software AG

webMethods EntireX

Software AG IDL Extractor for COBOL

Version 9.5 SP1

November 2013

webMethods EntireX

Table of Contents

I Introduction to the IDL Extractor for COBOL

1   Introduction to the IDL Extractor for COBOL

Introduction

The Software AG IDL Extractor for COBOL inspects a COBOL source and its copybooks for COBOL data items to extract. It can also extract directly from copybooks. In a user-driven process supported by an *Extractor Wizard*, the interface of a COBOL server is extracted and - with various features offered by a *Mapping Editor* - modelled to a client interface. The property view to which the editor is connected supports you in mapping the COBOL server to the client interface.
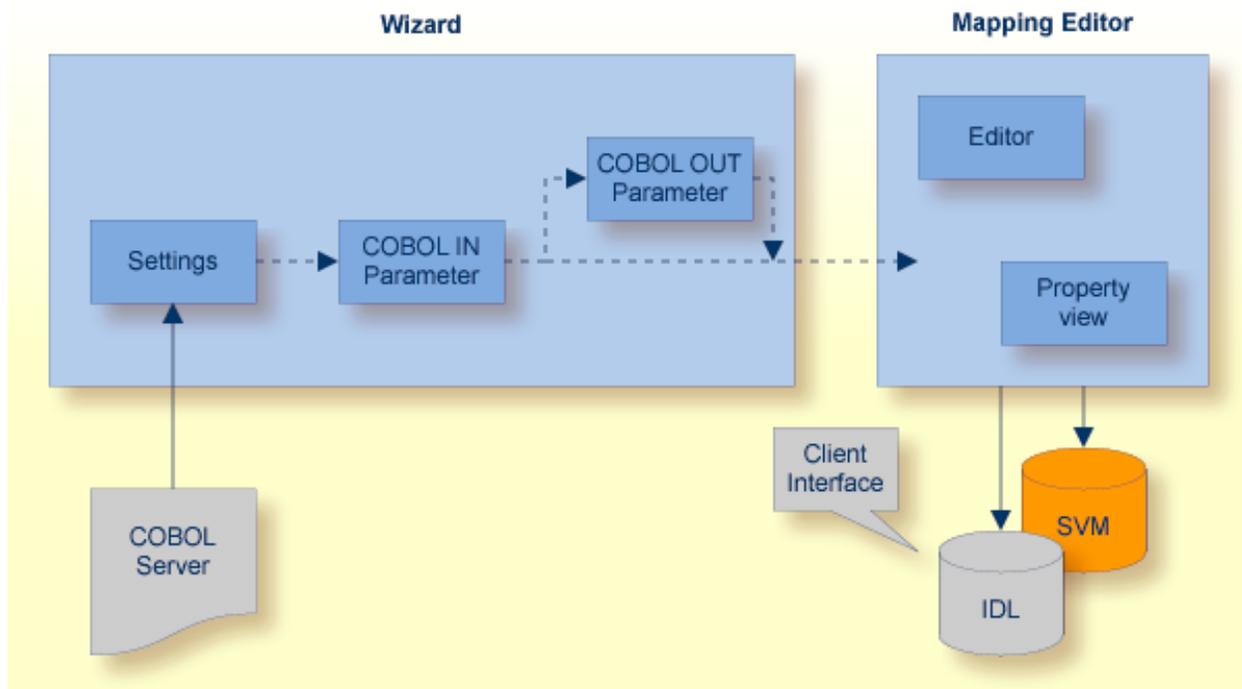
The results of this process are two related files mapping the client interface to the COBOL server:

- **IDL File**
  The Software AG IDL (interface definition language) file 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*.

- **SVM File**
  An SVM File (server mapping file) to complete the mapping is only generated if it is required by the RPC server during runtime to call the COBOL server. A *SVM File Deployment* integrated in the Workbench assists you to deploy the server mapping to the RPC server.
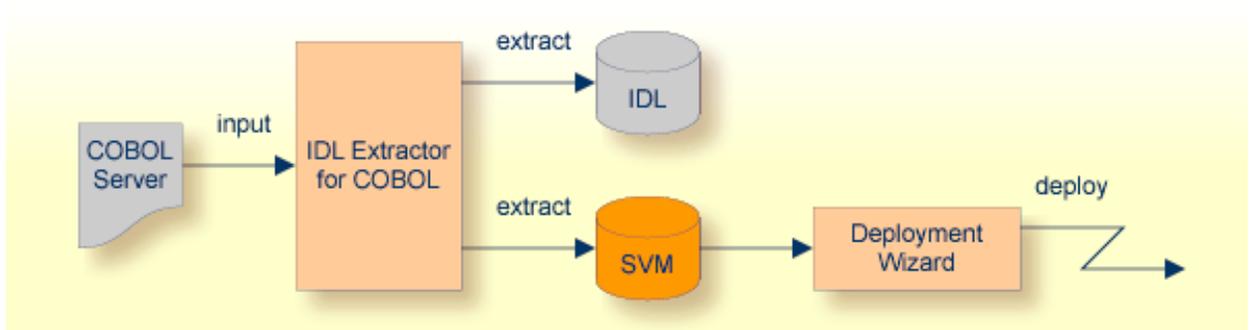
Extractor Wizard

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 with the *COBOL Parameter Selection* 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.

  - COBOL alphanumeric fields (PICTURE X, A, G, N). They can be mapped either to variable-length or fixed-length strings in the IDL. This option is provided for modern RPC clients that support variable-length strings, and also for legacy RPC clients that support fixed-length strings only.

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

Mapping Editor
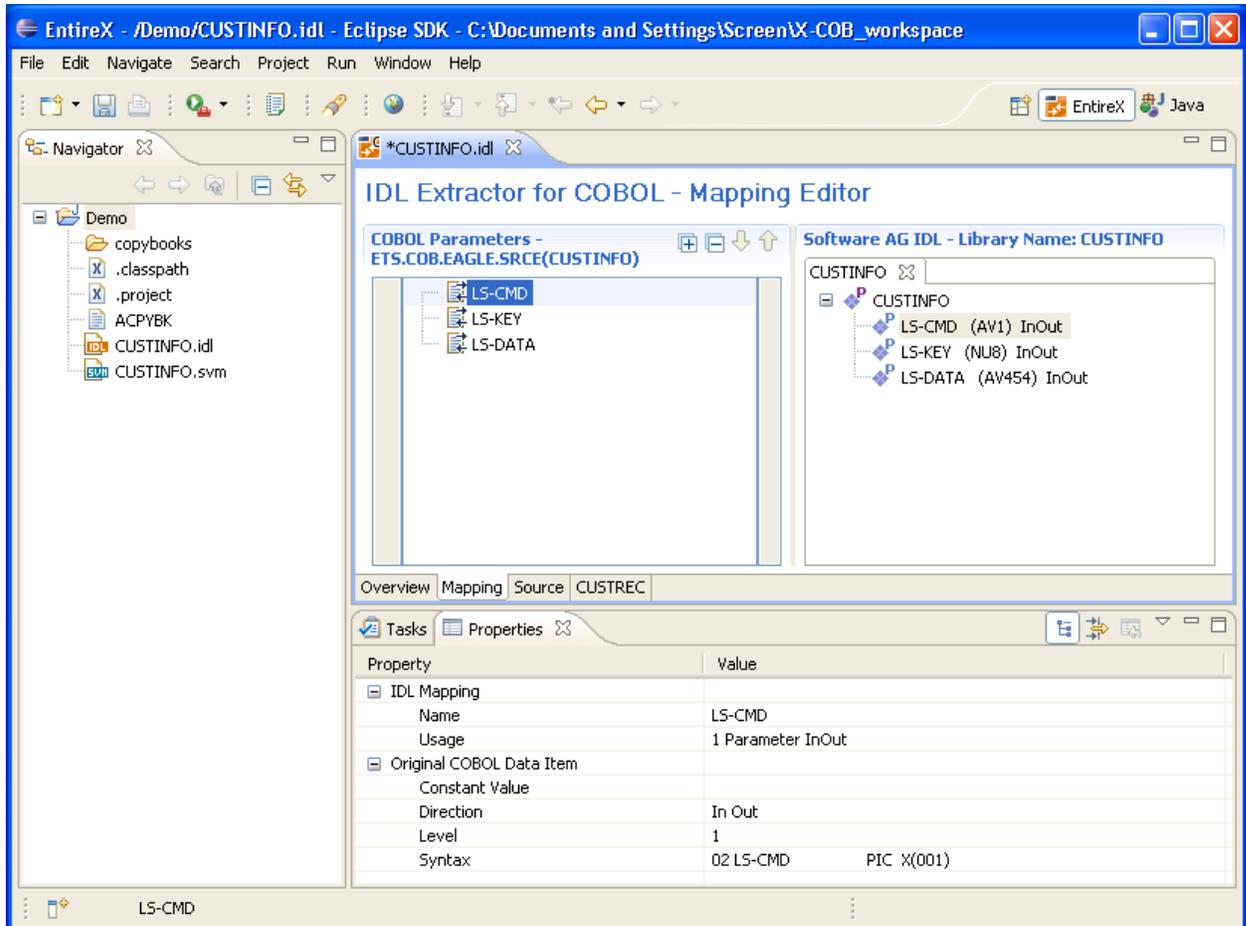
The *COBOL Mapping Editor* is the tool to map the COBOL server interface selected in the *Extractor Wizard* to IDL. This section gives a short overview of the mapping features provided:

- 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. See *Providing IDL Directions (`IN OUT INOUT`)*.

- 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). See *Selecting `REDEFINE` Paths*.

- Suppress or hide unneeded fields in the IDL. This keeps the IDL client interface lean and also minimizes the amount of data transferred during runtime. See *Suppressing or Hiding Unneeded Fields of the COBOL Server*.

- 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. See *Mapping COBOL Data Items to Constants*.

- For a COBOL server implementing multiple business functions controlled by an operation or function code field, multiple interfaces can be selected and modelled with the feature **Map Operations of the COBOL Server to IDL Programs**. See *Mapping COBOL Functions to Multiple IDL Programs*. 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:

- CICS with DFHCOMMAREA Calling Convention
- CICS with Channel Container Calling Convention
- CICS with DFHCOMMAREA Large Buffer Interface
- 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?

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

CICS with DFHCOMMAREA Calling Convention

The IDL Extractor for COBOL supports CICS programs using the standard `DFHCOMMAREA` calling convention.
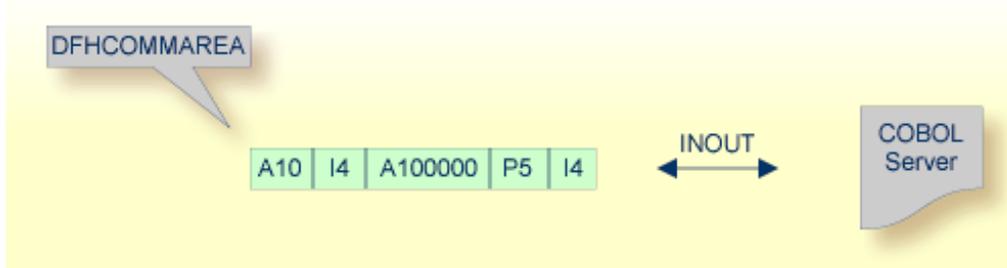
The following illustrates roughly how you can determine whether a COBOL server follows the DFHCOMMAREA calling convention standard:

```
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.
   . . .
```

