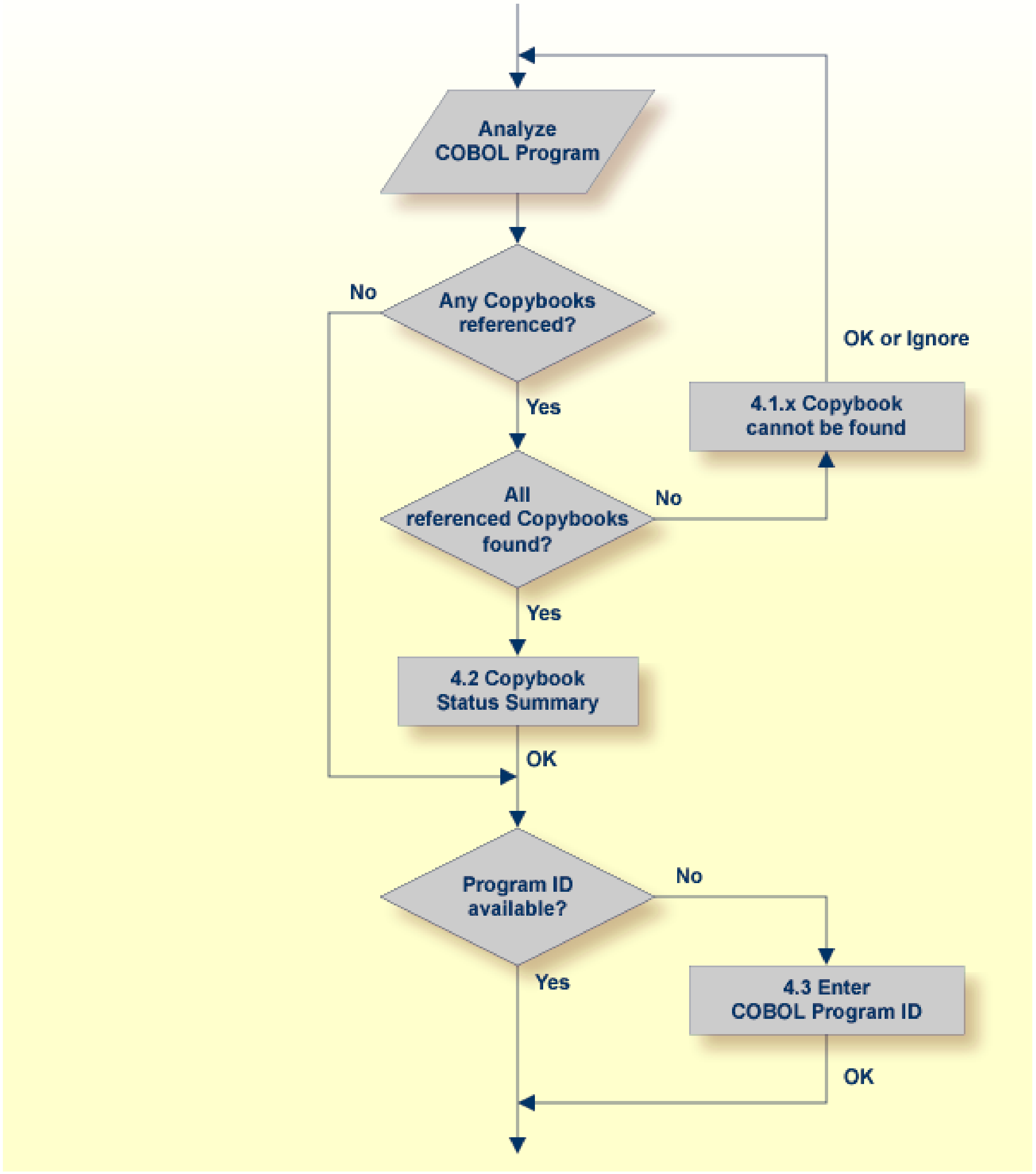
  For the interface type "Micro Focus with standard linkage calling convention" and if the COBOL server is an operating system standard library (.so|.sl on UNIX or .dll on Windows) or a Micro Focus proprietary library (*.lbr), the IDL library name used must match the operating system file name. For Micro Focus proprietary formats, intermediate code (*.int) and generated code (*.gnt), any IDL library name can be used. See *Locating and Calling the Target Server* under *Administering the Micro Focus RPC Server* in the Micro Focus RPC Server documentation.
- **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:

- If the target RPC clients support variable length strings without any restriction, we recommend you map alphanumeric fields to "Strings with variable length". This is true for most modern target environments such as Java, .NET, DCOM, C, Natural, SOAP, XML.
- If the target RPC clients do not support variable length strings or support them with restrictions, we recommend you map alphanumeric fields to "Strings with fixed length"
- 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.

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

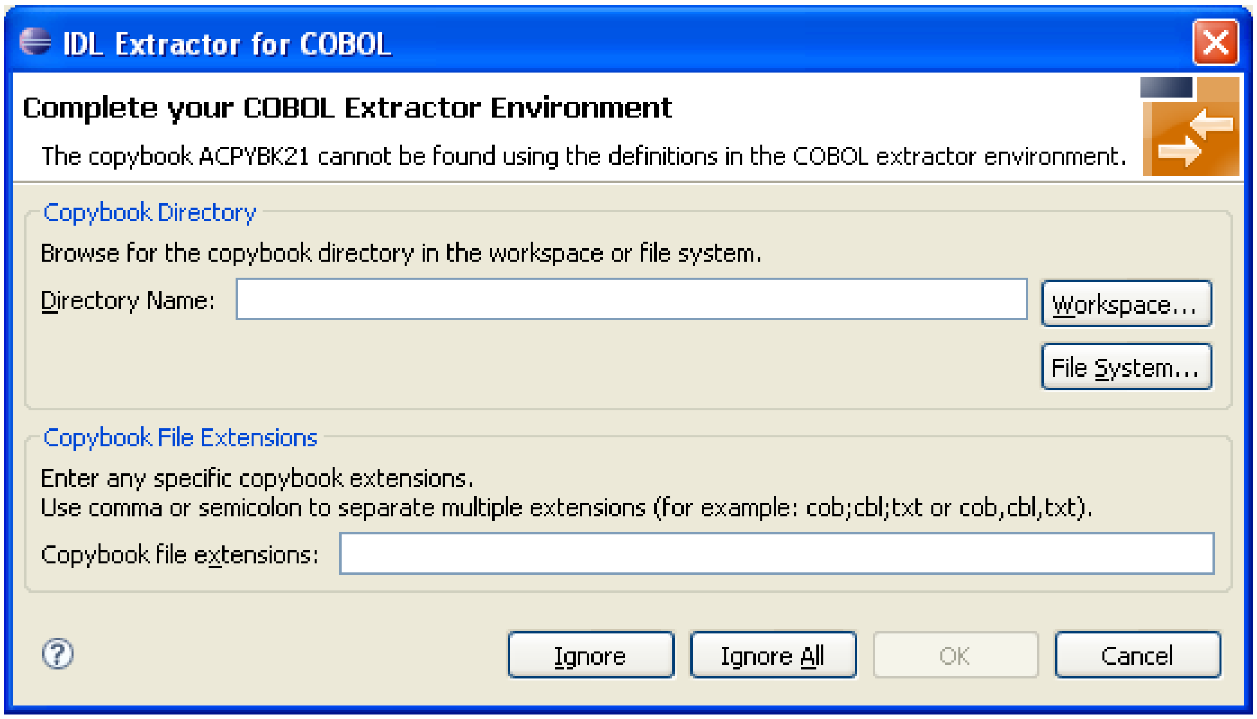

Analyze COBOL Program
Any Copybooks referenced?
No
Yes
All referenced Copybooks found?
No
OK or Ignore
4.1.x Copybook cannot be found
Yes
4.2 Copybook Status Summary
OK
Program ID available?
No
Yes
4.3 Enter COBOL Program ID
OK

- 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/OSD) in the IDL Extractor for COBOL documentation 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. 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.


IDL Extractor for COBOL

Complete your COBOL Extractor Environment

The copybook ACPYBK21 cannot be found using the definitions in the COBOL extractor environment.

Copybook Directory

Browse for the copybook directory in the workspace or file system.

Directory Name:

Copybook File Extensions

Enter any specific copybook extensions.
Use comma or semicolon to separate multiple extensions (for example: cob;cbl;txt or cob,cbl,txt).

Copybook file extensions:

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.

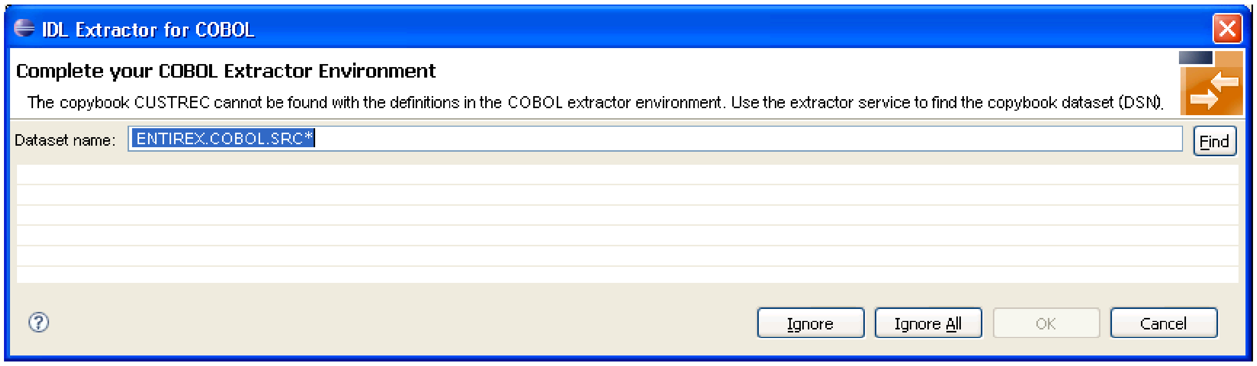
2 Check the copybook file extensions. Both will be saved in the COBOL extractor preferences and reused in further extractions.

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.


IDL Extractor for COBOL

Complete your COBOL Extractor Environment

The copybook CUSTREC cannot be found with the definitions in the COBOL extractor environment. Use the extractor service to find the copybook dataset (DSN).

Dataset name: ENTIREX.COBOL.SRC*

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

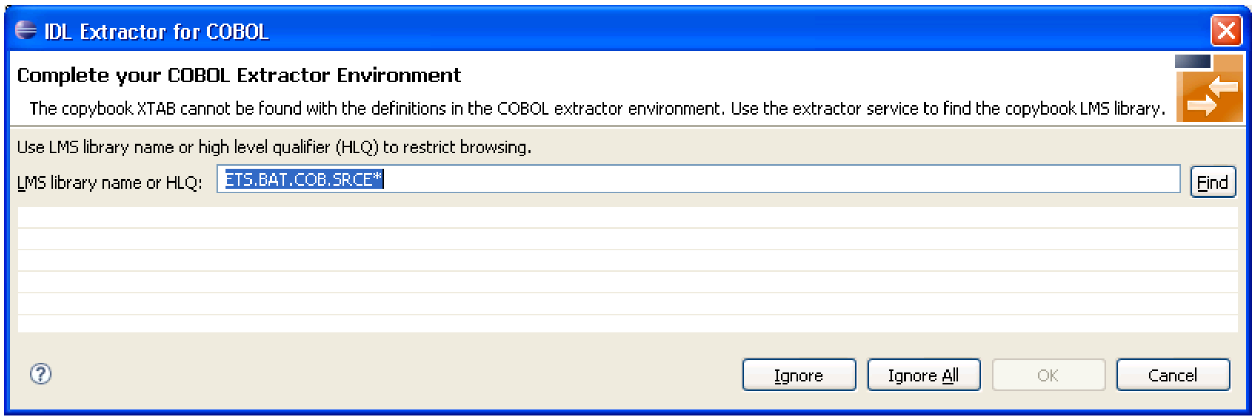
1. Choose **Ignore All** and go back to *Step 4: Define the Extraction Settings and Start Extraction*.
2. Choose **Next** to start extraction again.

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/OSD Remote Extraction

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


IDL Extractor for COBOL

Complete your COBOL Extractor Environment

The copybook XTAB cannot be found with the definitions in the COBOL extractor environment. Use the extractor service to find the copybook LMS library.

Use LMS library name or high level qualifier (HLQ) to restrict browsing.

LMS library name or HLQ: ETS.BAT.COB.SRCE*

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

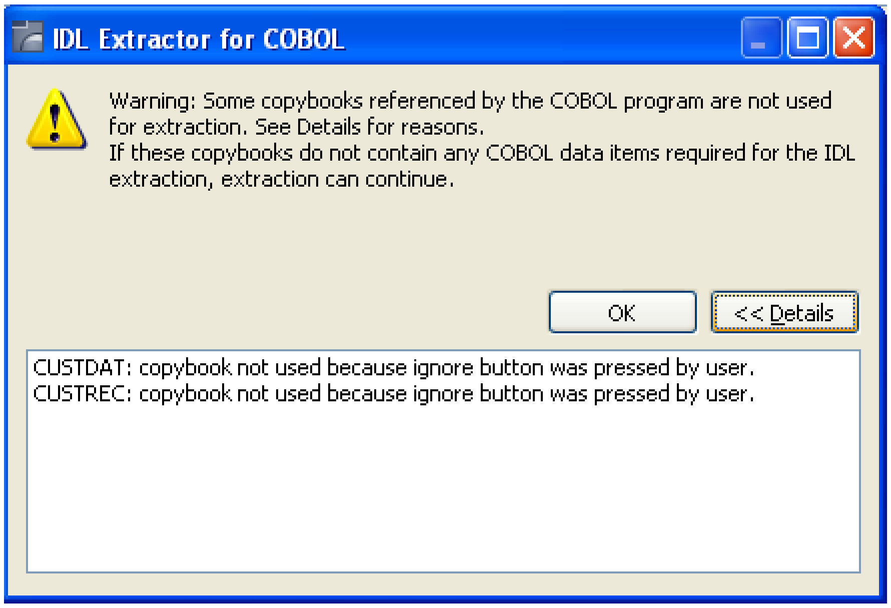
1 Choose **Find** to browse for the copybook LMS library. It will be saved in the COBOL extractor preferences and reused in further extractions.

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.



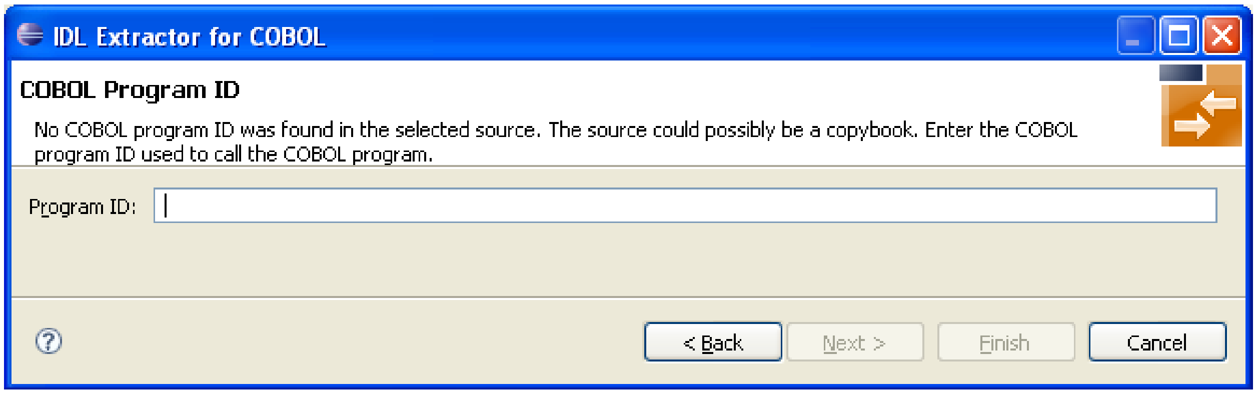
Warning: Some copybooks referenced by the COBOL program are not used for extraction. See Details for reasons.
If these copybooks do not contain any COBOL data items required for the IDL extraction, extraction can continue.

```
CUSTDAT: copybook not used because ignore button was pressed by user.
CUSTREC: copybook not used because ignore button was pressed by user.
```

- 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.



COBOL Program ID

No COBOL program ID was found in the selected source. The source could possibly be a copybook. Enter the COBOL program ID used to call the COBOL program.

Program ID:

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

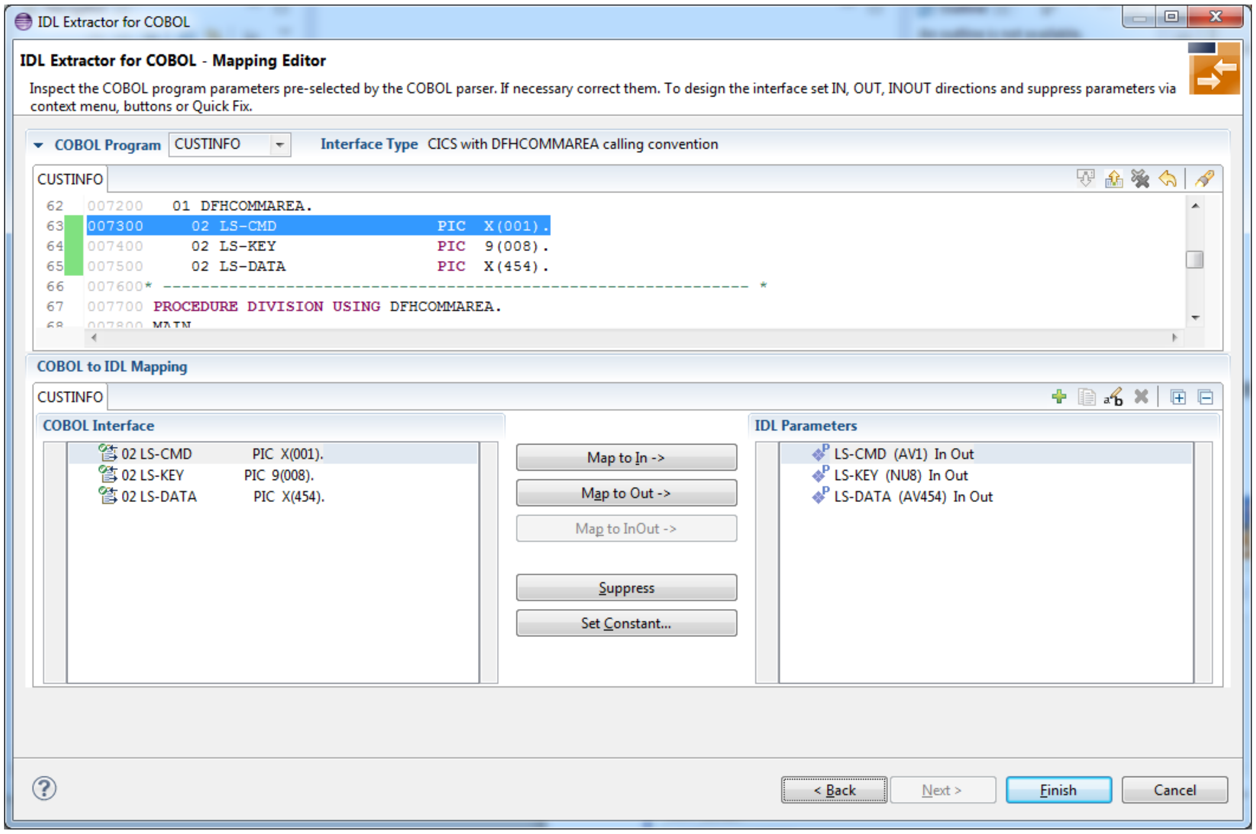
1. Enter the COBOL program ID.
2. Choose **OK** to continue with *Step 5: Select the COBOL Interface and Map to IDL Interface*.

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

In general, mapping the COBOL data items to IDL with the COBOL Mapping Editor is a two-step process:

1. First select the COBOL data items describing the COBOL interface from the COBOL source view. In this example the COBOL interface is preselected as defined in the `PROCEDURE DIVISION USING` clause.
2. Then map the COBOL interface to the IDL interface.

See the guidelines on IDL extraction below for further information.



IDL Extractor for COBOL - Mapping Editor

Inspect the COBOL program parameters pre-selected by the COBOL parser. If necessary correct them. To design the interface set IN, OUT, INOUT directions and suppress parameters via context menu, buttons or Quick Fix.

COBOL Program: CUSTINFO   Interface Type: CICS with DFHCOMMAREA calling convention

```
007200    01 DFHCOMMAREA.
007300       02 LS-CMD                  PIC  X(001).
007400       02 LS-KEY                  PIC  9(008).
007500       02 LS-DATA                 PIC  X(454).
007600* -------------------------------------------------------------- *
007700 PROCEDURE DIVISION USING DFHCOMMAREA.
007800 MAIN
```

COBOL Interface		IDL Parameters
02 LS-CMD	PIC X(001).	LS-CMD (AV1) In Out
02 LS-KEY	PIC 9(008).	LS-KEY (NU8) In Out
02 LS-DATA	PIC X(454).	LS-DATA (AV454) In Out

The following table provides guidelines on IDL extraction per interface type. 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	
Batch	Standard Linkage	
IMS	BMP with Standard Linkage	
	MPP Message Interface (IMS Connect)	
Micro Focus	Standard Linkage	

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 CICS input message is the same as the structure of the CICS output message.
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 CICS input message is different to the structure of the CICS output message (that is, the output overlays the input).
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 2: Redefines*
   - *Example 3: Buffer Technique*
   - *Example 4: COBOL SET ADDRESS Statements*
5. See the following COBOL server examples for CICS input message *different to* CICS output message:
   - *Example 2: Redefines*
   - *Example 3: Buffer Technique*
   - *Example 4: COBOL SET ADDRESS Statements*

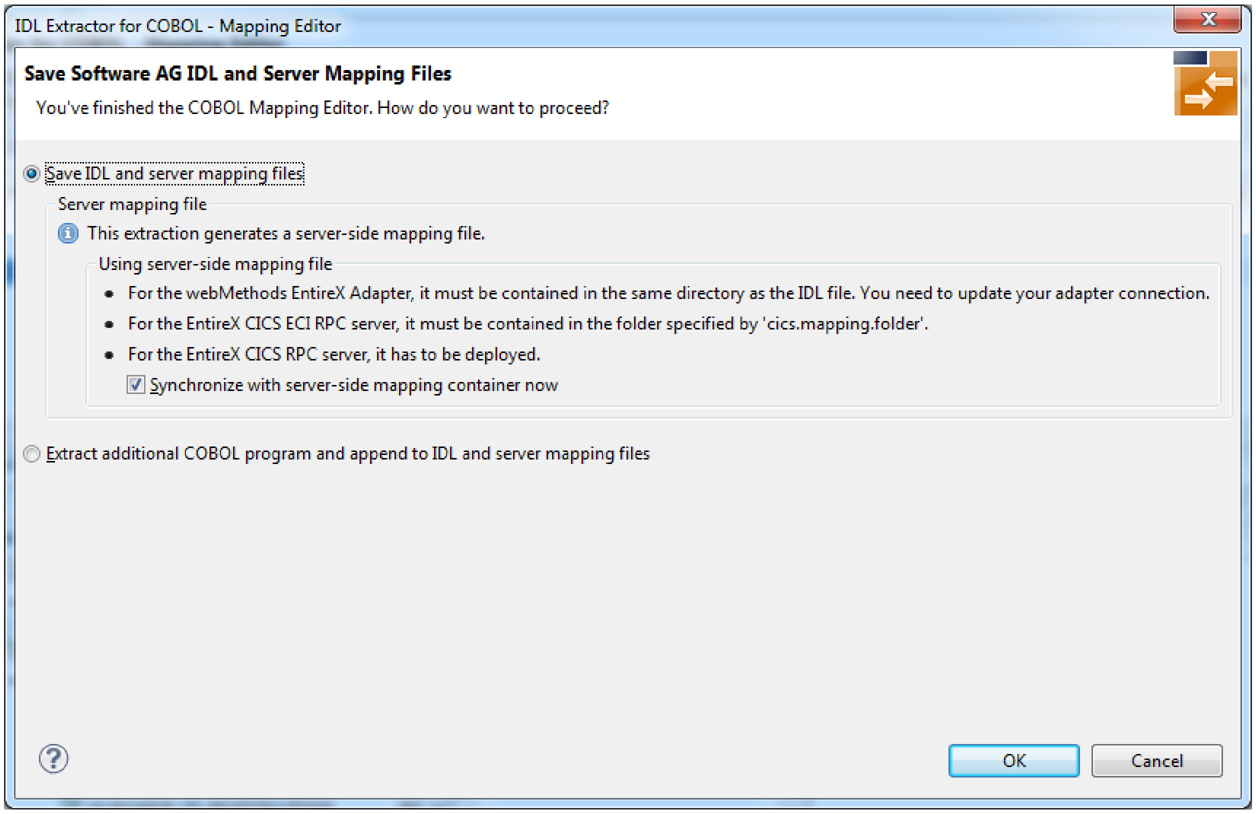
The outcome of the Mapping Editor is the IDL file and a server mapping file (optional). There are server-side mapping files (EntireX Workbench files with extension .svm) and client-side mapping files (extension .cvm). See *Server Mapping Files in the EntireX Workbench* and *How to Set the Type of Server Mapping Files*.

Step 6: Finishing the Mapping Editor

When you choose **Save** in the Mapping Editor, the IDL file is generated. If required, a server mapping file is generated,too. See *When is a Server Mapping File Required?* in the EntireX Workbench documentation The server mapping file is either of type client-side (extension .cvm) or server-side (extension .svm). See *How to Set the Type of Server Mapping Files*. Both files are written with the "File Name" entered for the IDL file in ***Step 4: Define the Extraction Settings and Start Extraction***.

- If you are using *client-side* mapping files, continue with ***Step 7: Validate the Extraction and Test the IDL File***.
- If you are using *server-side* mapping files, the dialog below is displayed whenever the COBOL Mapping Editor is saved. There are two options to choose from:
  - **Save IDL and server mapping files**
    will save the generated files into the workspace and quit the COBOL Mapping Editor

    The generated server-side mapping file need to be synchronized with the server-side mapping container of the target RPC server, except for IMS Connect and CICS ECI connections with the EntireX Adapter, where they are wrapped into the Integration Server connection - the same as client-side mapping files, see *Integration Server Wrapper*.
    - Check the option **Synchronize with server-side mapping container now** for the following RPC servers:
      - z/OS (CICS, Batch, IMS) | Micro Focus | BS2000/OSD | z/VSE (CICS, Batch)
    - Uncheck the option **Synchronize with server-side mapping container now** for
      - EntireX Adapter and IMS Connect and CICS ECI connections
      - the following RPC servers: CICS ECI | IMS Connect
      - later synchronization of other RPC servers
  - **Extract additional COBOL program and append to the IDL and server mapping files**
    will save the generated files into the workspace, quit the Mapping Editor and start the IDL Extractor for COBOL again. The additionally extracted COBOL source will then be added to the previously generated IDL and server mapping files.


IDL Extractor for COBOL - Mapping Editor

Save Software AG IDL and Server Mapping Files

You've finished the COBOL Mapping Editor. How do you want to proceed?

[✓ Save IDL and server mapping files]

Server mapping file

This extraction generates a server-side mapping file.

Using server-side mapping file

- For the webMethods EntireX Adapter, it must be contained in the same directory as the IDL file. You need to update your adapter connection.
- For the EntireX CICS ECI RPC server, it must be contained in the folder specified by 'cics.mapping.folder'.
- For the EntireX CICS RPC server, it has to be deployed.

[✓] Synchronize with server-side mapping container now

Extract additional COBOL program and append to IDL and server mapping files

To save the generated files into the workspace, quit the Mapping Editor and deploy the server-side mapping file

1 Select **Save IDL and server mapping files**.

2 Check the option **Synchronize with server-side mapping container now** and choose **OK**. This calls the Deployment Wizard. See *Server Mapping Deployment Wizard* in the EntireX Workbench documentation.

- If you are using the Server Mapping Deployment Wizard for first time with no predefined deployment environment preferences, continue with *Step 2a: Create a New Deployment Environment* in the Server Mapping Deployment Wizard documentation.
- If deployment environments are already defined, you may also continue with *Step 3: Select and Existing Deployment Environment and Deploy*.

3 Continue with *Step 7: Validate the Extraction and Test the IDL File*.

To save the generated files into the workspace and quit the Mapping Editor without deploying the server-side mapping file

1 Select **Save IDL and server mapping files**.

2 Clear the option **Synchronize with server-side mapping container now** and choose **OK**.

- Synchronize the server-side mapping container of the target RPC server later. See *Deploying Server-side Mapping Files to the RPC Server* in the respective sections of the documentation.
- For the webMethods EntireX Adapter and IMS Connect or CICS ECI connections, update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

3 Continue with *Step 7: Validate the Extraction and Test the IDL File*.

To save the generated files into the workspace, quit the Mapping Editor and start the IDL Extractor for COBOL again

- Select **Extract additional COBOL program and append to the IDL and server mapping files** and choose **OK**. Continue with *Step 2: Select a COBOL Extractor Environment or Create a New One*.

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.

Step 7: Validate the Extraction and Test the IDL File

The IDL file is used to build RPC clients using an EntireX Workbench wrapper of your choice. See *EntireX Wrappers* in the EntireX Workbench documentation.

If you are using client-side mapping files:

- You need to rebuild all RPC clients communicating with this RPC server program and re-generate the client interface objects.
- For connections with the webMethods EntireX Adapter you need to update your Adapter connection, see *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

For a quick validation of your extraction, you can

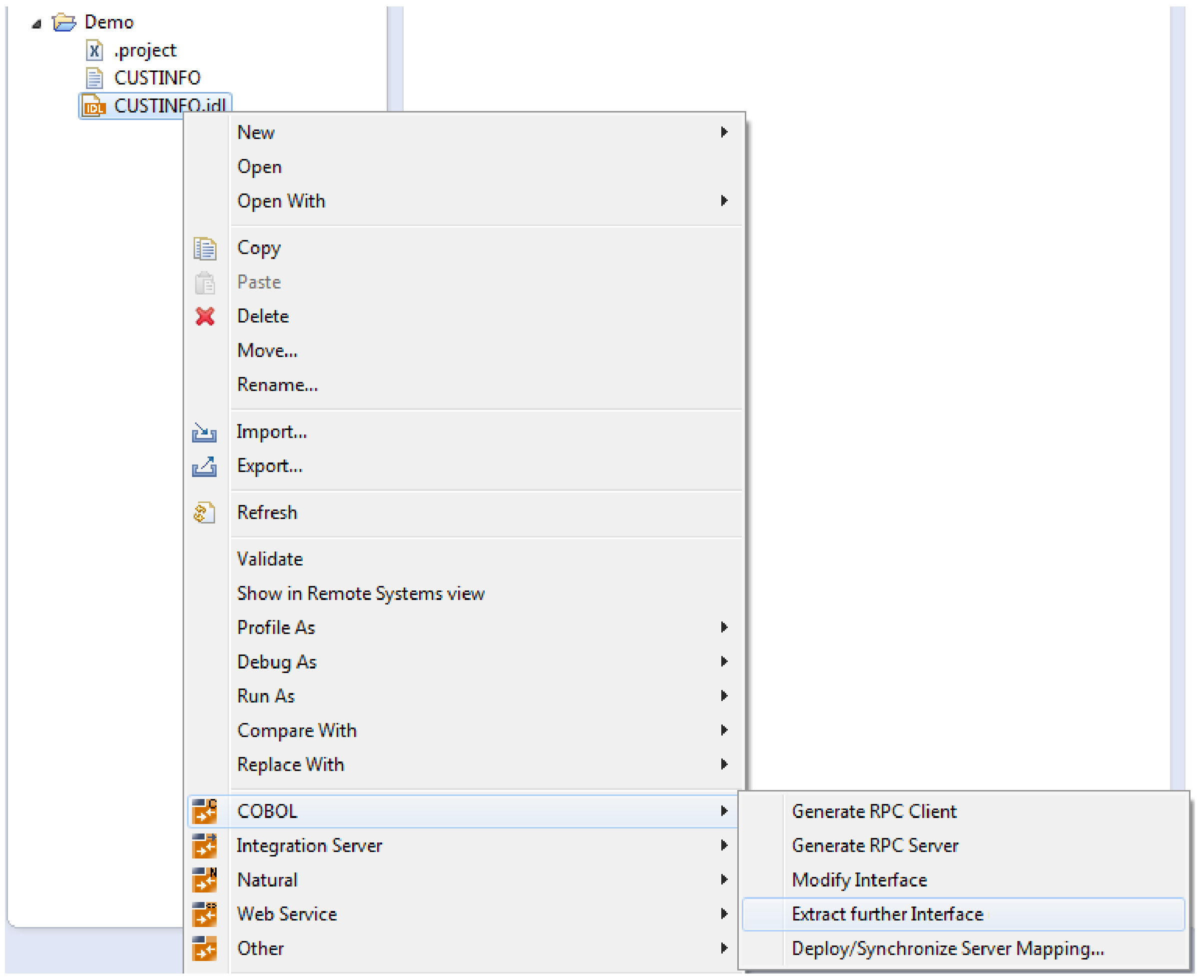
- use the IDL Tester to validate the extraction, see *EntireX IDL Tester* in the EntireX Workbench documentation.
- generate an XML mapping file (XMM) and use the XML Tester for verification. See *EntireX XML Tester* in the XML/SOAP Wrapper documentation.

3 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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

To start the IDL Extractor for COBOL

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



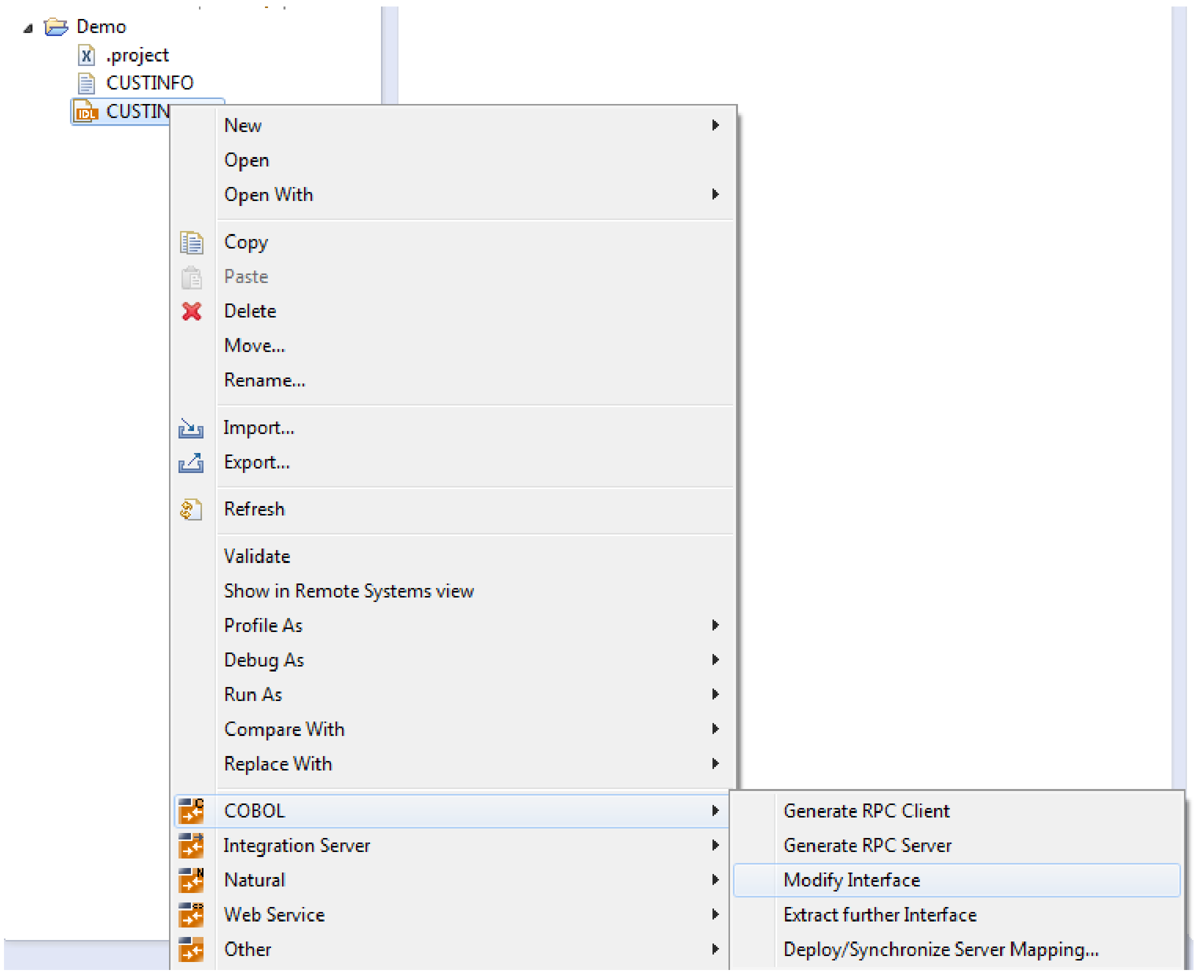
Continue with *Step 2: Select a COBOL Extractor Environment or Create a New One* as described under *Scenario I: Create New IDL and Server Mapping Files*.

4 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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

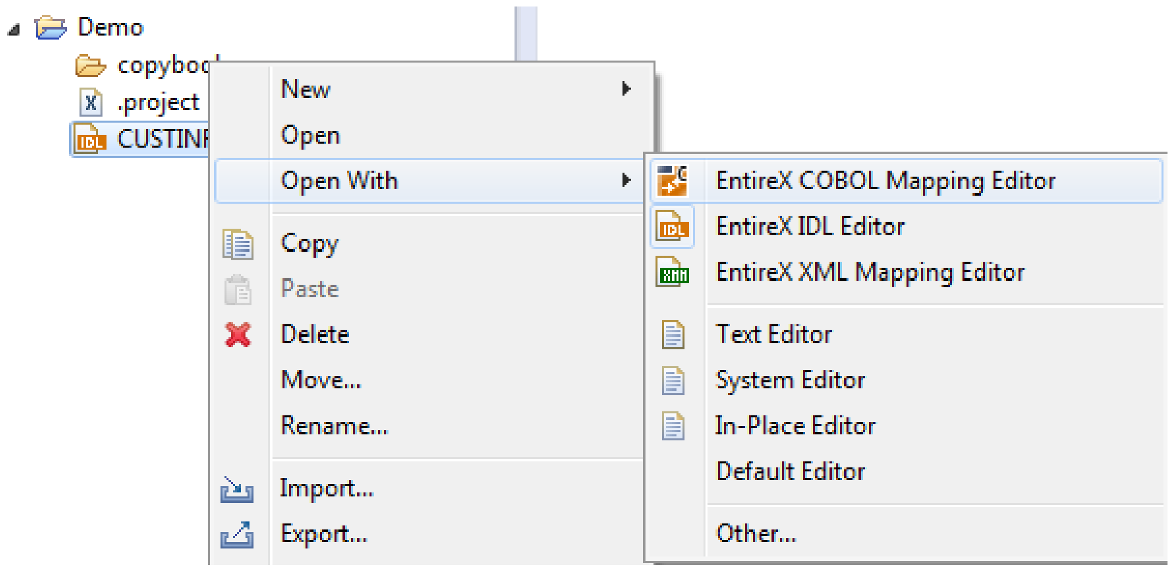
To start the COBOL Mapping Editor

- 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

A COBOL source program mostly does not contain all the information needed for IDL mapping. With the Mapping Editor you enter this missing information. The Mapping Editor allows you to map the COBOL server interface to Software AG IDL. With the Mapping Editor you

- select the COBOL data items of the COBOL interface
- define
  - which COBOL data items are mapped to IDL (**Select REDEFINE paths**, **Suppress COBOL Unneeded Data Items**)
  - the direction of the COBOL data items are mapped to IDL (**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**)
  - multiple IDL interfaces (**Map to Multiple IDL Interfaces**)

The following table provides guidelines on IDL extraction per interface type. 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	
Batch	Standard Linkage	
IMS	BMP with Standard Linkage	
	MPP Message Interface (IMS Connect)	

Environment	Interface Type	CICS Message on Input and Output
Micro Focus	Standard Linkage	

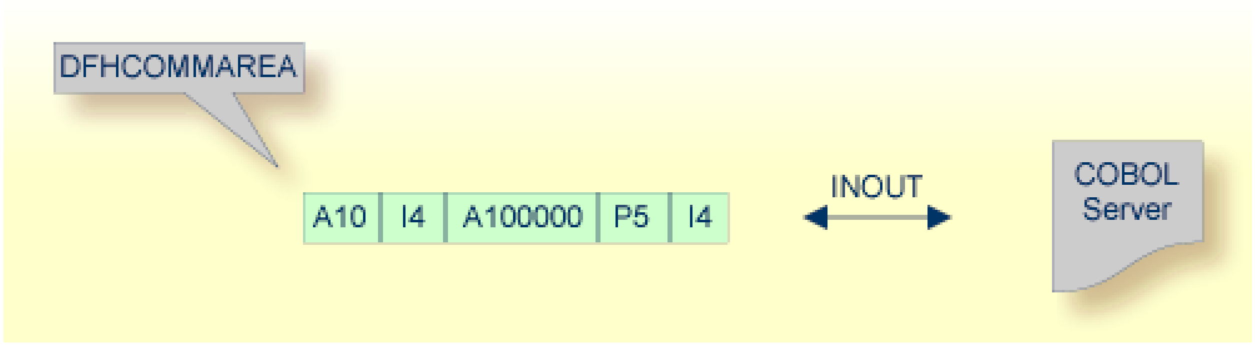
Notes:

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 CICS input message is the same as the structure of the CICS output message.
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 CICS input message is different to the structure of the CICS output message (that is, the output overlays the input).
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 2: Redefines*
   - *Example 3: Buffer Technique*
   - *Example 4: COBOL SET ADDRESS Statements*
5. See the following COBOL server examples for CICS input message *different to* CICS output message:
   - *Example 2: Redefines*
   - *Example 3: Buffer Technique*
   - *Example 4: COBOL SET ADDRESS Statements*

5 CICS with DFHCOMMAREA Calling Convention - In same as Out

- Introduction 60
- Extracting from a CICS DFHCOMMAREA Program 60
- Mapping Editor User Interface 61
- Mapping Editor IDL Interface Mapping Functions 68
- Programming Techniques 72



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. Please note also 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.

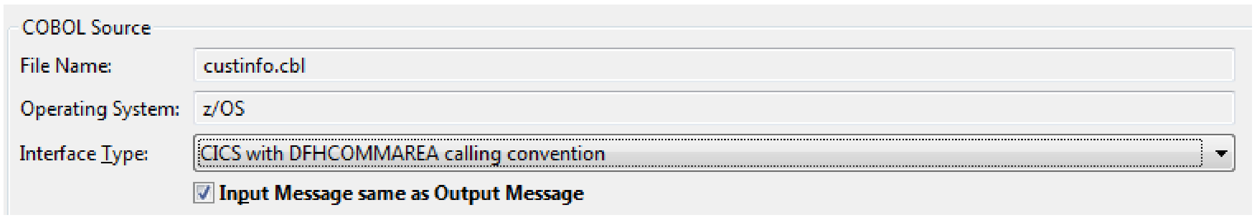
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

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.


COBOL Source

File Name: custinfo.cbl

Operating System: z/OS

Interface Type: CICS with DFHCOMMAREA calling convention

☑ Input Message same as Output Message

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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. See the examples provided under *Programming Techniques*.
4. If your COBOL server contain `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.

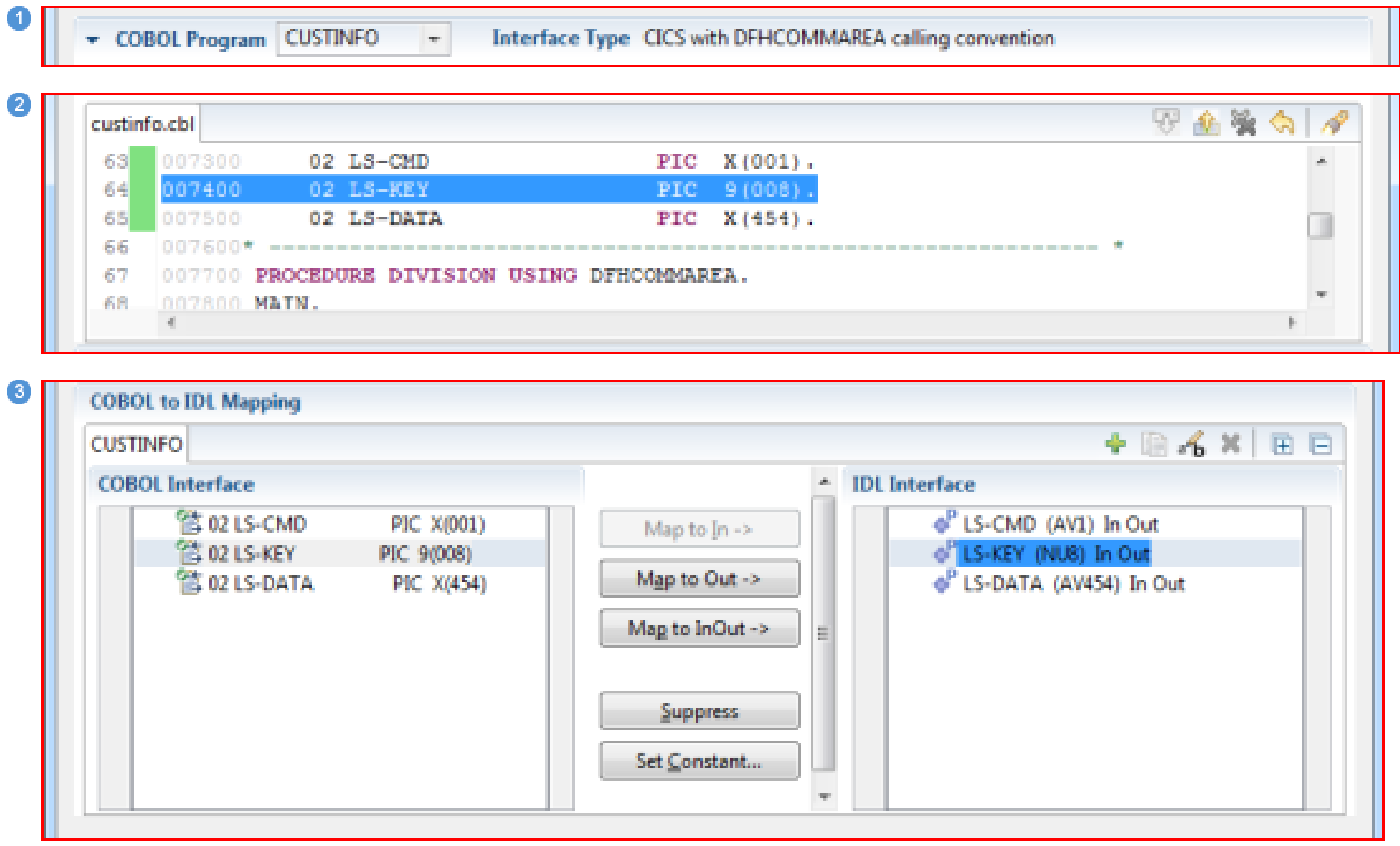
The user interface of the COBOL Mapping Editor is described below.

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:



COBOL Program CUSTINFO Interface Type CICS with DFHCOMMAREA calling convention

custinfo.cbl

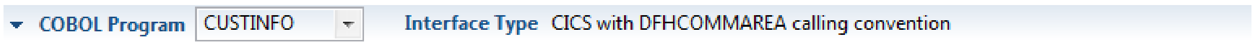
```
007300       02 LS-CMD                       PIC   X(001).
007400       02 LS-KEY                       PIC   9(008).
007500       02 LS-DATA                      PIC   X(454).
007600* ---------------------------------------------------------------- *
007700 PROCEDURE DIVISION USING DFHCOMMAREA.
007800 MAIN.
```

COBOL to IDL Mapping

COBOL Interface		IDL Interface
02 LS-CMD	PIC X(001)	LS-CMD (AV1) In Out
02 LS-KEY	PIC 9(008)	LS-KEY (NU8) In Out
02 LS-DATA	PIC X(454)	LS-DATA (AV454) In Out

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

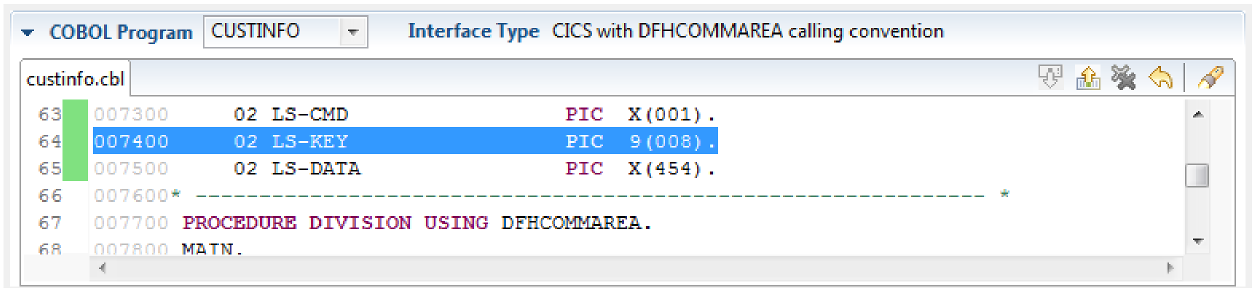


CUSTINFO

Interface Type CICS with DFHCOMMAREA calling convention

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

COBOL Source View



```
007300       02 LS-CMD                   PIC  X(001).
007400       02 LS-KEY                   PIC  9(008).
007500       02 LS-DATA                  PIC  X(454).
007600* ---------------------------------------------------------------- *
007700 PROCEDURE DIVISION USING DFHCOMMAREA.
007800 MAIN.
```

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

Add selected COBOL data item to COBOL Interface.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

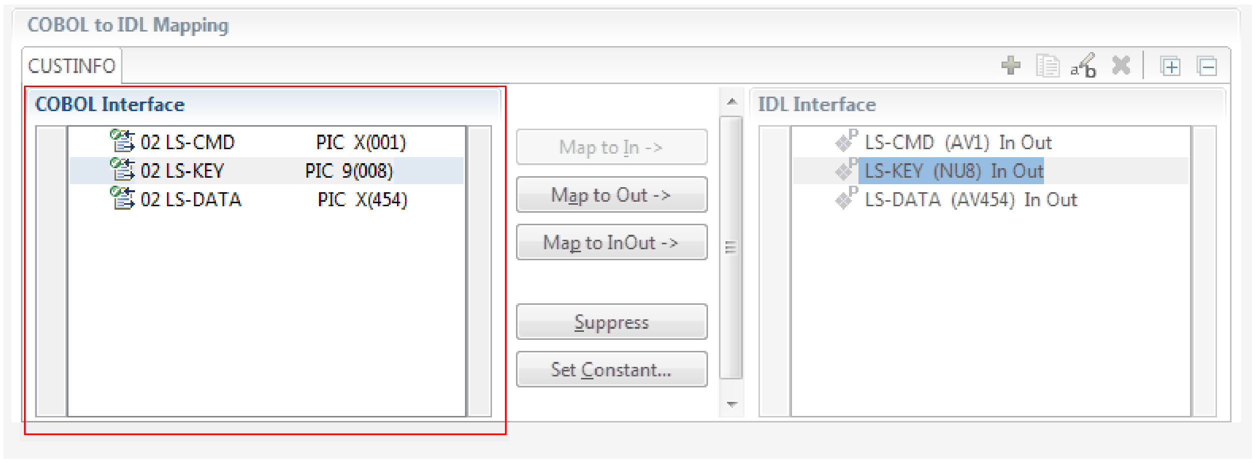
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



```
02 LS-CMD      PIC X(001)
02 LS-KEY      PIC 9(008)
02 LS-DATA     PIC X(454)
```

```
LS-CMD  (AV1)  In Out
LS-KEY  (NU8)  In Out
LS-DATA (AV454) In Out
```

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

Context Menu

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

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

Map to 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.
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*.

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".

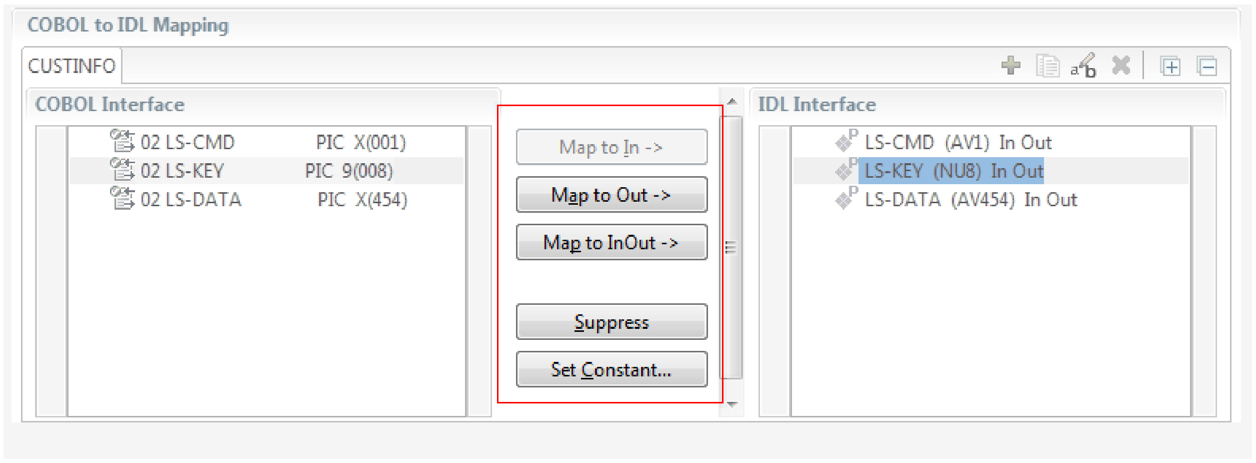
Mapping Icons

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.

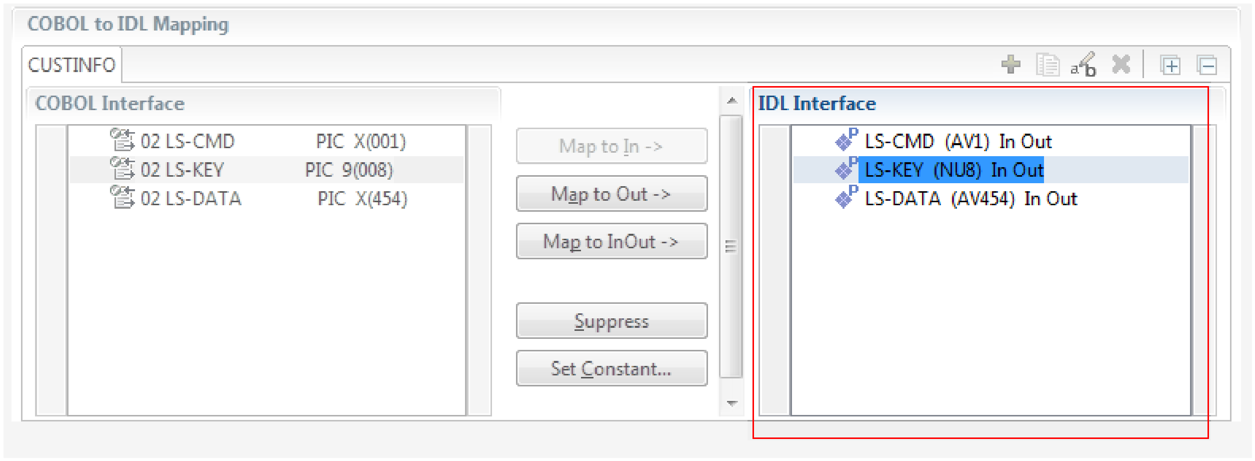
Suppress
: See *Suppress Unneeded COBOL Data Items*.

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
- 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.



```
02 LS-CMD     PIC X(001)
02 LS-KEY     PIC 9(008)
02 LS-DATA    PIC X(454)
```

```
LS-CMD  (AV1)  In Out
LS-KEY  (NU8)  In Out
LS-DATA  (AV454)  In Out
```

Mapping Editor IDL Interface Mapping Functions

This section covers the following topics:

- Map to In, Out, InOut
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

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 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 subsequent child COBOL data items and thus to the related IDL parameters in the IDL interface.
2. Subsequent child 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 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), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to In**, **Out** or **InOut** (see above) available in the context menu and as mapping button, 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 server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as `RECORD-TYPES`. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

To map COBOL data items to constants

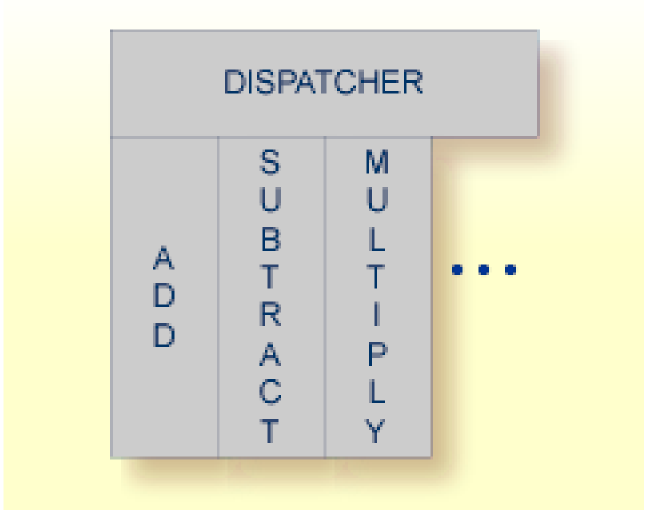
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Programming Techniques

This section covers the following topics:

- Example 1: COBOL Server with Multiple Functions
- Example 2: Redefines
- Example 3: Buffer Technique

- Example 4: COBOL SET ADDRESS Statements

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

Example 2: Redefines

The 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 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.
 02 BUFFER2 REDEFINES BUFFER1.
    03 FIELD-1                        PIC X(4).
    03 FIELD-2                        PIC X(2).
  . . .
 PROCEDURE DIVISION USING DFHCOMMAREA.
* process the BUFFER2 and provide result in BUFFER2
    EXEC CICS RETURN.
```

Example 3: 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.
```

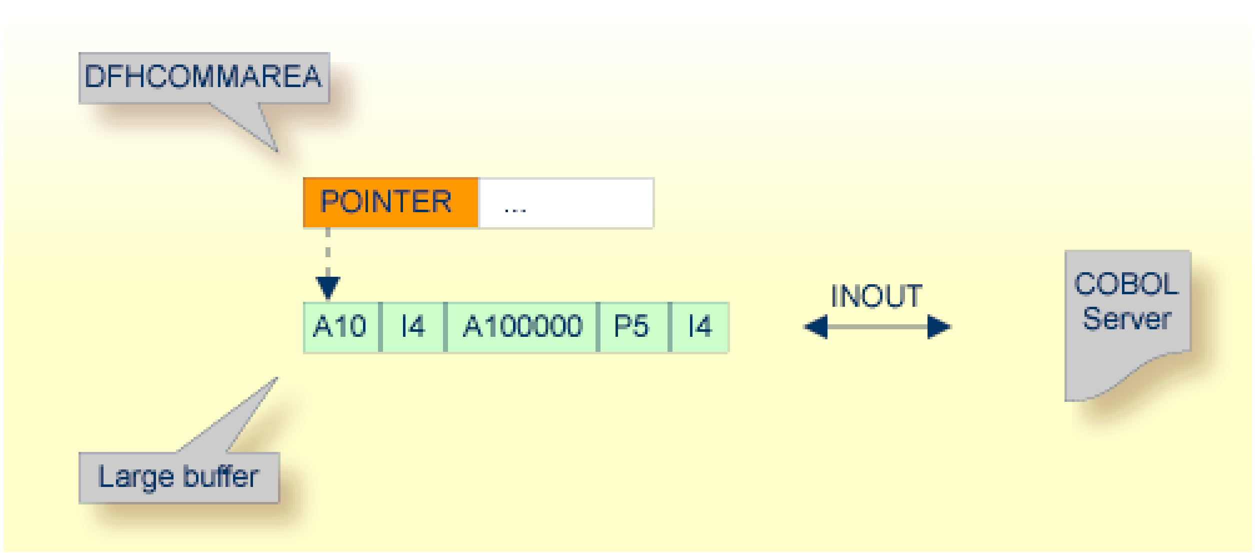
Example 4: COBOL SET ADDRESS Statements

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.
```

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

- Introduction 78
- Extracting from a CICS DFHCOMMAREA Large Buffer Program 79
- Mapping Editor User Interface 80
- Mapping Editor IDL Interface Mapping Functions 87
- Programming Techniques 92



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.

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

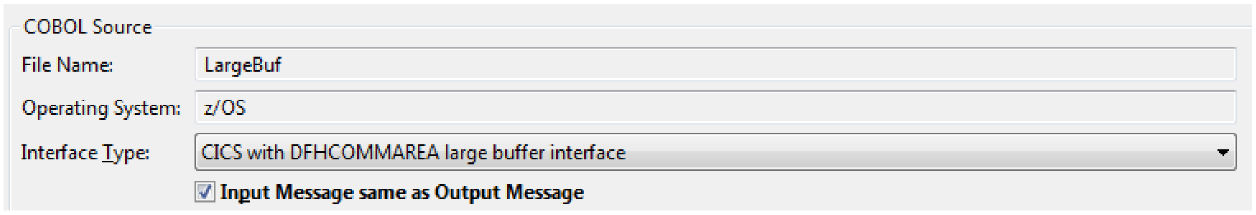
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

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.



COBOL Source

File Name: LargeBuf

Operating System: z/OS

Interface Type: CICS with DFHCOMMAREA large buffer interface

[✓] Input Message same as Output Message

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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).

3. If your COBOL server contain `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.

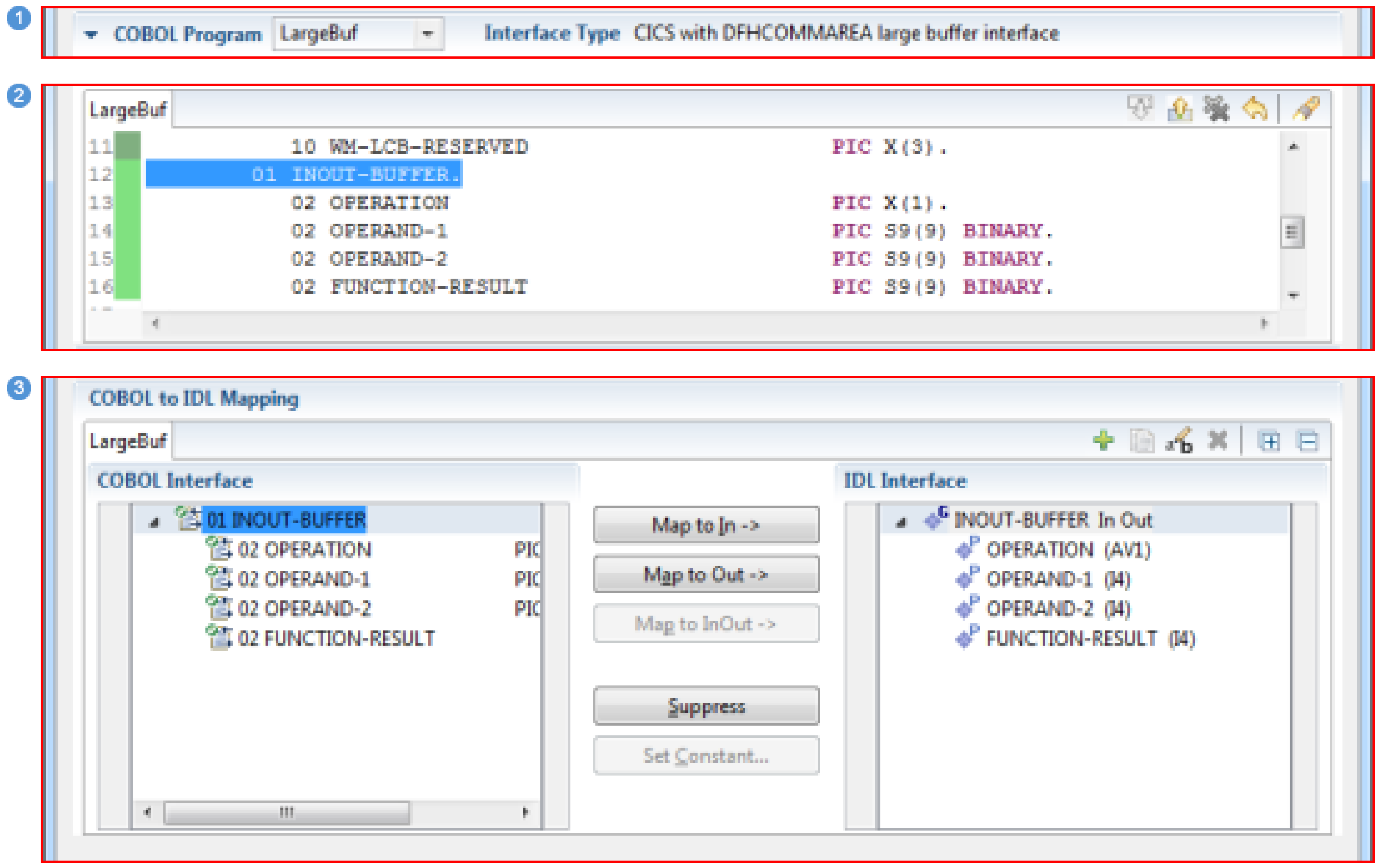
The user interface of the COBOL Mapping Editor is described below.

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:



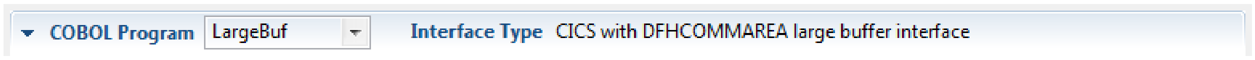
COBOL Program LargeBuf   Interface Type CICS with DFHCOMMAREA large buffer interface

```
LargeBuf
          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.
```

```
COBOL Interface                      IDL Interface
01 INOUT-BUFFER                      INOUT-BUFFER In Out
    02 OPERATION          PIC           OPERATION (AV1)
    02 OPERAND-1          PIC           OPERAND-1 (I4)
    02 OPERAND-2          PIC           OPERAND-2 (I4)
    02 FUNCTION-RESULT                  FUNCTION-RESULT (I4)
```

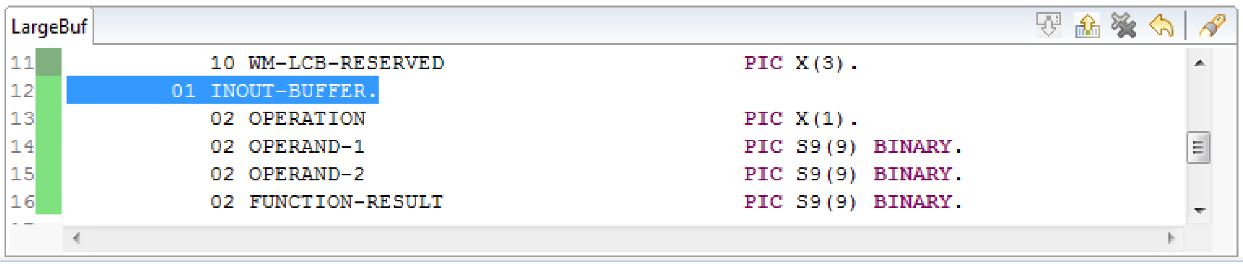
1. **COBOL Program Selection**. Currently selected program with interface type
2. **COBOL Source View**. Contains all related sources for the currently selected COBOL program
3. **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items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

COBOL Program Selection


COBOL Program LargeBuf Interface Type CICS with DFHCOMMAREA large buffer interface

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

COBOL Source View



```
        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.
```

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

Add selected COBOL data item to COBOL Interface.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

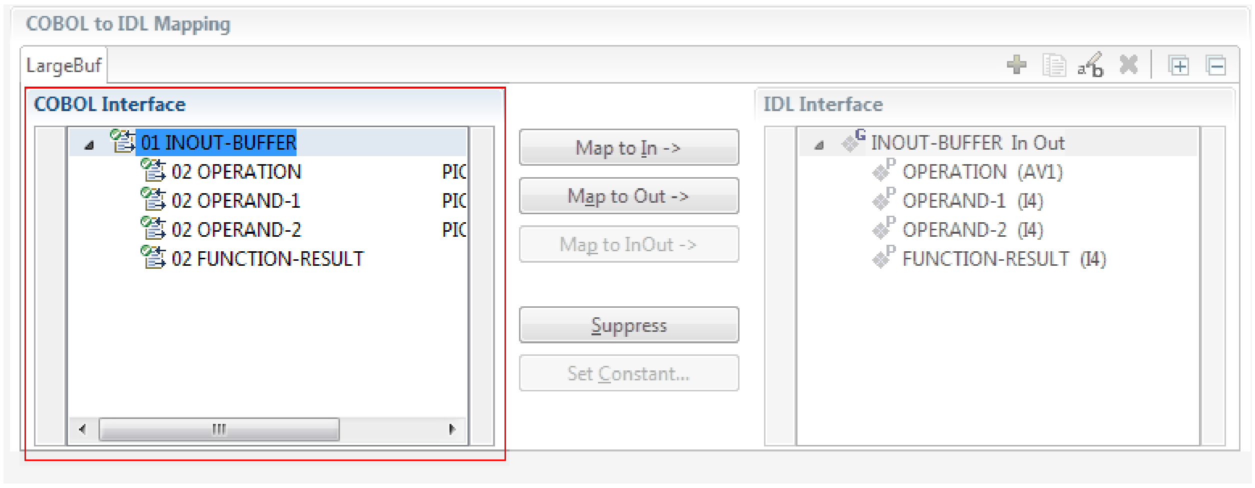
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to 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. **Note:** This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping could result in runtime errors.
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*.

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.

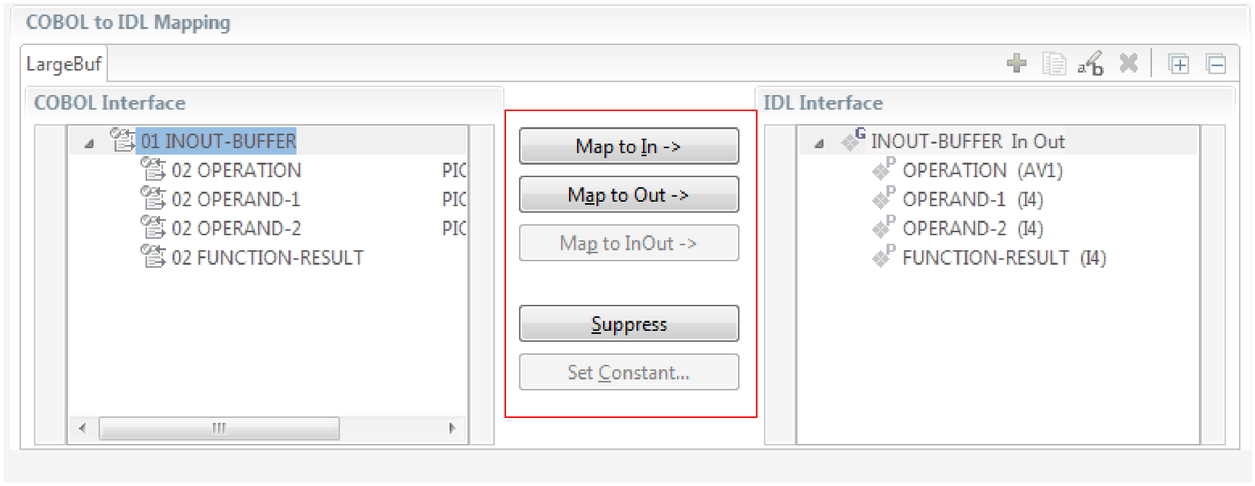
Group parameter, here mapped to InOut.

`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.

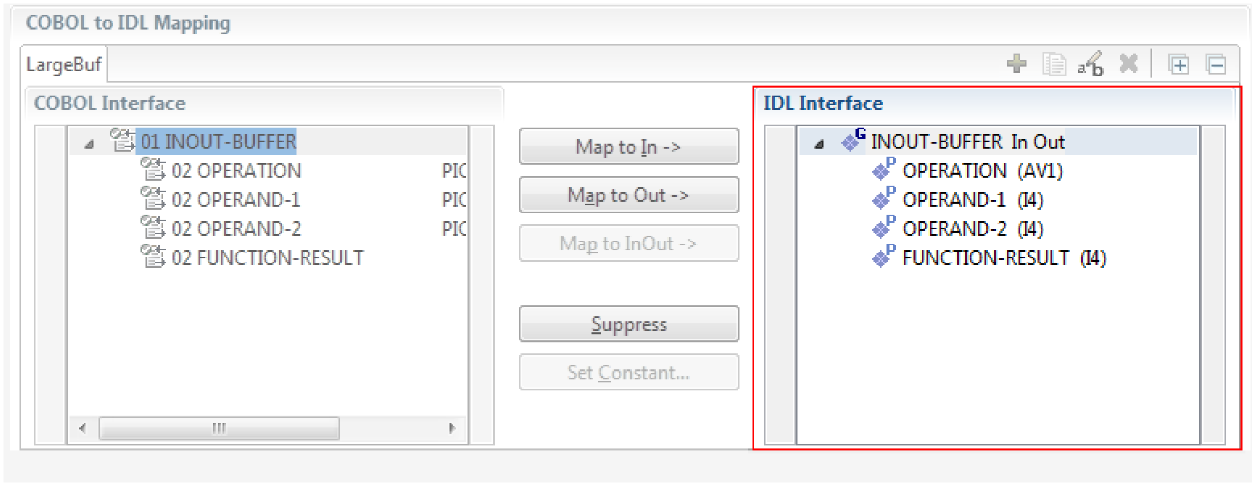
Suppress
: See *Suppress Unneeded COBOL Data Items*.

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

Map to 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 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 subsequent child COBOL data items and thus to the related IDL parameters in the IDL interface.

2. Subsequent child 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 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), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to In**, **Out** or **InOut** (see above) available in the context menu and as mapping button, 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 server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as `RECORD-TYPES`. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

To map COBOL data items to constants

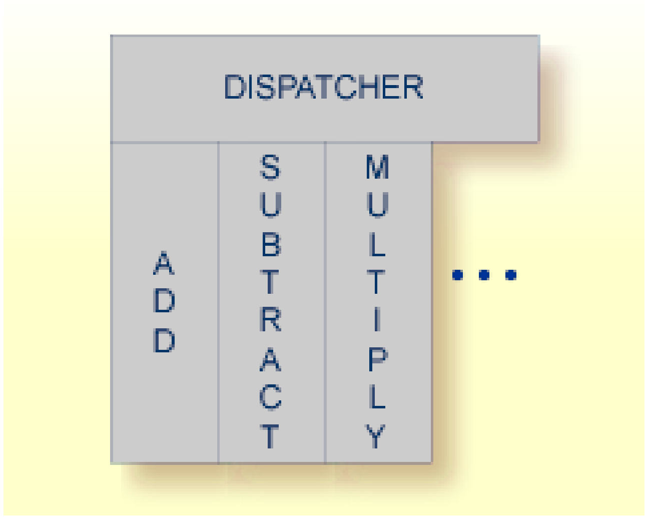
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Set Arrays (Fixed <-> Unbounded)

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

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

To set arrays from fixed to unbounded or vice versa

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

Notes:

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

Programming Techniques

This section covers the following topics:

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

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

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

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

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

          . . .

         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.
     10 COBOL-GROUP1.
             20 COBOL-TABLE-PREFIX           PIC X(1000).
           10 COBOL-TABLE-FIX                OCCURS 20.
             20 COBOL-GROUP2.
               25 COBOL-FIELD1               PIC X(30).
               25 COBOL-FIELD2               PIC X(20).
               25 COBOL-FIELD3               PIC X(50).
              . . .
         PROCEDURE DIVISION USING DFHCOMMAREA.
```

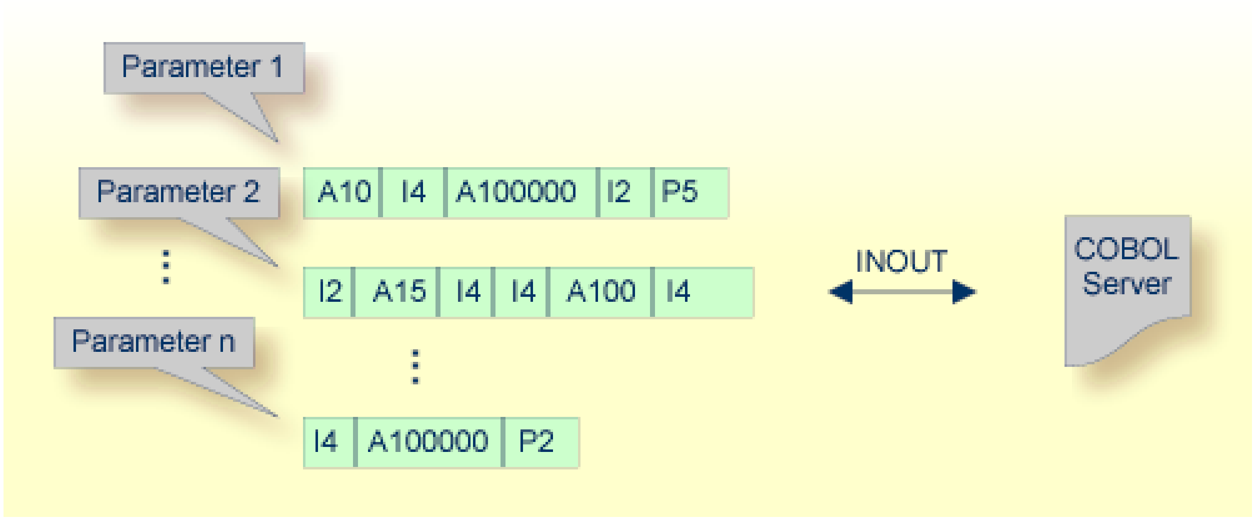
```
            SET ADDRESS OF INOUT-BUFFER TO WM-LCB-INPUT-BUFFER.
            SET ADDRESS OF INOUT-BUFFER TO WM-LCB-OUTPUT-BUFFER.
            COMPUTE NUMBER-OF-INCOMING-ELEMENTS = (WM-LCB-INPUT-BUFFER-SIZE
                  - LENGTH OF COBOL-GROUP1)
                  / LENGTH OF COBOL-GROUP2.

              . . .
            COMPUTE WM-LCB-OUTPUT-BUFFER-SIZE = LENGTH OF COBOL-GROUP2
                  + NUMBER-OF-OUTGOING-ELEMENTS * LENGTH OF COBOL-GROUP2

     EXEC CICS RETURN END-EXEC.
```

During input the COBOL CICS large buffer server program uses the large buffer input length `WM-LCB-INPUT-BUFFER-SIZE` to evaluate the `NUMBER-OF-INCOMING-ELEMENTS`. During output the large buffer output length is determined accordingly to the `NUMBER-OF-OUTGOING-ELEMENTS` and set in `WM-LCB-OUTPUT-BUFFER-SIZE`.