Typically DFHCOMMAREA programs have a DFHCOMMAREA data item in their Linkage Section and 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 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 *CICS DFHCOMMAREA* under *COBOL Parameter Selection* 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.
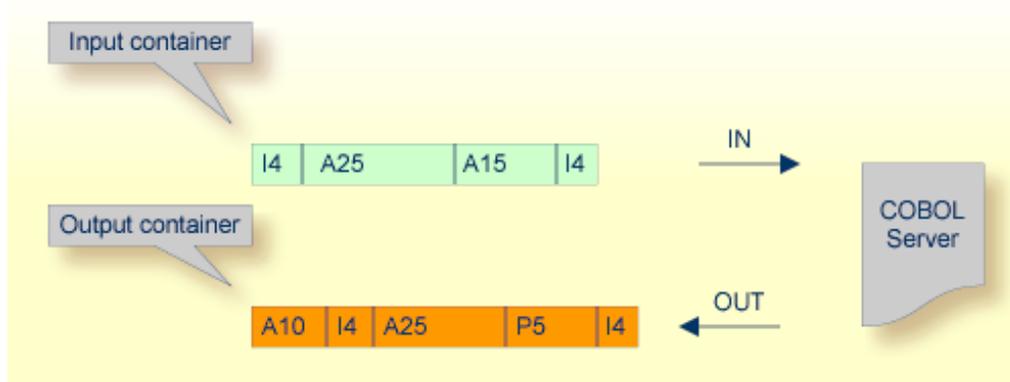
The following illustrates roughly how you can determine whether a COBOL server follows the Channel Container standard.

```
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.
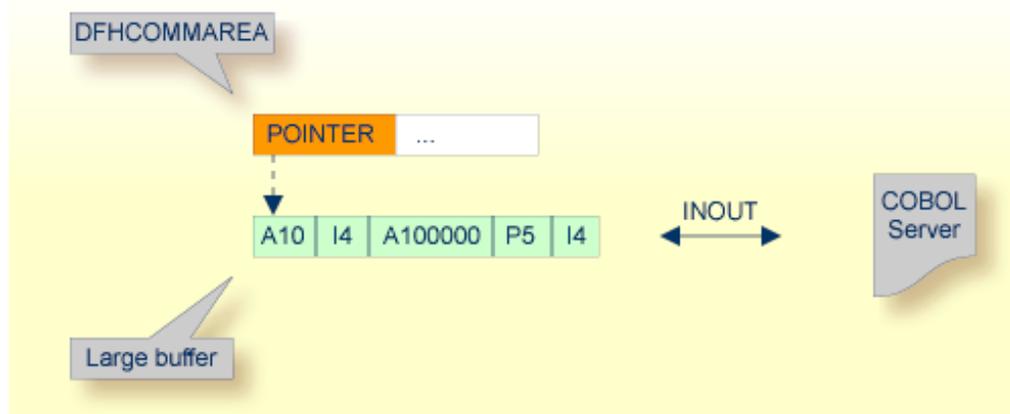
Technically,

■ 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.
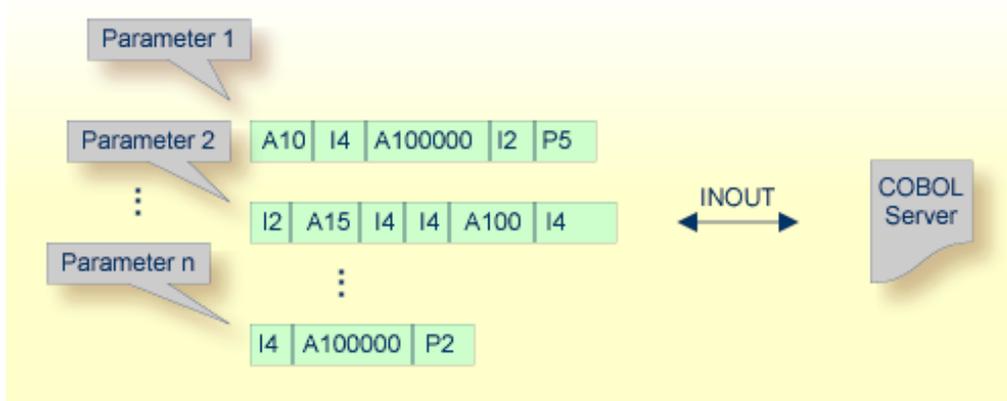
Technically, the generated COBOL server skeleton contains

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

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

Batch with Standard Linkage Calling Convention

Standard call interfaces with a given number of parameters are supported. Every parameter addresses a fixed COBOL structure.
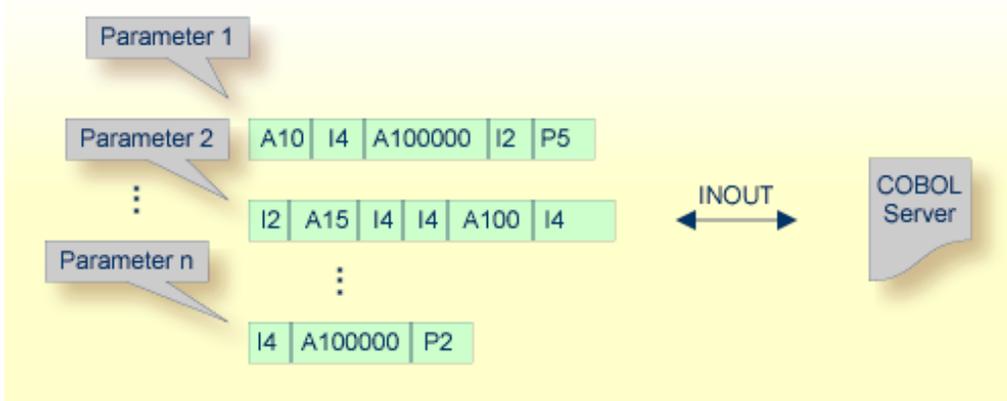
Technically, the COBOL server contains

- a parameter list: `PROCEDURE DIVISION USING PARM1 PARM2 ... PARM`*n*

- the parameters in the linkage section as COBOL data items on level 1

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

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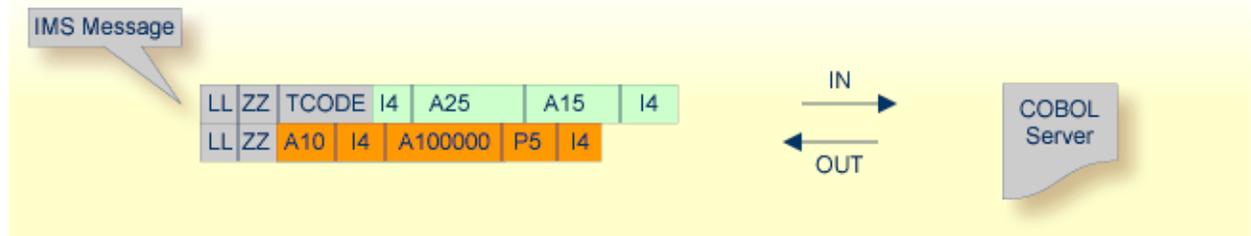
IMS message processing programs (MPP) get their parameters through an IMS message and return the result by sending an output message to IMS. The IDL Extractor for COBOL enables extractions from such programs.

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)* in *COBOL Parameter Selection* 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.
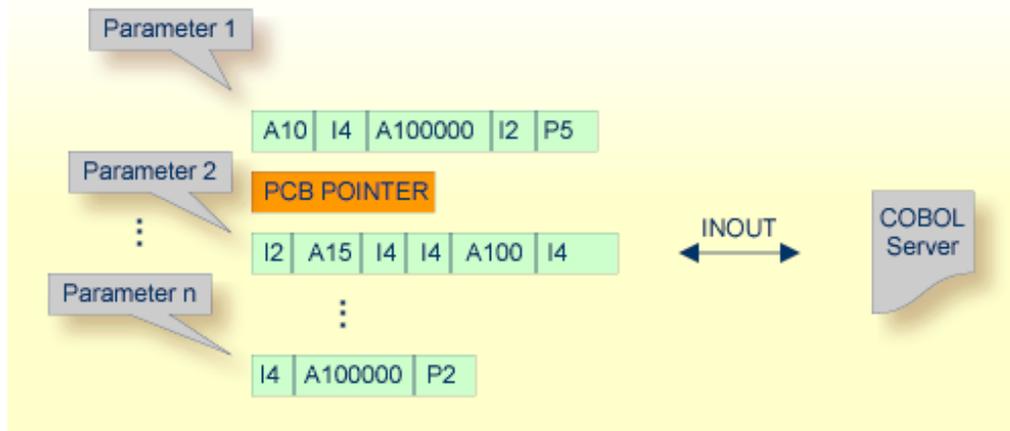
Technically, the COBOL server contains

■ 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 Standard Linkage* under *COBOL Parameter Selection* 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.

Server Mapping File (SVM)

The SVM file (Server Mapping File) completes the Software AG IDL file with a mapping from the programming-language-neutral parameter definition in the IDL file to the parameters and data types expected by the COBOL server. In contrast to a CVM file, an SVM file is deployed to the server environment, see *Server Mapping Deployment*.
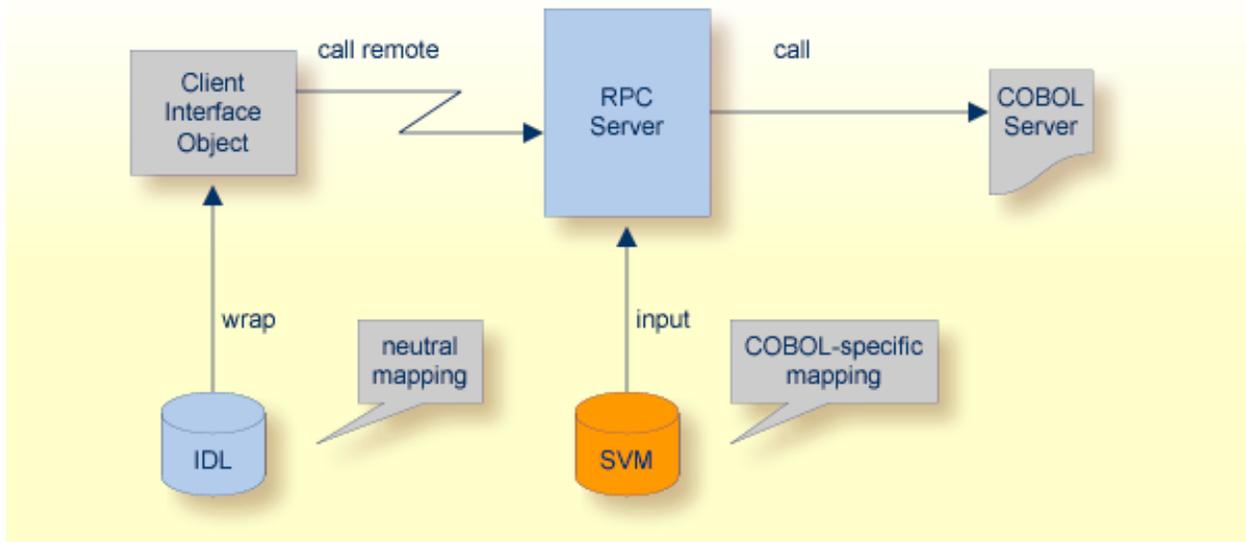
The SVM file contains

- special COBOL constructs such as `REDEFINE` and `OCCURS DEPENDING ON`
- FILLER pseudo-parameters
- map to constants
- map to operation bindings to the COBOL server
- IDL program names to be mapped to customized COBOL names; see *Customize Automatically Generated Server Names*
- other information ...

SVM File Deployment

SVM files obtained using the extraction process are required by the RPC server for correct mapping from the language-neutral parameter definition in the IDL file to the COBOL server. Therefore, the SVM file has to be deployed to the RPC server.
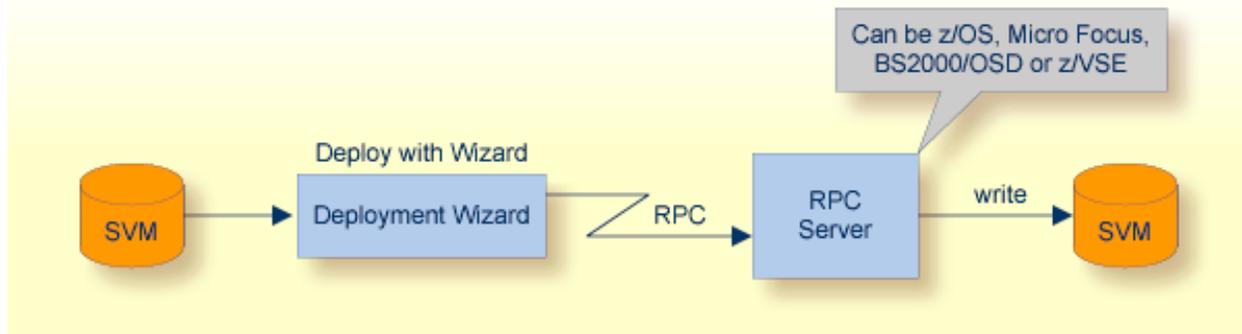
- The RPC server can be batch (z/OS, z/VSE); CICS (z/OS, z/VSE); IMS (z/OS); Micro Focus (UNIX, Windows); BS2000/OSD.

- A deployment wizard integrated in the EntireX Workbench supports in deploying the server mapping to the RPC server. See *Server Mapping Deployment to z/OS, using FTP and IDCAMS*. Most RPC servers work together with the deployment wizard to receive server mappings.

- Alternatively, depending on the RPC server, the server mapping can be deployed manually. See *Deploying a Server Mapping File*.

II Using the IDL Extractor for COBOL - Overview

This chapter describes how to extract IDL from a COBOL source, using the IDL Extractor for CO-BOL, 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 SVM*
- *Scenario II: Append to Existing IDL and SVM*
- *Scenario III: Modify Existing IDL and SVM*
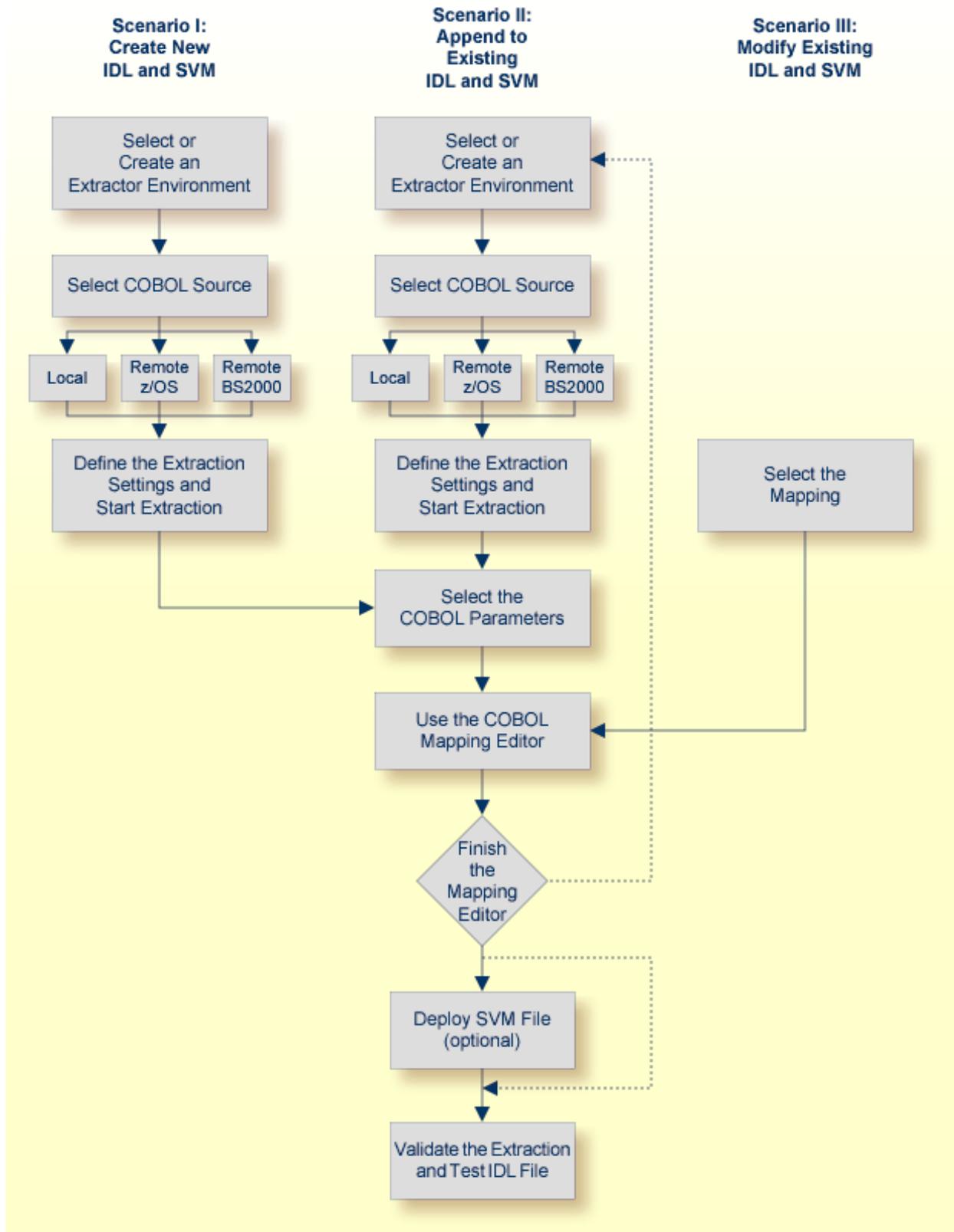
Before Starting an Extraction

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 *1*, *2*, *3* under *COBOL Parameter Selection*.

- 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. See *Mapping COBOL Functions to Multiple IDL Programs*.

  - Suppress or hide unneeded fields. See *Suppressing or Hiding Unneeded Fields of the COBOL Server*.

  - Map COBOL data items to constants. See *Mapping 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 SVM

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.
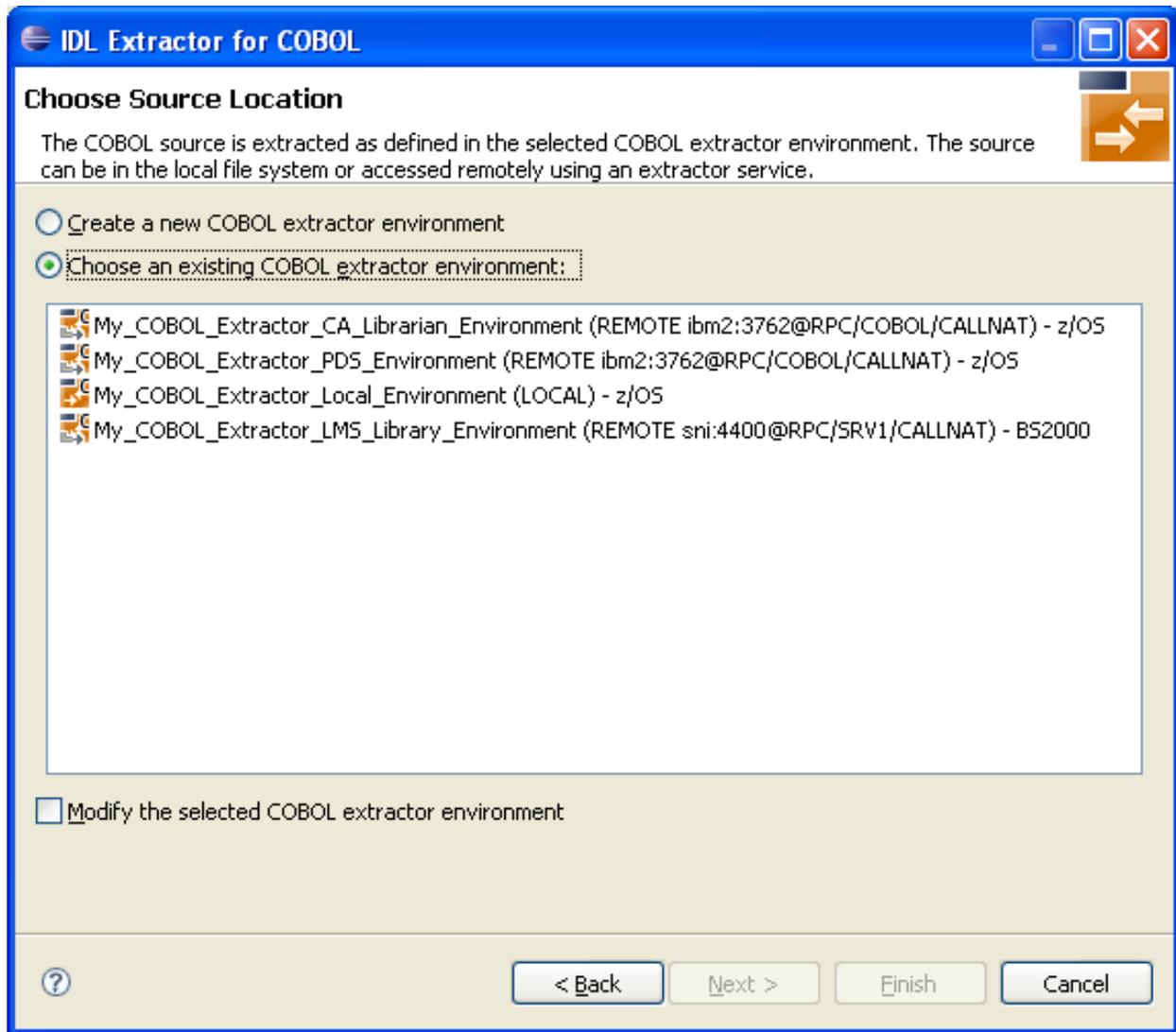
This page offers the following options:

▶ **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 COBOL Source from a Remote Host Computer (z/OS)
- Selecting a COBOL Source from a Remote Computer

Selecting a COBOL Source Stored Locally

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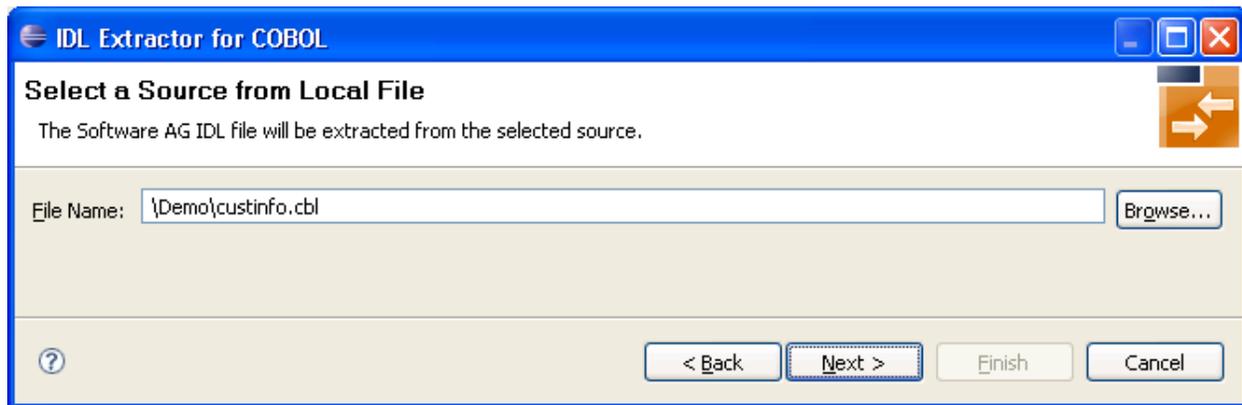
Selecting a COBOL Source from a Remote Host Computer (z/OS)

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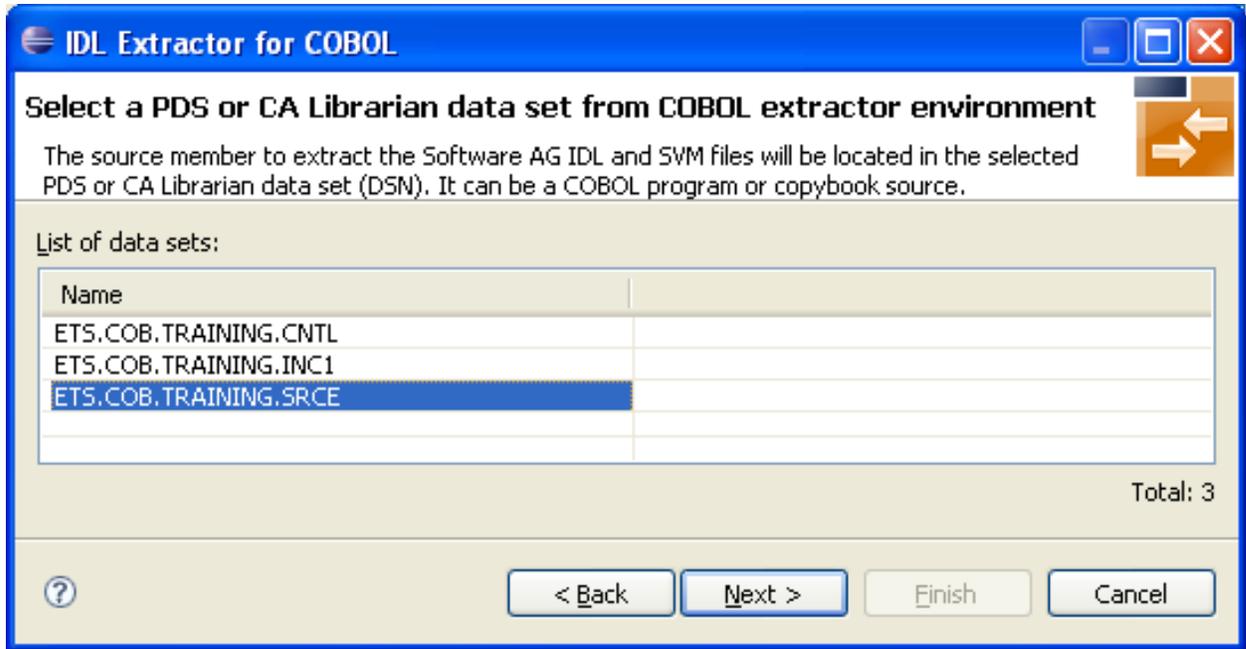
Select the partitioned data set or CA Librarian data set from which you want to extract and choose **Next**. Proceed depending on the selected data set type. See *Selecting a Member from a Partitioned Data Set (z/OS)* or *Selecting a Member from a CA Librarian Data Set (z/OS)*.

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

The following page offers all members contained in the partioned 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*.
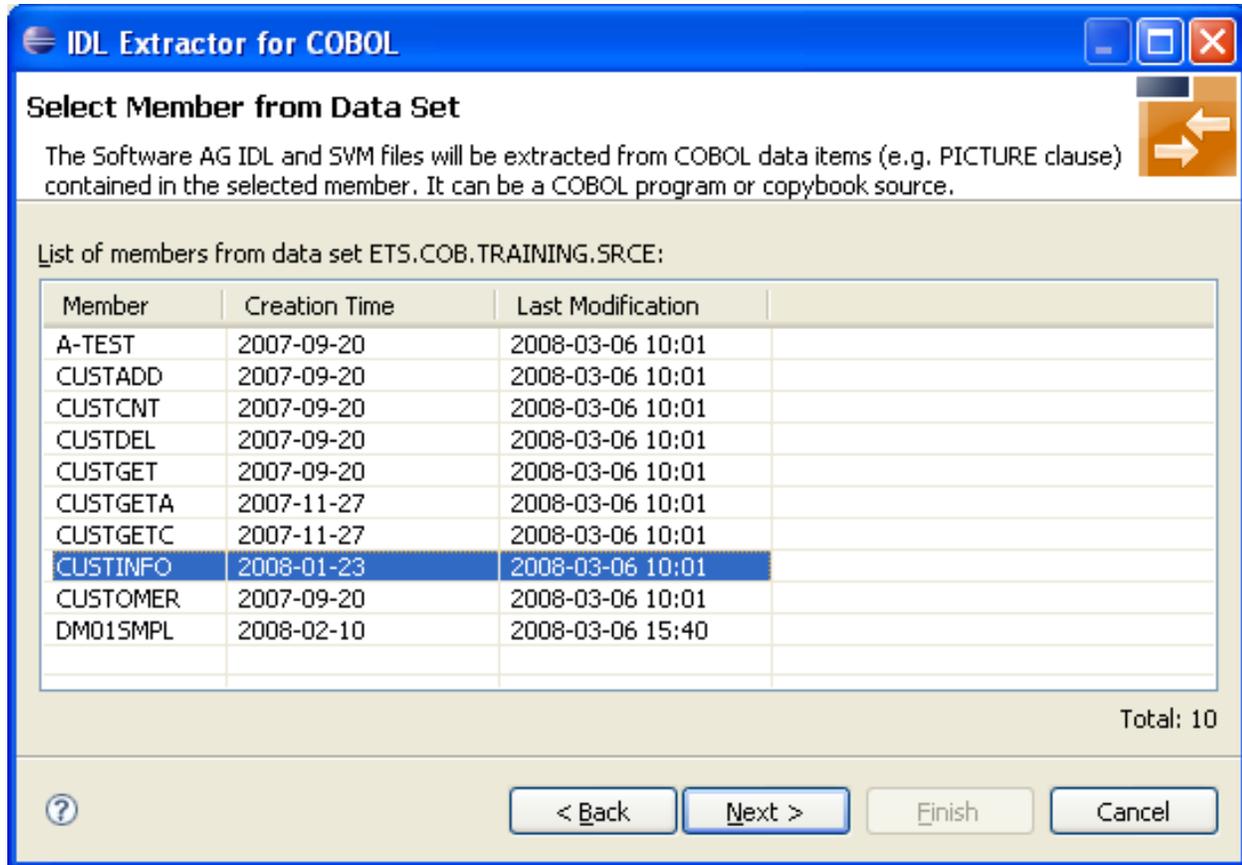
Select the member from which you want to extract. You can select only one COBOL source. The source can be a COBOL program or a COBOL copybook.

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

Selecting a Member from a CA Librarian Data Set (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*.
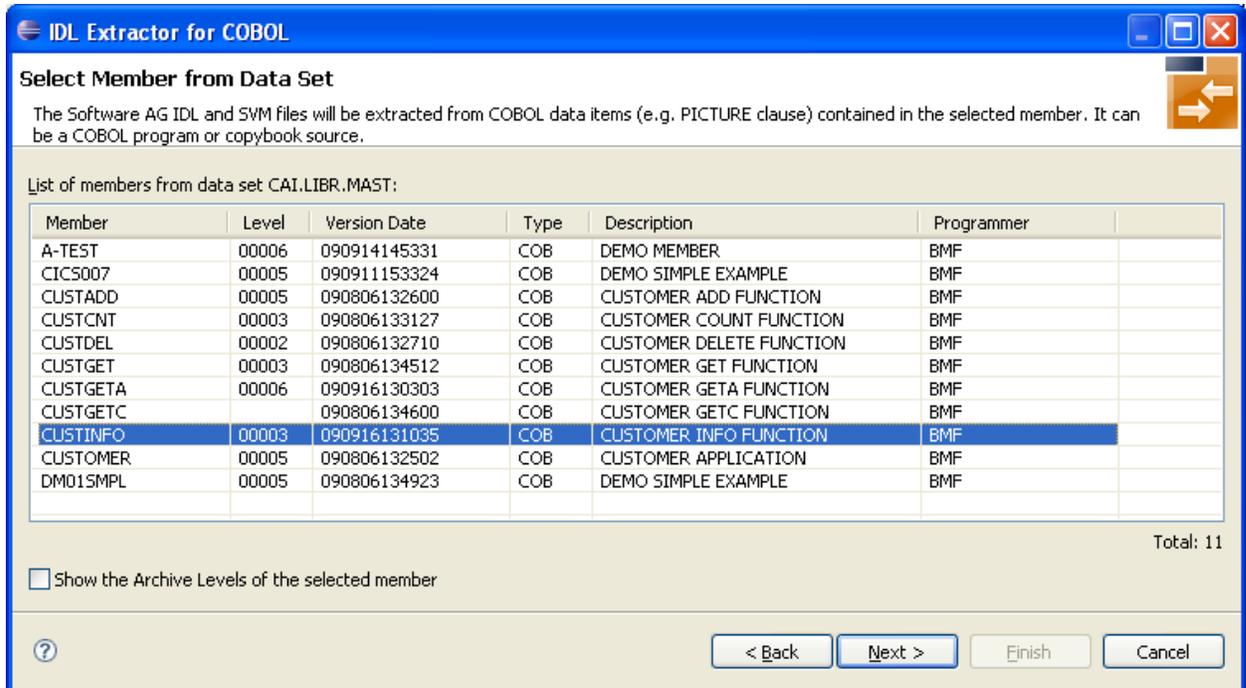
You can select only one COBOL source. The source can be a COBOL program or a COBOL copy-book. If you want to extract from

- 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*.

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

The following page offers all archive levels of the previously selected member.
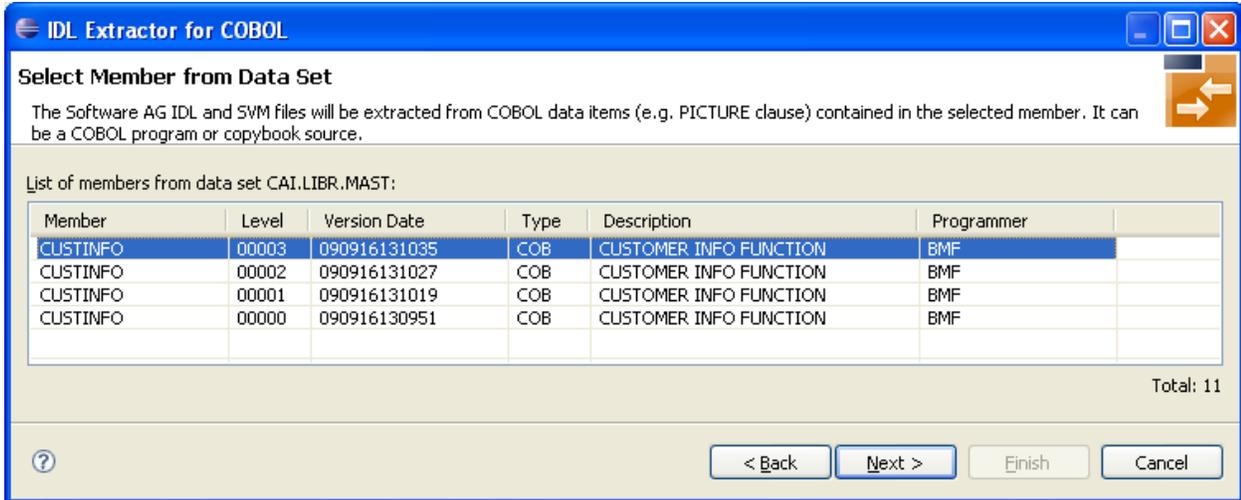
Select the member from which you want to extract. You can select only one archive level. Choose **Next** and continue with *Step 4: Define the Extraction Settings and Start Extraction* below.

Selecting a COBOL Source from a Remote Computer

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*.
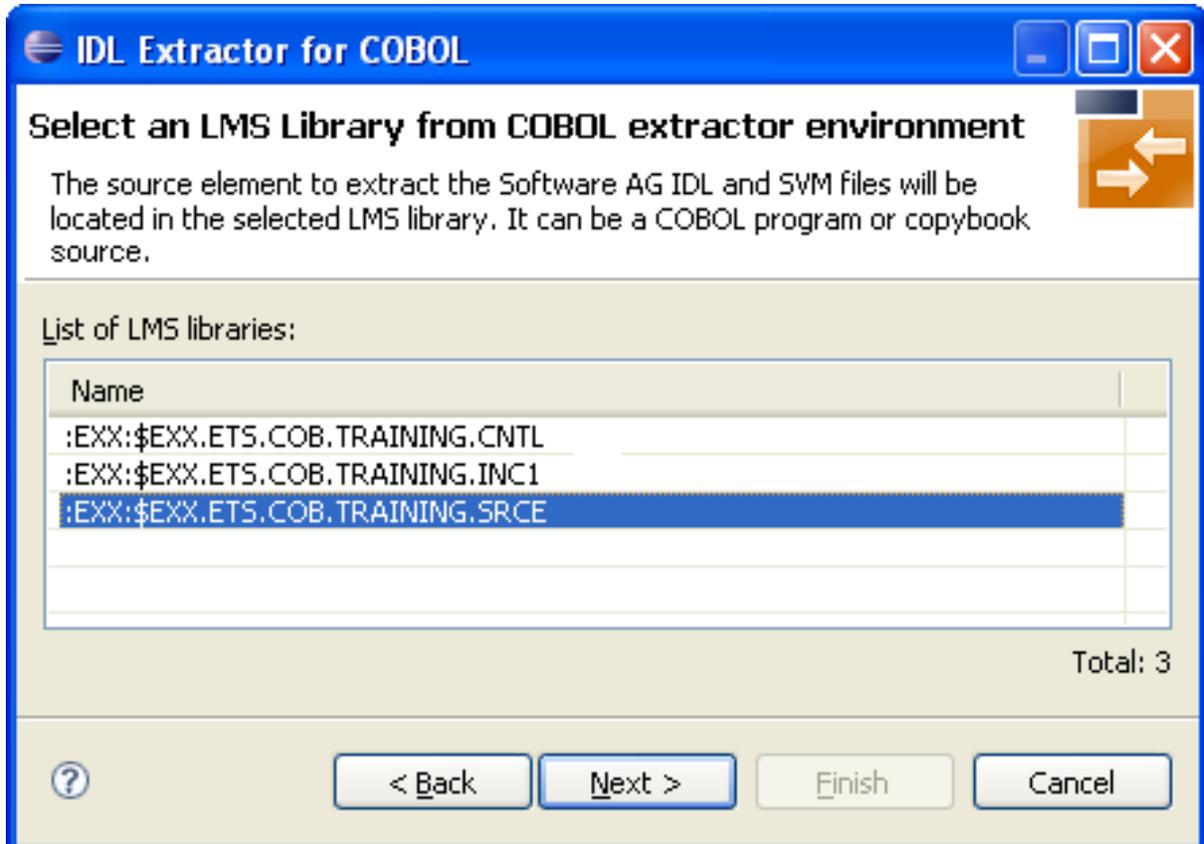
Selecting an Element (S) from an LMS Library (BS2000/OSD)

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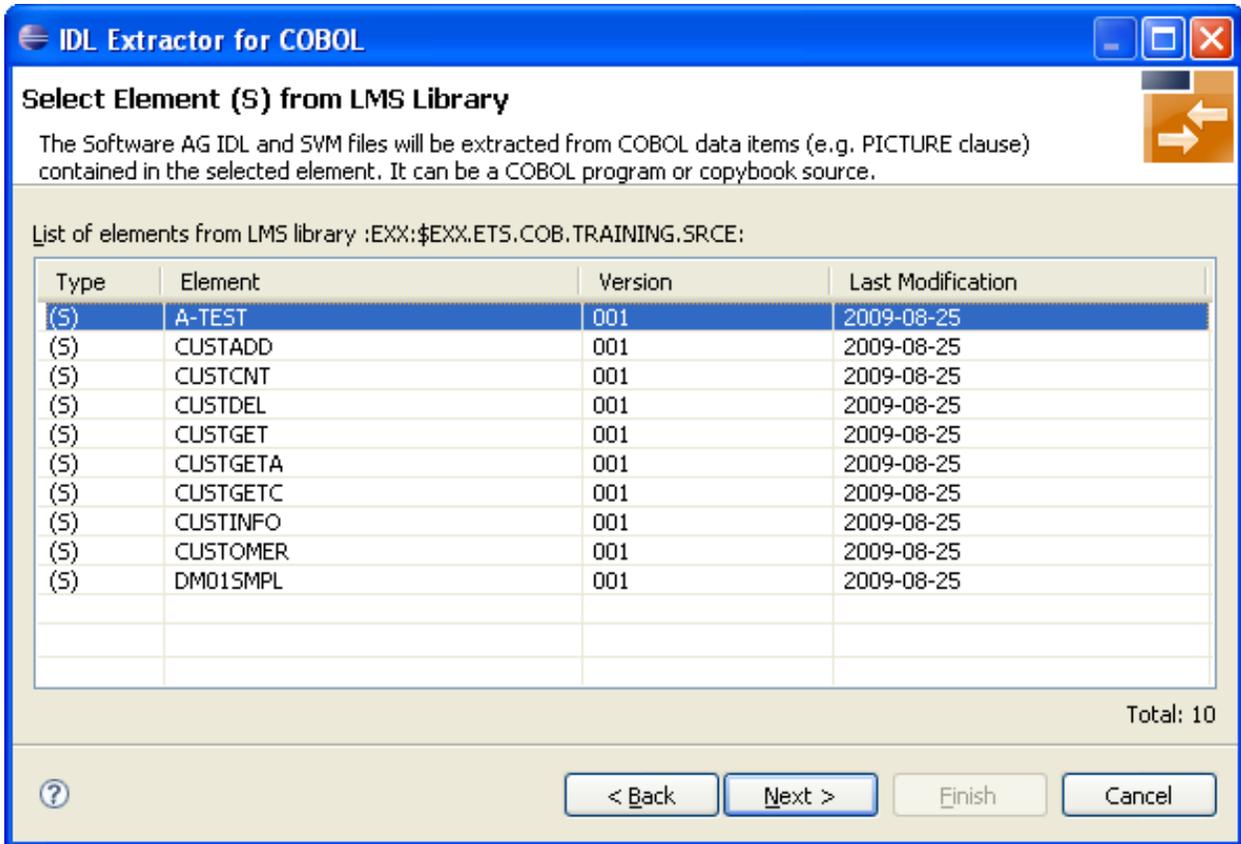
Select the element from which you want to extract. You can select only one COBOL source. The source can be a COBOL program or a COBOL copybook.

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.

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

The **Interface Type** must match the type of your COBOL server program. It is used by the RPC server at runtime to correctly call the COBOL server. Selection of an unsupported interface type is likely to result in problems during execution. To select an appropriate interface type, see:

- *Supported COBOL Interface Types*, for a list of interface types supported by the IDL Extractor for COBOL. (This section also provides an introduction to interface types).

- *Compatibility between Interface Type and RPC Server* for possible combinations of an interface type and a supporting RPC server.

Depending on the interface type, additional information can be set. For the interface type

- **CICS Channel Container Calling Convention Interface**, specify the channel name for the CICS RPC Server (max. 16 characters).