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

7 Batch with Standard Linkage Calling Convention

- Introduction 96
- Extracting from a Standard Call Interface 96
- Mapping Editor User Interface 97
- Mapping Editor IDL Interface Mapping Functions 104
- Programming Techniques 108



Introduction

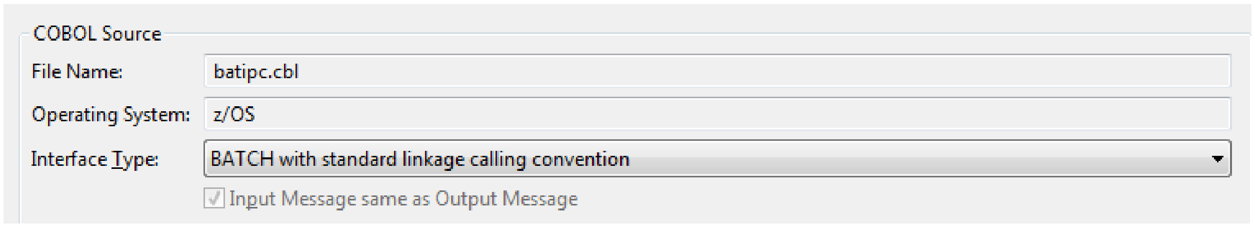
Because COBOL servers with a standard call interface always contain a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters, the COBOL data items of the interface can be evaluated by the IDL Extractor for COBOL and are already offered by the wizard. In most cases the offered 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.


COBOL Source

File Name:	batipc.cbl
Operating System:	z/OS
Interface Type:	BATCH with standard linkage calling convention

☑ Input Message same as Output Message

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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL server contain `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.

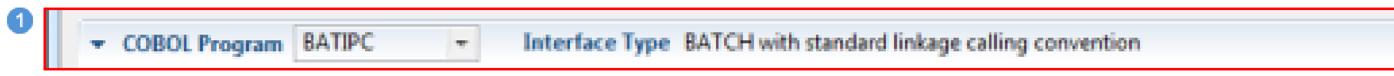
The user interface of the COBOL Mapping Editor is described below.

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:

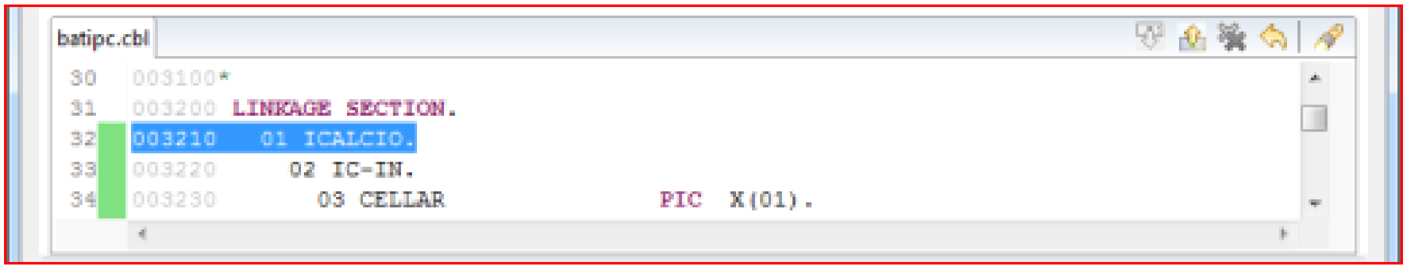
- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

For COBOL server programs with standard call interface types, the user interface of the COBOL Mapping Editor looks like this:



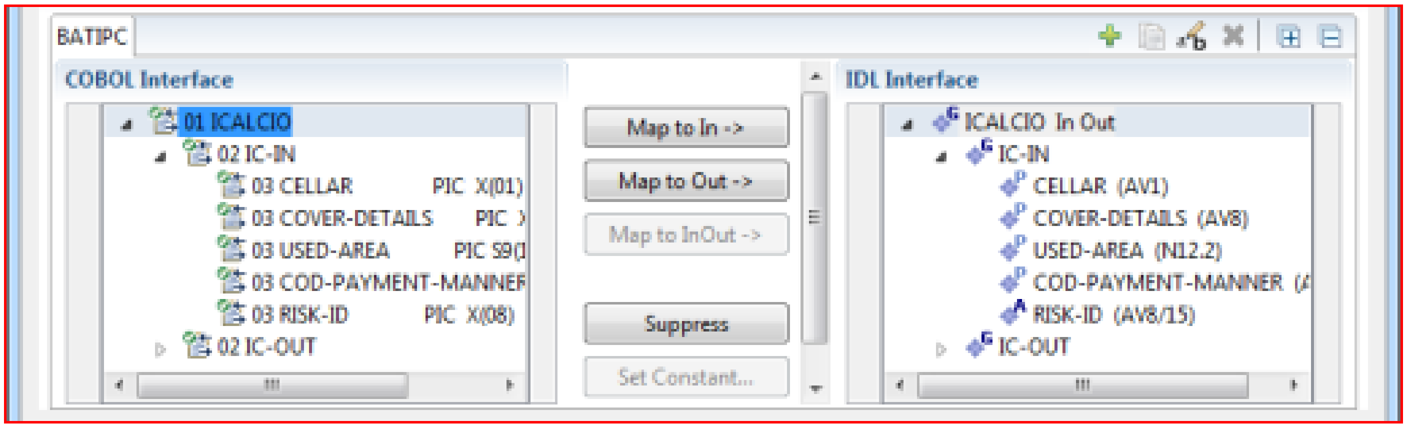
Interface Type BATCH with standard linkage calling convention

2



```
003100*
003200 LINKAGE SECTION.
003210    01 ICALCIO.
003220       02 IC-IN.
003230          03 CELLAR                   PIC  X(01).
```

3

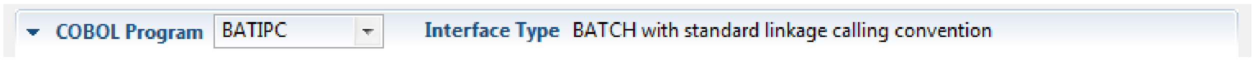


1 **COBOL Program Selection**. Currently selected program with interface type

2 **COBOL Source View**. Contains all related sources for the currently selected COBOL program

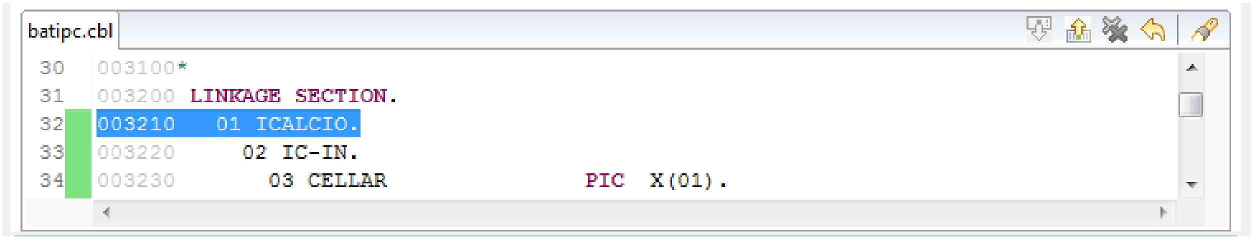
3 **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 associated IDL file, you can switch to another COBOL program with its mapping by selecting the name in the combo box.

COBOL Source View



```
003100*
003200 LINKAGE SECTION.
003210    01 ICALCIO.
003220       02 IC-IN.
003230          03 CELLAR              PIC  X(01).
```

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

Add selected COBOL data item to COBOL Interface.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

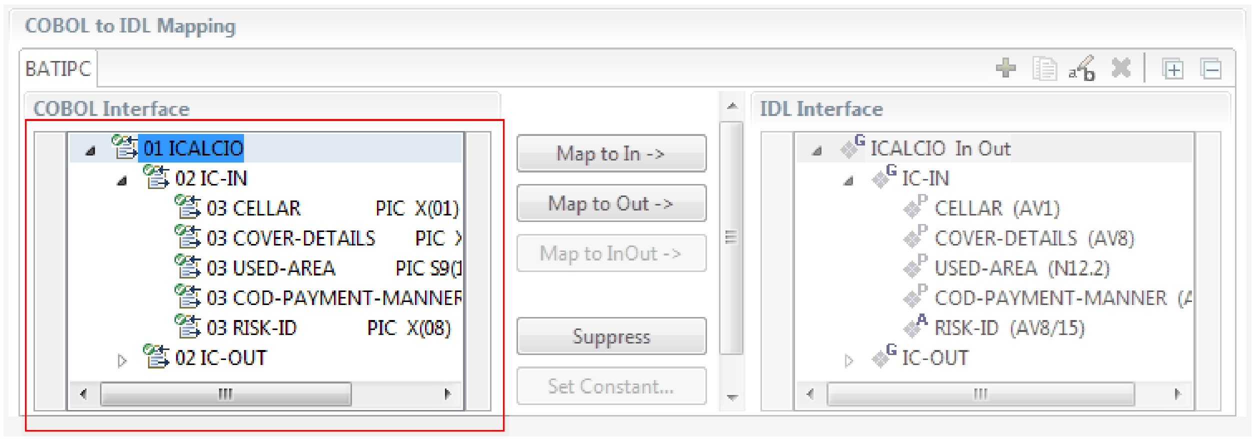
This section covers the following topics:

- COBOL Interface
- Mapping Buttons
- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to 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.
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*.

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.

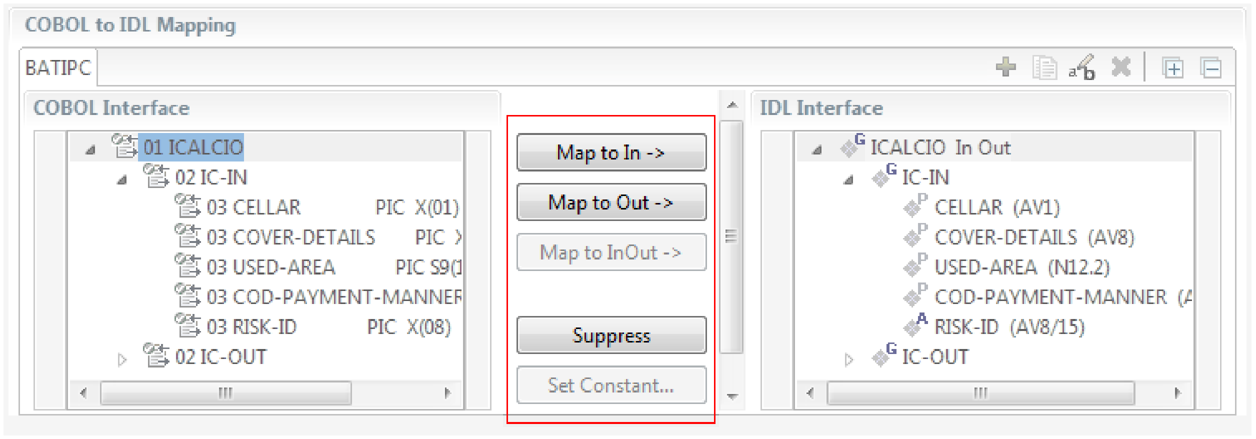
Group parameter, here mapped to InOut.

`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*.

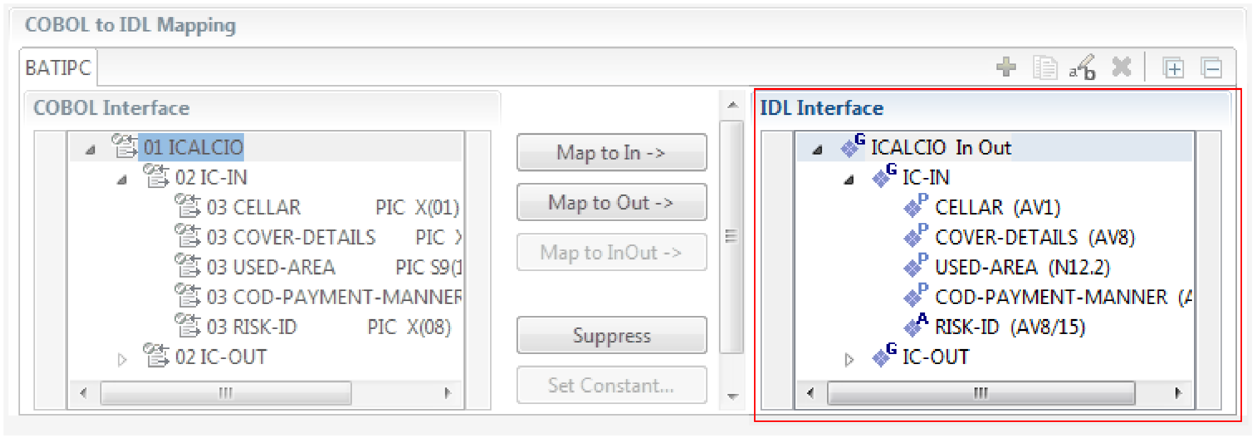
Set Constant...

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
- 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
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

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 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 subsequent child COBOL data items and thus to the related IDL parameters in the IDL interface.
2. Subsequent child 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 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), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to In**, **Out** or **InOut** (see above) available in the context menu and as mapping button, 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 server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as `RECORD-TYPES`. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

To map COBOL data items to constants

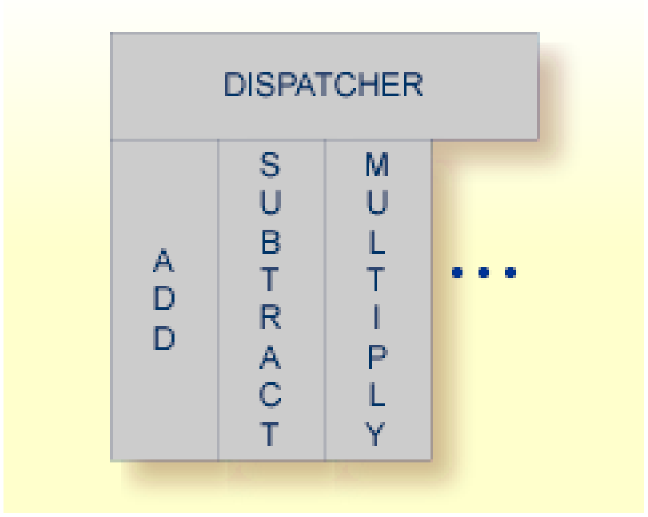
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Programming Techniques

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

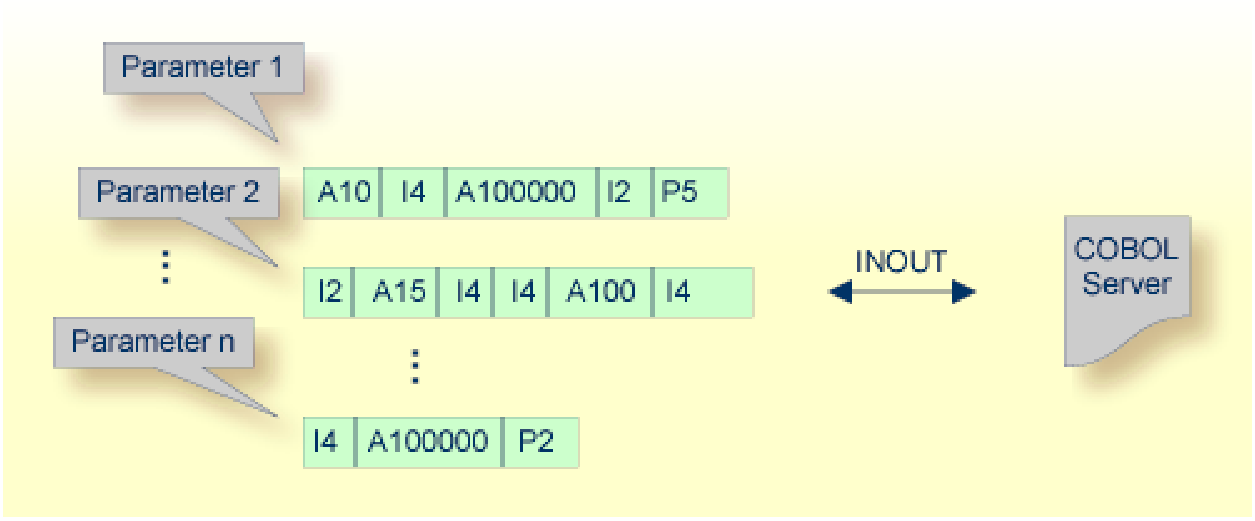
You can expose each COBOL server program function separately. The advantages or reasons for wanting this depend on the target endpoint. For example:

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

8 Micro Focus with Standard Linkage Calling Convention

- Introduction 112
- Extracting from a Standard Call Interface 112
- Mapping Editor User Interface 113
- Mapping Editor IDL Interface Mapping Functions 120
- Programming Techniques 124



Introduction

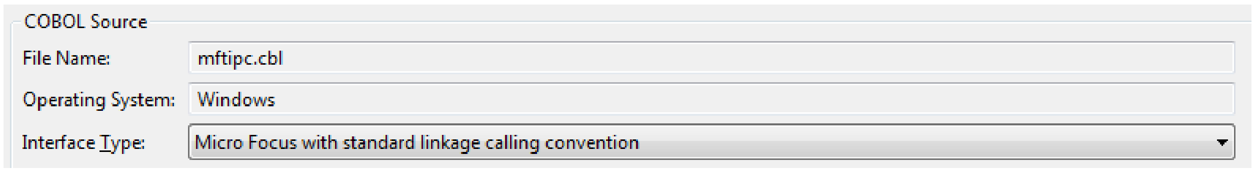
Because COBOL servers with a standard call interface always contain a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters, the COBOL data items of the interface can be evaluated by the IDL Extractor for COBOL and are already offered by the wizard. In most cases the offered 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 Micro Focus 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.



COBOL Source

File Name:	mftipc.cbl
Operating System:	Windows
Interface Type:	Micro Focus with standard linkage calling convention

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 **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 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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL server contain `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.

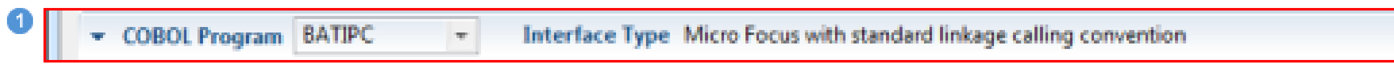
The user interface of the COBOL Mapping Editor is described below.

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:

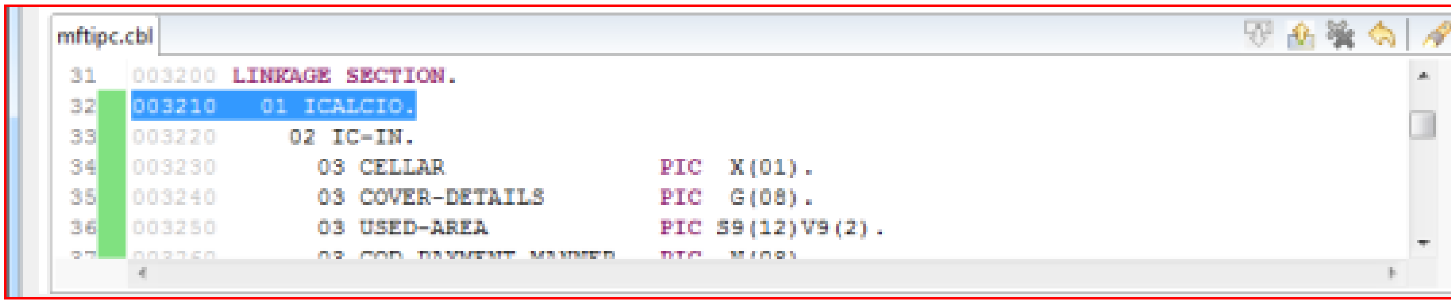
- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

For COBOL server programs with standard call interface types, the user interface of the COBOL Mapping Editor looks like this:



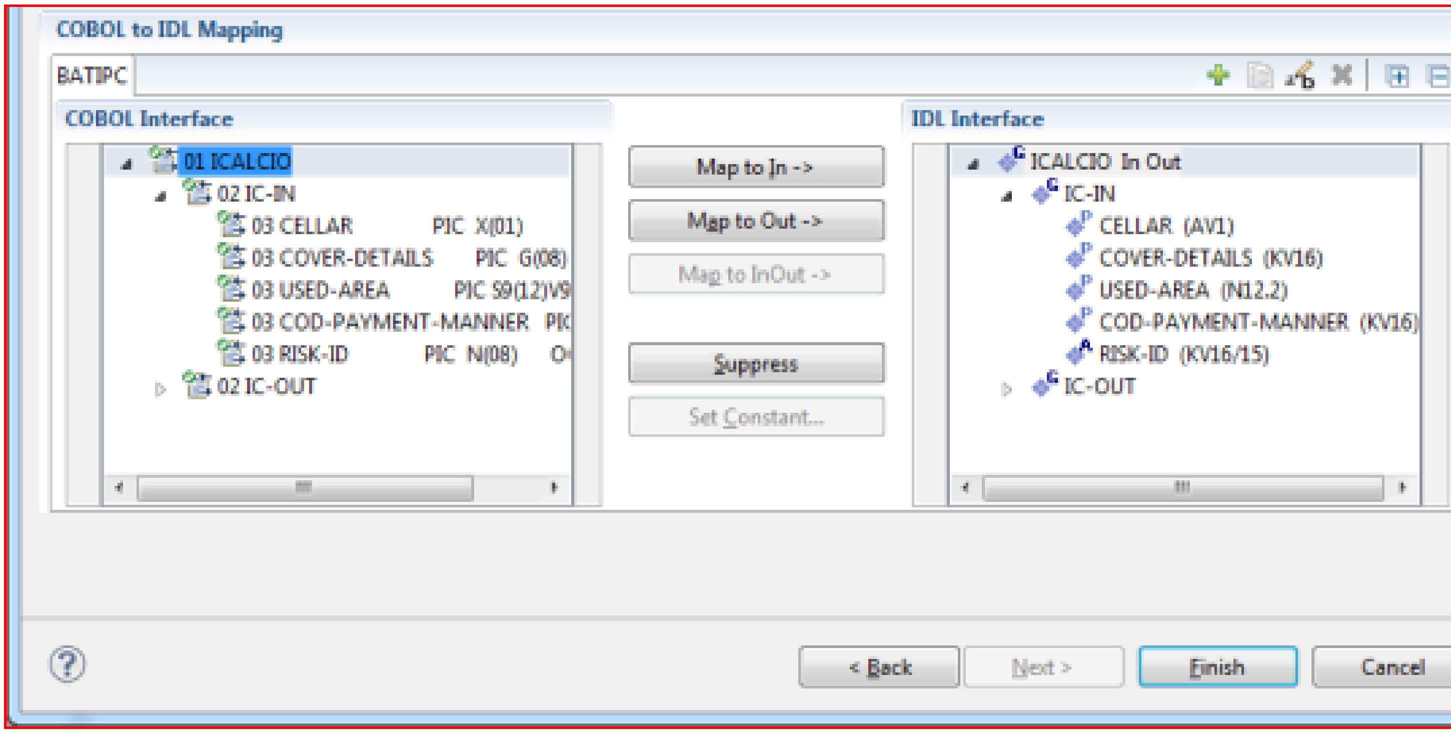
Interface Type Micro Focus with standard linkage calling convention

2



```
003200 LINKAGE SECTION.
003210    01 ICALCIO.
003220       02 IC-IN.
003230          03 CELLAR                PIC  X(01).
003240          03 COVER-DETAILS         PIC  G(08).
003250          03 USED-AREA             PIC S9(12)V9(2).
003260          03 COD-PAYMENT-MANNER    PIC  N(08)
```

3



```
01 ICALCIO
   02 IC-IN
      03 CELLAR               PIC X(01)
      03 COVER-DETAILS        PIC G(08)
      03 USED-AREA            PIC S9(12)V9
      03 COD-PAYMENT-MANNER   PI
      03 RISK-ID              PIC N(08)   O
   02 IC-OUT
```

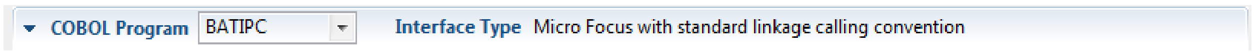
```
ICALCIO In Out
   IC-IN
      CELLAR (AV1)
      COVER-DETAILS (KV16)
      USED-AREA (N12.2)
      COD-PAYMENT-MANNER (KV16)
      RISK-ID (KV16/15)
   IC-OUT
```

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

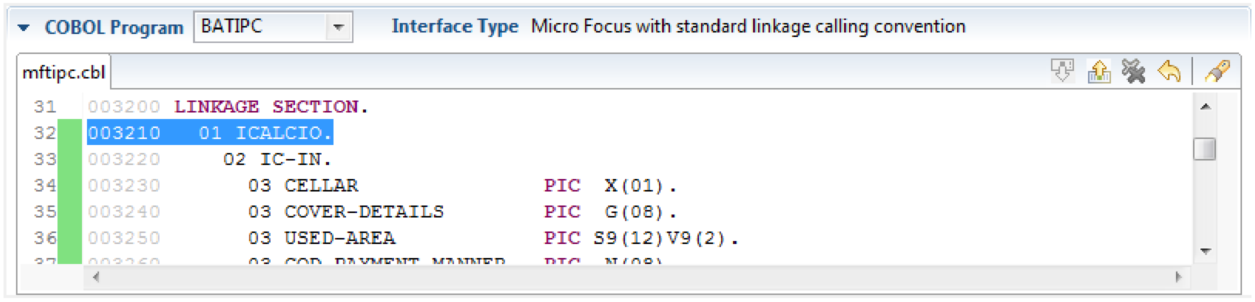
COBOL Program Selection


COBOL Program BATIPC

Interface Type Micro Focus with standard linkage calling convention

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

COBOL Source View



```
003200 LINKAGE SECTION.
003210    01 ICALCIO.
003220       02 IC-IN.
003230          03 CELLAR                PIC  X(01).
003240          03 COVER-DETAILS         PIC  G(08).
003250          03 USED-AREA             PIC S9(12)V9(2).
```

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

Add selected COBOL data item to COBOL Interface.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

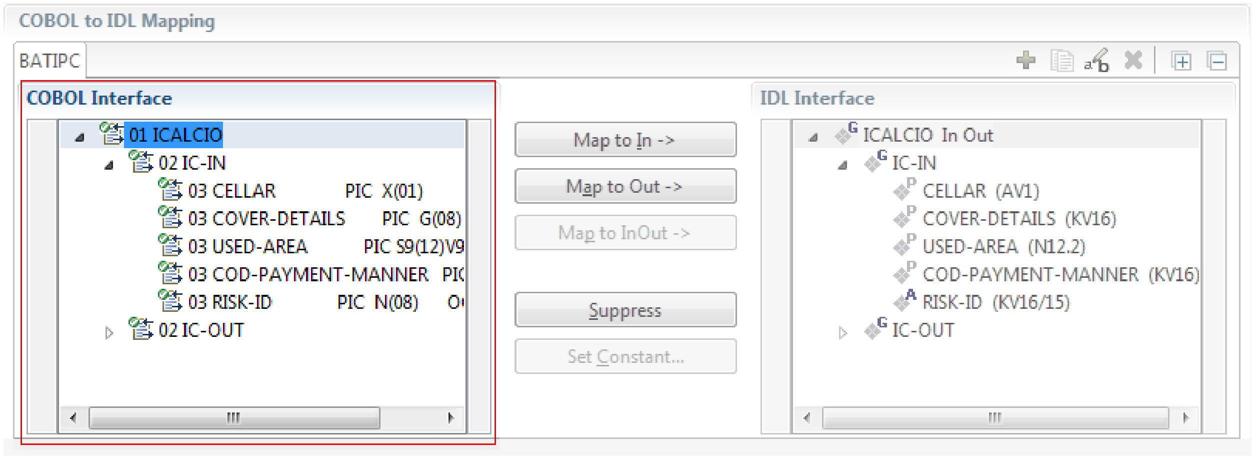
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to 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.
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*.

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".

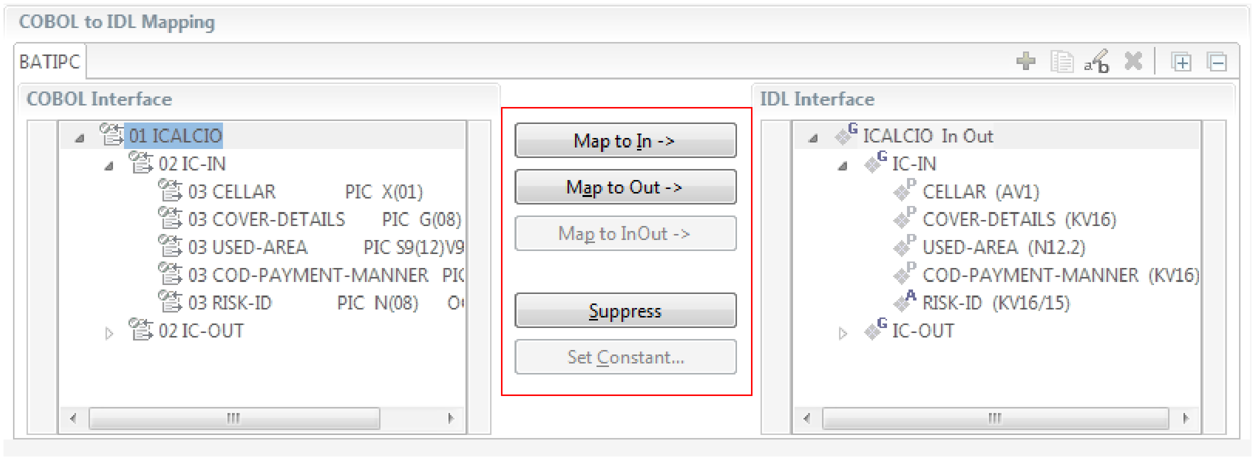
Mapping Icons

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.

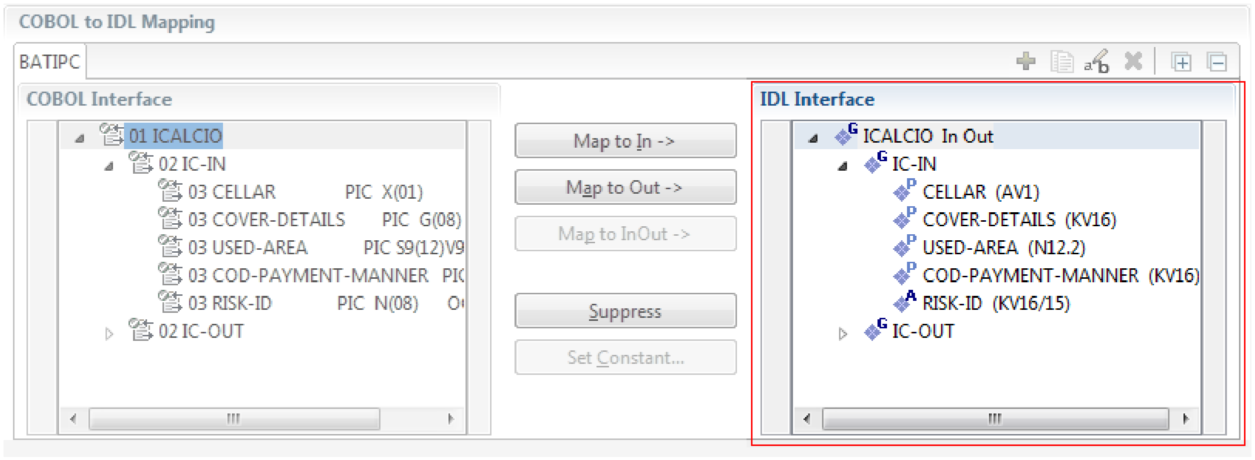
Suppress
See *Suppress Unneeded COBOL Data Items*.

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
- 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
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

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 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 subsequent child COBOL data items and thus to the related IDL parameters in the IDL interface.
2. Subsequent child 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 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), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to In**, **Out** or **InOut** (see above) available in the context menu and as mapping button, 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 server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as `RECORD-TYPES`. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

To map COBOL data items to constants

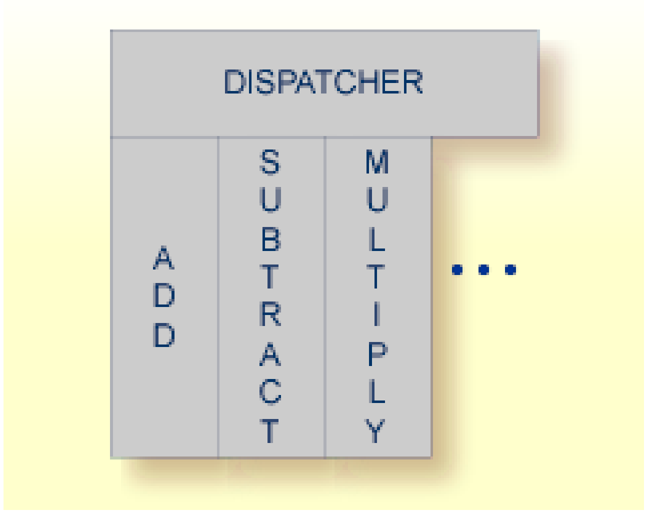
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Programming Techniques

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

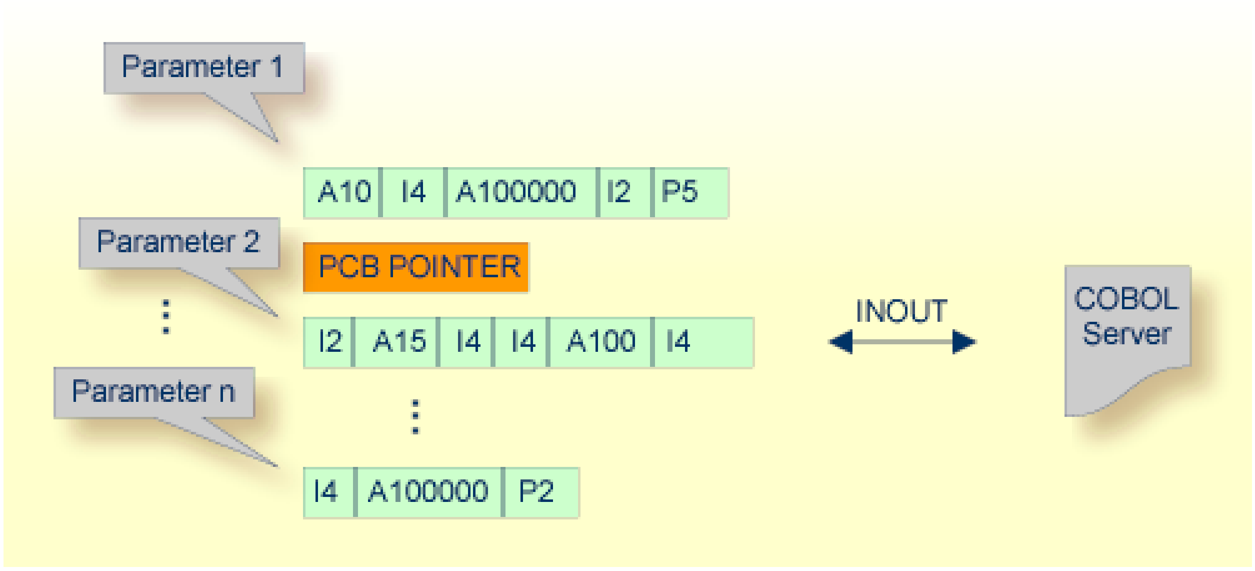
You can expose each COBOL server program function separately. The advantages or reasons for wanting this depend on the target endpoint. For example:

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

9 IMS BMP with Standard Linkage Calling Convention

- Introduction 128
- Extracting from an IMS BMP Standard Call Interface 128
- Mapping Editor User Interface 130
- Mapping Editor IDL Interface Mapping Functions 137
- Programming Techniques 141



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.