- **IMS MPP Message Interface (IMS Connect)**, specify the length of the transaction code field which is mostly 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".

Select either **Transaction Name** or **Create IDL parameter for Transaction Name - specification at runtime**. For a constant transaction name, specifiy the transaction code value here. In the latter case, the client application is responsible to set the correct transaction code value at runtime.

- **IMS BMP with Standard Linkage Calling Convention**, 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* under *Using the COBOL Wrapper for IMS BMP (z/OS)* in the COBOL Wrapper documentation.

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

With the **COBOL-to-IDL Mapping** 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:
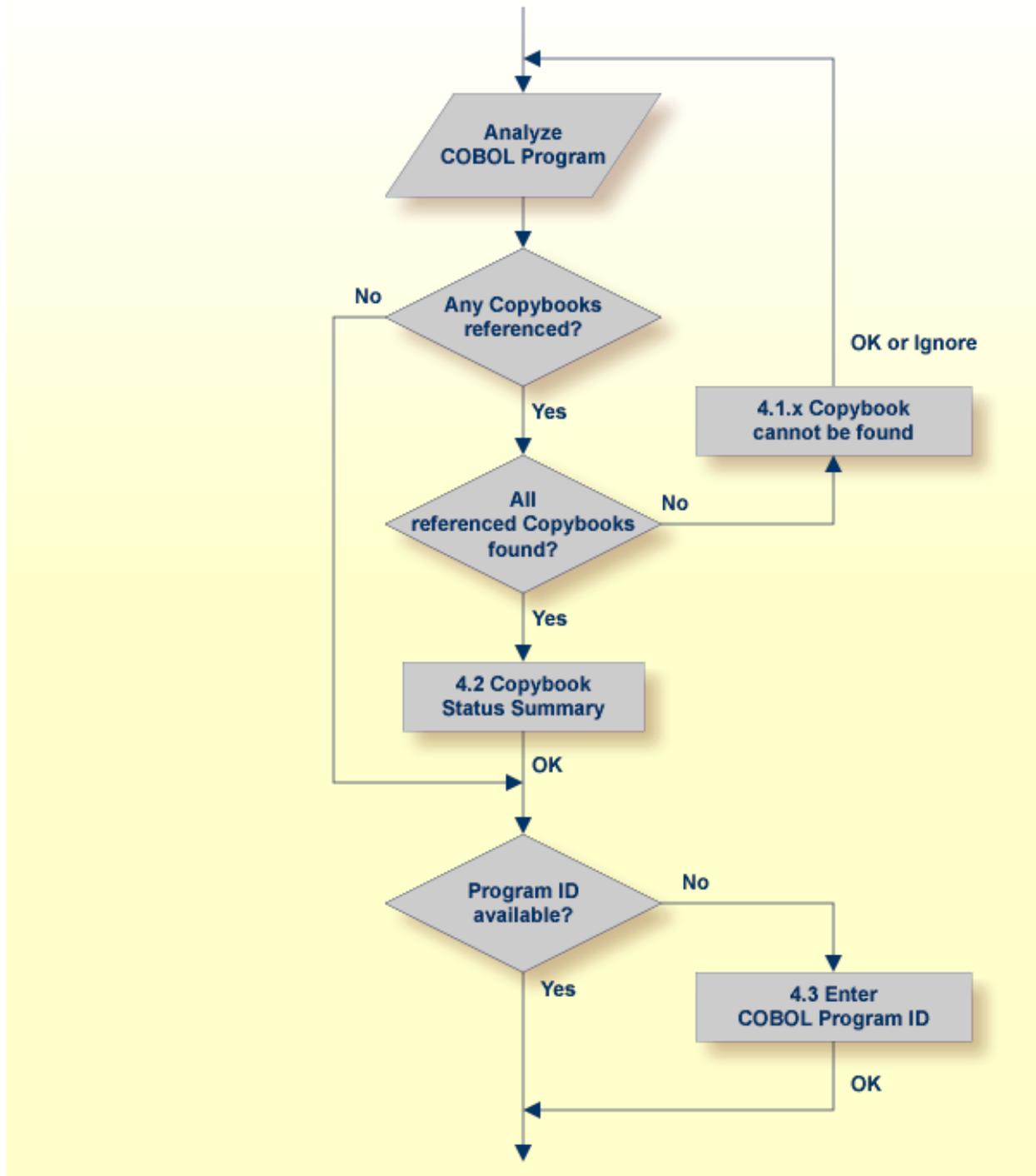
■ Referenced copybooks cannot be found. In this case the wizard prompts you for every missing copybook. Continue with optional step *Step 4.1x: Copybook Cannot be Found* - Local Extraction | Remote Extraction (z/OS) | Remote Extraction (BS2000/OSD) in the IDL Extractor for COBOL documentation depending 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 Parameters* under *Scenario I: Create New IDL and SVM*.

Step 4.1a: Copybook Cannot be Found - Local Extraction

This dialog enables you to browse directories where missing copybooks might be found. If there are any specific copybook file extensions, you can define them here.
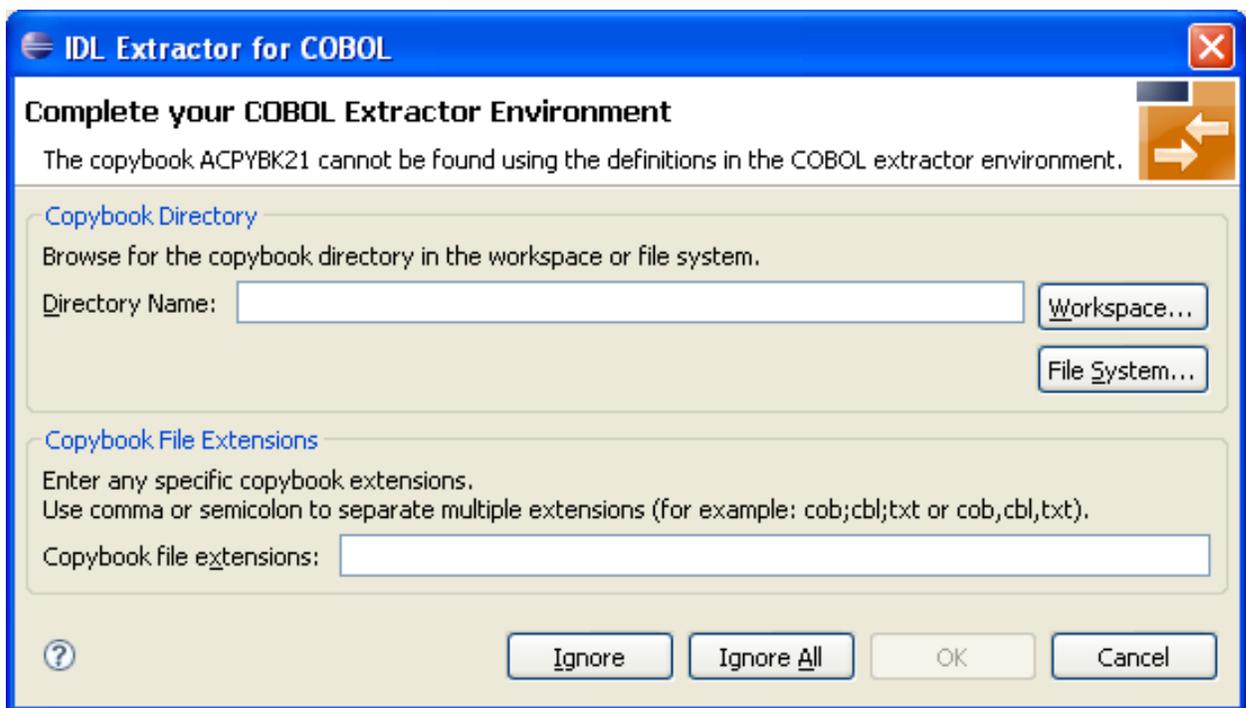
The copybook that cannot be found is given in the window, here its name is "ACPYBK21". In the extractor Preferences, the copybook directory that contains the copybook or the copybook file extension is not defined.

Continue with one of the following actions:

▶ **To ignore this copybook only**

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

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

▶ **To ignore this and all further copybooks**

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

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

▶ **To complete the extractor environment**

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

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

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

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

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

This dialog enables you to browse remote locations (partitioned or CA Librarian data sets) where missing copybooks might be found.
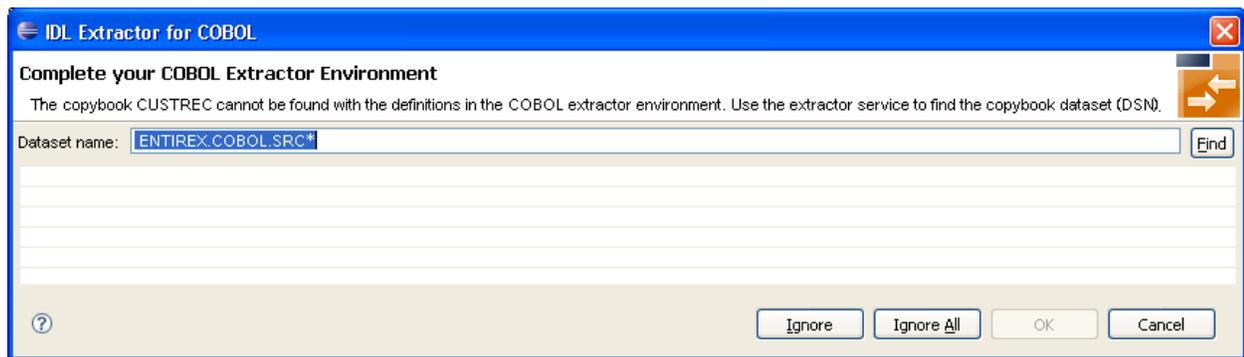
The copybook that cannot be found is given in the window; here its name is "CUSTREC". In the extractor preferences, the copybook data set that contains the copybook is not defined.

Continue with one of the following choices:

▶ **To ignore this copybook only**

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

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

▶ **To ignore this and all further copybooks**

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

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

▶ **To complete the extractor environment**

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

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

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

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

This dialog enables you to browse remote locations (LMS libraries) where missing copybooks might be found.
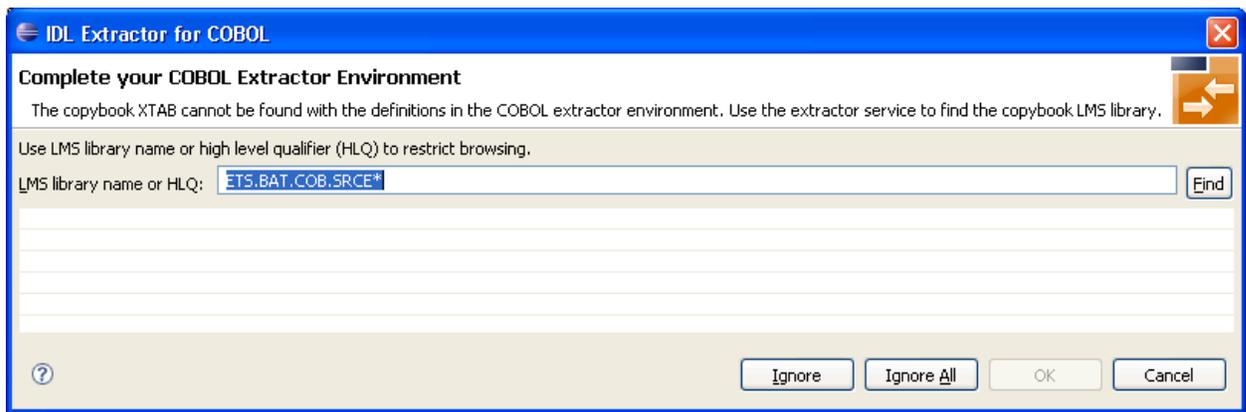
The copybook that cannot be found is given in the window; here its name is "XTAB". In the extractor preferences, the copybook LMS library that contains the copybook is not defined.

Continue with one of the following choices:

▶ **To ignore this copybook only**

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

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

▶ **To ignore this and all further copybooks**

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

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

▶ **To complete the extractor environment**

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

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

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

Step 4.2: Copybook Status Summary (Optional)

This summary page lists all COBOL copybooks which were not available during extraction.
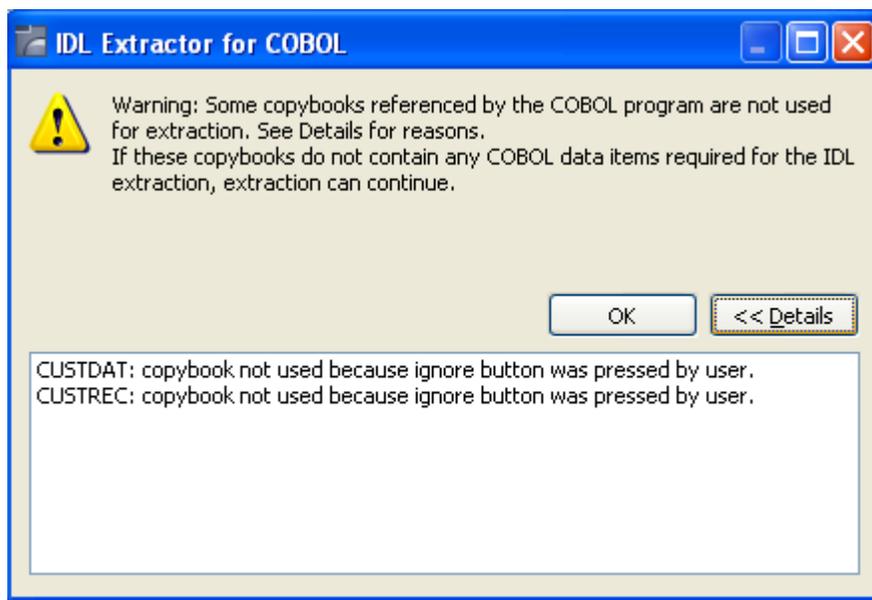
■ If any relevant COBOL parameter describing the server interface is contained in one of the listed copybooks, you cannot continue. Terminate the extraction and try to get the missing copybooks.

■ If no relevant COBOL parameter describing the server interface is contained in the copybooks, you can continue. Choose **OK**.

Step 4.3: Enter COBOL Program ID (Optional)

This page is shown whenever the program ID of the COBOL source is missing. Entering a COBOL program name is compulsory.
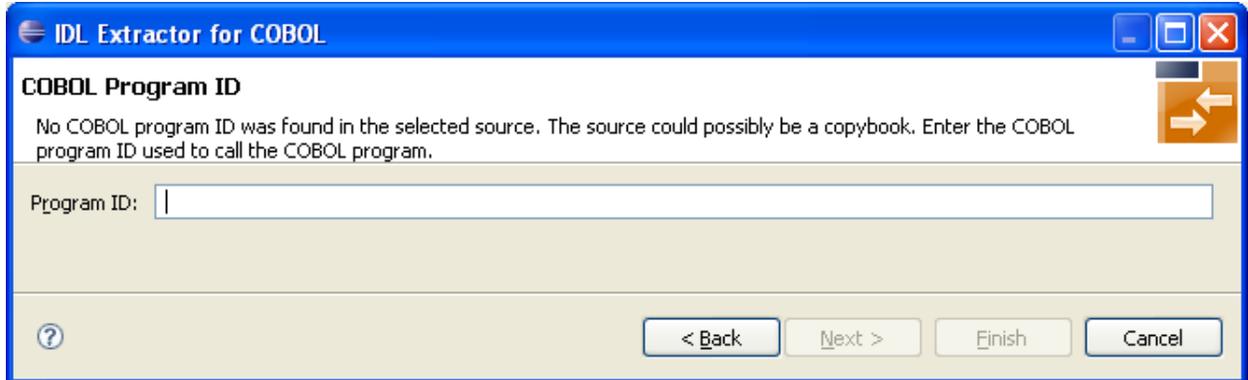
No COBOL program ID can be located if you extract, for example, from a copybook that contains COBOL data items only. The COBOL program ID

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

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

▶ **To complete the extraction**

1   Enter the COBOL program ID.

2   Choose **OK** to continue with *Step 5: Select the COBOL Parameters*.

Step 5: Select the COBOL Parameters

In this page you select the COBOL data items that form the COBOL `INOUT` parameters of your COBOL server program. For a detailed description of the functionality, context menus, toolbars and main panes of this page, see *COBOL Parameter Selection*.
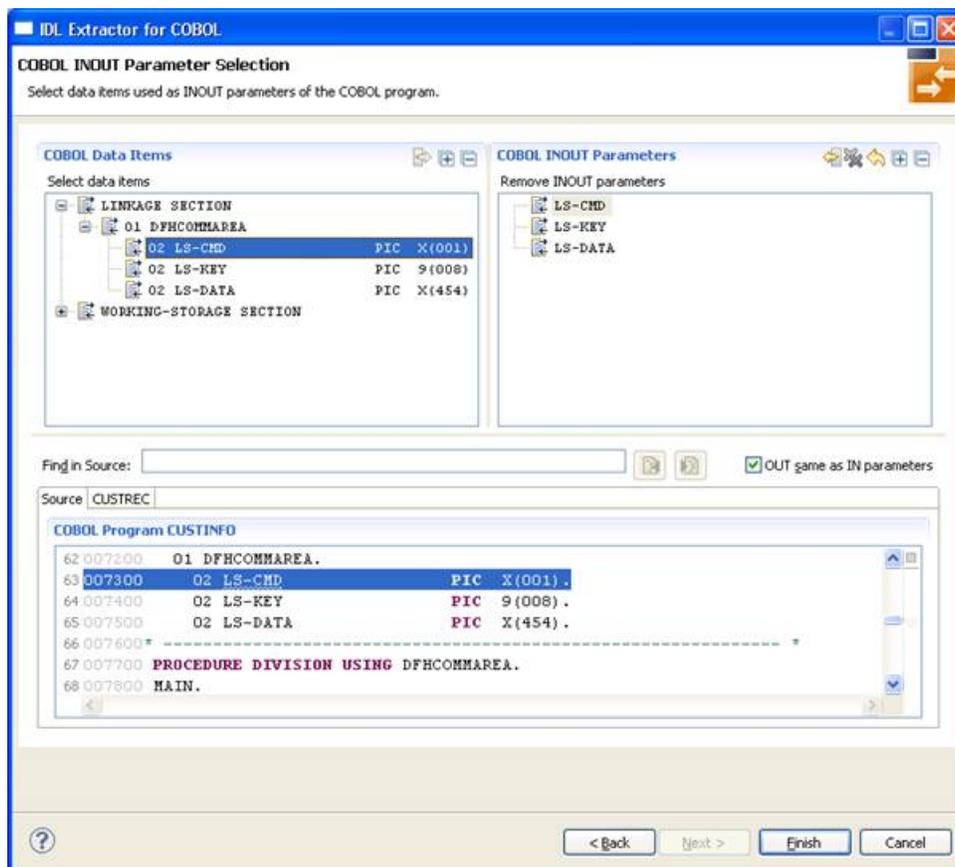
Selecting the COBOL parameters differs depending on the interface type and is described under *How to Select COBOL Parameters*.

Interface Type	Proceed with ...
CICS with `DFHCOMMAREA` calling convention	*CICS `DFHCOMMAREA`*
CICS with channel container calling convention	*CICS Channel Container*
CICS with `DFHCOMMAREA` large buffer interface	*CICS `DFHCOMMAREA` Large Buffer*
Batch with standard linkage calling convention	*Standard Linkage*
Micro Focus with standard linkage calling convention	*Standard Linkage*
IMS MPP message interface (IMS Connect)	*IMS MPP Message Interface (IMS Connect)*
IMS BMP with standard linkage calling convention	*IMS BMP Standard Linkage*

Step 6: Map the COBOL Interface to IDL with the Mapping Editor

With the Mapping Editor you map the COBOL server interface selected in *Step 5: Select the COBOL Parameters* to Software AG IDL. The outcome of the Mapping Editor is the IDL and an SVM file.
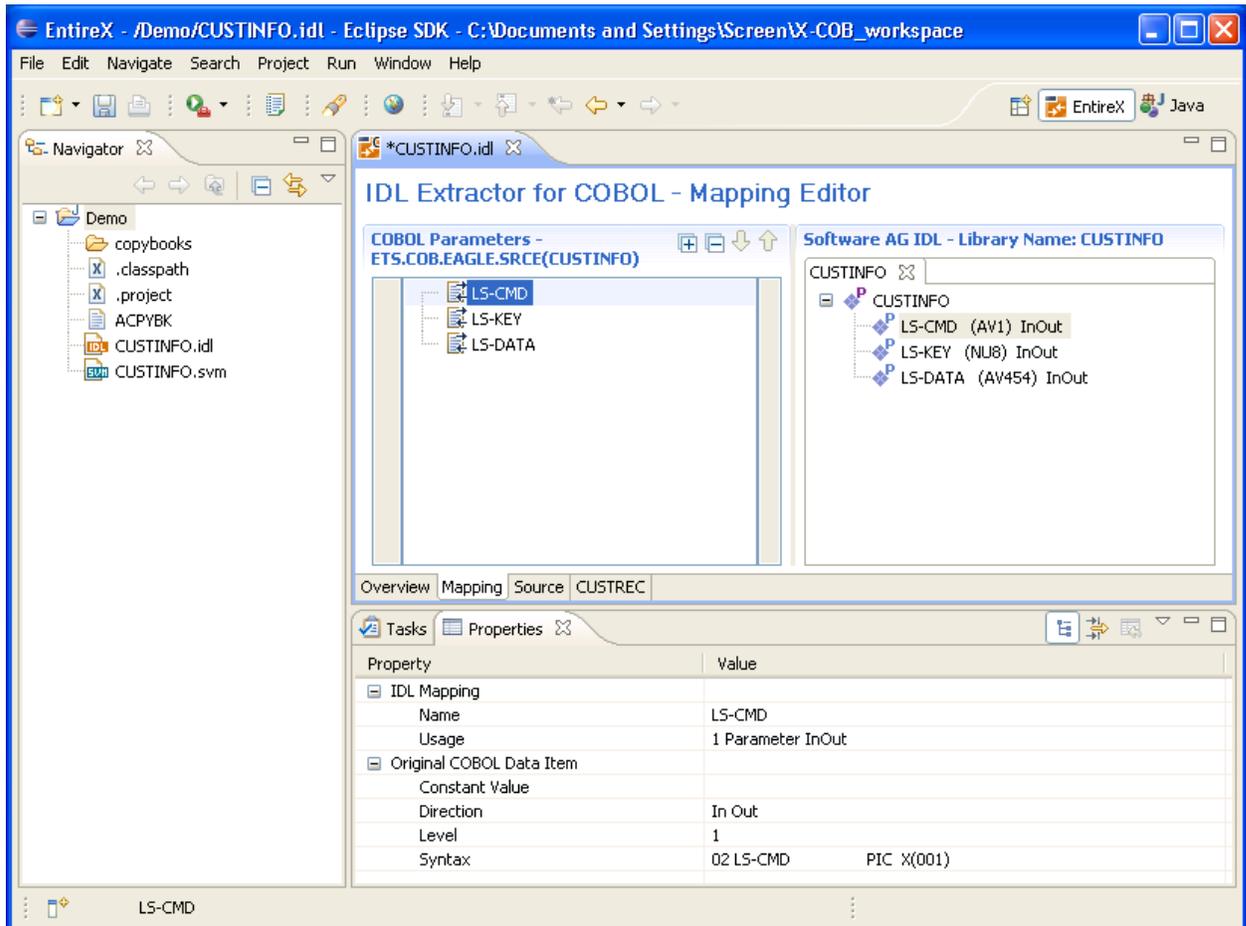
For more information on the COBOL mapping editor's user interface, its decision icons, mapping icons, context menus, toolbars and main panes, see *Mapping Editor User Interface*.

Before you start to define the mapping in the COBOL mapping editor, clarify whether your COBOL server implements multiple business functions controlled by an operation or function code field and you have to extract all the functions, or whether the COBOL server implements just a single function.

■ **Multiple Business Functions**
For COBOL servers with multiple business functions, the best approach is to extract them as multiple IDL programs. Start with *Mapping COBOL Functions to Multiple IDL Programs*. In this case, 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 class with a single method.

- etc.

- **Single Business Function**
  For COBOL servers with a single business function, start with *Mapping a COBOL Interface to an IDL Program*.

Mapping COBOL Functions to Multiple IDL Programs