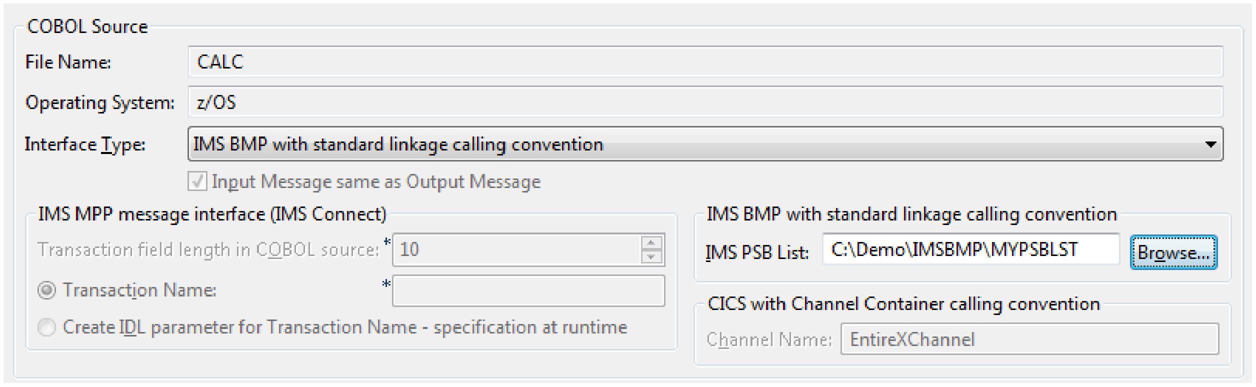
If the IMS PSB list is correctly assigned, the COBOL data items (including the PCB pointers) can be evaluated by the extractor because this type of COBOL server contains a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters. In most cases the offered COBOL data items 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.



COBOL Source

File Name: CALC

Operating System: z/OS

Interface Type: IMS BMP with standard linkage calling convention

[✓] Input Message same as Output Message

IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: * 10

(●) Transaction Name: *

( ) Create IDL parameter for Transaction Name - specification at runtime

IMS BMP with standard linkage calling convention

IMS PSB List: C:\Demo\IMSBMP\MYPSBLST

CICS with Channel Container calling convention

Channel Name: EntireXChannel

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 IMS RPC Server 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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL server contain `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.

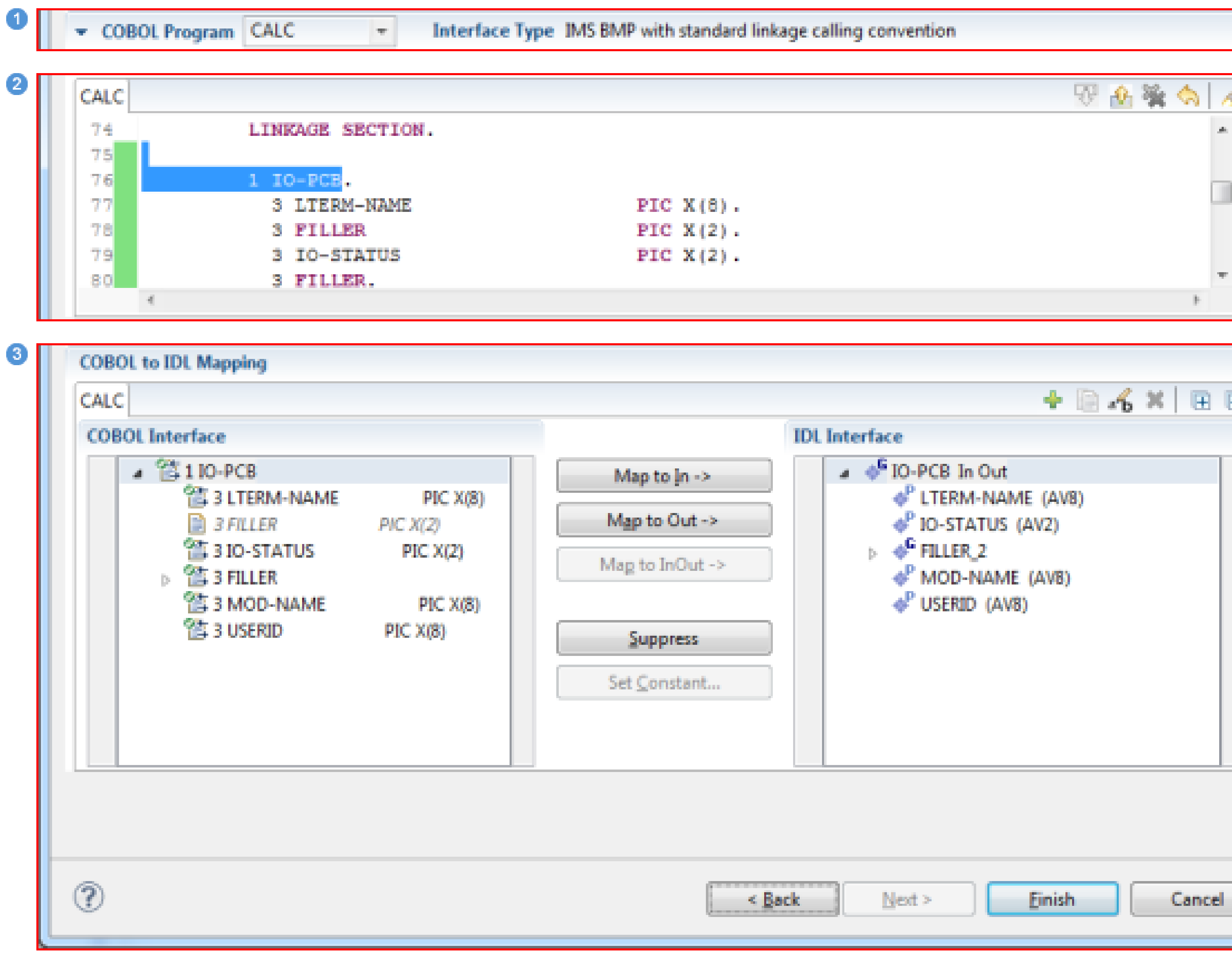
The user interface of the COBOL Mapping Editor is described below.

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:



COBOL Program CALC Interface Type IMS BMP with standard linkage calling convention

CALC

```
        LINKAGE SECTION.

          1 IO-PCB.
            3 LTERM-NAME                PIC X(8).
            3 FILLER                    PIC X(2).
            3 IO-STATUS                 PIC X(2).
            3 FILLER.
```

COBOL Interface:

```
1 IO-PCB
   3 LTERM-NAME          PIC X(8)
   3 FILLER              PIC X(2)
   3 IO-STATUS           PIC X(2)
   3 FILLER
   3 MOD-NAME            PIC X(8)
   3 USERID              PIC X(8)
```

IDL Interface:

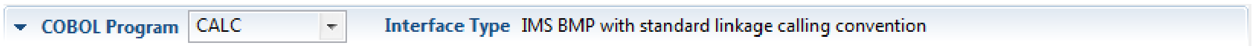
```
IO-PCB In Out
   LTERM-NAME (AV8)
   IO-STATUS (AV2)
   FILLER_2
   MOD-NAME (AV8)
   USERID (AV8)
```

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

2 **COBOL Source View**. Contains all related sources for the currently selected COBOL program

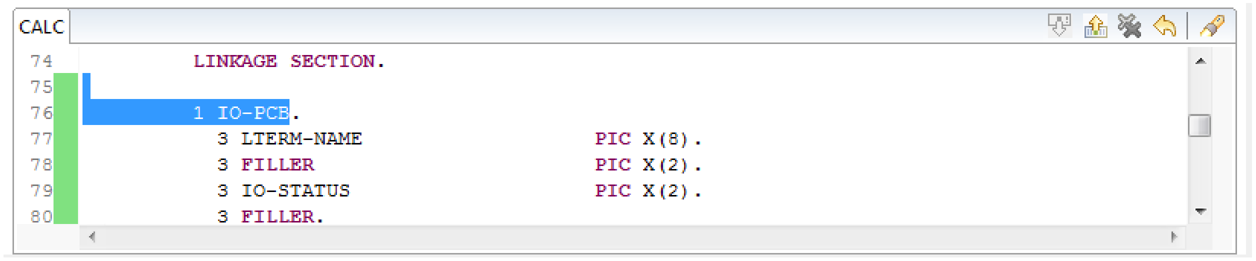
3 **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

COBOL Program Selection


COBOL Program CALC
Interface Type IMS BMP with standard linkage calling convention

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

COBOL Source View



```
       LINKAGE SECTION.

       1 IO-PCB.
         3 LTERM-NAME                PIC X(8).
         3 FILLER                    PIC X(2).
         3 IO-STATUS                 PIC X(2).
         3 FILLER.
```

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

Add selected COBOL data item to COBOL Interface.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

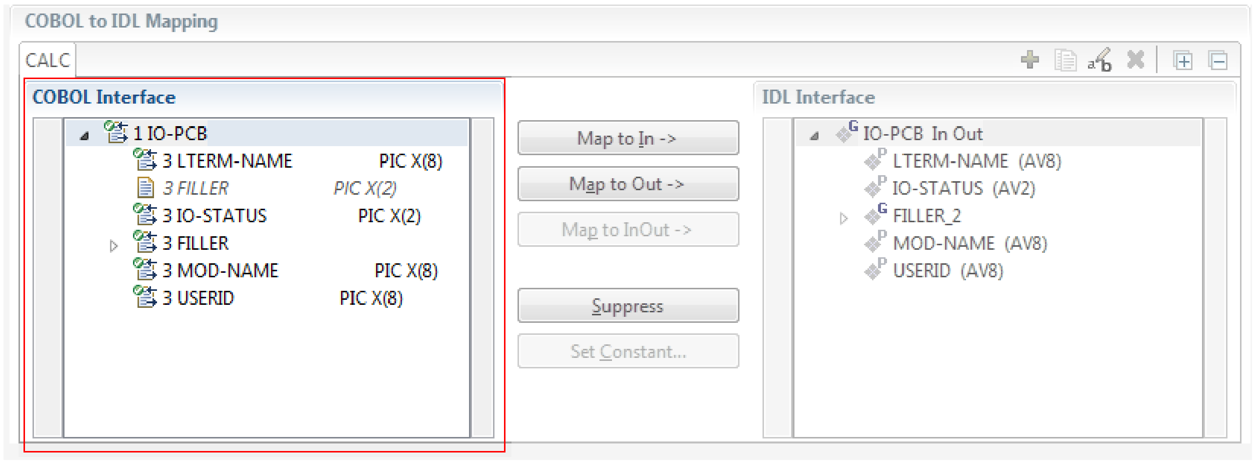
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to 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.
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***.

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".

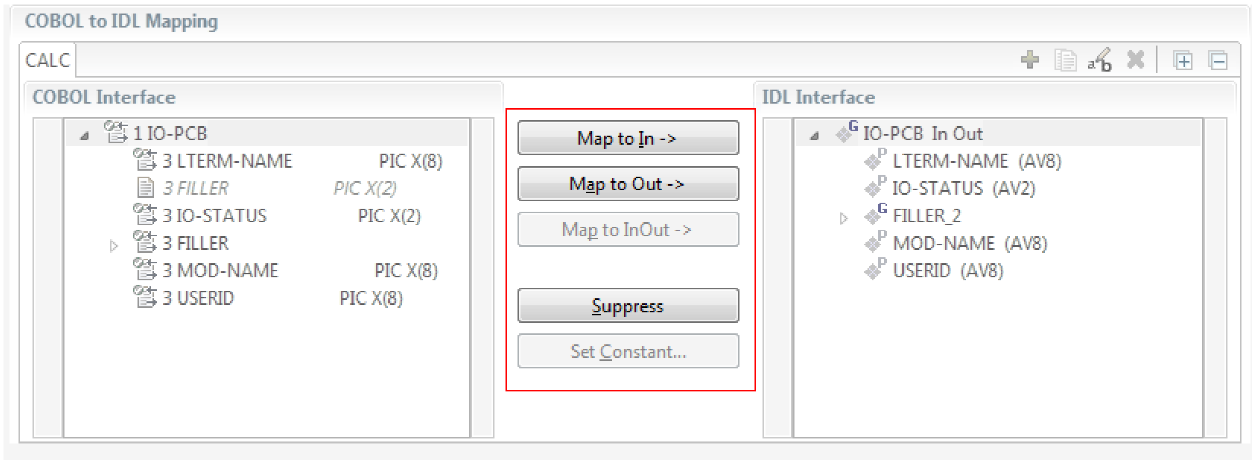
Mapping Icons

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.

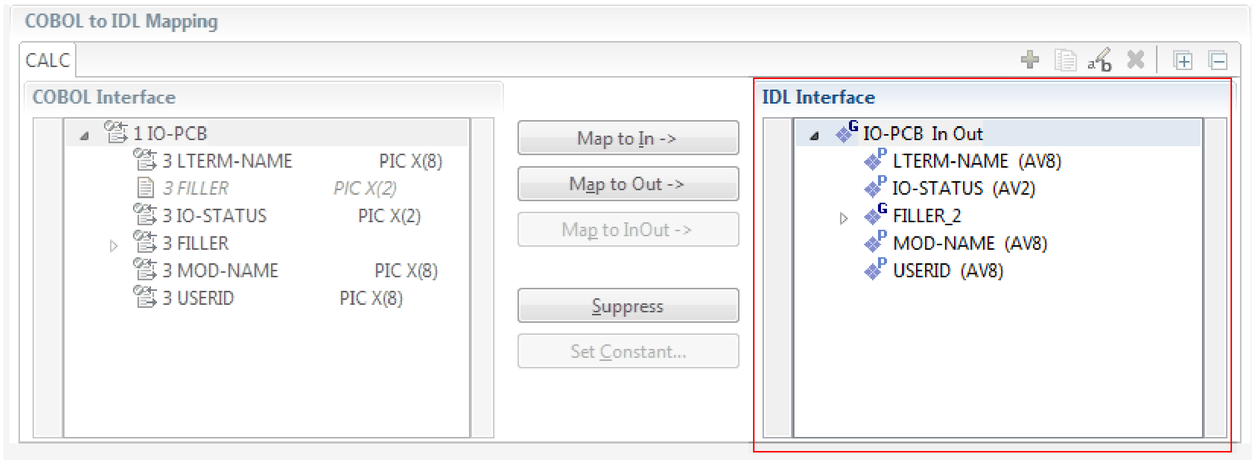
Suppress
See *Suppress Unneeded COBOL Data Items*.

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
- 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
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

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 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 subsequent child COBOL data items and thus to the related IDL parameters in the IDL interface.
2. Subsequent child 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 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), z/VSE (CICS | Batch), Micro Focus or BS2000/OSD RPC server, the amount of data to be transferred to/from the RPC client is reduced with correct IDL directions.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.
2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to In**, **Out** or **InOut** (see above) available in the context menu and as mapping button, 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 server program can be made invisible in the IDL interface and initialized with the required constant values. This is useful for keeping the IDL interface short and tidy. RPC clients or Adapter Services are not bothered with IDL parameters that always contain constants, such as `RECORD-TYPES`. This function is often used in conjunction with **Map to Multiple IDL Interfaces** (see below).

To map COBOL data items to constants

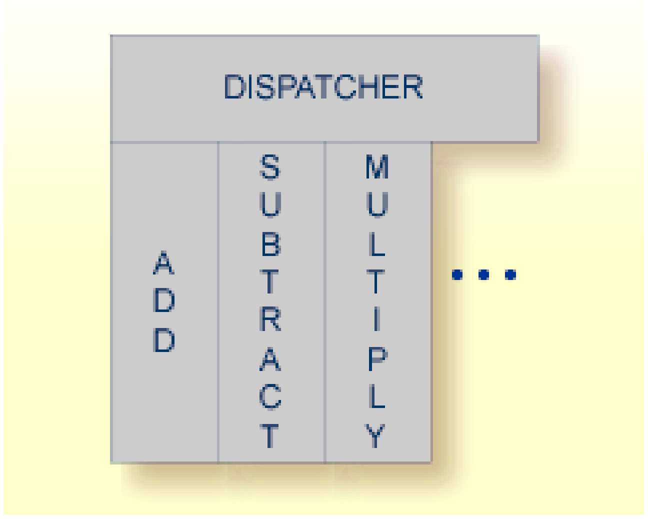
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

2 Give the IDL interfaces meaningful names with the toolbar function ✎.

3 Define the required constant values to the *function-code* or *operation-code* parameter, see *Set COBOL Data Items to Constants* above.

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Programming Techniques

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

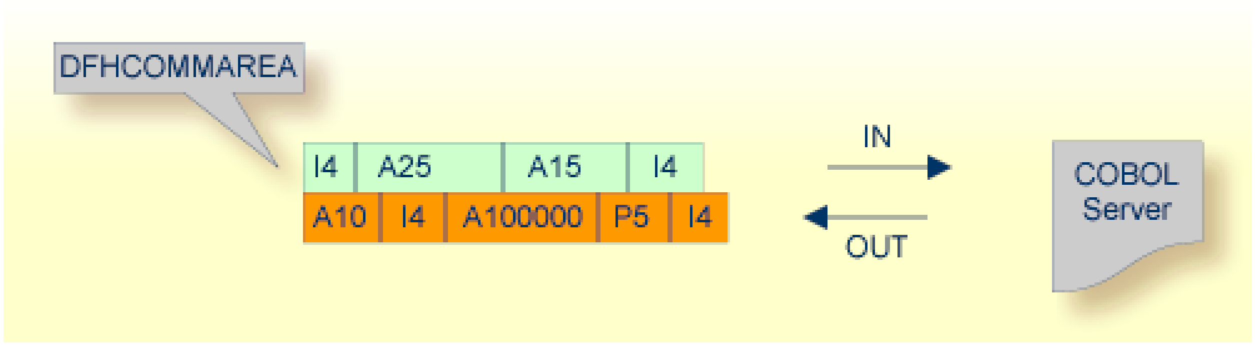
You can expose each COBOL server program function separately. The advantages or reasons for wanting this depend on the target endpoint. For example:

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

10 CICS with DFHCOMMAREA Calling Convention - In different to Out

- Introduction 144
- Extracting from a CICS DFHCOMMAREA Program 144
- Mapping Editor User Interface 145
- Mapping Editor IDL Interface Mapping Functions 152
- Programming Techniques 156



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. Please note also 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.

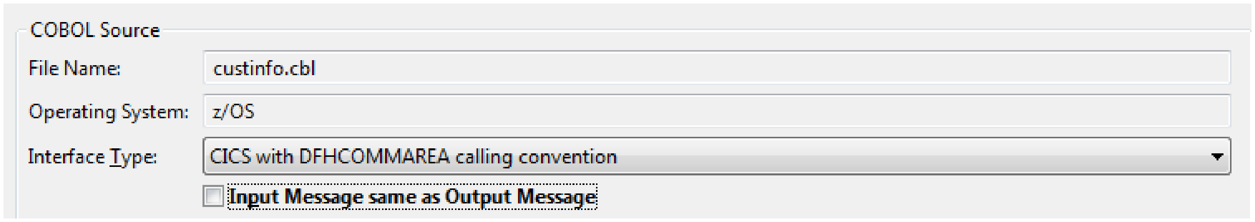
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

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.


COBOL Source
File Name: custinfo.cbl
Operating System: z/OS
Interface Type: CICS with DFHCOMMAREA calling convention
☐ Input Message same as Output Message

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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL server contain `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. See the examples provided under *Programming Techniques*.

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

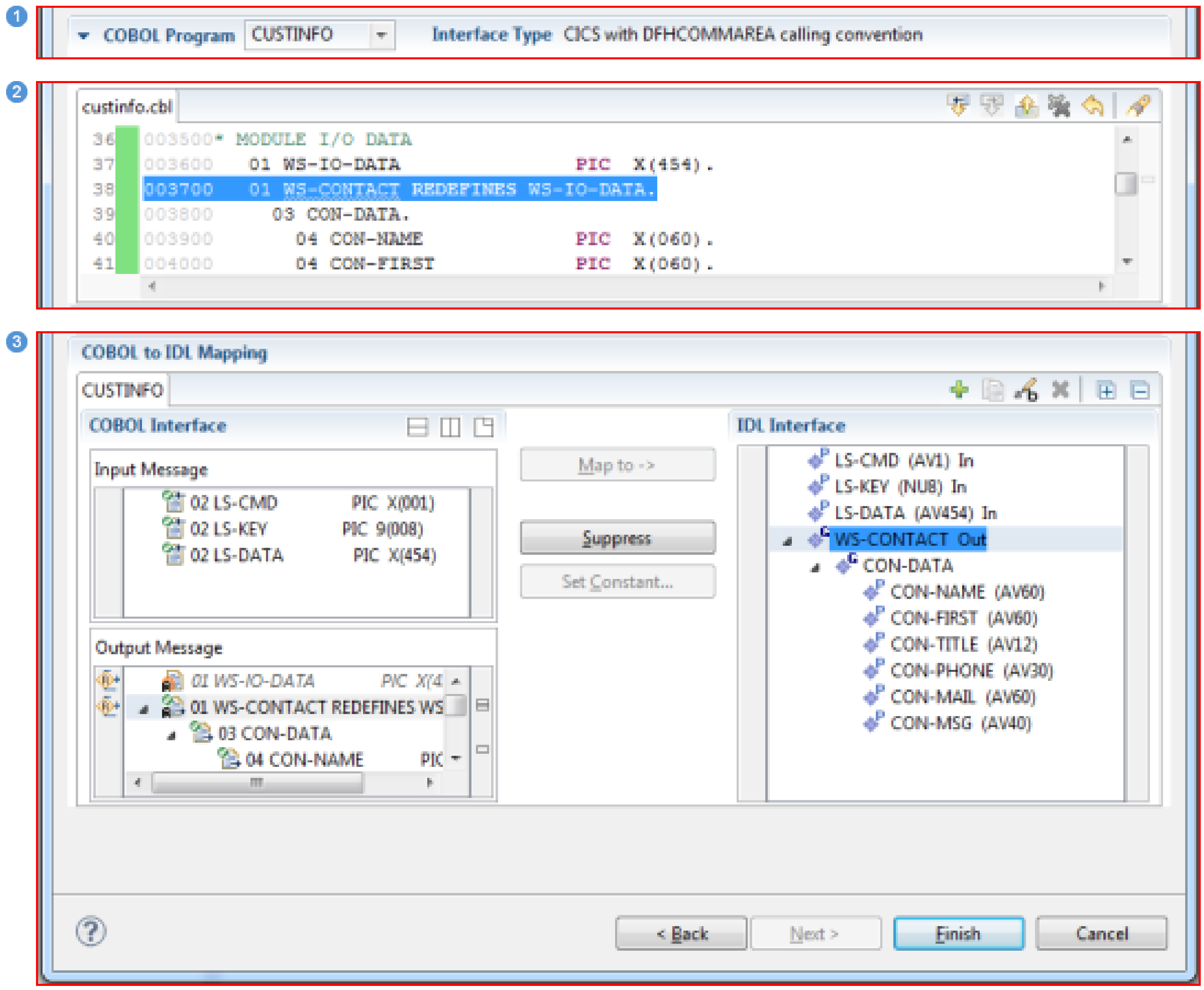
Mapping Editor User Interface

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:



COBOL Program CUSTINFO Interface Type CICS with DFHCOMMAREA calling convention

custinfo.cbl

```
003500* MODULE I/O DATA
003600    01 WS-IO-DATA                    PIC   X(454).
003700    01 WS-CONTACT REDEFINES WS-IO-DATA.
003800       03 CON-DATA.
003900          04 CON-NAME                PIC   X(060).
004000          04 CON-FIRST               PIC   X(060).
```

COBOL to IDL Mapping

CUSTINFO

COBOL Interface

Input Message

```
02 LS-CMD      PIC X(001)
02 LS-KEY      PIC 9(008)
02 LS-DATA     PIC X(454)
```

Output Message

```
01 WS-IO-DATA              PIC X(4
01 WS-CONTACT REDEFINES WS
   03 CON-DATA
      04 CON-NAME          PIC
```

IDL Interface

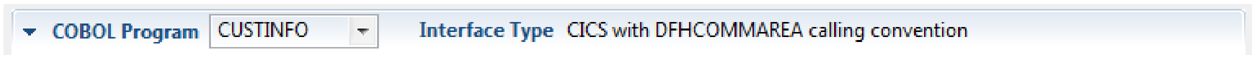
```
LS-CMD (AV1) In
LS-KEY (NU8) In
LS-DATA (AV454) In
WS-CONTACT Out
   CON-DATA
      CON-NAME (AV60)
      CON-FIRST (AV60)
      CON-TITLE (AV12)
      CON-PHONE (AV30)
      CON-MAIL (AV60)
      CON-MSG (AV40)
```

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

2 **COBOL Source View**. Contains all related sources for the currently selected COBOL program

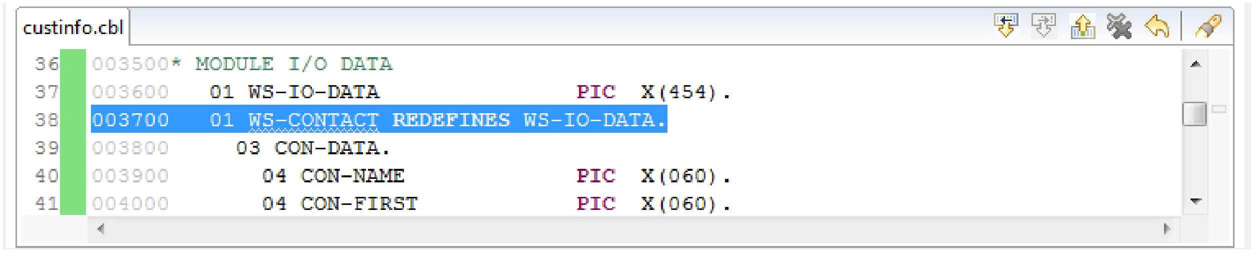
3 **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

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 associated IDL file, you can switch to another COBOL program with its mapping by selecting the name in the combo box.

COBOL Source View


custinfo.cbl

```
003500* MODULE I/O DATA
003600    01 WS-IO-DATA                PIC  X(454).
003700    01 WS-CONTACT REDEFINES WS-IO-DATA.
003800      03 CON-DATA.
003900        04 CON-NAME              PIC  X(060).
004000        04 CON-FIRST             PIC  X(060).
```

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

Add selected COBOL data item to COBOL Interface as Input Message.

Add selected COBOL data item to COBOL Interface as Output Message.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

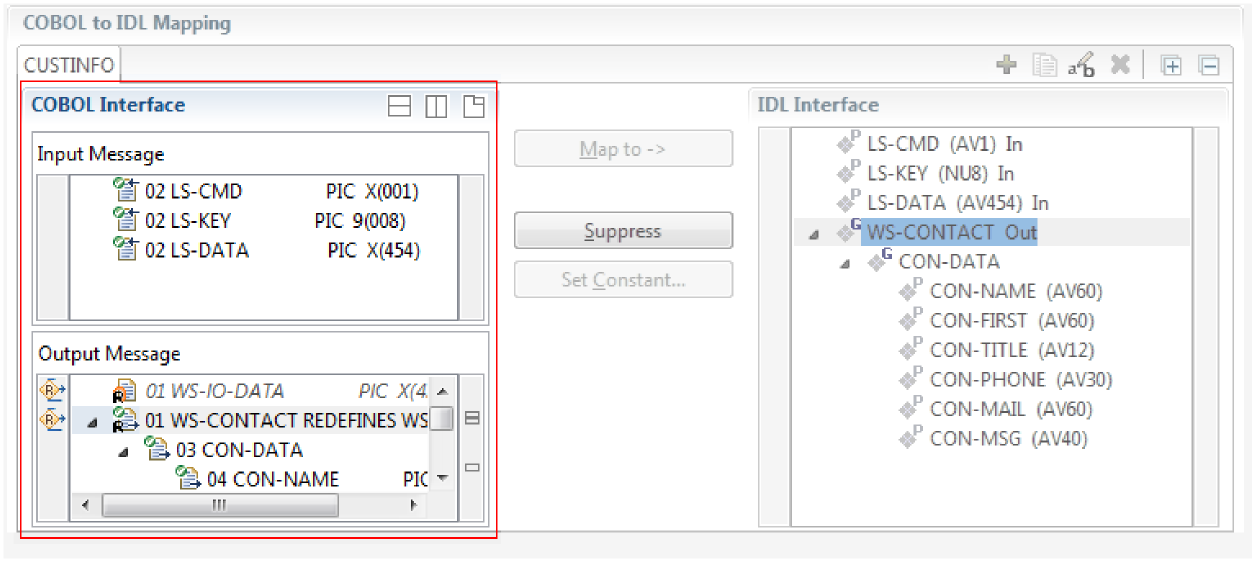
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to	A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.
Suppress	Suppress unneeded COBOL data items.
Set Constant	Set COBOL data items to constant.

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*.

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".

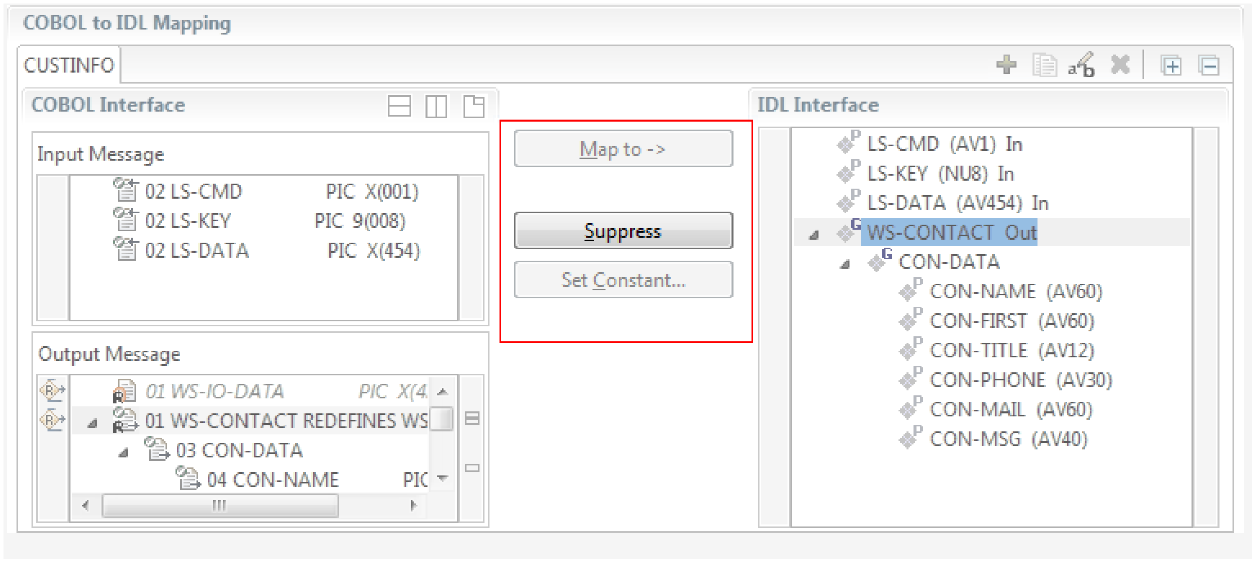
Mapping Icons

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.

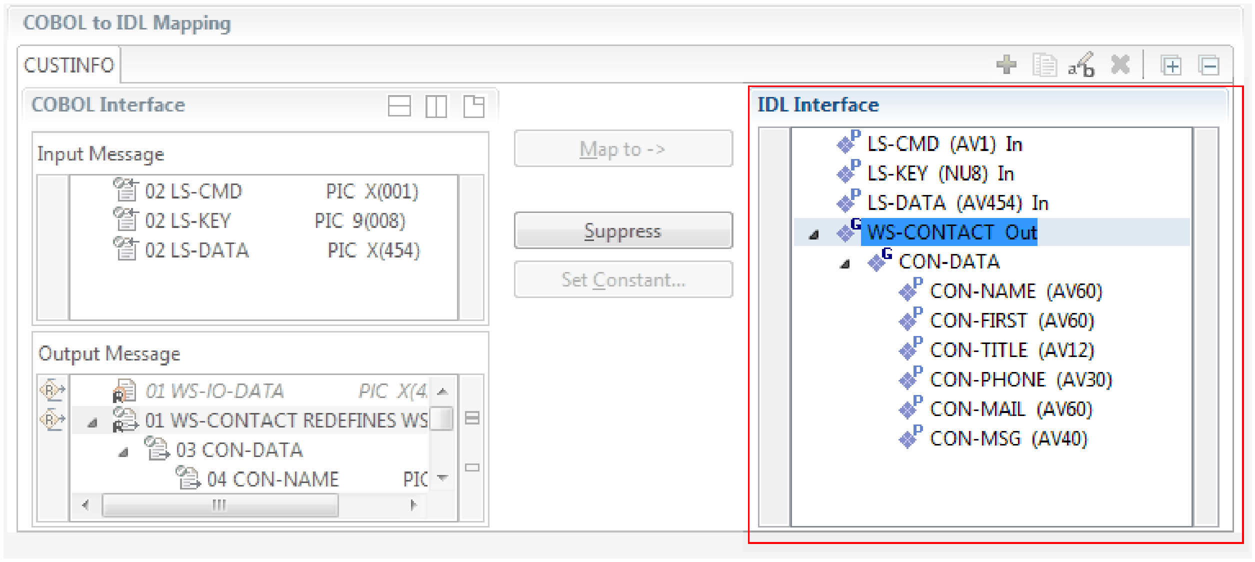
Suppress
: See *Suppress Unneeded COBOL Data Items*.

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
- 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
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

Map to

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

To map a COBOL data item to IDL interface

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

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

Notes:

1. If a COBOL group is mapped, all subsequent child COBOL data items are also made visible in the IDL interface.

2. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu and as mapping button, a COBOL data item can be removed from the IDL interface.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

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

Set COBOL Data Items to Constants

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

To map COBOL data items to constants

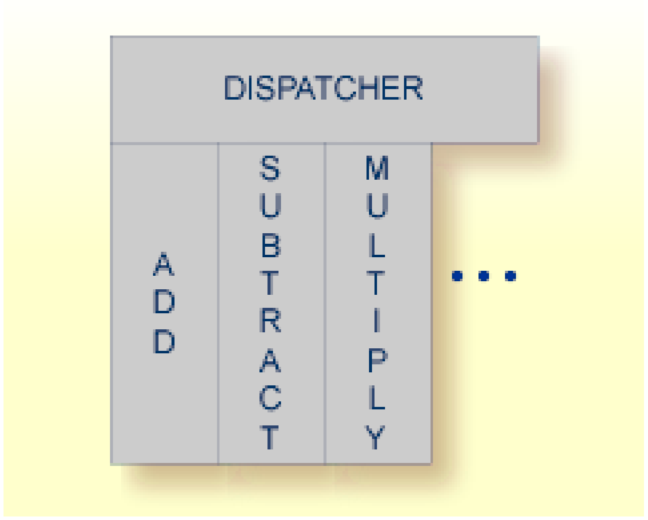
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Programming Techniques

This section covers the following topics:

- Example 1: COBOL Server with Multiple Functions
- Example 2: Redefines
- Example 3: Buffer Technique
- Example 4: COBOL SET ADDRESS Statements

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

Example 2: 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.

```
 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.
```

Example 3: 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.

```
 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.
```

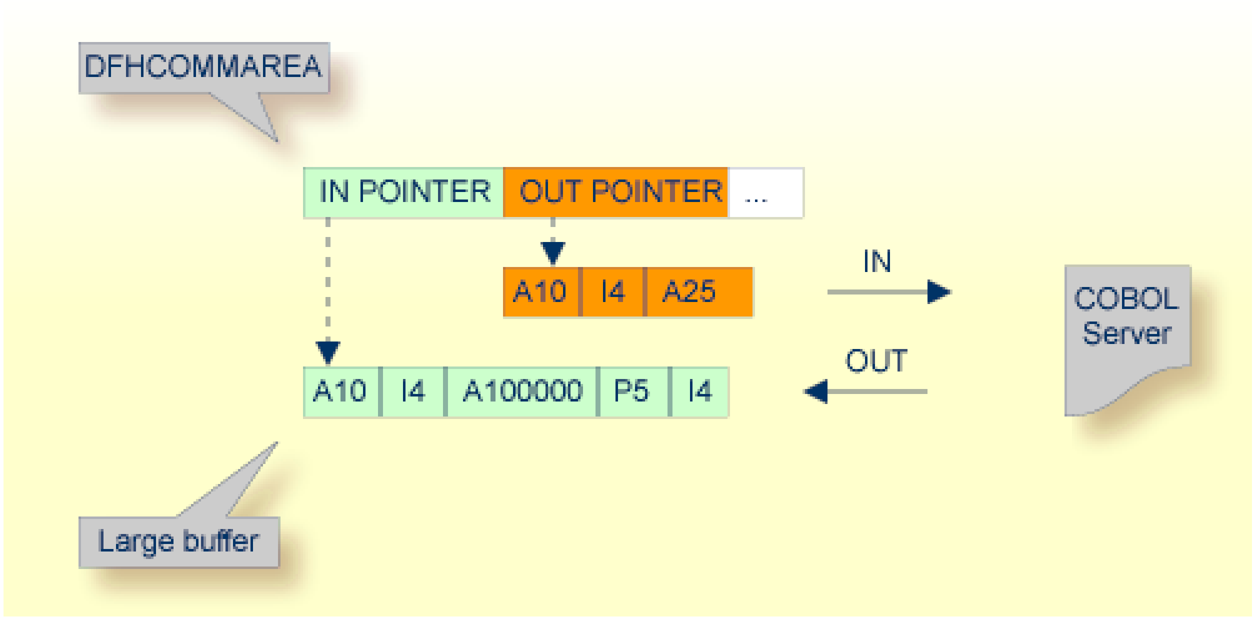
Example 4: COBOL SET ADDRESS Statements

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.

```
 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.
```

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

- Introduction 162
- Extracting from a CICS DFHCOMMAREA Large Buffer Program 163
- Mapping Editor User Interface 164
- Mapping Editor IDL Interface Mapping Functions 171
- Programming Techniques 176



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.

```
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.
```

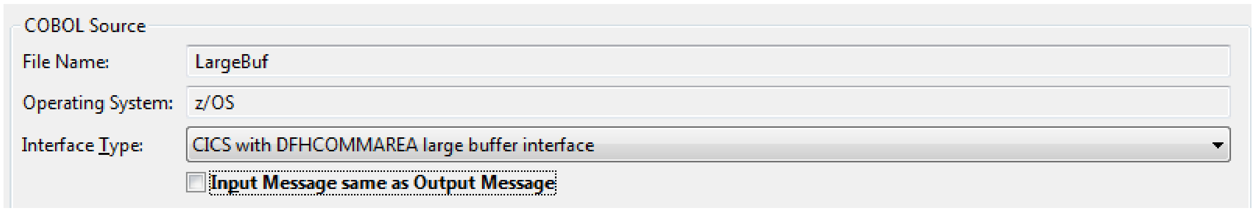
If you have selected an IDL file and opened the COBOL Mapping Editor with an existing COBOL to IDL mapping, continue with *Mapping Editor User Interface*.

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.


COBOL Source
File Name: LargeBuf
Operating System: z/OS
Interface Type: CICS with DFHCOMMAREA large buffer interface
☐ Input Message same as Output Message

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 interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
3. If your COBOL server contain `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.

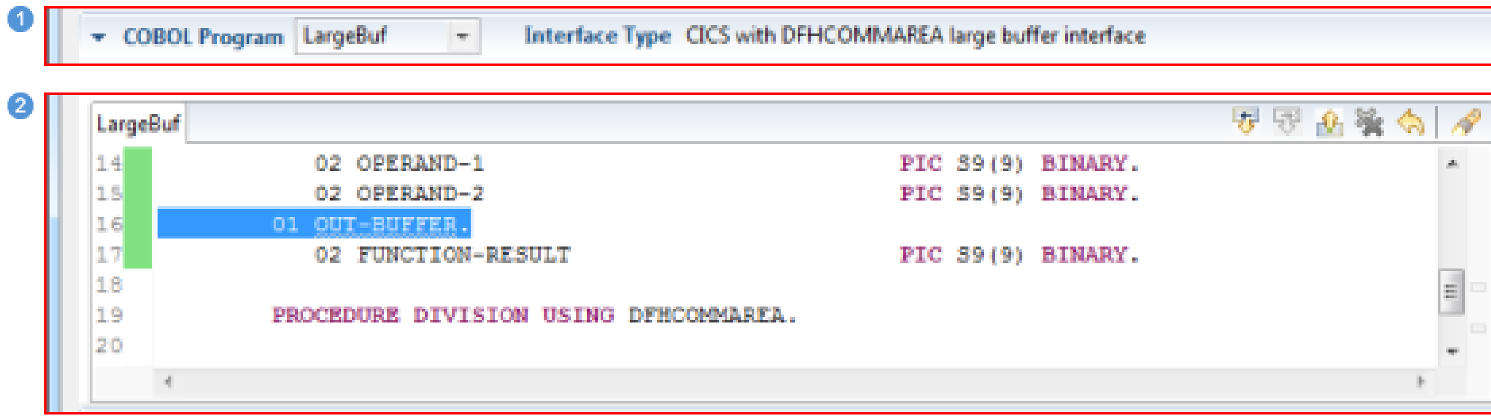
The user interface of the COBOL Mapping Editor is described below.

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:

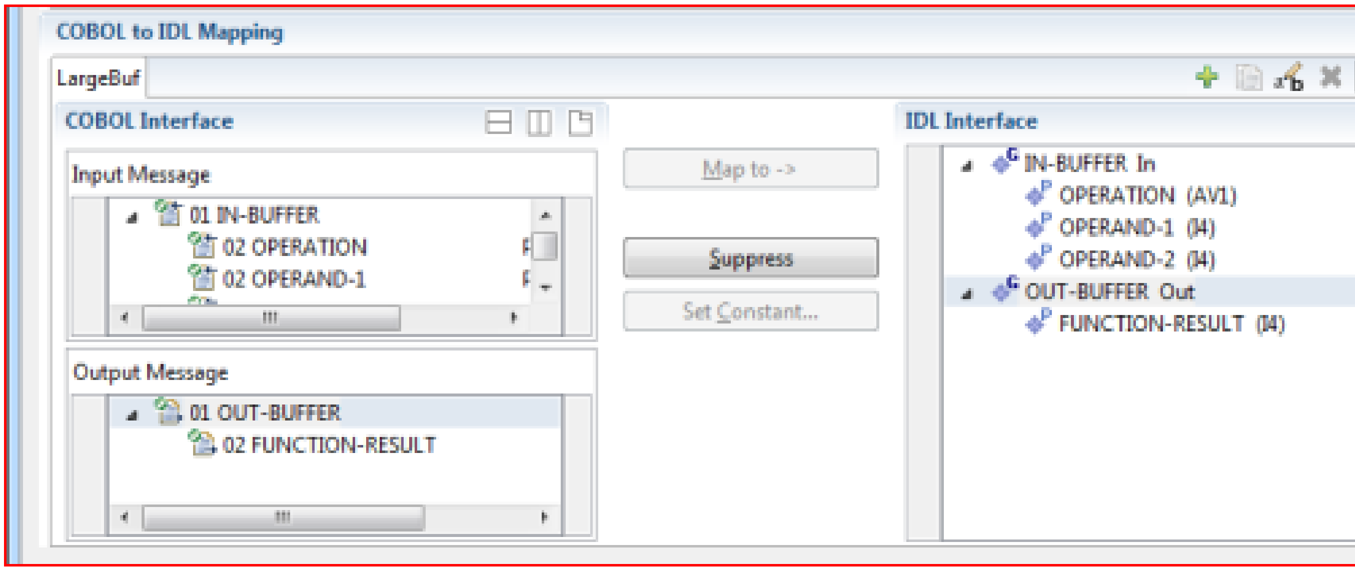
- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

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



```
           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.
```

3

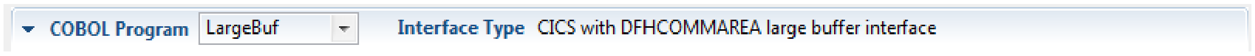


1 **COBOL Program Selection**. Currently selected program with interface type

2 **COBOL Source View**. Contains all related sources for the currently selected COBOL program

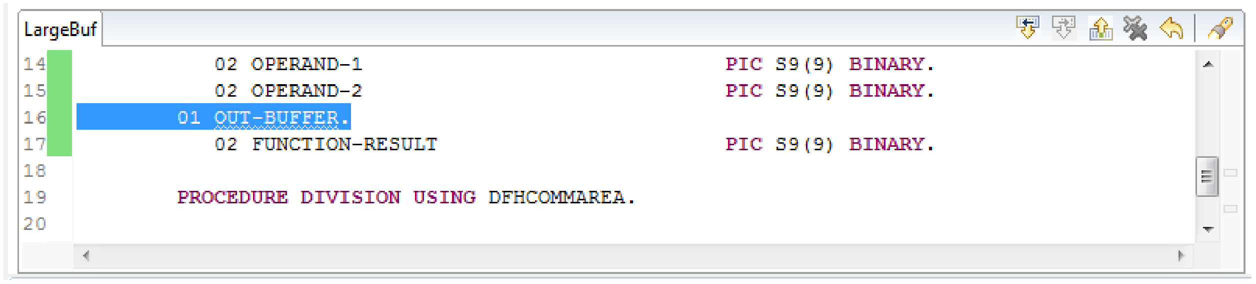
3 **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

COBOL Program Selection


COBOL Program LargeBuf Interface Type CICS with DFHCOMMAREA large buffer interface

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

COBOL Source View


LargeBuf

```
            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.
```

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

Add selected COBOL data item to COBOL Interface as Input Message.

Add selected COBOL data item to COBOL Interface as Output Message.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

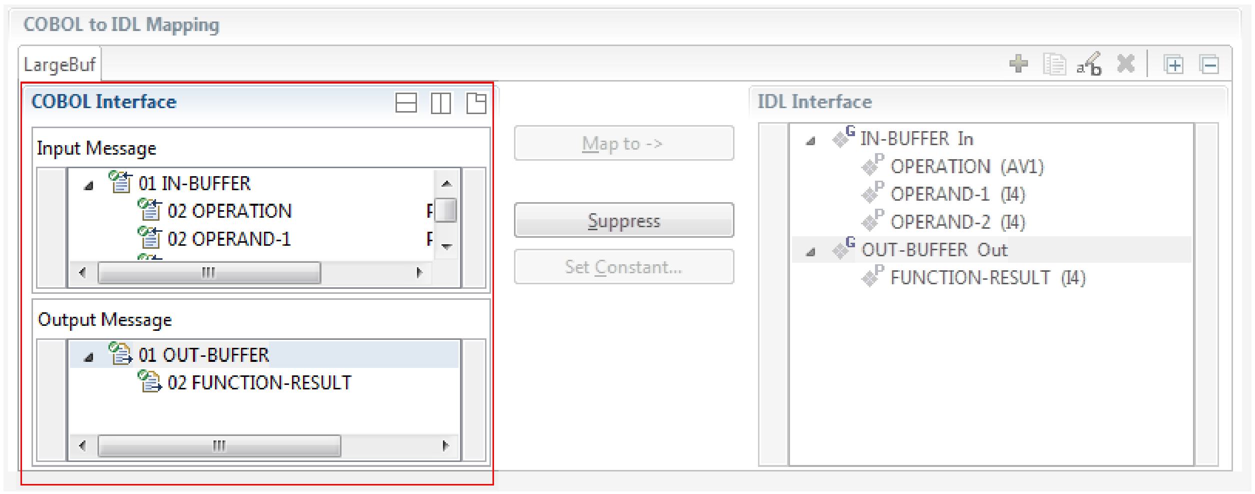
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



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

Context Menu

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

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

Map to	A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.
Suppress	Suppress unneeded COBOL data items.
Set Constant	Set COBOL data items to constant.

Set Array Mapping	Map an array to a fixed sized or unbounded array. **Note:** This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping could result in runtime errors.
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*.

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:

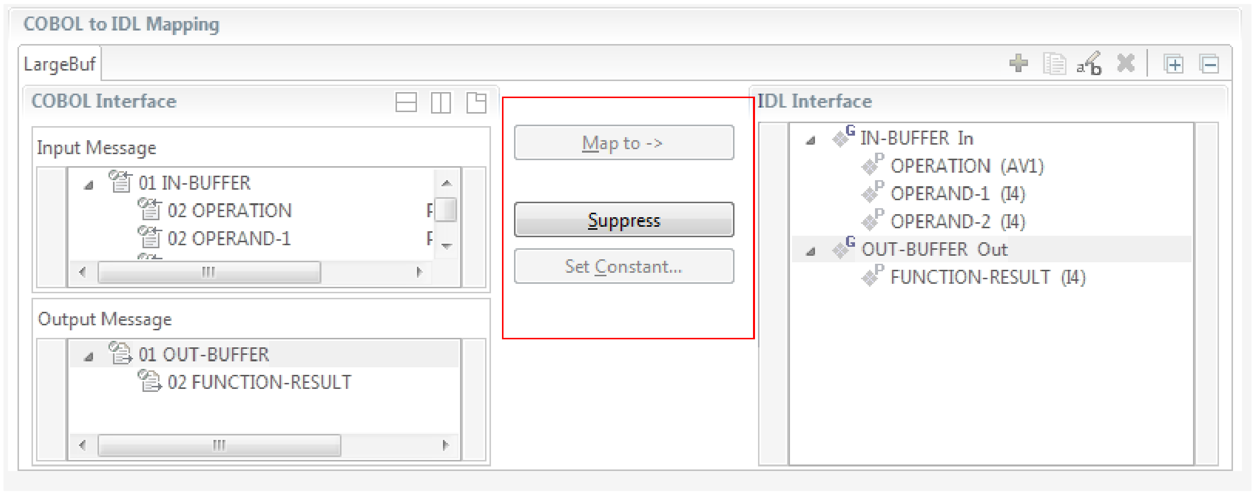
- Scalar parameter, mapped to In.
- Scalar parameter, mapped to Out.
- Group parameter, here mapped to In.

`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.

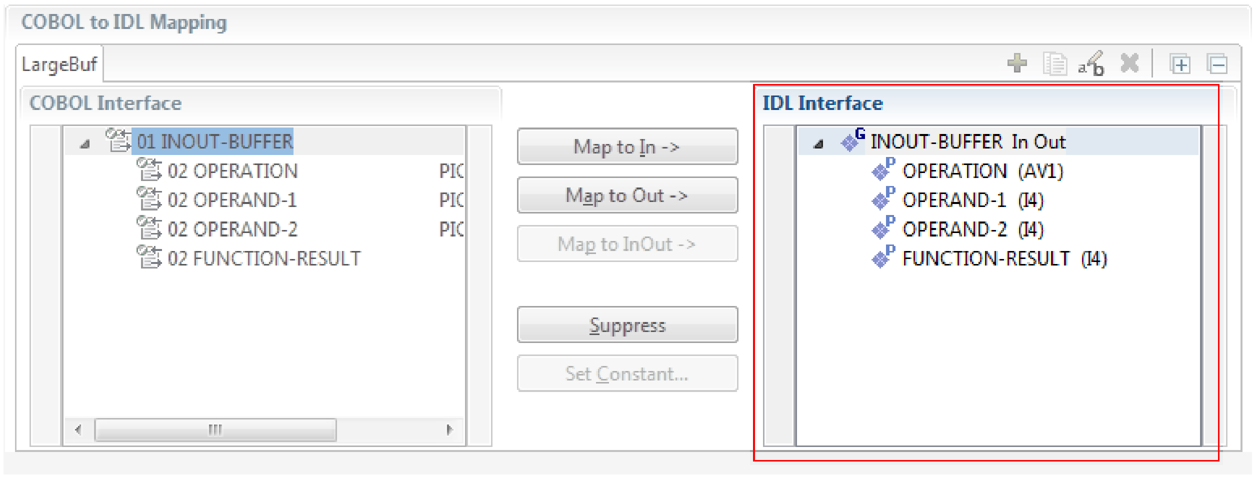
Suppress
See *Suppress Unneeded COBOL Data Items*.

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

Map to

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

To map a COBOL data item to IDL interface

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

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

Notes:

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

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

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

Set COBOL Data Items to Constants

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

To map COBOL data items to constants

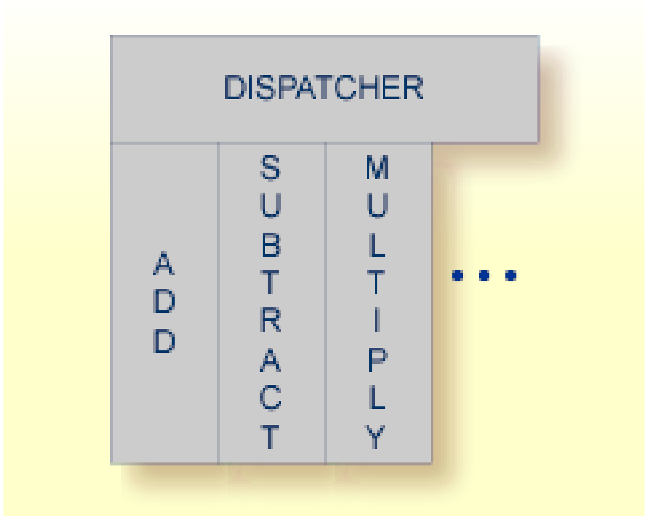
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

2 Give the IDL interfaces meaningful names with the toolbar function ✎.

3 Define the required constant values to the *function-code* or *operation-code* parameter, see *Set COBOL Data Items to Constants* above.

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Set Arrays (Fixed <-> Unbounded)

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

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

To set arrays from fixed to unbounded or vice versa

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

Notes:

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

table and must not contain any `DEPENDING ON` clause (see ***Tables with Variable Size - `DEPENDING ON` Clause***).

Programming Techniques

This section covers the following topics:

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

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

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

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

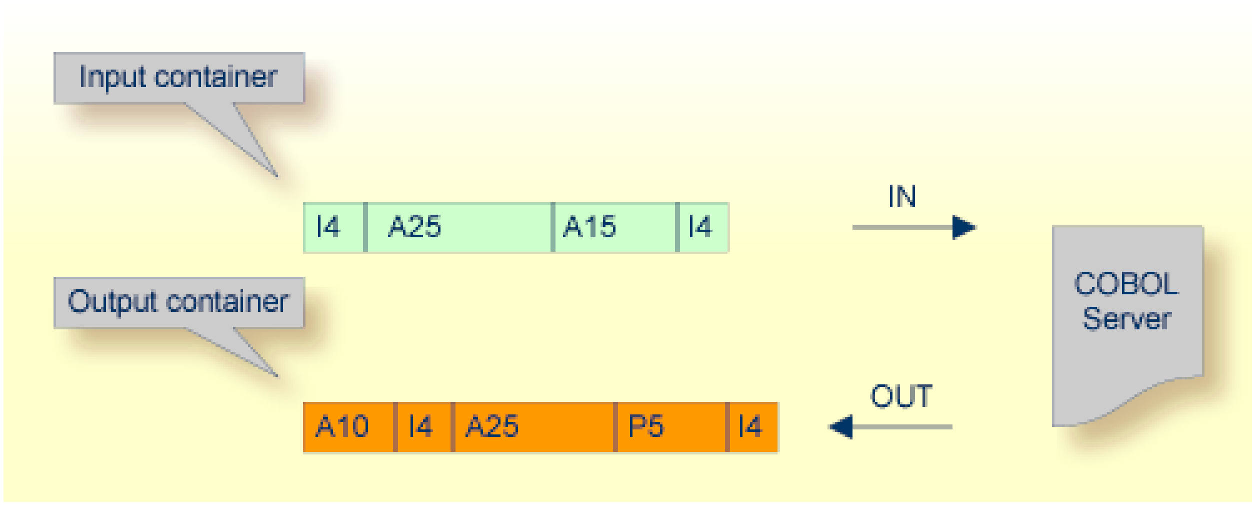
```
         WORKING-STORAGE SECTION.
         01 NUMBER-OF-INCOMING-ELEMENTS       PIC S9(8) BINARY.
         01 NUMBER-OF-OUTGOING-ELEMENTS       PIC S9(8) BINARY.

          . . .

         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.
     10 COBOL-GROUP1.
             20 COBOL-TABLE-PREFIX            PIC X(1000).
           10 COBOL-TABLE-FIX                 OCCURS 20.
             20 COBOL-GROUP2.
               25 COBOL-FIELD1                PIC X(30).
               25 COBOL-FIELD2                PIC X(20).
               25 COBOL-FIELD3                PIC X(50).
              . . .
         PROCEDURE DIVISION USING DFHCOMMAREA.
            SET ADDRESS OF INOUT-BUFFER TO WM-LCB-INPUT-BUFFER.
            SET ADDRESS OF INOUT-BUFFER TO WM-LCB-OUTPUT-BUFFER.
            COMPUTE NUMBER-OF-INCOMING-ELEMENTS = (WM-LCB-INPUT-BUFFER-SIZE
                  - LENGTH OF COBOL-GROUP1)
                  / LENGTH OF COBOL-GROUP2.

              . . .
            COMPUTE WM-LCB-OUTPUT-BUFFER-SIZE = LENGTH OF COBOL-GROUP2
                  + NUMBER-OF-OUTGOING-ELEMENTS * LENGTH OF COBOL-GROUP2

     EXEC CICS RETURN END-EXEC.
```

During input the COBOL CICS large buffer server program uses the large buffer input length `WM-LCB-INPUT-BUFFER-SIZE` to evaluate the `NUMBER-OF-INCOMING-ELEMENTS`. During output the large buffer output length is determined accordingly to the `NUMBER-OF-OUTGOING-ELEMENTS` and set in `WM-LCB-OUTPUT-BUFFER-SIZE`.

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

12 CICS with Channel Container Calling Convention

- Introduction 180
- Extracting from a CICS Channel Container Program 180
- Mapping Editor User Interface 182
- Mapping Editor IDL Interface Mapping Functions 189
- Programming Techniques 194



Introduction

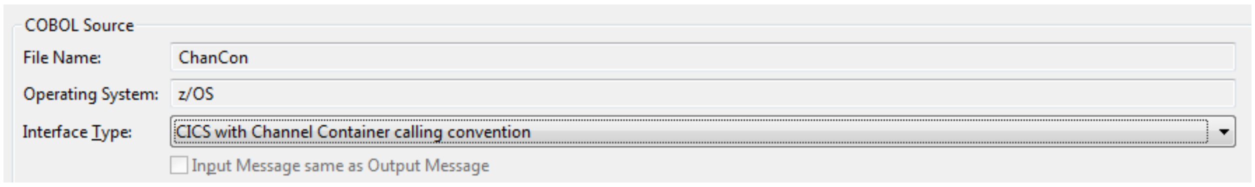
Modern CICS programs may use the CICS channels and containers model. During extraction, containers are mapped to IDL structures. See `structure-parameter-definition` (IDL) under *Software AG IDL Grammar* in the *IDL Editor* documentation.

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.


COBOL Source

File Name: ChanCon

Operating System: z/OS

Interface Type: CICS with Channel Container calling convention

☐ Input Message same as Output Message

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 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>`. The `EXEC CICS PUT` statement can be executed multiple times (for example in a loop) for the same container definition, creating an array of container. If this is true, set the column Array in the wizard to "Yes" and enter the maximum number of occurrences for the container in the **Max** column. You can select multiple CICS output containers. See **Notes**.

3 Continue with *COBOL to IDL Mapping*.

Notes:

1. It is very important to select the right COBOL data items describing the interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).

2. If your COBOL server contain `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.
4. Container arrays will enlarge the container name, because the number of occurrences (**Max** column) will be added to the name (max. 16 characters). Example:
   For `MYCONTAINER` as the container name and 99999 as the number of occurrences, the container names are `MYCONTAINER00001` - `MYCONTAINER99999`.

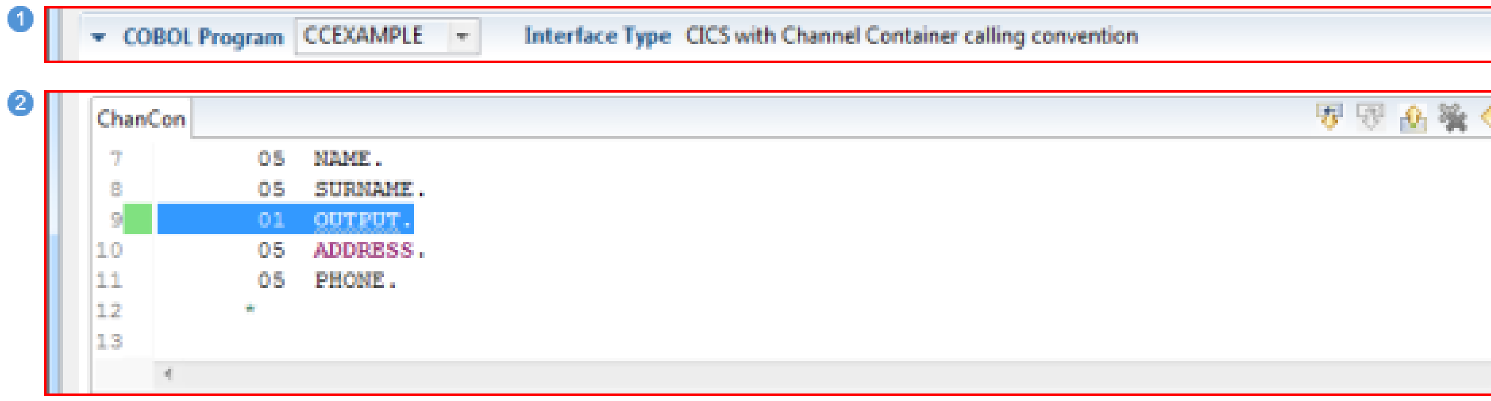
The user interface of the COBOL Mapping Editor is described below.

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:

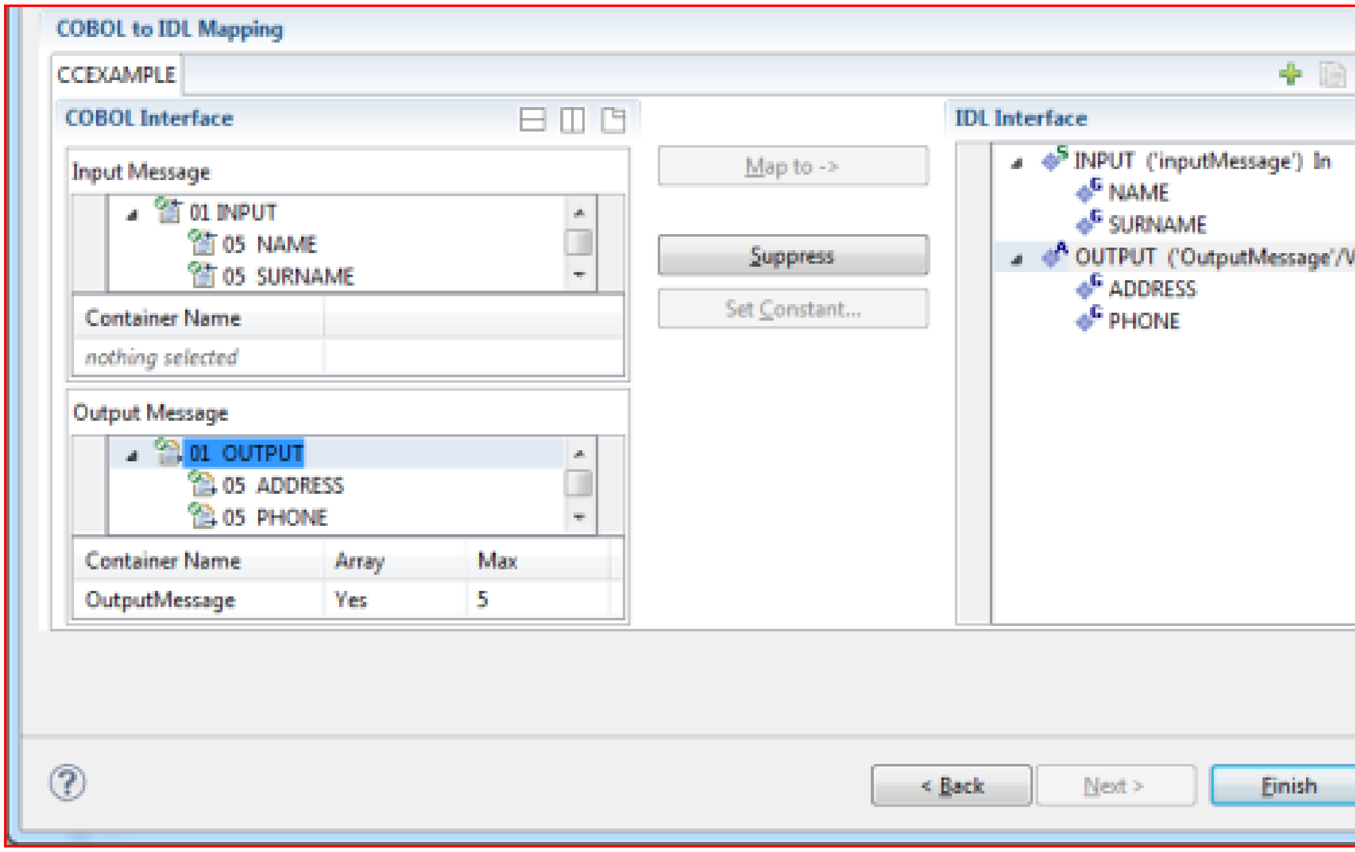
- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

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



```
           05   NAME.
           05   SURNAME.
           01   OUTPUT.
           05   ADDRESS.
           05   PHONE.
          *
```

3



Container Name
nothing selected

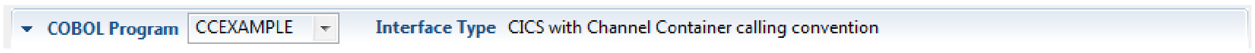
Container Name	Array	Max
OutputMessage	Yes	5

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

2 **COBOL Source View**. Contains all related sources for the currently selected COBOL program

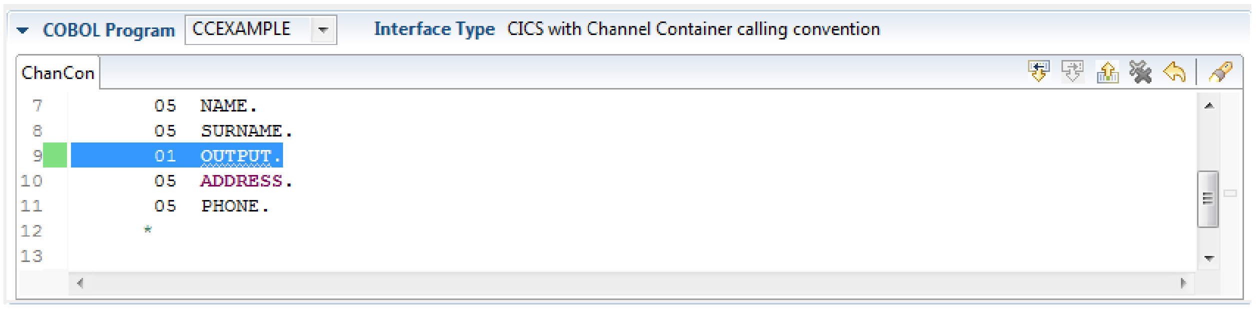
3 **COBOL to IDL Mapping**. Tree view of your selected COBOL data items and mapping buttons with which you can map these items to your IDL interface

COBOL Program Selection


COBOL Program CCEXAMPLE Interface Type CICS with Channel Container calling convention

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

COBOL Source View



```
        05  NAME.
        05  SURNAME.
        01  OUTPUT.
        05  ADDRESS.
        05  PHONE.
       *
```

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

Add selected COBOL data item to COBOL Interface as Input Message.

Add selected COBOL data item to COBOL Interface as Output Message.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

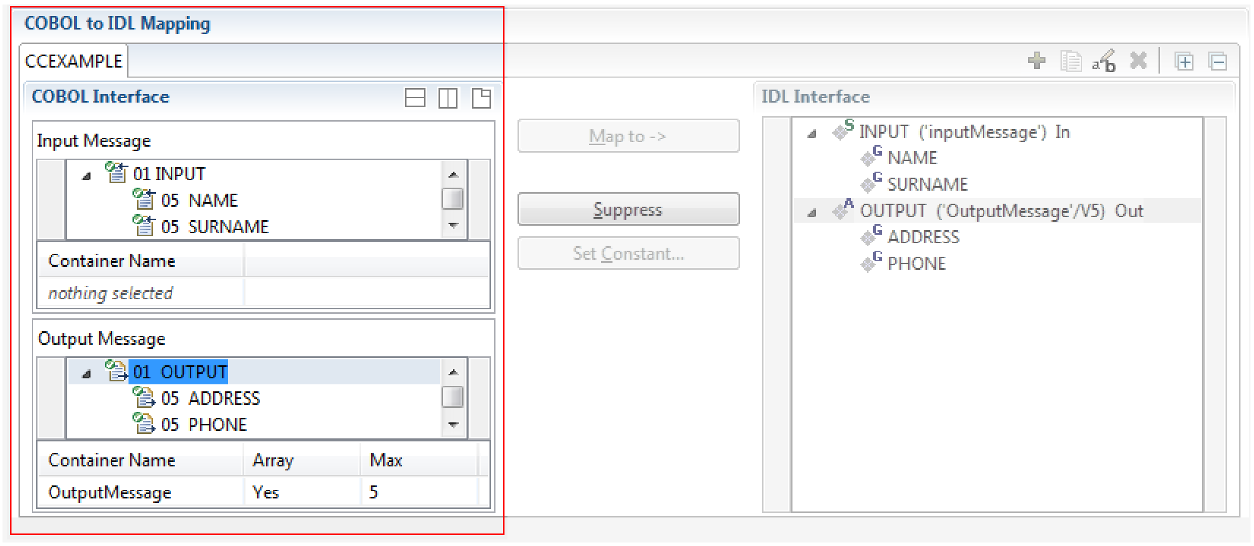
- COBOL Interface
- Mapping Buttons

- IDL Interface

COBOL Interface

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

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



Container Name		
nothing selected		

Container Name	Array	Max
OutputMessage	Yes	5

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

Context Menu

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

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

Map to	A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.
Suppress	Suppress unneeded COBOL data items.
Set Constant	Set COBOL data items to constant.

Set Array Mapping	Map an array to a fixed sized or unbounded array. **Note:** This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping could result in runtime errors.
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*.

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:

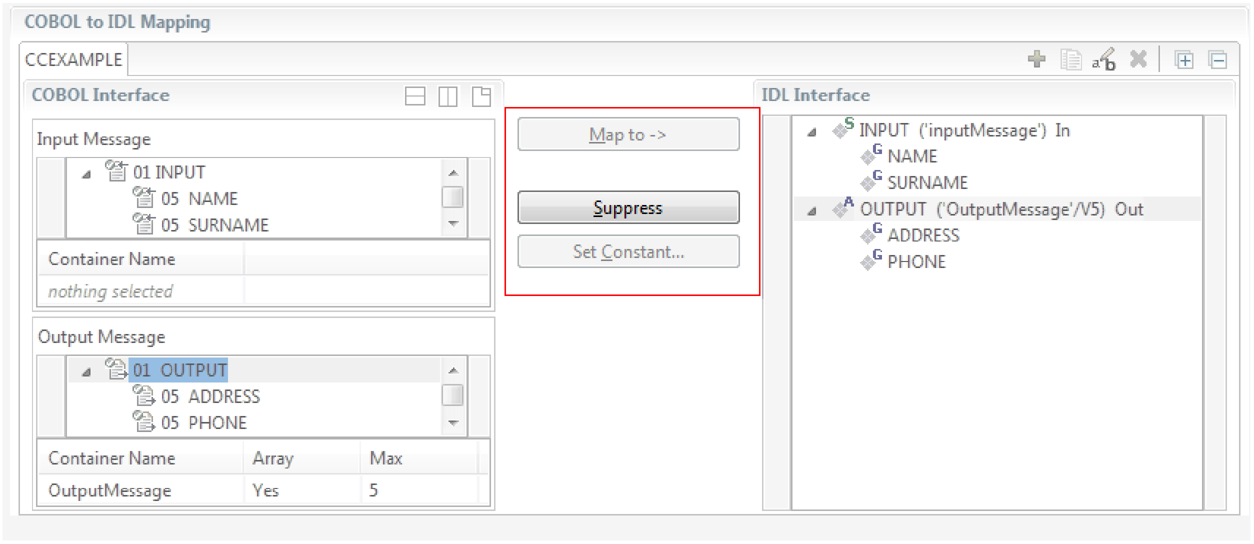
- Scalar parameter, mapped to In.
- Scalar parameter, mapped to Out.
- Group parameter, here mapped to In.

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.

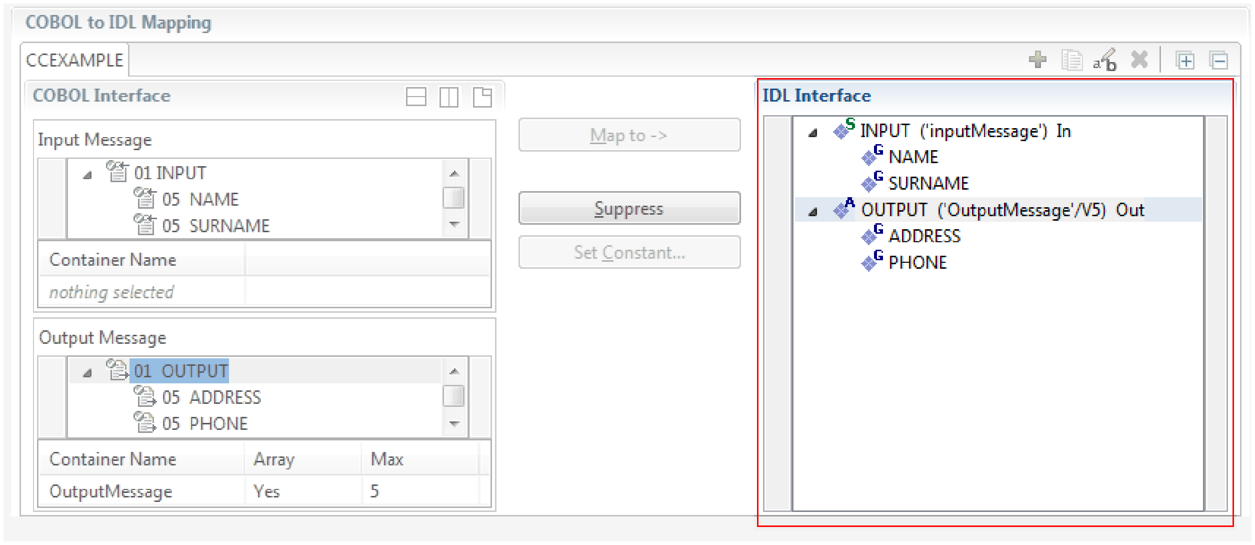
Suppress
See *Suppress Unneeded COBOL Data Items*.

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
- 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.



Container Name	Array	Max
OutputMessage	Yes	5

Mapping Editor IDL Interface Mapping Functions

This section covers the following topics:

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

Map to

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

To map a COBOL data item to IDL interface

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

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

Notes:

1. If a COBOL group is mapped, all subsequent child COBOL data items are also made visible in the IDL interface.

2. With the inverse function **Suppress Unneeded COBOL Data Items** (see below) available in the context menu and as mapping button, a COBOL data item can be removed from the IDL interface.

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

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

Set COBOL Data Items to Constants

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

To map COBOL data items to constants

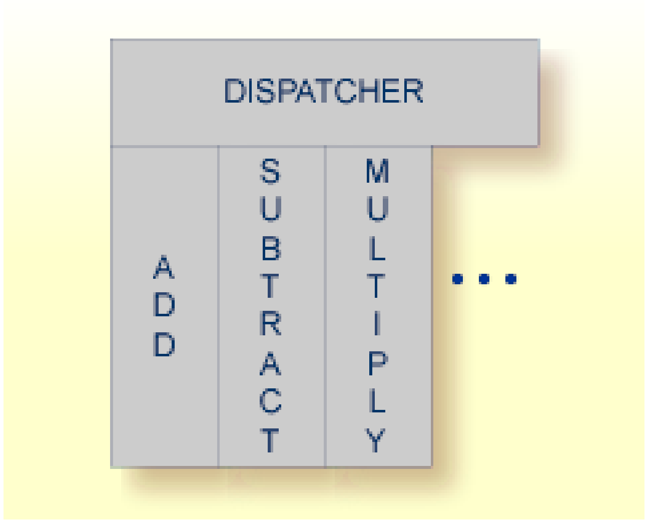
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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.*

Set Arrays (Fixed <-> Unbounded)

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

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

To set arrays from fixed to unbounded or vice versa

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

Notes:

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

table and must not contain any `DEPENDING ON` clause (see ***Tables with Variable Size -*** `DEPENDING ON` ***Clause***).