Optional. COBOL level-88 data items can be mapped to operation using the **Map to operation** function from the context menu.

▶ **To map operations of the COBOL server to IDL programs**

1 Call the context menu of the COBOL level-88 data item of the operation or function code field you want to map to an IDL program.

2 Use the **Map to Operation** function for this purpose. The following rules apply:

   - Each map to operation results in an IDL program with its own tab.

   - By default, the name of the IDL program is the name of the level-88 data item. If required you can rename the IDL program, see the Software AG IDL tree pane. See *The Software AG IDL Tree Pane*.

   - Only one COBOL data item can be the **operation or function code** of your COBOL server, thus the related base data item of the level-88 data items mapped first to operation becomes the **operation or function code** field.

   - If the **operation or function code** field is decided by the first map to operation action, the map to operation functions are removed on all level-88 data items which do not relate to the **operation or function code** field.

   - You can still map all other level-88 siblings of the **operation or function code** to operation, resulting in IDL programs and all rules listed here.

   - The RPC server provides the level-88 constant value to your COBOL at runtime.

Perform all the other mappings for the COBOL function described under *Mapping a COBOL Interface to an IDL Program*.

Mapping a COBOL Interface to an IDL Program

The following actions are required in the Mapping Editor if your COBOL server met the conditions:

- If your COBL server contains COBOL `REDEFINES`, you need to select `REDEFINE` paths. See *Selecting REDEFINE Paths*.

- Provide IDL directions (see *Providing IDL Directions (`IN OUT INOUT`)*). This step can be skipped for CICS servers that overlay the `IN` parameters with different `OUT` parameters. See `DFHCOMMAREA` examples *1*, *2*, *3*. In this case the check box **OUT same as IN** in *Step 5: Select the COBOL Parameters* was cleared and you selected the `OUT` parameters separately.

The following mapping editor actions are optional. They remove uneeded COBOL parameters during the mapping process, which simplifies the extracted IDL. See the following sections:

- *Suppressing or Hiding Unneeded Fields of the COBOL Server*
- *Mapping COBOL Data Items to Constants*

Providing IDL Directions (IN OUT INOUT)

A very important task in the mapping editor is to choose the correct IDL directions (see `direction-attribute` in `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation) of the COBOL parameters, because in COBOL programs there is no parameter direction. You have to do this in the mapping editor because it is not possible to do this after extraction by simply editing the IDL file extracted - the related SVM file would no longer match the IDL. For CICS servers that overlay the `IN` parameters with different `OUT` parameters, it is not possible to provide IDL directions manually - there is only one IDL direction offered. It is also not necessary, because the correct IDL directions are already implicit in step *Step 5: Select the COBOL Parameters*.

▶ **To provide IDL directions**

- Go *step-by-step* through all top level COBOL data items and use the **Map to IN, OUT and INOUT** functions available in the *Context Menu* of the COBOL **Parameters** pane to provide the IDL directions.

Please note the following:

- We recommend you define IDL directions at the *top-level* COBOL data items. For CICS COBOL servers, the *top-level* COBOL data items are the COBOL data items directly under `DFHCOMMAREA`. For all other COBOL servers, the *top-level* COBOL data items are defined with COBOL level 1.