Programming Techniques

This section covers the following topics:

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

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

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

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

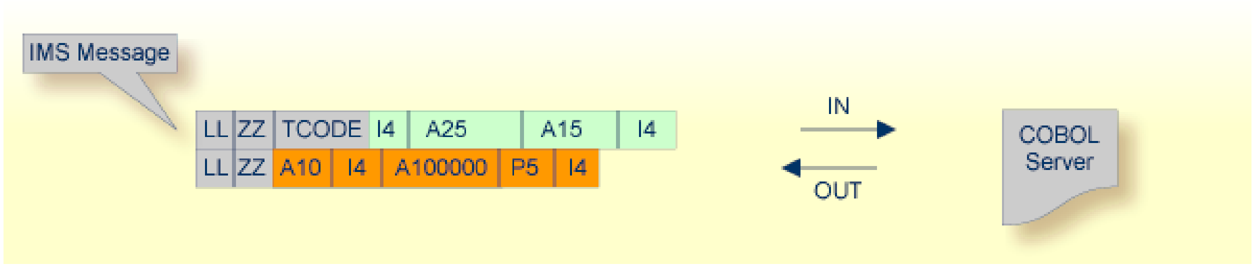
```
         WORKING-STORAGE SECTION.
         01 LS-CONTAINER-NAME                PIC X(16) VALUE "VAR-INPUT".
         01 WS-CONTAINER-NAME                PIC X(16) VALUE "VAR-OUTPUT".
         01 NUMBER-OF-INCOMING-ELEMENTS      PIC S9(8) BINARY.
         01 NUMBER-OF-OUTGOMING-ELEMENTS     PIC S9(8) BINARY.

          . . .

         01 WS-CONTAINER-LAYOUT.
           10 COBOL-GROUP1.
             20 COBOL-TABLE-PREFIX           PIC X(1000).
           10 COBOL-TABLE-FIX                OCCURS 20.
             20 COBOL-GROUP2.
               25 COBOL-FIELD1               PIC X(4).
               25 COBOL-FIELD2               PIC X(3).
               25 COBOL-FIELD3               PIC X(50).
         LINKAGE SECTION.
         01 LS-CONTAINER-LAYOUT.
           10 COBOL-GROUP1.
             20 COBOL-TABLE-PREFIX           PIC X(1000).
           10 COBOL-TABLE-FIX                OCCURS 20.
             20 COBOL-GROUP2.
               25 COBOL-FIELD1               PIC X(30).
               25 COBOL-FIELD2               PIC X(20).
               25 COBOL-FIELD3               PIC X(50).
              . . .
         PROCEDURE DIVISION.
            EXEC CICS GET
                CONTAINER (LS-CONTAINER-NAME
                SET       (ADDRESS OF LS-CONTAINER-LAYOUT)
                FLENGTH   (WS-CONTAINER-LENGTH)
                RESP      (WS-RESP)
                RESP2     (WS-RESP2)
            END-EXEC.
            COMPUTE NUMBER-OF-INCOMING-ELEMENTS = (WS-CONTAINER-LENGTH
                  - LENGTH OF COBOL-GROUP1 IN AREA LS-CONTAINER-LAYOUT)
                  / LENGTH OF COBOL-GROUP2 IN AREA LS-CONTAINER-LAYOUT.
              . . .
            COMPUTE WS-CONTAINER-LENGTH = LENGTH OF COBOL-GROUP2 IN AREA ↩
WS-CONTAINER-LAYOUT
                  + (NUMBER-OF-OUTGOING-ELEMENTS
                 * LENGTH OF COBOL-GROUP2 IN AREA WS-CONTAINER-LAYOUT).

            EXEC CICS PUT
                 CONTAINER (WS-CONTAINER-NAME)
                 FROM      (WS-CONTAINER-LAYOUT)
                 FLENGTH   (WS-CONTAINER-LENGTH)
                 RESP      (WS-RESP)
                 RESP2     (WS-RESP2)
            END-EXEC.
     EXEC CICS RETURN END-EXEC.
```

During input the COBOL channel container server program uses the container length `WS-CONTAINER-LENGTH` to evaluate the `NUMBER-OF-INCOMING-ELEMENTS`. During output the `WS-CONTAINER-LENGTH` is determined according to the `NUMBER-OF-OUTGOING-ELEMENTS` and set in the `EXEC CICS PUT CONTAINER` statement.

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

13 IMS MPP Message Interface (IMS Connect)

- Introduction 198
- Extracting from an IMS MPP Message Interface Program 199
- Mapping Editor User Interface 202
- Mapping Editor IDL Interface Mapping Functions 210
- Programming Techniques 215



Introduction

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

IMS Message Processing Programs (MPPs) work as follows:

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

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

Notes:

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

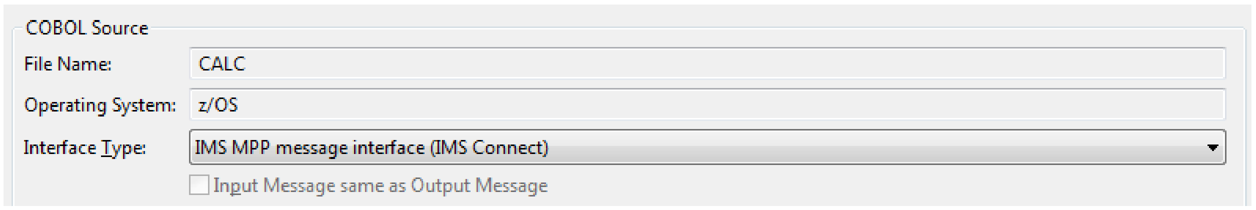
```
WORKING-STORAGE SECTION.
    . . .
 *  DLI Function Codes
    77 FCT-GU                          PIC X(4) VALUE 'GU  '.
    77 FCT-ISRT                        PIC X(4) VALUE 'ISRT'.
    . . .
    LINKAGE SECTION.
    . . .
    1 IO-PCB.
      3 LTERM-NAME                     PIC X(8).
      3 FILLER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

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

Extracting from an IMS MPP Message Interface Program

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

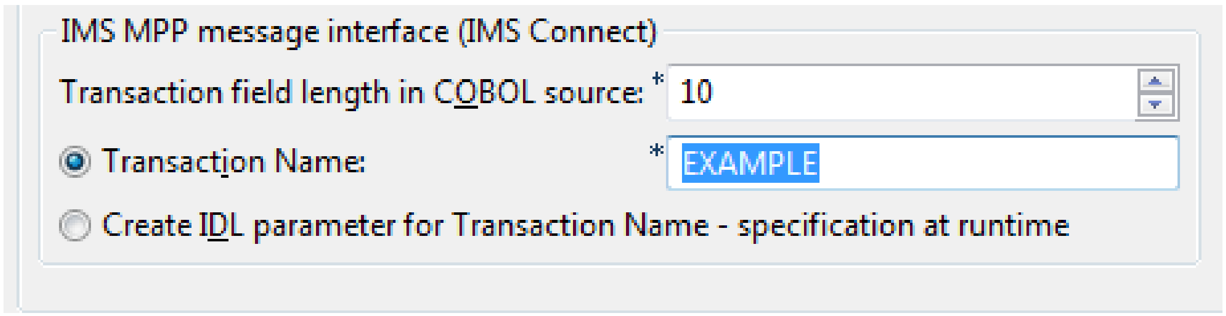
Make sure the interface type is correct and specify how you want the transaction name to be determined.


COBOL Source

File Name:	CALC
Operating System:	z/OS
Interface Type:	IMS MPP message interface (IMS Connect)
	☐ Input Message same as Output Message

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

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



Transaction field length in COBOL source: * 10

[✓ Transaction Name:] * EXAMPLE

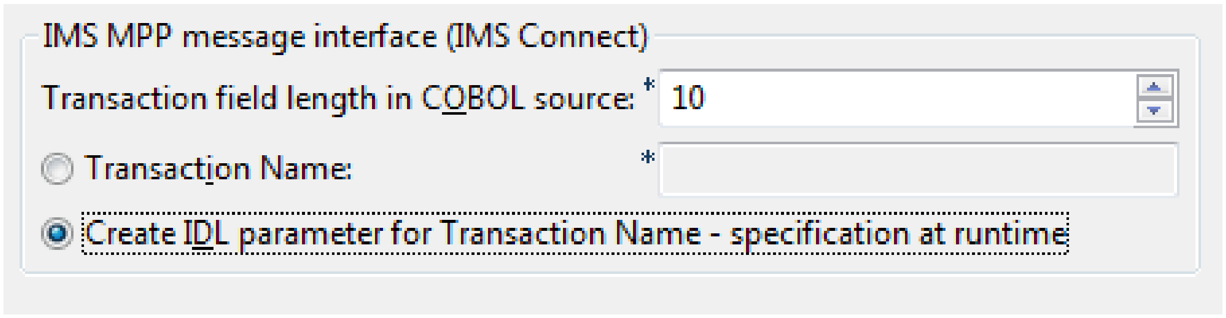
Create IDL parameter for Transaction Name - specification at runtime

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

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

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

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



Transaction field length in COBOL source: * 10

Transaction Name: *

[✓ Create IDL parameter for Transaction Name - specification at runtime]

To select the COBOL interface data items of your COBOL server

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

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

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

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

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

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

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

3. Continue with *COBOL to IDL Mapping*.

Notes:

1. It is very important to select the right COBOL data items describing the interface of the COBOL server correctly. This means the COBOL data items used as parameters must match in number and in sequence of formats (COBOL usage clause).
2. If your COBOL server contain `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.

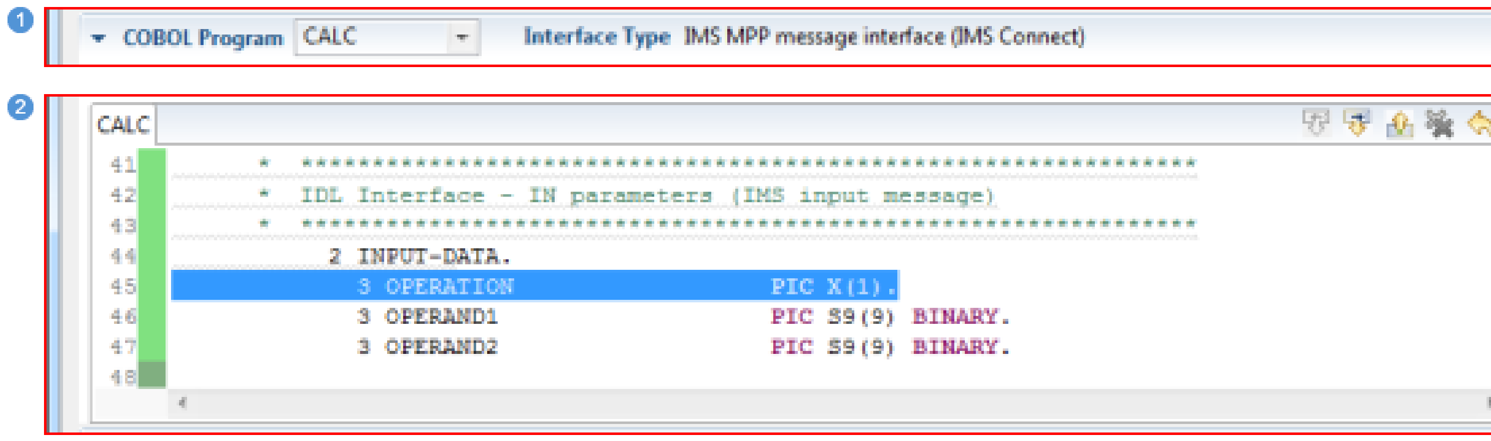
The user interface of the COBOL Mapping Editor is described below.

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:

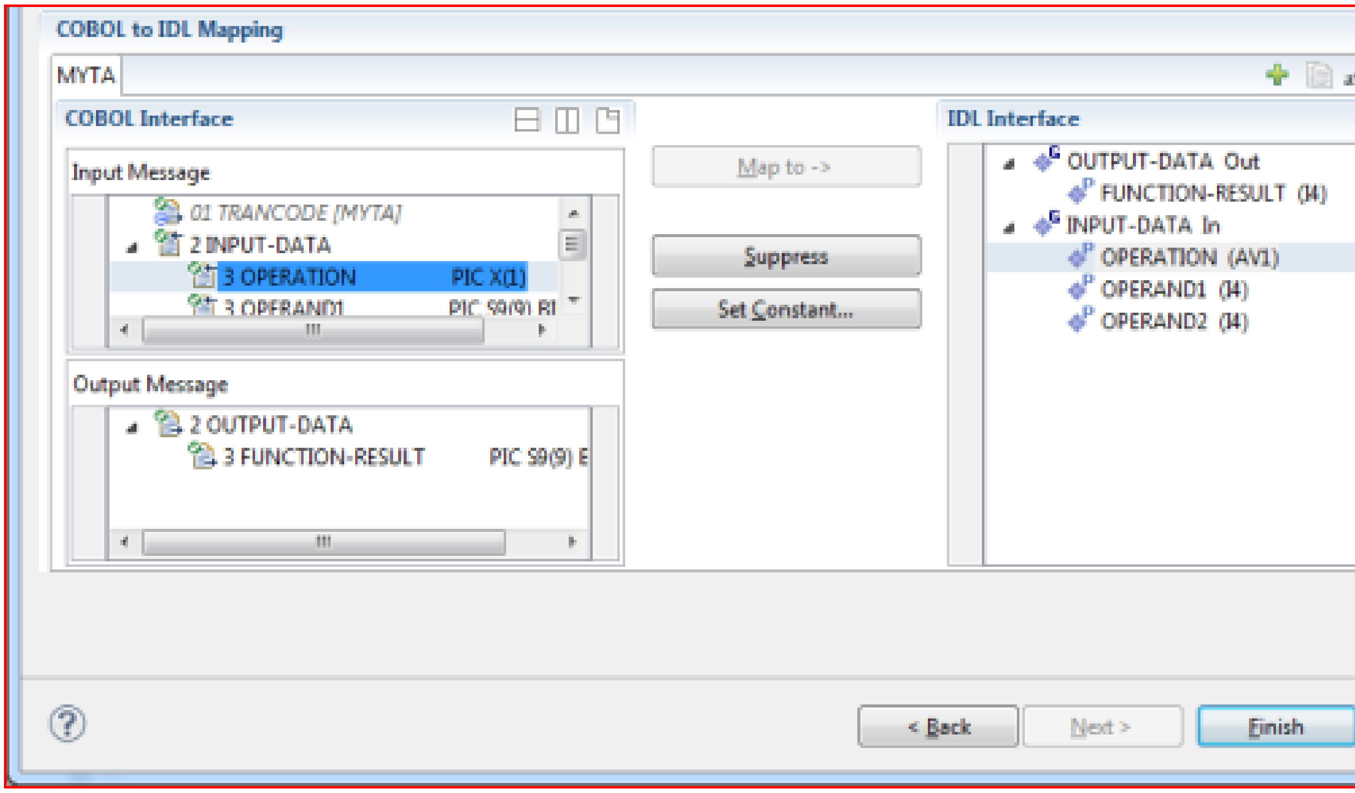
- COBOL Program Selection
- COBOL Source View
- COBOL to IDL Mapping

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



```
      *   ****************************************************************
      *   IDL Interface - IN parameters (IMS input message)
      *   ****************************************************************
           2 INPUT-DATA.
             3 OPERATION                PIC X(1).
             3 OPERAND1                 PIC S9(9) BINARY.
             3 OPERAND2                 PIC S9(9) BINARY.
```

③

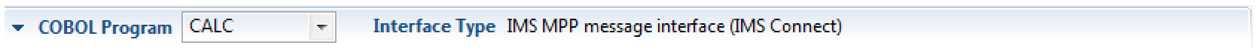


① **COBOL Program Selection**. Currently selected program with interface type

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

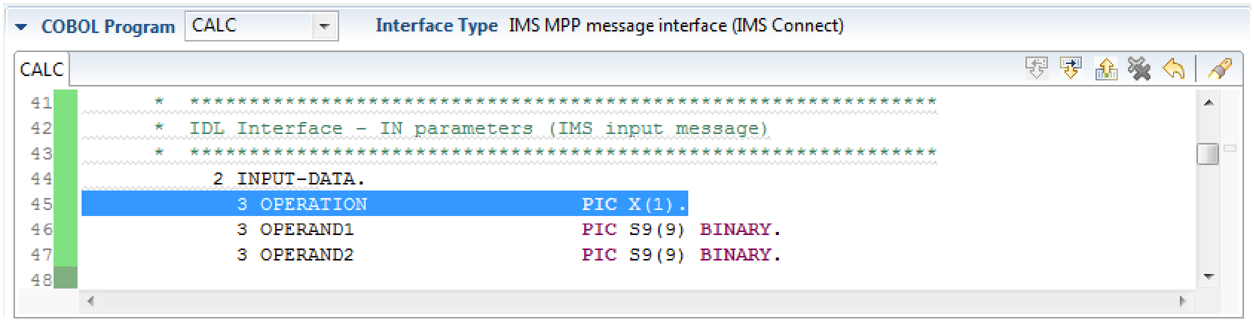
COBOL Program Selection



IMS MPP message interface (IMS Connect)

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

COBOL Source View



```
      *   ****************************************************************
      *   IDL Interface - IN parameters (IMS input message)
      *   ****************************************************************
            2 INPUT-DATA.
              3 OPERATION                   PIC X(1).
              3 OPERAND1                    PIC S9(9) BINARY.
              3 OPERAND2                    PIC S9(9) BINARY.
```

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

Add selected COBOL data item to COBOL Interface as Input Message.

Add selected COBOL data item to COBOL Interface as Output Message.

Remove selected COBOL data item from COBOL Interface.

Remove all COBOL data items from COBOL Interface.

Reset COBOL Interface to initial state.

Show dialog to find text in Source.

The same functionality is also available from the context menu.

COBOL to IDL Mapping

This section covers the following topics:

- COBOL Interface
- Mapping Buttons

- IDL Interface

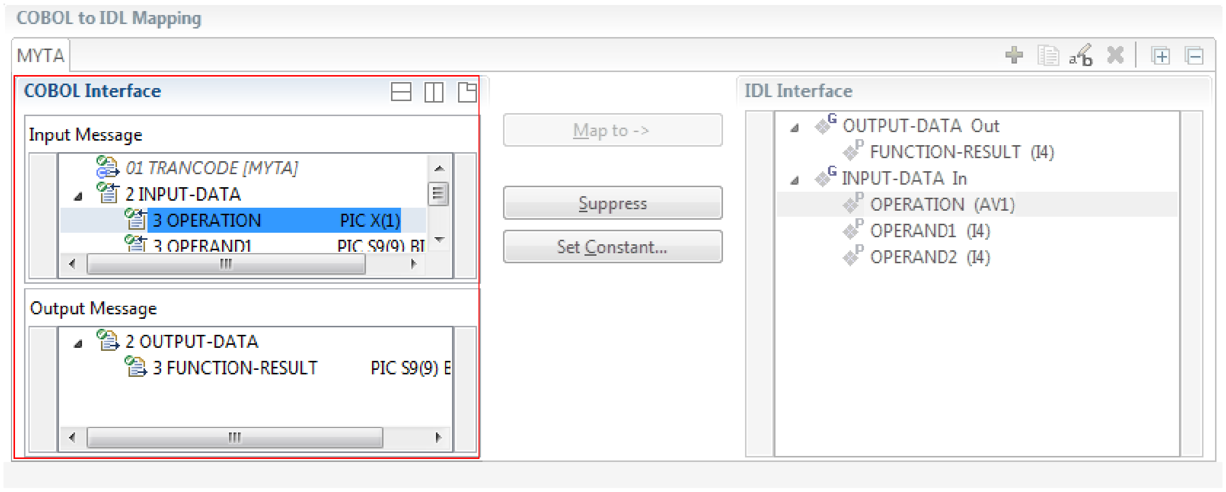
COBOL Interface

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

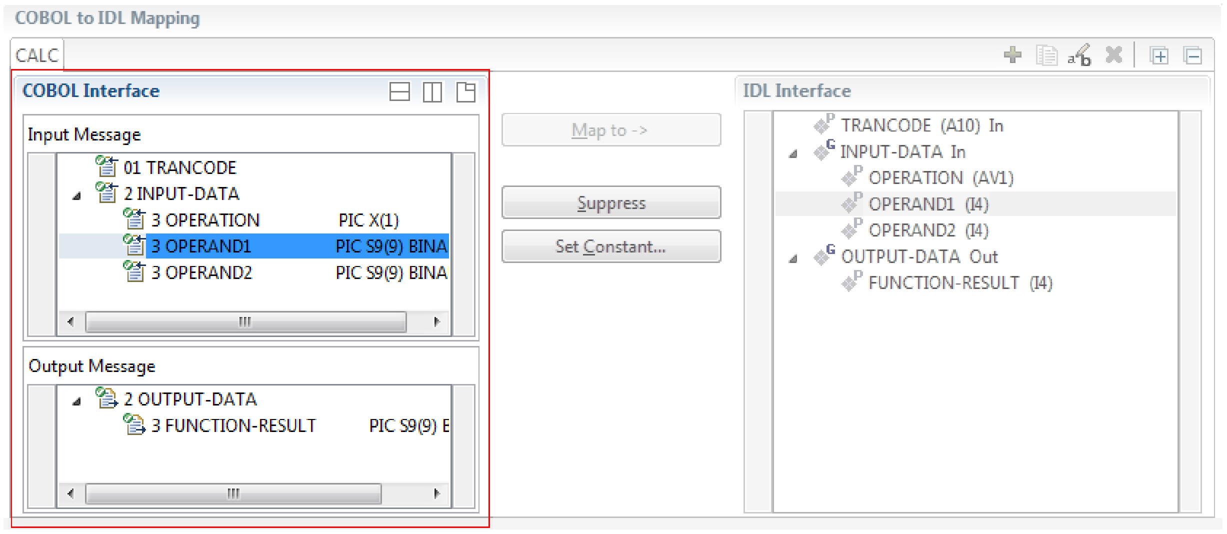
The COBOL data item names are derived from the COBOL source from which they were extracted. If your COBOL interface contains parameters without a name, that is, the keyword `FILLER` is used, those COBOL data items are shown as `[FILLER]`. See ***`FILLER` Pseudo-Parameter***.

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

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



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



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

Context Menu

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

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

Map to	A suppressed COBOL data item becomes visible in the IDL interface. Used also to select another `REDEFINE` path.
Suppress	Suppress unneeded COBOL data items.
Set Constant	Set COBOL data items to constant.
Set Array Mapping	Map an array to a fixed sized or unbounded array. **Note:** This option should be used carefully and requires knowledge of the COBOL server program. Be aware that an incorrect mapping could result in runtime errors.
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*.

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.

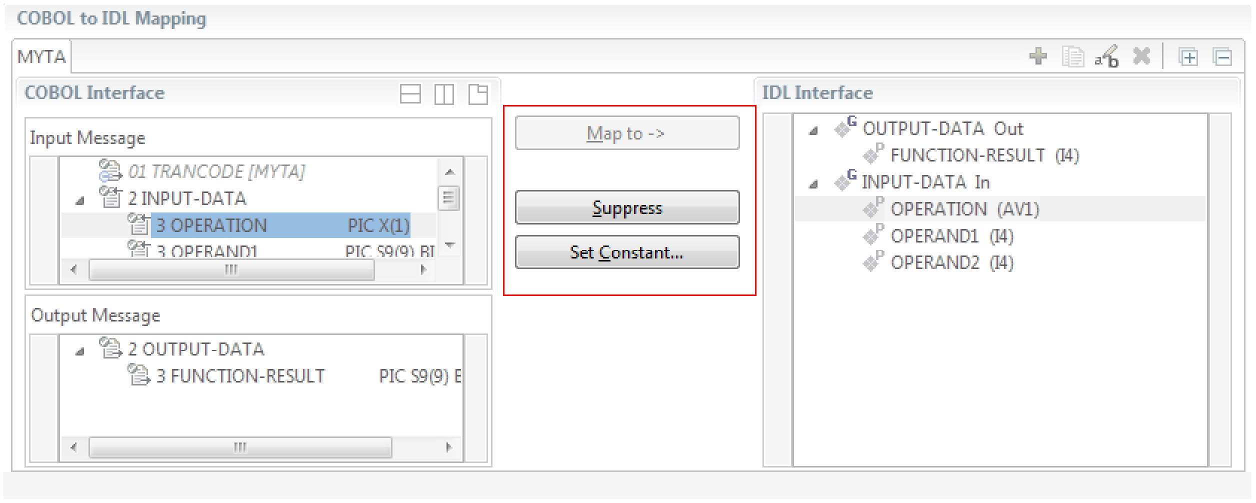
Group parameter, here mapped to In.

`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*.

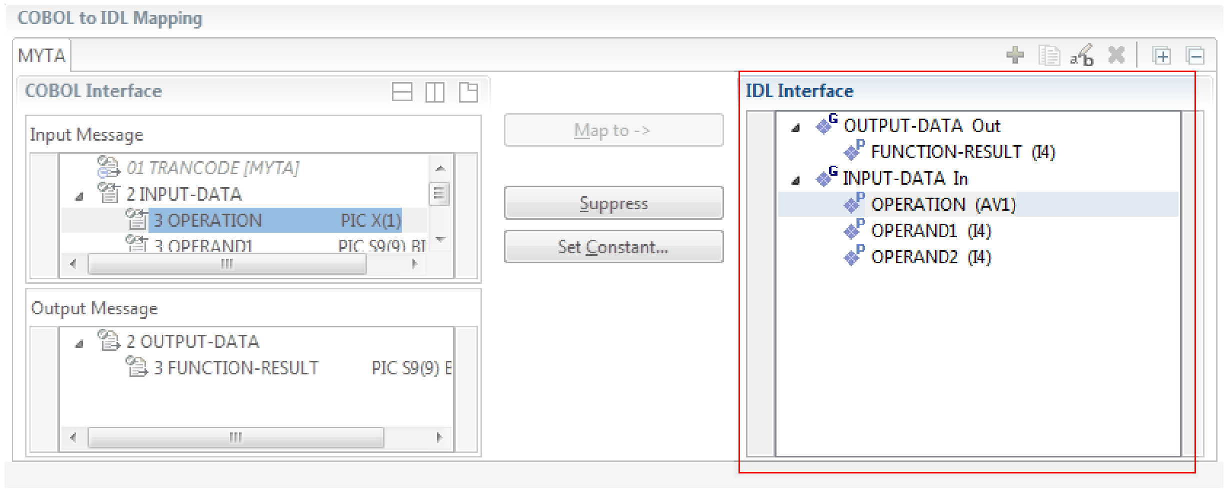
IDL Interface

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:

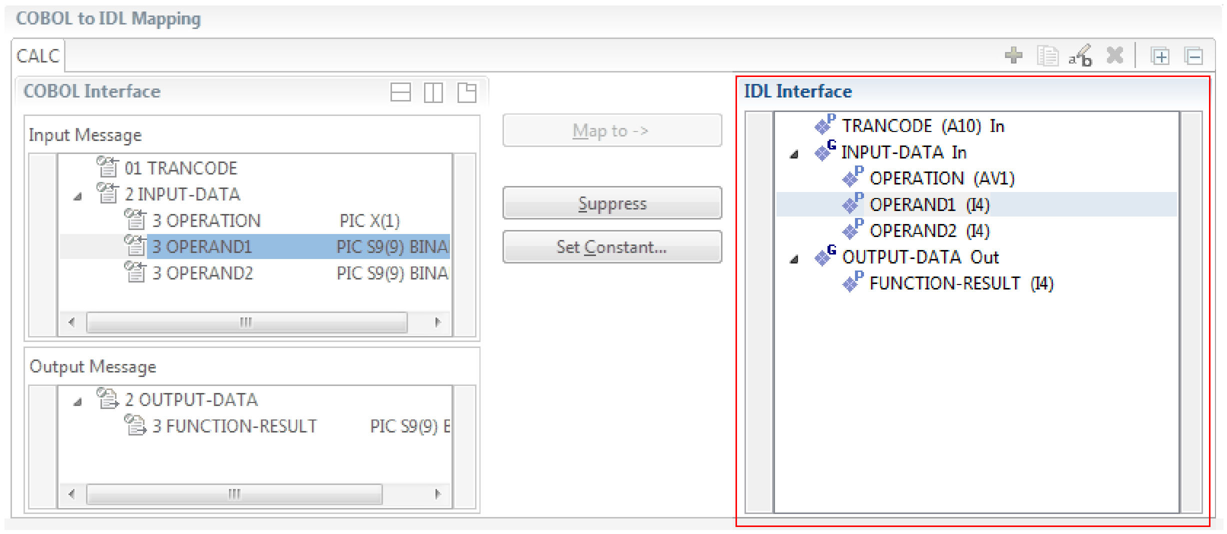
- Rename
- Remove from COBOL Interface. This also removes the mapped IDL parameter from all IDL interfaces for the current COBOL program. See *COBOL Program Selection* above.

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

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



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



Mapping Editor IDL Interface Mapping Functions

This section covers the following topics:

- Map to
- Suppress Unneeded COBOL Data Items
- Set COBOL Data Items to Constants
- Map to Multiple IDL Interfaces
- Select REDEFINE Paths

- Set Arrays (Fixed <-> Unbounded)

Map to

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

To map a COBOL data item to IDL interface

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

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

Notes:

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

Suppress Unneeded COBOL Data Items

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

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

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

To suppress unneeded COBOL data items

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

Notes:

1. The COBOL data item is not part of the IDL interface. It is invisible for RPC clients or Adapter Services.

2. The RPC server or Adapter Service provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.
3. If a COBOL group is suppressed, all subsequent child COBOL data items are suppressed as well.
4. With the inverse function **Map to** (see above) available in the context menu and as mapping button, a COBOL data item can be made visible in the IDL interface again.

Set COBOL Data Items to Constants

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

To map COBOL data items to constants

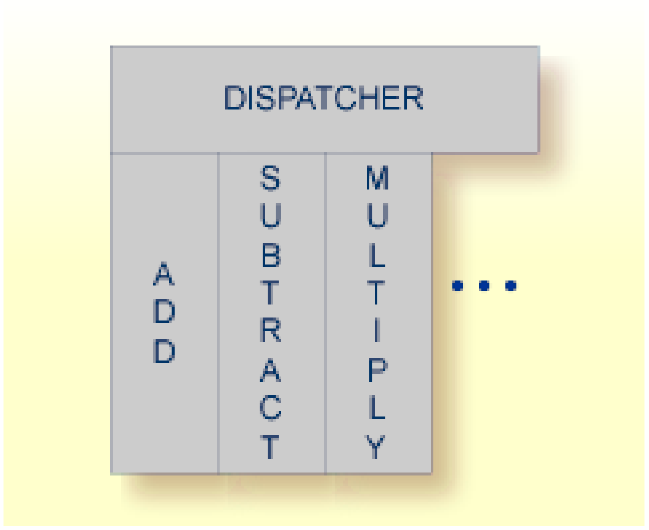
- Use the **Set Constant** function available in the context menu and as mapping button to define a constant value for a COBOL data item. You are prompted with a window to enter the constant value.

Notes:

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

Map to Multiple IDL Interfaces

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



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

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

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

To map a COBOL interface to multiple IDL interfaces

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

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

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

Notes:

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

Icon	Function	Description
✚	Create IDL Interface	Creates a new IDL interface based on the current COBOL interface. All IDL parameters are of IDL direction InOut; no IDL parameters are set to constant; for COBOL `REDEFINE`, the first `REDEFINE` path is mapped to IDL; `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** function available in the context menu and as mapping button to make the COBOL `REDEFINE` path available in the IDL interface.

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

Notes:

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*.

Set Arrays (Fixed <-> Unbounded)

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

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

To set arrays from fixed to unbounded or vice versa

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

Notes:

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

Programming Techniques

This section covers the following topics:

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

Example 1: COBOL Server with Multiple Functions

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

```
. . .

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

     END-EVALUATE.
. . .
```

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

- **Web Service**
  Instead having a Web service with a single operation, you want a web service with multiple operations, one operation for each COBOL function.
- **Java or .NET**
  Instead having a class with a single method, you want a class with multiple methods, one method for each COBOL function.
- etc.

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

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

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

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

         01 INPUT-MESSAGE.
           05 INPUT-IMS-META.
             10 INPUT-LL                       PIC S9(3) BINARY.
             10 INPUT-ZZ                       PIC S9(3) BINARY.
             10 INPUT-TRANSACTION              PIC X(10).
           05 INPUT-DATA.
              10 COBOL-GROUP1.
              20 COBOL-TABLE-PREFIX            PIC X(1000).
            10 COBOL-TABLE-FIX                 OCCURS 20.
              20 COBOL-GROUP2.
                25 COBOL-FIELD1                PIC X(4).
                25 COBOL-FIELD2                PIC X(3).
                25 COBOL-FIELD3                PIC X(50).
         01 OUTPUT-MESSAGE.
           05 OUTPUT-IMS-META.
            10 OUTPUT-LL                       PIC S9(3) BINARY.
            10 OUTPUT-ZZ                       PIC S9(3) BINARY.
           05 OUTPUT-DATA.
            10 COBOL-GROUP1.
              20 COBOL-TABLE-PREFIX            PIC X(1000).
            10 COBOL-TABLE-FIX                 OCCURS 20.
              20 COBOL-GROUP2.
                25 COBOL-FIELD1                PIC X(30).
                25 COBOL-FIELD2                PIC X(20).
                25 COBOL-FIELD3                PIC X(50).
         LINKAGE SECTION.
              . . .
         PROCEDURE DIVISION USING IO-PCB.
            CALL "CBLTDLI" USING GU, IO-PCB, INPUT-MESSAGE.
            . . .
            COMPUTE NUMBER-OF-INCOMING-ELEMENTS = (INPUT-LL
                  - LENGTH OF COBOL-GROUP1 IN AREA INPUT-MESSAGE)
                  / LENGTH OF COBOL-GROUP2 IN AREA INPUT-MESSAGE.
            . . .
            COMPUTE OUTPUT-LL = LENGTH OF COBOL-GROUP2 IN AREA OUTPUT-MESSAGE
                  + (NUMBER-OF-OUTGOING-ELEMENTS
                  * LENGTH OF COBOL-GROUP2 IN AREA OUTPUT-MESSAGE).

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

         GOBACK.
```

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

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

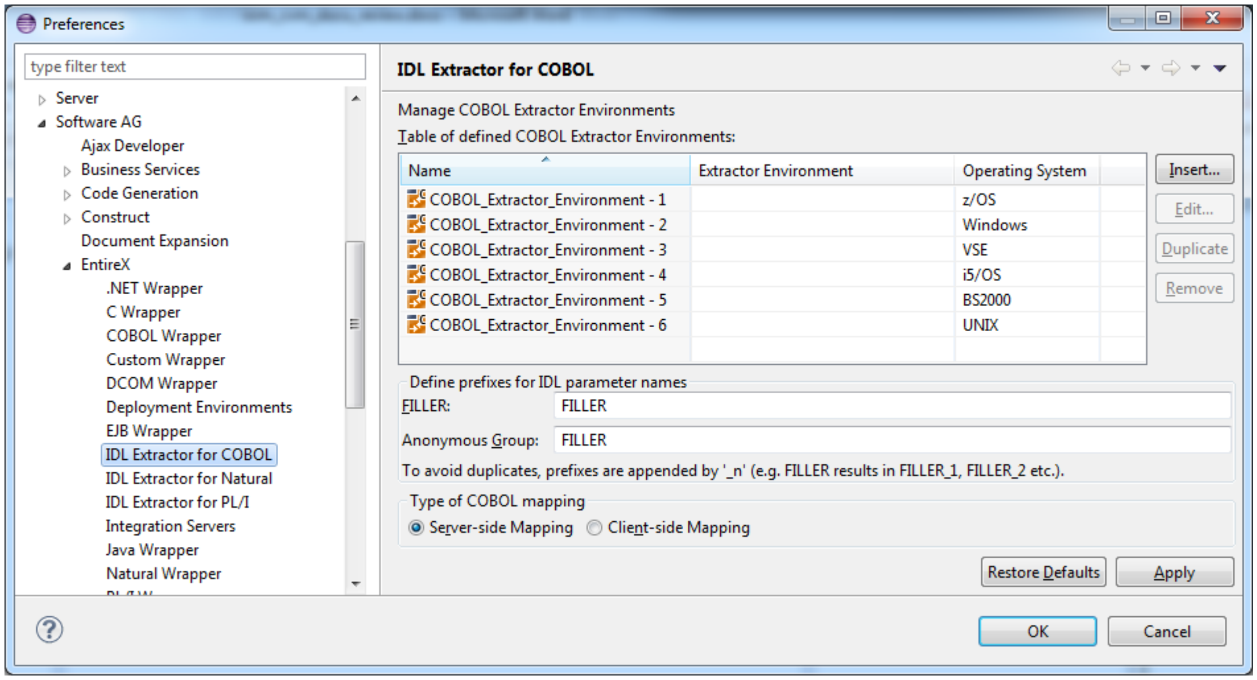
14 IDL Extractor for COBOL Preferences

- Create New Local Extractor Environment (z/OS, z/VSE, BS2000/OSD and IBM i) 222
- Create New Local Extractor Environment for Micro Focus (UNIX and Windows) 226
- Create New Remote Extractor Environment (z/OS) 230
- Create New Remote Extractor Environment (BS2000/OSD) 234

The IDL Extractor for COBOL preferences are used to 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 EntireX Workbench 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/OSD. See *Extractor Service* in the z/OS administration and BS2000/OSD Batch RPC Server documentation.

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 and use the buttons on the right.


Preferences

type filter text

- Server
- Software AG
  - Ajax Developer
  - Business Services
  - Code Generation
  - Construct
  - Document Expansion
  - EntireX
    - .NET Wrapper
    - C Wrapper
    - COBOL Wrapper
    - Custom Wrapper
    - DCOM Wrapper
    - Deployment Environments
    - EJB Wrapper
    - IDL Extractor for COBOL
    - IDL Extractor for Natural
    - IDL Extractor for PL/I
    - Integration Servers
    - Java Wrapper
    - Natural Wrapper

IDL Extractor for COBOL

Manage COBOL Extractor Environments

Table of defined COBOL Extractor Environments:

Name	Extractor Environment	Operating System
COBOL_Extractor_Environment - 1		z/OS
COBOL_Extractor_Environment - 2		Windows
COBOL_Extractor_Environment - 3		VSE
COBOL_Extractor_Environment - 4		i5/OS
COBOL_Extractor_Environment - 5		BS2000
COBOL_Extractor_Environment - 6		UNIX

Define prefixes for IDL parameter names

FILLER: FILLER

Anonymous Group: FILLER

To avoid duplicates, prefixes are appended by '_n' (e.g. FILLER results in FILLER_1, FILLER_2 etc.).

Type of COBOL mapping

[✓ Server-side Mapping] Client-side Mapping

The **Preferences** page contains further settings valid for all COBOL extractor environments:

- **Define prefixes for IDL parameter names**
  The defined prefixes are used for `FILLER` ***Pseudo-Parameter***.
- **Type of COBOL mapping**
  Every EntireX Workbench (Eclipse) workspace is either in client-side mapping mode (generating EntireX Workbench server mapping files with extension .cvm) or server-side mapping mode (generating EntireX Workbench server mapping files with extension .svm). See *Server Mapping Files for COBOL* for an introduction. You can adjust the mode here, which will also set the mode

of the COBOL Wrapper to the same value. See *Generation Settings - Preferences* in the COBOL Wrapper documentation.

Server mapping files are generated automatically for RPC servers if required. See *When is a Server Mapping File Required? - IDL Extractor for COBOL* in the *EntireX Workbench* documentation.

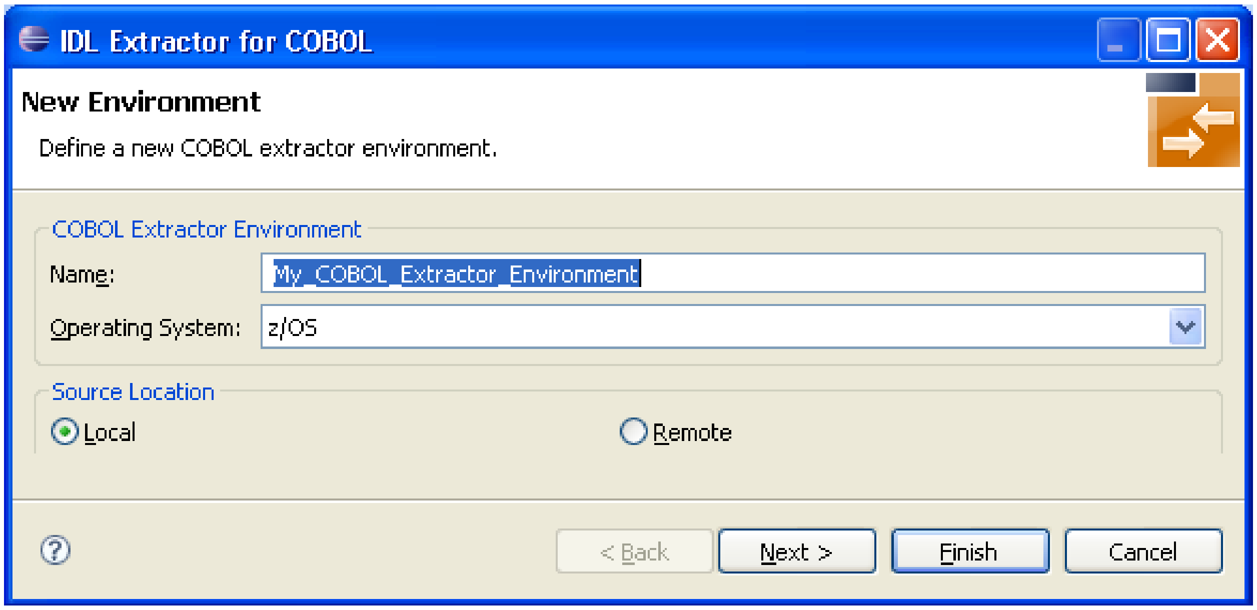
Create New Local Extractor Environment (z/OS, z/VSE, BS2000/OSD and IBM i)

This section describes the four steps for creating a new local COBOL extractor environment to extract z/OS, z/VSE, BS2000/OSD or IBM i COBOL programs.

- Step 1: Define the New Local Environment
- Step 2: Define the Default Settings
- Step 3: Define the Local Extractor Environment
- Step 4: Define the Local Copybook Locations

Step 1: Define the New Local Environment

On the New Environment page you can specify **Name** and **Operating system**.


IDL Extractor for COBOL

New Environment

Define a new COBOL extractor environment.

COBOL Extractor Environment

Name: My_COBOL_Extractor_Environment

Operating System: z/OS

Source Location

[✓ Local] Remote

To define the new environment settings

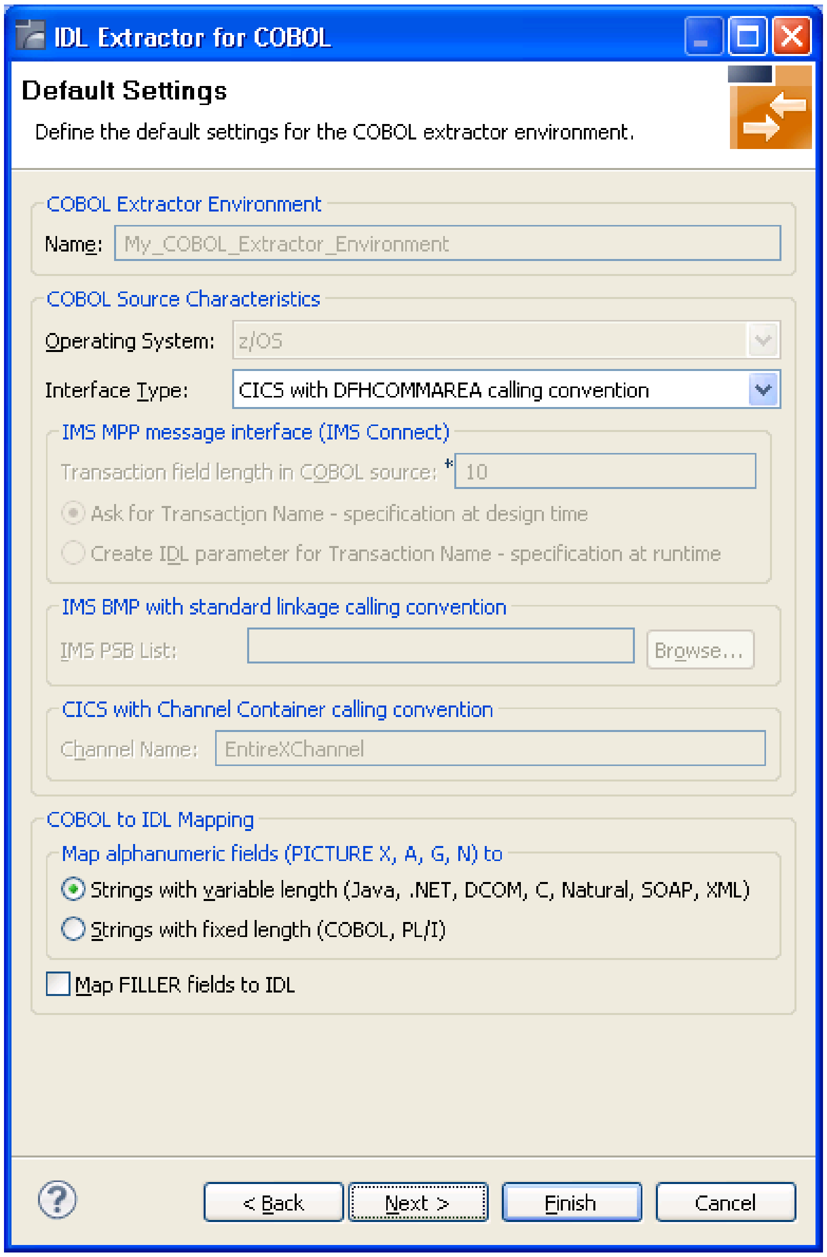
1 Enter a unique **Name** for the COBOL extractor environment.

2 Select the **Operating system**.

3 Select "Local" for **Source Location**.

Step 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.



IDL Extractor for COBOL

Default Settings

Define the default settings for the COBOL extractor environment.

COBOL Extractor Environment

Name: My_COBOL_Extractor_Environment

COBOL Source Characteristics

Operating System: z/OS

Interface Type: CICS with DFHCOMMAREA calling convention

IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: * 10

[✓ Ask for Transaction Name - specification at design time]

Create IDL parameter for Transaction Name - specification at runtime

IMS BMP with standard linkage calling convention

IMS PSB List:

CICS with Channel Container calling convention

Channel Name: EntireXChannel

COBOL to IDL Mapping

Map alphanumeric fields (PICTURE X, A, G, N) to

[✓ Strings with variable length (Java, .NET, DCOM, C, Natural, SOAP, XML)]

Strings with fixed length (COBOL, PL/I)

☐ Map FILLER fields to IDL

To define the default extraction settings

1 Select the default **Interface Type**. See *Supported COBOL Interface Types*.

2 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**.

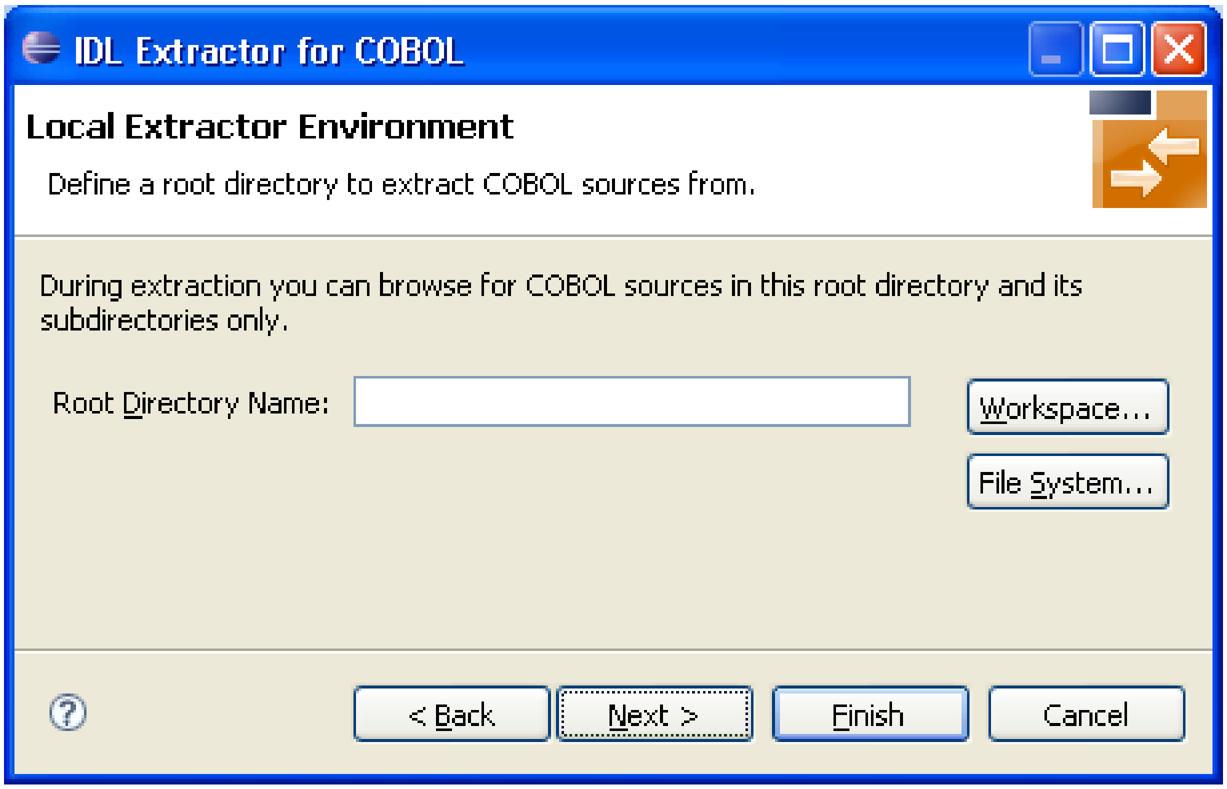
For more information refer to *Step 4: Define the Extraction Settings and Start Extraction*.

3 Specify a default value for **COBOL to IDL Mapping**. See *COBOL to IDL Mapping*.

Press **Next** and continue with *Step 3: Define the Local Extractor Environment* below.

Step 3: Define the Local Extractor Environment

On the **Local Extractor Environment** page you can provide a default directory name for the COBOL programs:


IDL Extractor for COBOL

Local Extractor Environment

Define a root directory to extract COBOL sources from.

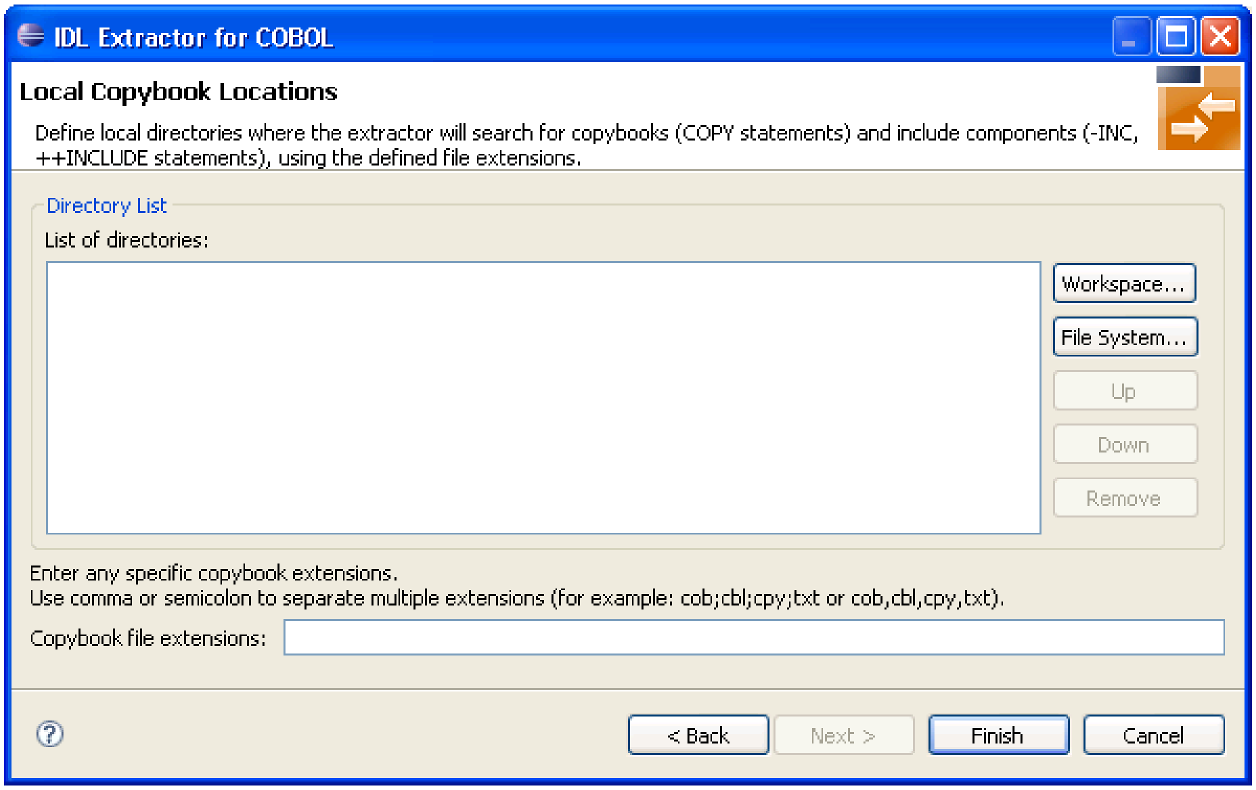
During extraction you can browse for COBOL sources in this root directory and its subdirectories only.

Root Directory Name:

1. Choose **Workspace...** or **File System...** to browse for a folder.
2. Choose **Next** and continue with *Step 4: Define the Local Copybook Locations* below.

Step 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:



Local Copybook Locations

Define local directories where the extractor will search for copybooks (COPY statements) and include components (-INC, ++INCLUDE statements), using the defined file extensions.

Enter any specific copybook extensions.
Use comma or semicolon to separate multiple extensions (for example: cob;cbl;cpy;txt or cob,cbl,cpy,txt).

Copybook file extensions:

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**.

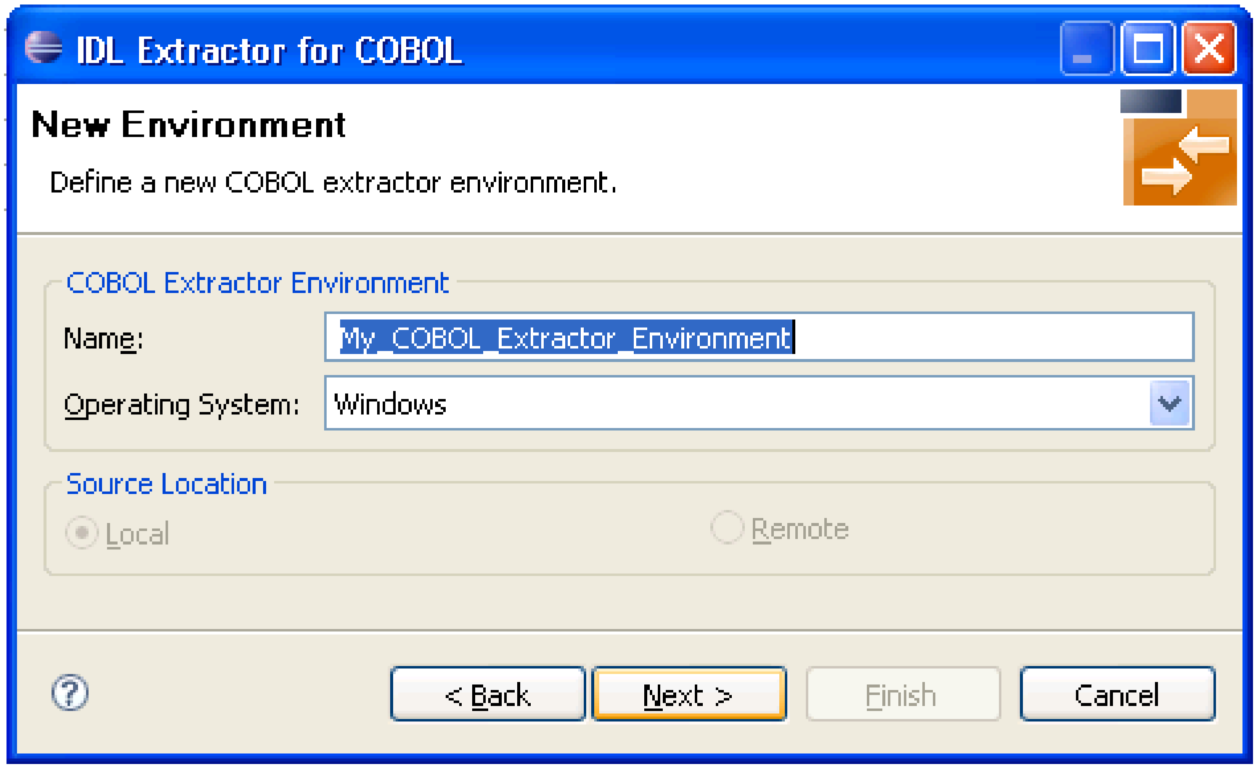
Create New Local Extractor Environment for Micro Focus (UNIX and Windows)

This section describes the four steps for creating a new local COBOL extractor environment to extract Micro Focus COBOL programs.

- Step 1: Define the New Local Environment
- Step 2: Define the Default Settings
- Step 3: Define the Local Extractor Environment

- Step 4: Define the Local Copybook Locations

Step 1: Define the New Local Environment



Name: My_COBOL_Extractor_Environment
Operating System: Windows
Source Location: [✓ Local] Remote

On the **New Environment** page you can specify the **Name** and **Operating system**. Only UNIX and Windows operating systems can be used for Micro Focus COBOL.

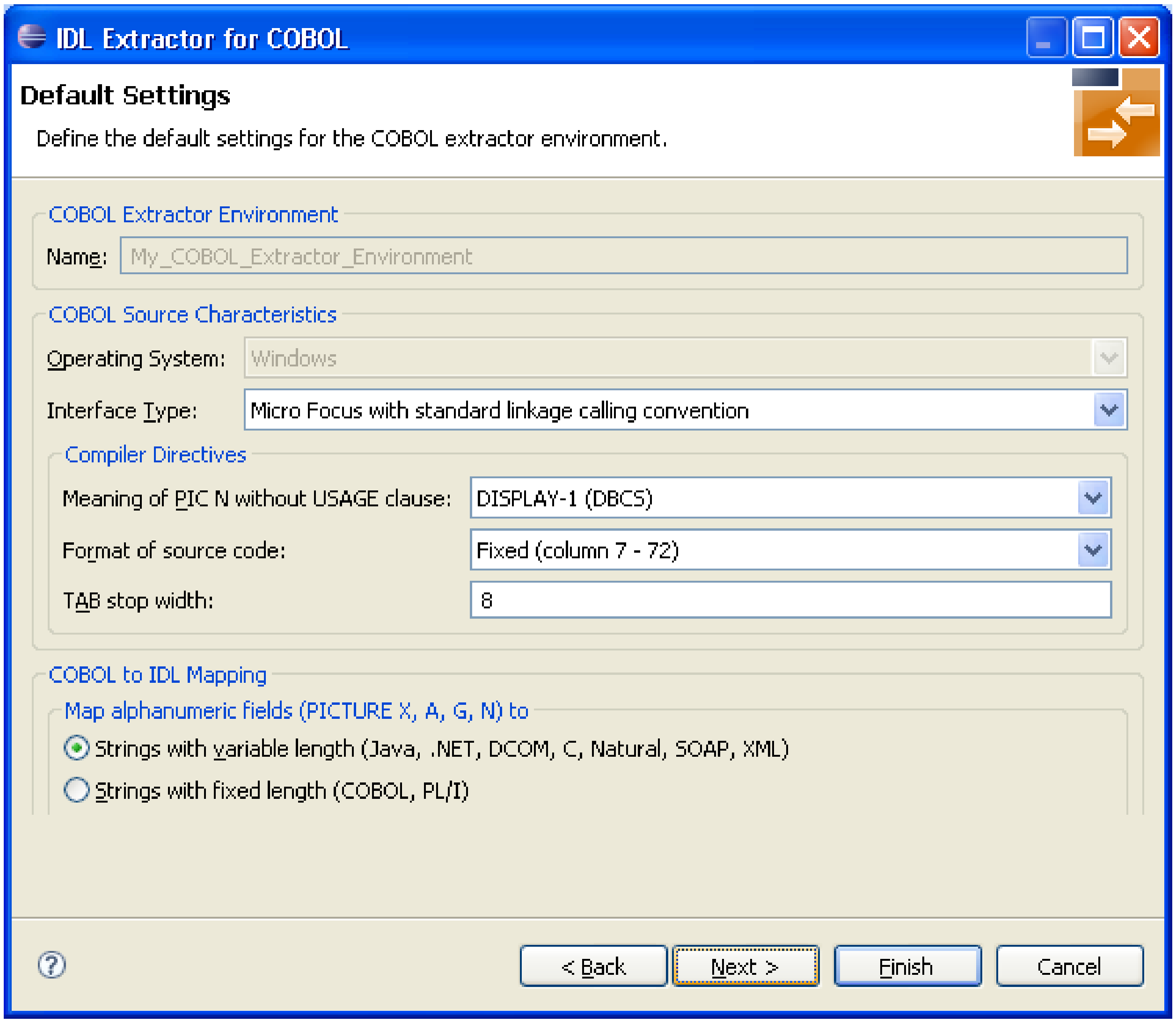
To define the default extraction settings

1. Enter a unique name for the COBOL extractor environment.
2. Select the **Operating system** "UNIX" or "Windows".
3. Select "Local" for **Source location**.

Step 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**, **Compiler directives** and **COBOL to IDL Mapping**.


IDL Extractor for COBOL
Default Settings
Define the default settings for the COBOL extractor environment.
COBOL Extractor Environment
COBOL Source Characteristics
Compiler Directives
COBOL to IDL Mapping
Map alphanumeric fields (PICTURE X, A, G, N) to
< Back
Next >
Finish
Cancel

Name: My_COBOL_Extractor_Environment

Operating System: Windows

Interface Type: Micro Focus with standard linkage calling convention

Meaning of PIC N without USAGE clause: DISPLAY-1 (DBCS)

Format of source code: Fixed (column 7 - 72)

TAB stop width: 8

[✓ Strings with variable length (Java, .NET, DCOM, C, Natural, SOAP, XML)]

Strings with fixed length (COBOL, PL/I)

To define the default extraction settings

1. Refer to *Step 2: Define the Default Settings* for a local extractor environment for field descriptions. Select the default **Interface type**. See *Supported COBOL Interface Types*.
2. Select a value for **Meaning of PIC N without USAGE clause**. Select "NATIONAL" for IDL mapping to data type U, or "DISPLAY-1" (DBCS) for data type K. "DISPLAY-1" (DBCS) is the default, which is the same as Micro Focus compilers. See also *COBOL to IDL Mapping*.

3 Select the source code format. Use "Fixed" (default) or "Variable" to change the interpreted source code columns. Refer to your Micro Focus documentation for further information.

4 Enter the **TAB stop width**. Typical values are 4 or 8 (default).

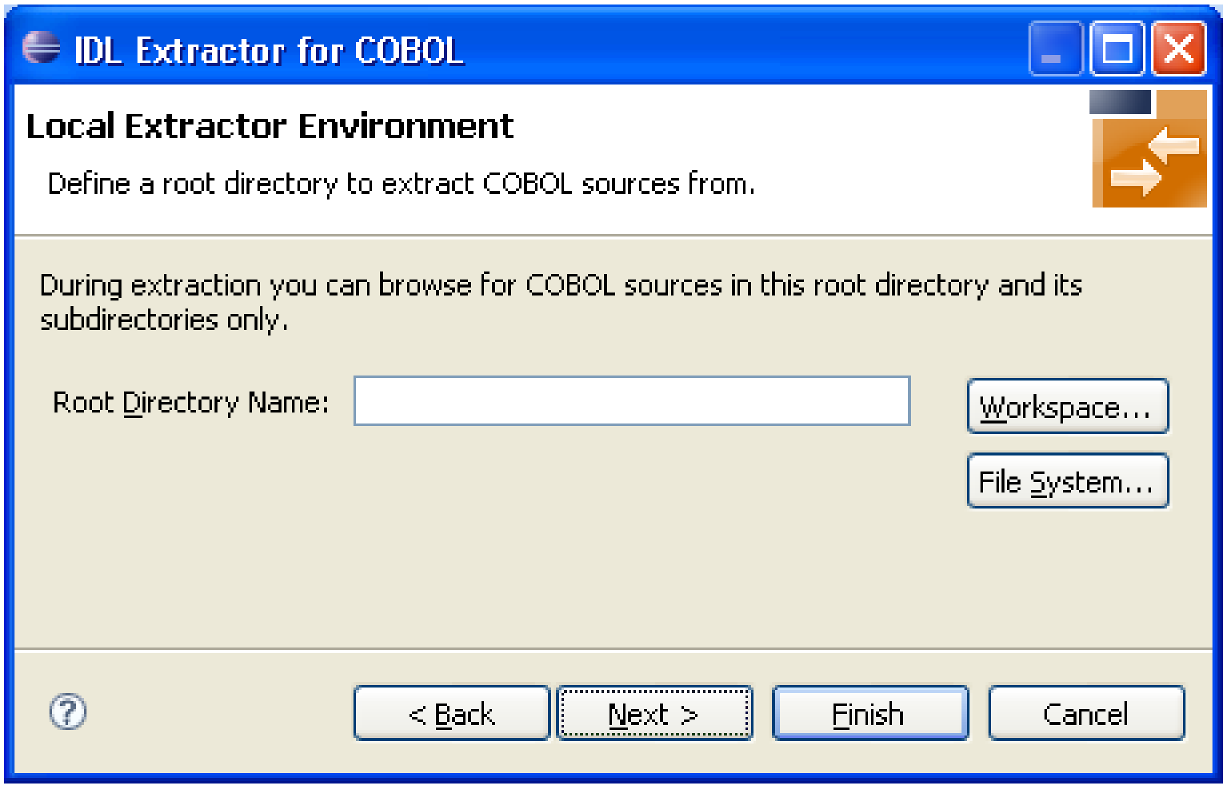
5 Specify the default **COBOL to IDL Mapping**. See *COBOL to IDL Mapping*.

6 Choose **Next** and continue with the *Step 3: Define the Local Extractor Environment* below.

Refer to *Step 2: Define the Default Settings* for a local extractor environment for field descriptions.

Step 3: Define the Local Extractor Environment

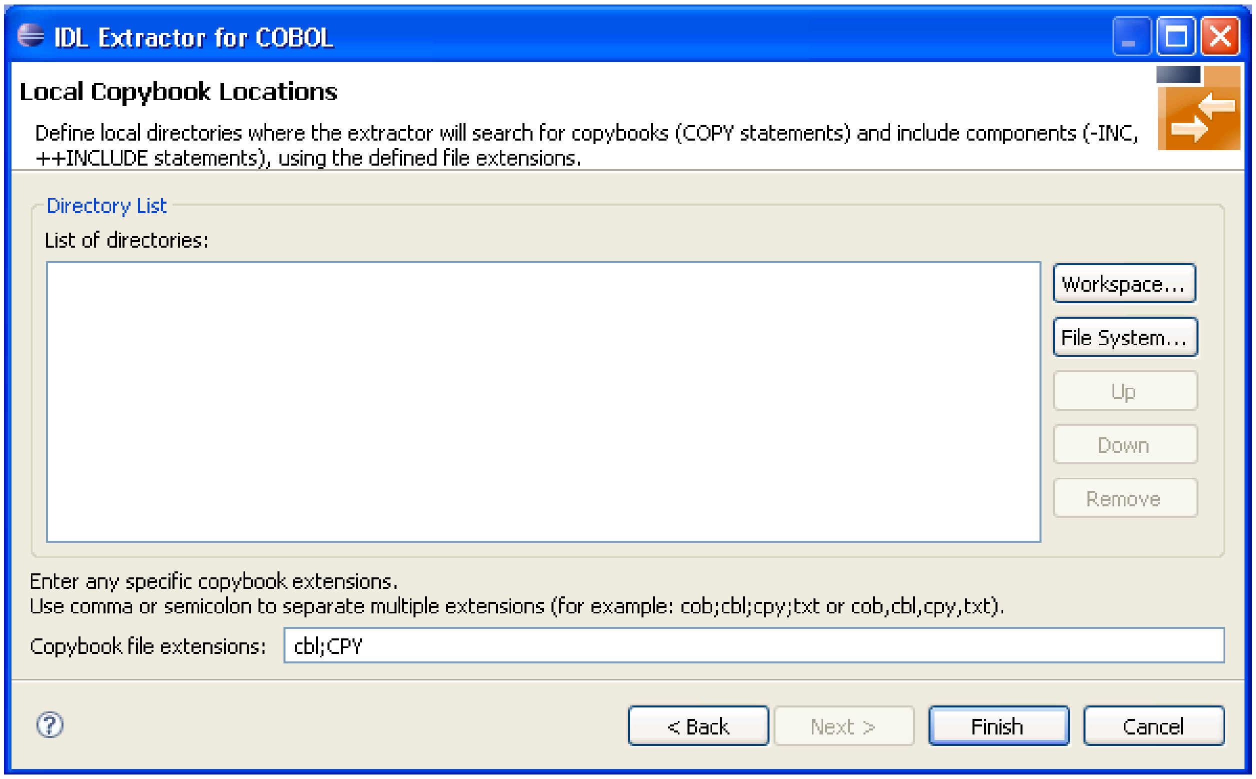
On the **Local Extractor Environment** page you can provide a default directory name for the COBOL programs:


IDL Extractor for COBOL
Local Extractor Environment
Define a root directory to extract COBOL sources from.
During extraction you can browse for COBOL sources in this root directory and its subdirectories only.
Root Directory Name:
Workspace...
File System...
< Back
Next >
Finish
Cancel

1. Choose **Workspace...** or **File System...** to browse for a folder.
2. Choose **Next** and continue with *Step 4: Define the Local Copybook Locations* below.

Step 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:



Define local directories where the extractor will search for copybooks (COPY statements) and include components (-INC, ++INCLUDE statements), using the defined file extensions.

List of directories:

Enter any specific copybook extensions.
Use comma or semicolon to separate multiple extensions (for example: cob;cbl;cpy;txt or cob,cbl,cpy,txt).

Copybook file extensions: cbl;CPY

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.

Choose **Workspace...** or **File System...** to browse for a folder.

Choose **Finish**.

Create New Remote Extractor Environment (z/OS)

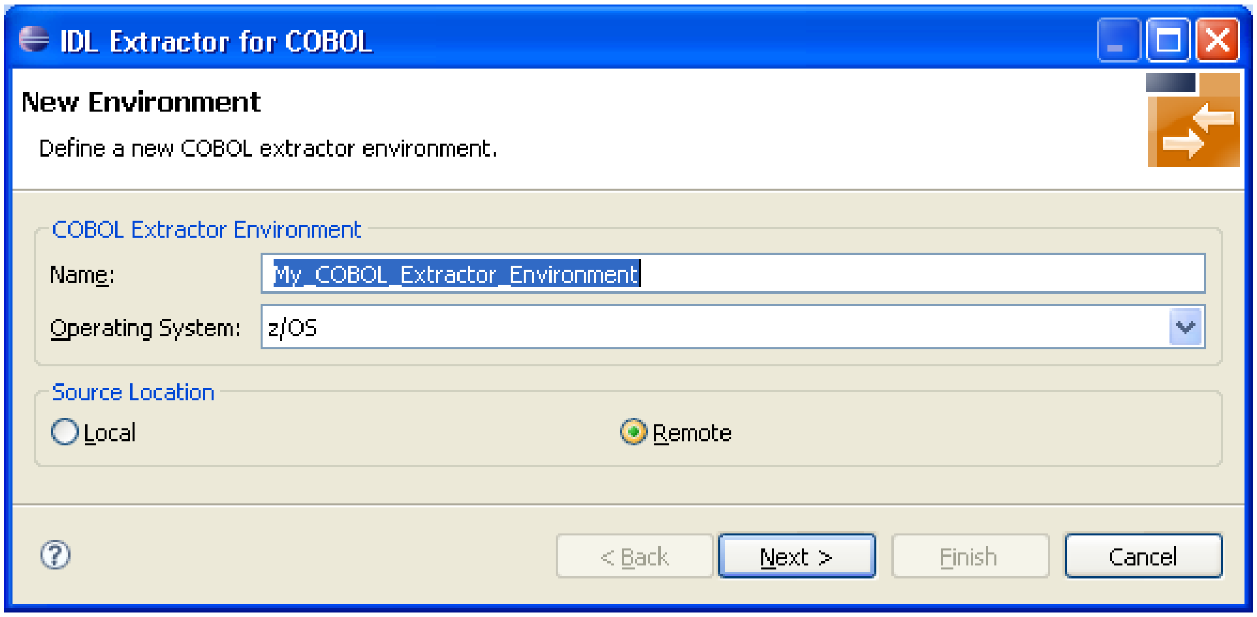
This section describes the four steps for creating a new remote COBOL extractor environment to extract remotely z/OS COBOL programs stored in partitioned data sets or CA Librarian data sets.

- Step 1: Define the New Remote Environment
- Step 2: Define the Default Settings
- Step 3: Define the Remote Extractor Environment

- Step 4: Define the Remote Copybook Locations

Step 1: Define the New Remote Environment

On the **New Environment** page you can specify **Name**, **Operating system** and the **Remote Source Location**.


IDL Extractor for COBOL

New Environment

Define a new COBOL extractor environment.

COBOL Extractor Environment

Name: My_COBOL_Extractor_Environment

Operating System: z/OS

Source Location

Local [✓ Remote]

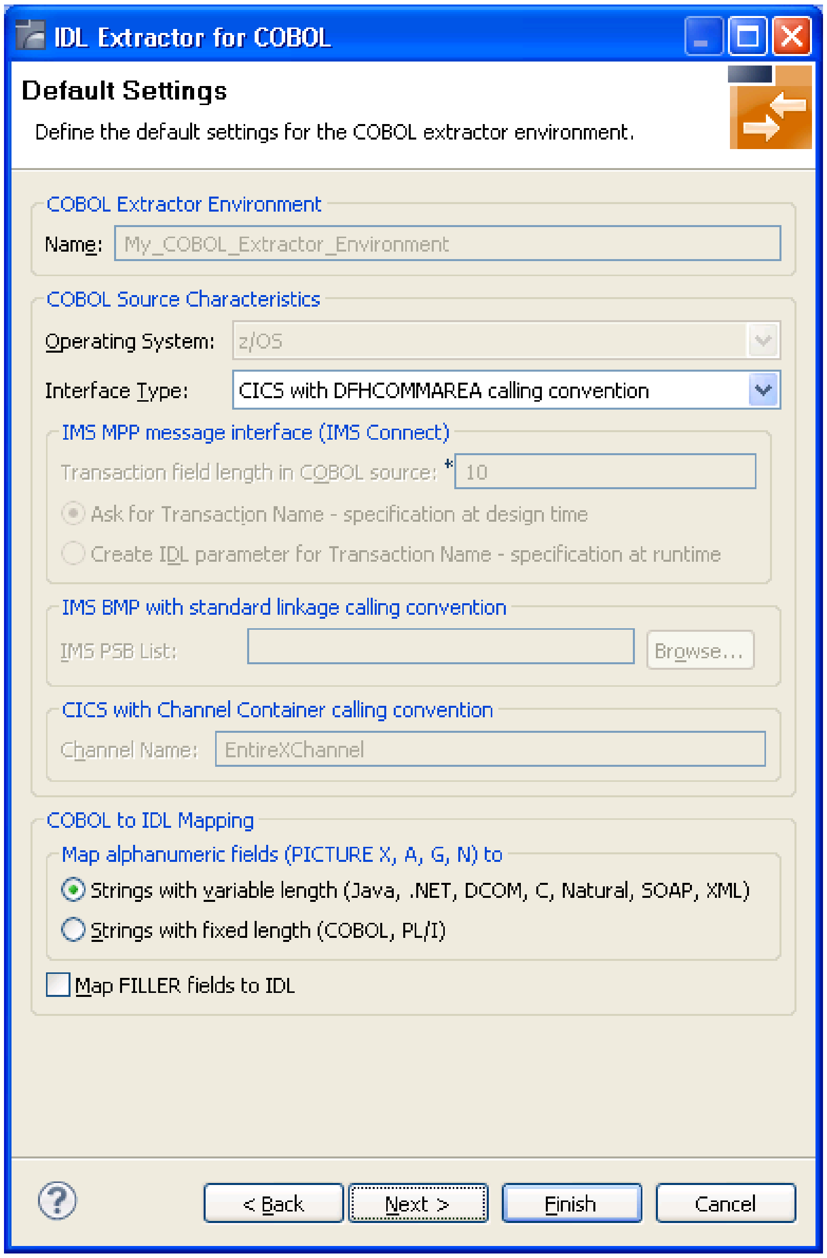
To define the new environment settings

1. Enter a unique name for the COBOL extractor environment.
2. Select the **Operating system**.
3. Select "Remote" for **Source location**.