- On *top-level* COBOL data items, IDL directions can always be set with the **Map to IN, OUT or INOUT** functions. If set on a *top-level* COBOL group, the direction is inherited by all subsequent child COBOL data items.

- On *higher-level* COBOL data items, the IDL direction can be set to the same direction as the *top-level* COBOL data item - it is not permissible to set a conflicting direction.

- Go *step-by-step* through all top level COBOL data items and use the **Map to IN, OUT and INOUT** functions available in the *Context Menu* of the COBOL **Parameters** pane to provide the IDL directions.

Selecting REDEFINE Paths

If your COBOL server contains COBOL`REDEFINES`, a required and very important task in the mapping editor is to choose the correct `REDEFINE` paths of a COBOL `REDEFINE` data item for your mapping.

▶ **To select redefine paths**

1 Use the **Map to IN, OUT or INOUT** functions of the *Context Menu* for this purpose. The following rules apply:

- Only one `REDEFINE` path of a COBOL `REDEFINE` can be mapped to IDL. All other siblings are suppressed.

- If a `REDEFINE` path is actively mapped to `IN`, `OUT` or `INOUT` in the IDL with the context menu functions, all siblings are mapped automatically to "suppress" by the mapping editor.

- You can suppress all `REDEFINE` paths of a COBOL `REDEFINE`. Simply suppress the active `REDEFINE` path, using the suppress function from the context menu.

2 Select the correct redefine paths beginning with a COBOL `REDEFINE` defined at the top-level first. For a CICS COBOL server, the top-level COBOL data items are the COBOL data items directly under `DFHCOMMAREA`. For all other COBOL servers, the top-level COBOL data items are defined with COBOL level 1.

3 Work through all inner COBOL `REDEFINE` data items, going from higher levels to lower levels.

4 We recommend using the **Up** and **Down** buttons of the *Toolbar* to locate the COBOL `REDEFINE` data items in the right order.

Suppressing or Hiding Unneeded Fields of the COBOL Server

Optional. With the mapping editor you can make COBOL data items of the COBOL server interface invisible for IDL clients. This means the fields are not mapped to IDL, which makes sense for COBOL data items that do not have any meaning, for example `FILLER`.

▶ **To suppress or hide unneeded fields**

- Use the **Map to suppress** function from the *Context Menu* to suppress the COBOL *scalar* data items. The following rules apply:

- The COBOL data item is not part of the IDL.

- The RPC server provides the COBOL data item to your COBOL server with low value, managing the offset to the next COBOL data item.

COBOL *groups* can be suppressed as well, which means that all subsequent child COBOL data items are suppressed.

Mapping COBOL Data Items to Constants

Optional. With the mapping editor you can provide fixed values to COBOL data items of the COBOL server at runtime. In this case, the COBOL data items are also invisible for IDL clients, which means the fields are not mapped to IDL. This feature keeps the IDL short and tidy: IDL clients are not bothered by fields that always contain constants, such as RECORD-TYPES.

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

- Use the **Map to constant** function from the *Context Menu* to map COBOL *level-88* data items to constants. The following rules apply:

  - The related level-88 base variable is not part of the IDL.

  - Only one COBOL level-88 data item can be selected per level-88 base data item as a constant. All COBOL level-88 siblings are set automatically to "suppress" by the mapping editor

  - The RPC server provides the constant in the related level-88 base variable to your COBOL server.

  COBOL scalar data items can be mapped to constants as well. The following rules apply here:

  - The COBOL scalar data item is not part of the IDL.