Step 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**.



Default Settings

Define the default settings for the COBOL extractor environment.

COBOL Extractor Environment

Name: My_COBOL_Extractor_Environment

COBOL Source Characteristics

Operating System: z/OS

Interface Type: CICS with DFHCOMMAREA calling convention

IMS MPP message interface (IMS Connect)

Transaction field length in COBOL source: * 10

Ask for Transaction Name - specification at design time

Create IDL parameter for Transaction Name - specification at runtime

IMS BMP with standard linkage calling convention

IMS PSB List:

CICS with Channel Container calling convention

Channel Name: EntireXChannel

COBOL to IDL Mapping

Map alphanumeric fields (PICTURE X, A, G, N) to

[✓ Strings with variable length (Java, .NET, DCOM, C, Natural, SOAP, XML)]

Strings with fixed length (COBOL, PL/I)

☐ Map FILLER fields to IDL

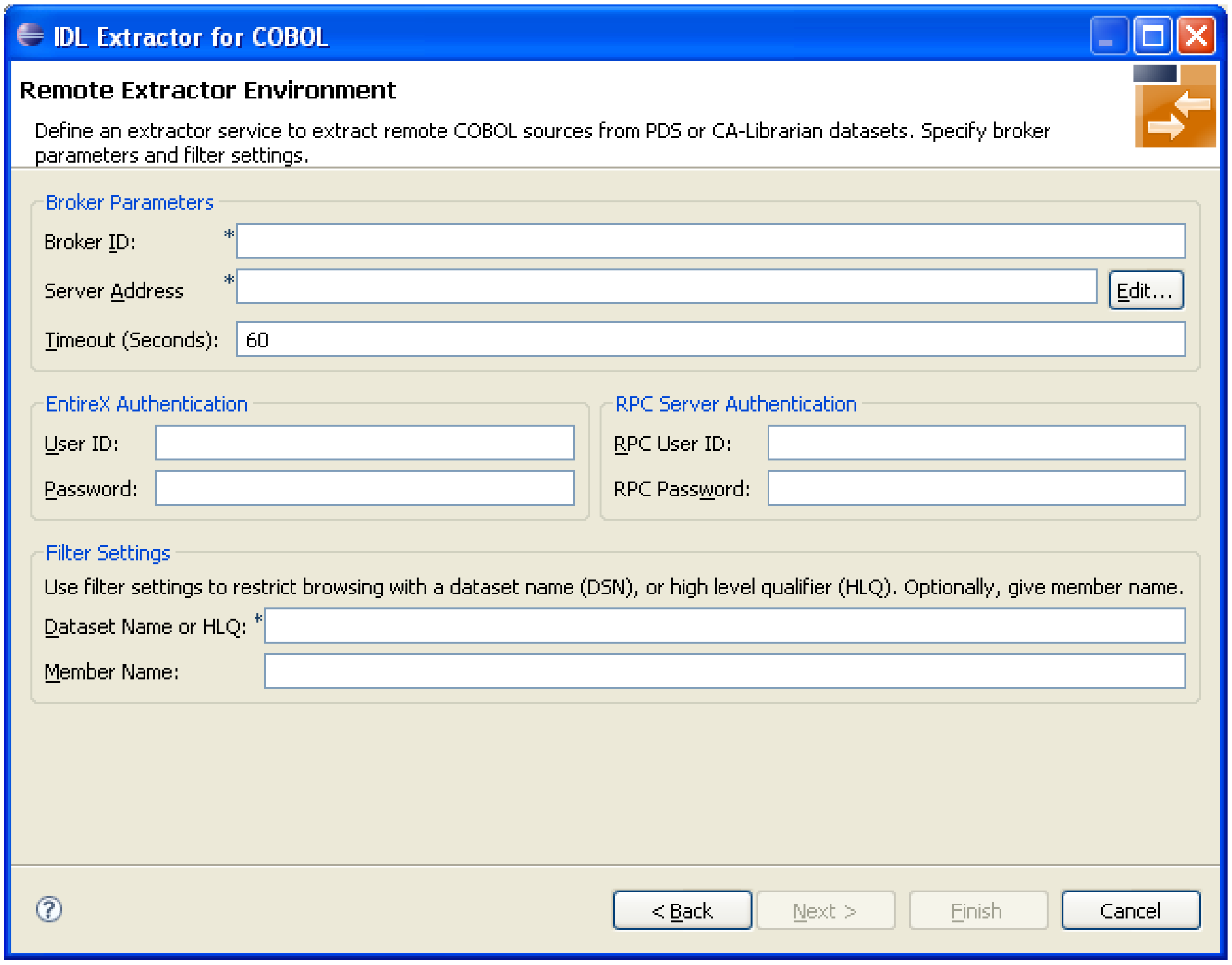
To define the default extraction settings

- See *Step 2: Define the Default Settings* in section *Create New Local Extractor Environment (z/OS, z/VSE, BS2000/OSD and IBM i)*.

Press **Next** and continue with *Step 3: Define the Remote Extractor Environment* below.

Step 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*.


IDL Extractor for COBOL

Remote Extractor Environment

Define an extractor service to extract remote COBOL sources from PDS or CA-Librarian datasets. Specify broker parameters and filter settings.

Broker Parameters

Broker ID: *

Server Address *

Timeout (Seconds): 60

EntireX Authentication

User ID:

Password:

RPC Server Authentication

RPC User ID:

RPC Password:

Filter Settings

Use filter settings to restrict browsing with a dataset name (DSN), or high level qualifier (HLQ). Optionally, give member name.

Dataset Name or HLQ: *

Member Name:

To define the remote extractor environment

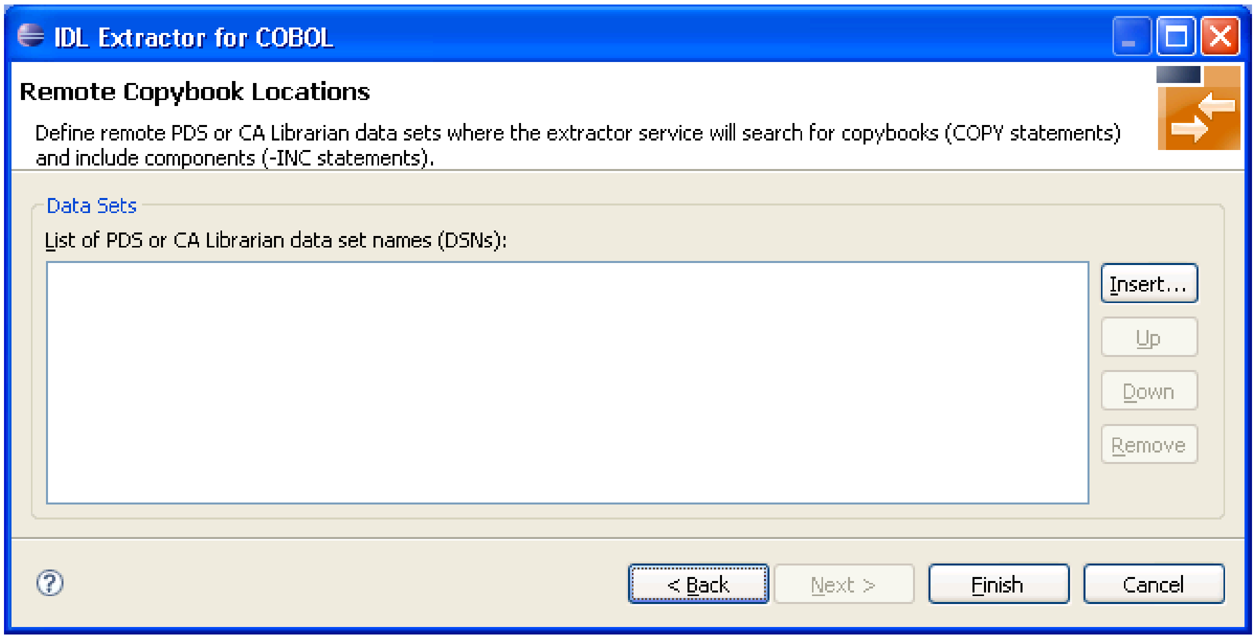
1. Under **Broker Parameters**, enter the required fields Broker ID and Server Address, which will have the default format brokerID@serverAddress. The last part (broker service) of the server address must always be "EXTRACTOR". The timeout value must be in the range 1-9999 seconds (default is 60).
2. The **EntireX Authentication** parameters describe the settings for the broker. See *Authentication of User* under *Overview of EntireX Security* in the EntireX Security documentation.
3. The **RPC Server Authentication** parameters describe the settings for the RPC server. See *Administering the Batch RPC Server* | *Administering the EntireX RPC Server under z/OS IMS*.
4. 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.

5 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.

Press **Next** and continue with *Step 4: Define the Remote Copybook Locations* below.

Step 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:



Define remote PDS or CA Librarian data sets where the extractor service will search for copybooks (COPY statements) and include components (-INC statements).

List of PDS or CA Librarian data set names (DSNs):

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**.

Create New Remote Extractor Environment (BS2000/OSD)

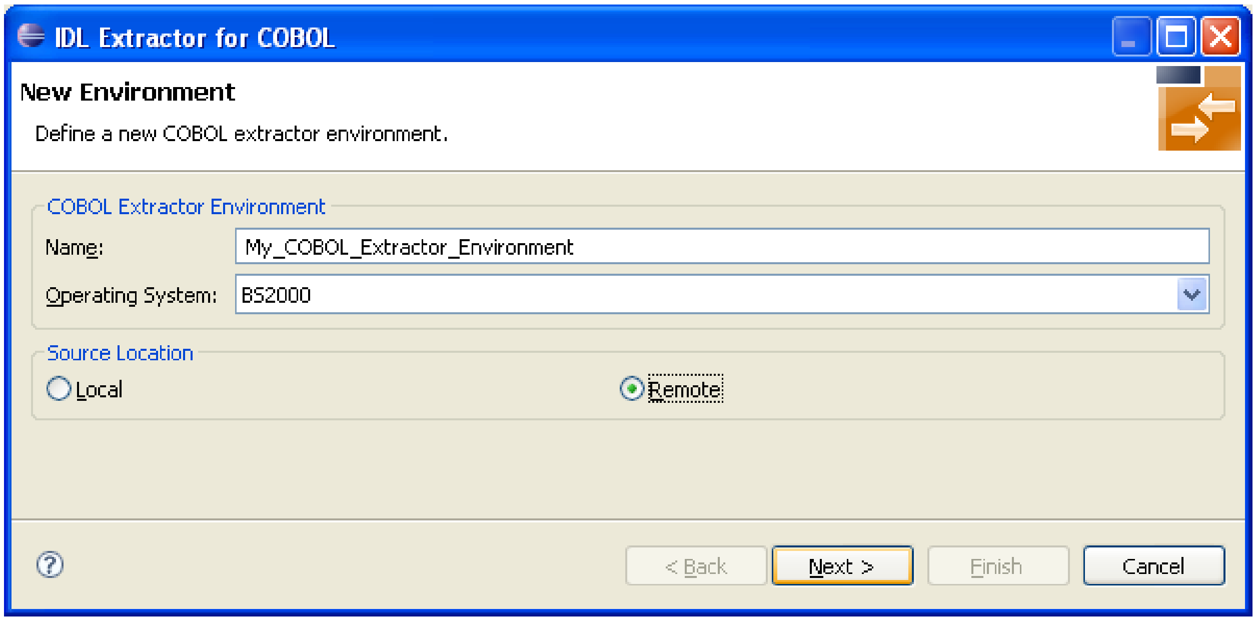
This section describes the four steps for creating a new remote COBOL extractor environment to extract remotely BS2000/OSD COBOL programs stored in LMS libraries.

- Step 1: Define the New Remote Environment
- Step 2: Define the Default Settings
- Step 3: Define the Remote Extractor Environment

- Step 4: Define the Remote Copybook Locations

Step 1: Define the New Remote Environment

On the **New Environment** page you can specify **Name**, **Operating system** and the **Remote Source Location**.



Name: My_COBOL_Extractor_Environment

Operating System: BS2000

Source Location: Local [✓ Remote]

To define the new environment settings

1 Enter a unique name for the COBOL extractor environment.

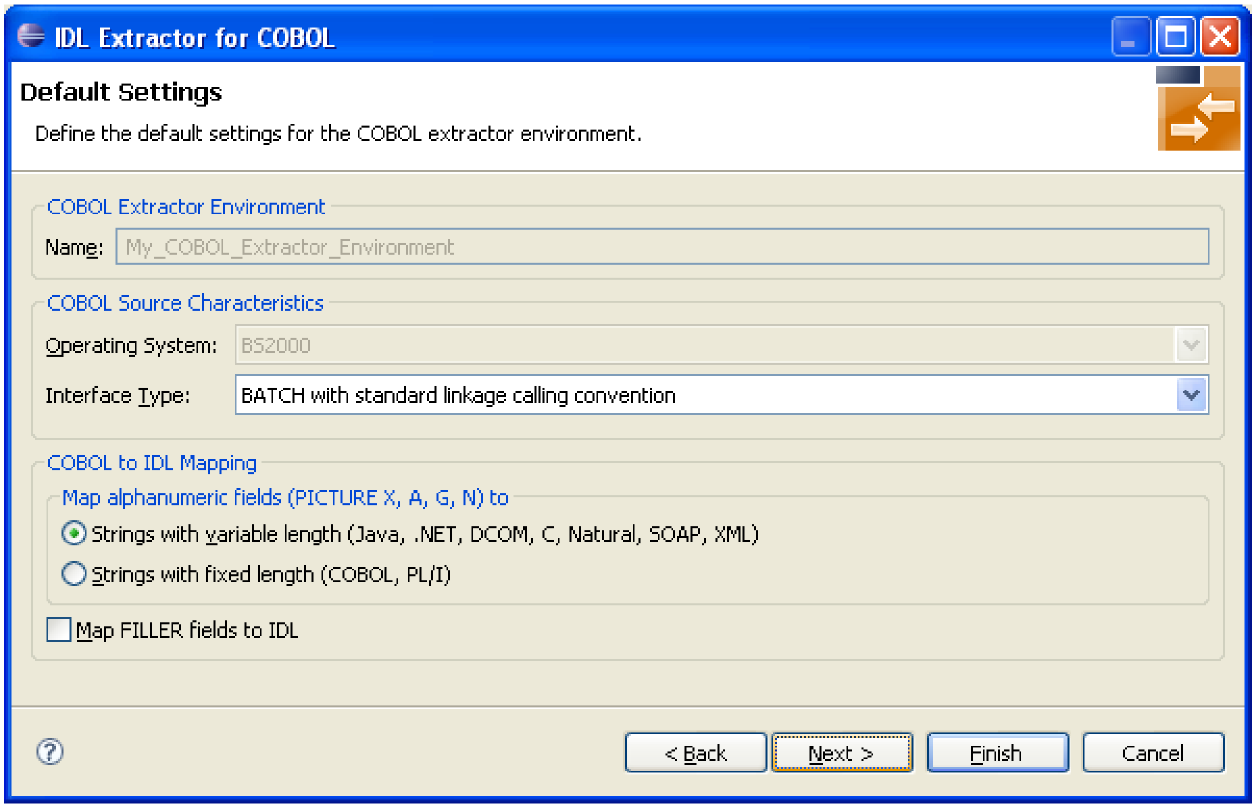
2 Select the **Operating system**.

3 Select "Remote" for **Source location**.

Step 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**.


IDL Extractor for COBOL

Default Settings

Define the default settings for the COBOL extractor environment.

COBOL Extractor Environment

Name: My_COBOL_Extractor_Environment

COBOL Source Characteristics

Operating System: BS2000

Interface Type: BATCH with standard linkage calling convention

COBOL to IDL Mapping

Map alphanumeric fields (PICTURE X, A, G, N) to

[✓ Strings with variable length (Java, .NET, DCOM, C, Natural, SOAP, XML)] Strings with fixed length (COBOL, PL/I)

☐ Map FILLER fields to IDL

To define the default extraction settings

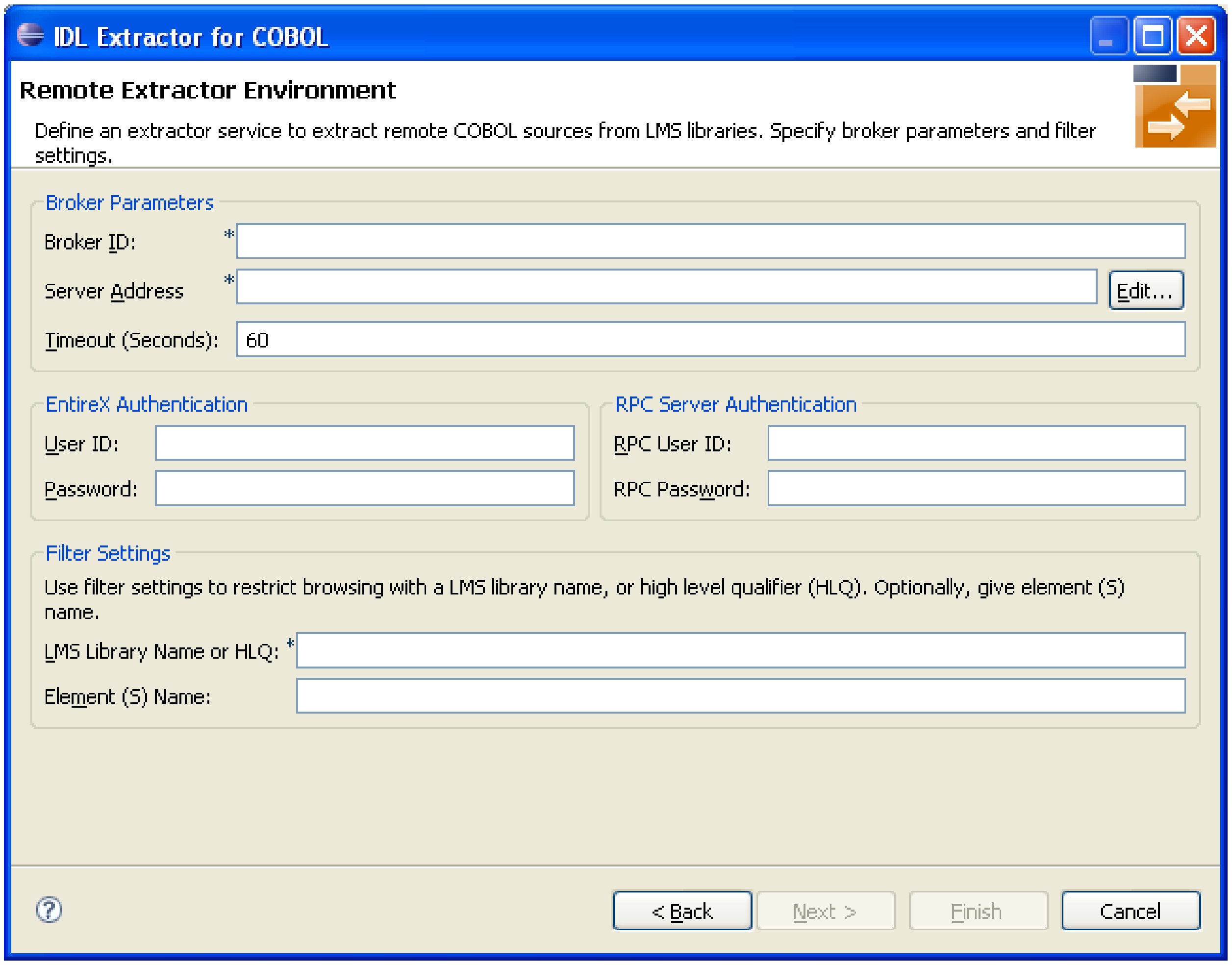
1 Select the default **Interface Type**. See *Supported COBOL Interface Types*.

2 Specify the default **COBOL to IDL Mapping**. See *COBOL to IDL Mapping*.

Press **Next** and continue with *Step 3: Define the Remote Extractor Environment* below.

Step 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* in the BS2000/OSD Batch RPC Server documentation.



Remote Extractor Environment

Define an extractor service to extract remote COBOL sources from LMS libraries. Specify broker parameters and filter settings.

Broker Parameters

Broker ID: *	
Server Address *	
Timeout (Seconds):	60

EntireX Authentication

User ID:	
Password:	

RPC Server Authentication

RPC User ID:	
RPC Password:	

Filter Settings

Use filter settings to restrict browsing with a LMS library name, or high level qualifier (HLQ). Optionally, give element (S) name.

LMS Library Name or HLQ: *	
Element (S) Name:	

To define the remote extractor environment

1 Under **Broker Parameters**, enter the required fields Broker ID and Server Address, which will have the default format brokerID@serverAddress. The last part (broker service) of the server address must always be "EXTRACTOR". The **Timeout** value must be in the range 1-9999 seconds (default is 60).

2 The **EntireX Authentication** parameters describe the settings for the broker. See *Authentication of User* under *Overview of EntireX Security* in the EntireX Security documentation.

3 The **RPC Server Authentication** parameters describe the settings for the RPC server. See *Configuring the RPC Server* in the BS2000/OSD administration documentation.

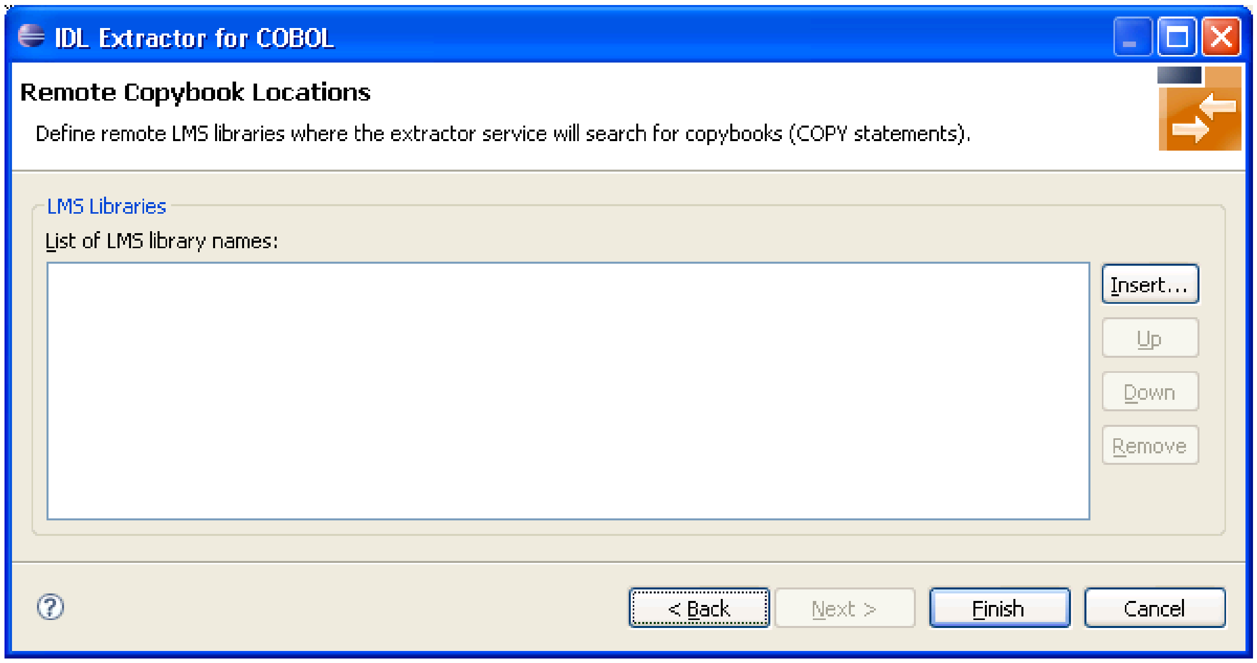
4 A high-level qualifier can be entered in the **LMS Library Name or HLQ** field. The extractor service will then offer only LMS libraries with this high-level qualifier. You can use wildcard notation with asterisk to specify a range of values.

5 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.

Press **Next** and continue with *Step 4: Define the Remote Copybook Locations* below.

Step 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:


IDL Extractor for COBOL

Remote Copybook Locations

Define remote LMS libraries where the extractor service will search for copybooks (COPY statements).

LMS Libraries

List of LMS library names:

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**.

15 COBOL to IDL Mapping

- COBOL Data Type to Software AG IDL Mapping 241
- DATA DIVISION Mapping 244
- PROCEDURE DIVISION Mapping 254
- Copybooks 255

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***.

COBOL Data Type to Software AG IDL Mapping

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

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

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

COBOL Data Type			Software AG IDL Data Type		Notes
Alphabetic		PIC A(*n*)	A*n*, AV*n*	Alphanumeric	1,2
DBCS		PIC G(*n*)	K*n**2, KVn*2	Kanji	1,2,3
DBCS		PIC N(*n*) [USAGE] [IS] DISPLAY-1	K*n**2, KV*n**2	Kanji	1,2,3
Unicode or DBCS		PIC N(*n*)	U*n*, UV*n* or K*n**2, KV*n**2	Unicode or Kanji	1,2,3,10
Unicode		PIC N(*n*) [USAGE] [IS] NATIONAL	U*n*, UV*n*	Unicode	1,2
Alphanumeric		PIC X(*n*)	A*n*, AV*n*	Alphanumeric	1,2
Numeric	Zoned decimal	PIC 9(*n*)[V9(*m*)]	NU*n*[,*m*]	Unpacked decimal unsigned	2,4,8
	Zoned decimal	PIC S9(*n*)[V9(*m*)]	N*n*[,*m*]	Unpacked decimal	2,4,8
	Packed decimal	PIC 9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	PU*n*[,*m*]	Packed decimal unsigned	2,4,8
	Packed decimal	PIC S9(*n*) [V9(*m*)] COMP[UTATIONAL]-3	P*n*[,*m*]	Packed decimal	2,4,8
	Packed decimal	PIC 9(*n*) [V9(*m*)] PACKED-DECIMAL	PU*n*[,*m*]	Packed decimal unsigned	2,4,8
	Packed decimal	PIC S9(*n*) [V9(*m*)] PACKED-DECIMAL	P*n*[,*m*]	Packed decimal	2,4,8
	Binary	PIC [S]9(*n*) BINARY (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6,7
	Binary	PIC [S]9(*n*) BINARY (5<=*n*<=9)	I4	Integer (large)	2,4,5,6,7
	Binary	PIC [S]9(*n*) COMP[UTATIONAL][-4] (1<=*n*<=4)	I2	Integer (medium)	2,4,5,6,7
	Binary	PIC [S]9(*n*) COMP[UTATIONAL][-4] (5≤*n*<=9)	I4	Integer (large)	2,4,5,6,7

COBOL Data Type			Software AG IDL Data Type		Notes
	Binary	`PIC [S]9(n) COMP-5 (1<=n<=4)`	`I2`	Integer (medium)	2,4,6,7
	Binary	`PIC [S]9(n) COMP-5 (5<=n<=9)`	`I4`	Integer (medium)	2,4,6,7
	Floating point	`COMP[UTATIONAL]-1`	`F4`	Floating point (small)	9
	Floating point	`COMP[UTATIONAL]-2`	`F8`	Floating point (large)	9
Alphanumeric-edited		Alphanumeric item containing "0" or "/"	`A(length of PIC)`	Alphanumeric	11
Numeric-edited		Numeric item containing "DB", "CR", "Z", "$", ".", ",", "+", "-", "*", "B", "0" or "/"	`A(length of PIC)`	Alphanumeric	11

Notes:

1. Mapping to fixed-length or variable-length Software AG IDL data type is controlled in the extraction settings of the extraction wizard, see ***Step 4: Define the Extraction Settings and Start Extraction***.
2. Equivalent alternative forms of the `PICTURE` clause, e.g. `XXX`, `AAA`,`NNN`, `GGG` or `999` may also be used.
3. The length for IDL data type is given in bytes. For COBOL the length is in DBCS characters (2 bytes).
4. The character "`P[(n)]`" stands for a decimal scaling position, this character has no effect on the length of the generated data type. Only the data fraction will be mapped to the Software AG IDL:

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

   will be mapped to IDL:

```
1 GROUP1
 2 FIELD1 NU4
```

5. Behavior depends on the COBOL compiler settings:
   - With COBOL 85 standard, the value range depends on the number of digits in the `PICTURE` clause. This differs from the value range of the IDL data type using the binary field size instead. If the parameter is of direction "In" your RPC client application has to ensure the integer value sent is within the allowed range. See *Software AG IDL Grammar* 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
  - with operating systems UNIX and Windows and a Micro Focus compiler, see option `NOTRUNC` in your Micro Focus COBOL documentation.

6. For unsigned COBOL data types (without "`S`" in the `PICTURE` clause) the value range of the IDL data type differs:
   - IDL allows negative values, COBOL does not.
   - For I2, the maximum is 32767 for IDL instead of 65535 for COBOL.
   - For I4, the maximum is 2147483647 for IDL instead of 4294967294 for COBOL.
7. Binaries with more than 9 digits in the `PICTURE` clause cannot be mapped to IDL. See the following table:

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

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

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

   Only the IDL range 0=*n*=7 and 1=(*m*+*n*)=29 is supported.
9. `COMPUTATIONAL-1` (4-byte, single precision) and `COMPUTATIONAL-2` items (8-byte, double precision) items are an IBM-specific extension. When floating-point data types are used, rounding errors can occur, so the values of senders and receivers might differ slightly.
10. If this form is extracted from a COBOL program originally written for Micro Focus COBOL and operating system UNIX or Windows, the mapping to the IDL data type depends on the setting in the IDL Extractor for COBOL Preferences. See **Meaning of PIC N without USAGE clause** within pane **Compiler Directives** of ***Step 2: Define the Default Settings***. For all other COBOL program extractions, the mapping is always to IDL data type `U`*n*/`Uv`*n*.
11. COBOL alphanumeric/numeric edited items will force the generation of IDL data type `A` with an inline comment containing the original `COBOL PICTURE` clause. The `CURRENCY SIGN` clause in the `SPECIAL-NAMES` and the `CURRENCY` compiler option is not considered.

DATA DIVISION Mapping

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

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

BLANK WHEN ZERO Clause

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

Condition Names - Level-88 Data Items

See the following COBOL syntax:

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

Semantically, level-88 condition names can be

- **Enumeration Type Values**
  If your COBOL server requires the level-88 value to be provided on a call-by-call basis, that is, the value may change with every call, map the level-88 base variable to a simple IDL parameter with the desired direction In, Out or InOut. RPC clients have to pass correct values, same as defined by the level-88 condition names.
- **Single Constant Values**
  If your COBOL server interface expects for your purpose always a constant value, map the level-88 condition names to a constant.
- **Function or Operation Codes**
  If the level-88 values are function or operation codes, map the level-88 condition names to an operation.

Continuation Lines

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

DATE FORMAT Clause

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

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

FILLER Pseudo-Parameter

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

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

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

This will be mapped to Software AG IDL:

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

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

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

This will be mapped to Software AG IDL:

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

GLOBAL and EXTERNAL Clause

The `GLOBAL` clause

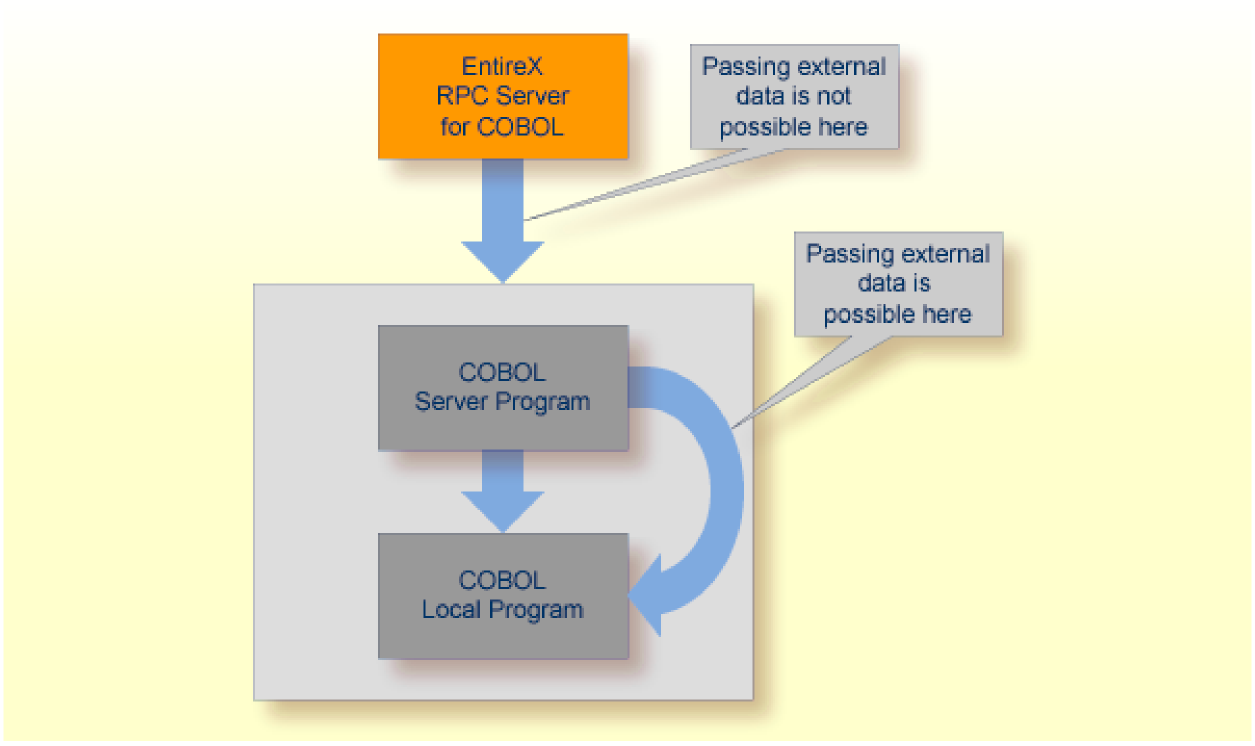
- specifies that a data-name is available to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name.
- is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `GLOBAL` clause.

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

The `EXTERNAL` clause

- can only be specified on data description entries that are in the Working-Storage section of a program.
- is not considered by the IDL Extractor for COBOL. The `DATA DIVISION` is parsed as without the `EXTERNAL` clause.

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


EntireX RPC Server for COBOL
Passing external data is not possible here
Passing external data is possible here
COBOL Server Program
COBOL Local Program

JUSTIFIED Clause

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

OBJECT REFERENCE Phrase

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

Parameter Names

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

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

by default will be mapped to Software AG IDL:

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

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

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

see ***`FILLER` Pseudo-Parameter*** above.

POINTER Phrase

The `POINTER` phrase is an IBM-specific extension.

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

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

PROCEDURE-POINTER Phrase

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

REDEFINE Clause

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

- ***COBOL Mapping Editor***
  - Select the single `REDEFINE` path for a level 1 `REDEFINE` unit (all redefine paths addressing the same storage location) in the parameter selection window. See ***Step 5: Select the COBOL Interface and Map to IDL Interface*** in *Using the IDL Extractor for COBOL*. This is the simplest

and most straightforward approach for a COBOL server with a single interface, because All `REDEFINE` siblings are no longer considered. Further processing is like a single parameter for the level 1 `REDEFINE` path.

- Select the complete `REDEFINE` unit on level 1 with all paths for a COBOL server with multiple interfaces that have to be mapped to operation, and where each operation interface is described by a level 1 `REDEFINE` path in the parameter selection window. See ***Step 5: Select the COBOL Interface and Map to IDL Interface*** in *Using the IDL Extractor for COBOL*. Then model the operation interfaces in the Mapping Editor to IDL programs.
- Only `REDEFINE` units on level 1 can be selected in the parameter selection. `REDEFINE` units on level greater than 1 have to be selected in the Mapping Editor, see below.

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

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

COBOL Syntax	Software AG IDL Syntax
`1 [ `*`name_1`*` ] REDEFINES `*`name_2`*	`1 `*`name_1`*
`1 FILLER REDEFINES `*`name_2`*	`1 FILLER_`*`n`*

RENAMES Clause - LEVEL 66 Data Items

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

SIGN LEADING and TRAILING SEPARATE Clause

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

SYNCHRONIZED Clause

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

Tables with Fixed Size

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

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

Rules

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

Tables with Variable Size - DEPENDING ON Clause

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

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

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

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

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

Rules

- The data item referenced by the `OCCURS DEPENDING ON` clause has to be part of the COBOL server interface as well - in the same COBOL data item direction. This means that if the variable-size table is selected as a
  - COBOL InOut Parameter (see ***Step 5: Select the COBOL Interface and Map to IDL Interface***), the *`index`* data item (ODO subject) must be selected as a COBOL InOut parameter as well.
  - COBOL In Parameter, the *`index`* data item (ODO subject) must be selected as a COBOL In parameter as well.
  - COBOL Out Parameter, the *`index`* data item (ODO subject) must be selected as a COBOL Out parameter as well.
- If the *`index`* data item (ODO subject) is not selected correctly with the variable-size table, unexpected behavior occurs.
- The COBOL from value, *`n`* above, is semantically different from the IDL lower bound and means a lower-bound of elements which must not be crossed. It is the duty of the calling RPC client to take care of this and set the corresponding number of elements correctly. Do not send less than the COBOL lower bound.
- The combination of the `ASCENDING` and `INDEXED BY` phrase as well as `DESCENDING` and `INDEXED BY` phrase is also supported.

Unstructured Data Types - LEVEL 77 Data Items

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

USAGE Clause on Group Level

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

USAGE IS INDEX Clause

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

VALUE Clause

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

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

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

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

PROCEDURE DIVISION Mapping

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

- PROCEDURE DIVISION Header
- BY VALUE Phrase
- RETURNING Phrase

PROCEDURE DIVISION Header

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

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

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

BY VALUE Phrase

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

RETURNING Phrase

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

Copybooks

Copybook Support

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

If copybooks cannot be located, the following rules apply:

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

You can also predefine the following in the preferences:

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

Copybooks with REPLACE Option

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

- a copybook ACPYBK with REPLACE option

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

- referencing the copybook above

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