  - You are prompted with a window to enter the constant value.

  - The RPC server provides the constant in the COBOL scalar data item to your COBOL server during runtime.

Step 7: Finishing the Mapping Editor

When you choose **Save** in the Mapping Editor, the IDL and SVM files are generated. Both files are written with the "File Name" entered for the IDL file in the IDL Extractor for COBOL wizard. See *Step 4: Define the Extraction Settings and Start Extraction*. The extension used for the IDL file is *.idl and *.svm for the SVM file.

The following dialog is displayed whenever the Mapping Editor is saved. There are two options to choose from:

■ **Save Software AG IDL and SVM files** will save the generated files into the workspace and quit the Mapping Editor. Additionally, the generated SVM file can be deployed if the option **Deploy SVM** is checked.

■ **Extract additional COBOL program and append to the IDL and SVM 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 SVM files.
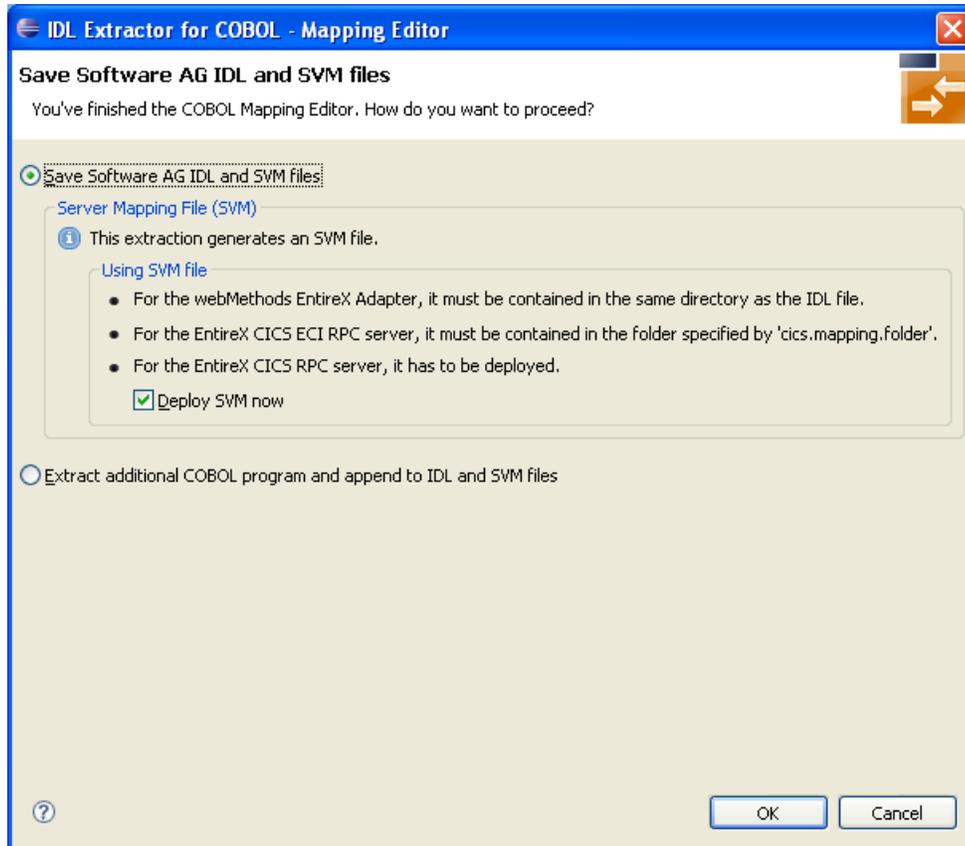
▶ **To save the generated files into the workspace, quit the Mapping Editor and deploy the SVM file**

1 Select **Save Software AG IDL and SVM files**.

2 Check the option **Deploy SVM** and choose **OK**. Continue with *Step 8: Deploy the SVM File (Optional)*.

▶ **To save the generated files into the workspace and quit the Mapping Editor without deploying the SVM file**

1 Select **Save Software AG IDL and SVM files**.

2 Clear the option **Deploy SVM** and choose **OK**.

▶ **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 SVM 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 SVM file will be lost, resulting in unexpected behavior.

⛔ **Caution:** An SVM 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 existing COBOL server.

Step 8: Deploy the SVM File (Optional)

If an SVM file is required, it has to be provided. For RPC servers this is done by deployment, either by using a wizard or by a manual approach specific to the RPC server used, see *Server Mapping Deployment*. For the webMethods EntireX Adapter, it is picked up automatically together with the IDL file when the adapter connection is generated. For further information, see the latest version of the EntireX Adapter under *https://empower.softwareag.com/Products/Documentation/default.asp*.

Step 9: 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.

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 SVM

Step 1: Start the COBOL Mapping Editor

The IDL Extractor for COBOL can be started from an existing pair of IDL and SVM files.

▶ **To open the COBOL Mapping Editor**

■ Open the context menu of an IDL file and choose **Edit COBOL Mapping of Software AG IDL**.

Or:
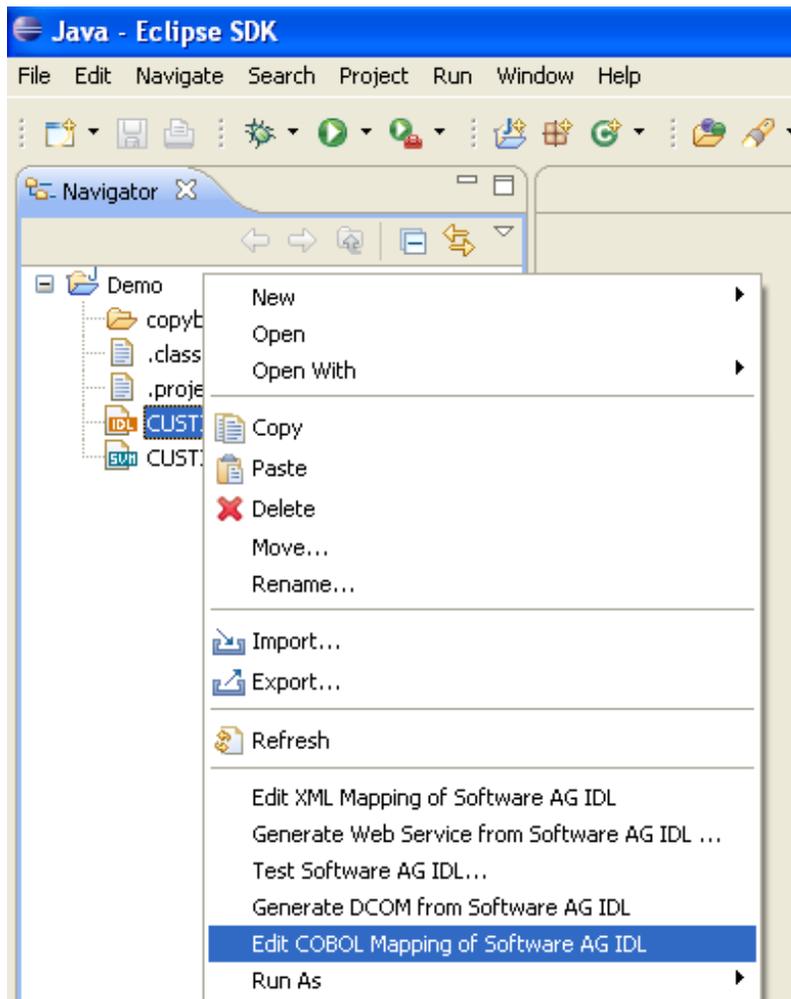
Choose **Open With > EntireX COBOL Mapping Editor**.
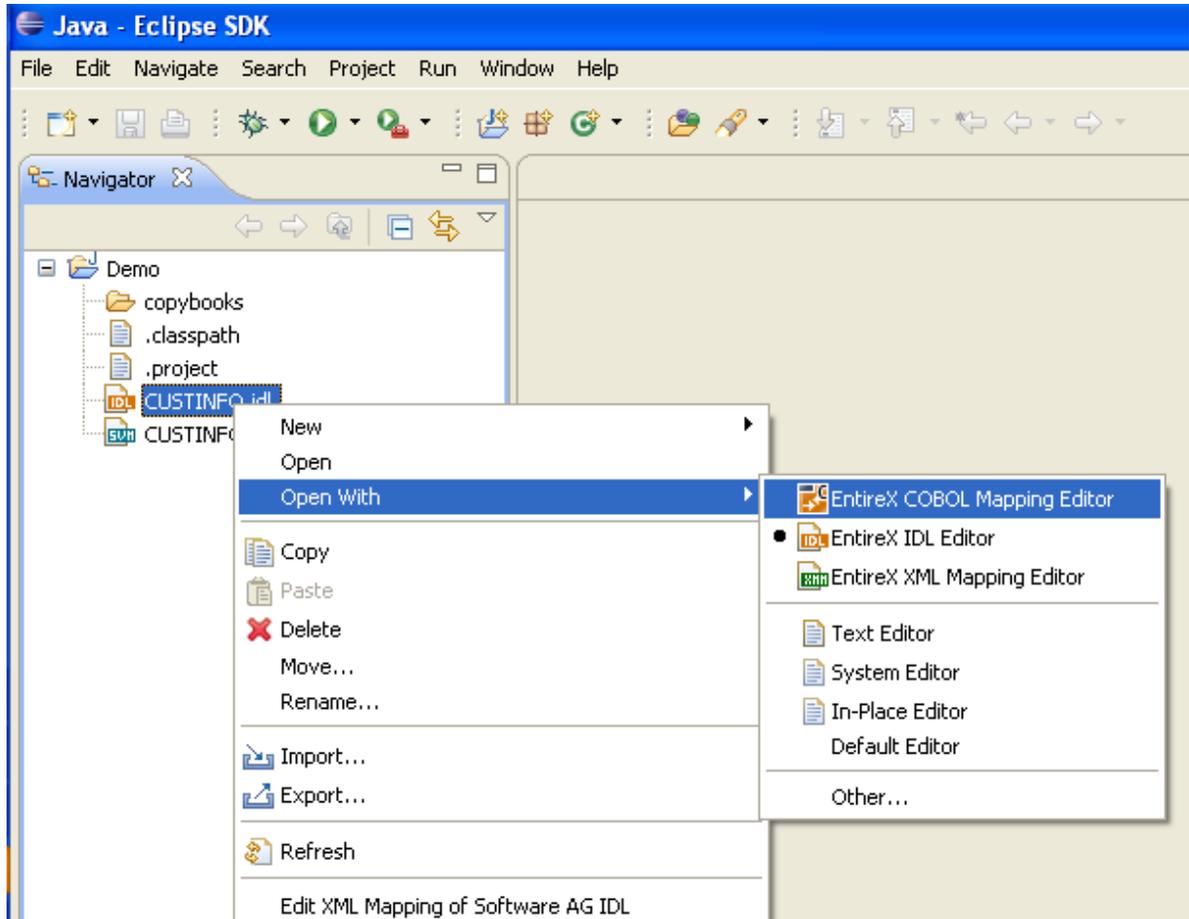
Continue with *Step 2: Select to Extract Additional COBOL Program* below.

Step 2: Select to Extract Additional COBOL Program

The following dialog appears. There are two options to choose from:

▪ Extract additional COBOL program and append to IDL and SVM files

▪ Modify existing IDL and SVM files
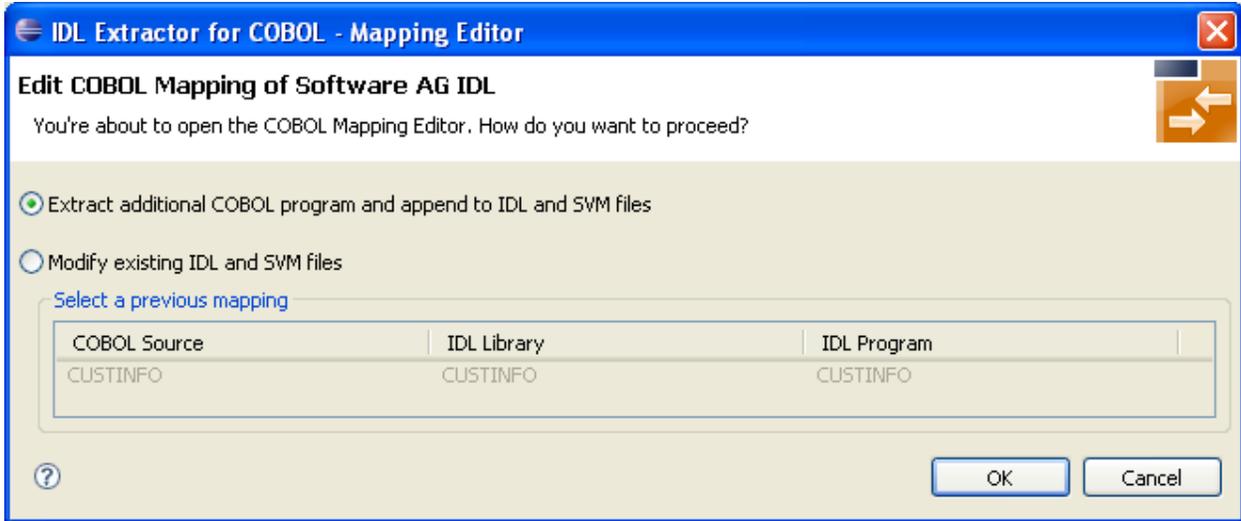
▶ **To start IDL Extractor for COBOL for extracting further COBOL programs**

■ Check the option **Extract additional COBOL program and append to IDL and SVM files** and choose **OK**.

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

4 Scenario III: Modify Existing IDL and SVM

Step 1: Start the COBOL Mapping Editor

The IDL Extractor for COBOL can be started from an existing pair of IDL and SVM files.

▶ **To open the COBOL Mapping Editor**

■ Open the context menu of an IDL file and choose **Edit COBOL Mapping of Software AG IDL**.

Or:
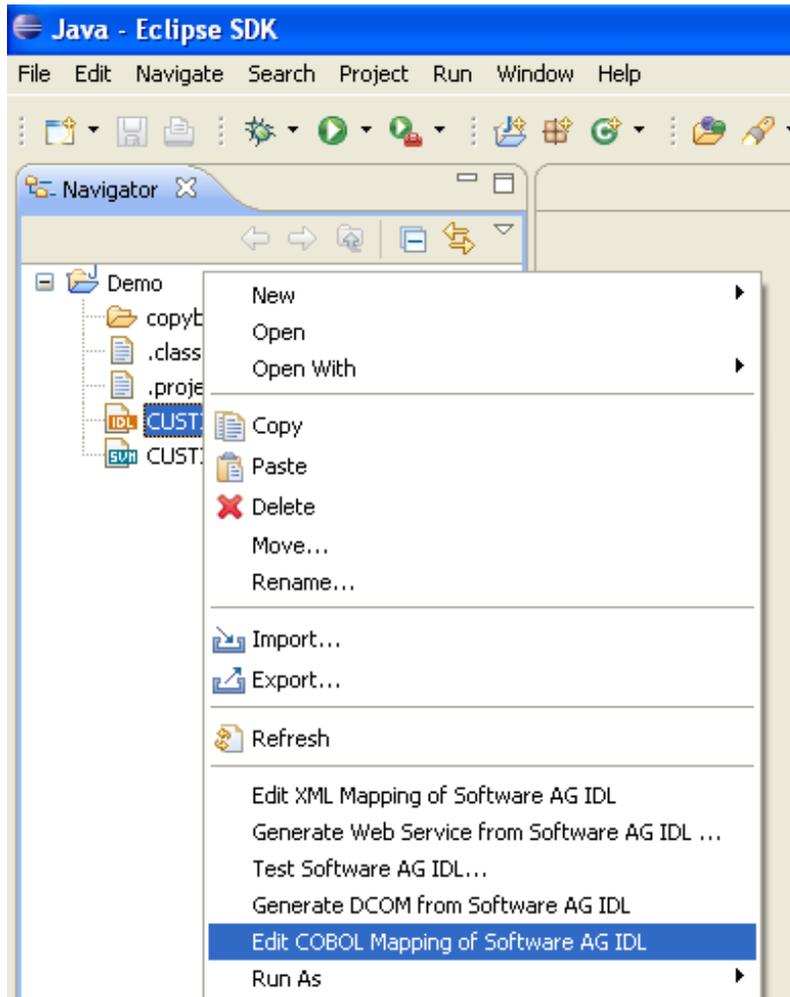
Choose **Open With > EntireX COBOL Mapping Editor**.
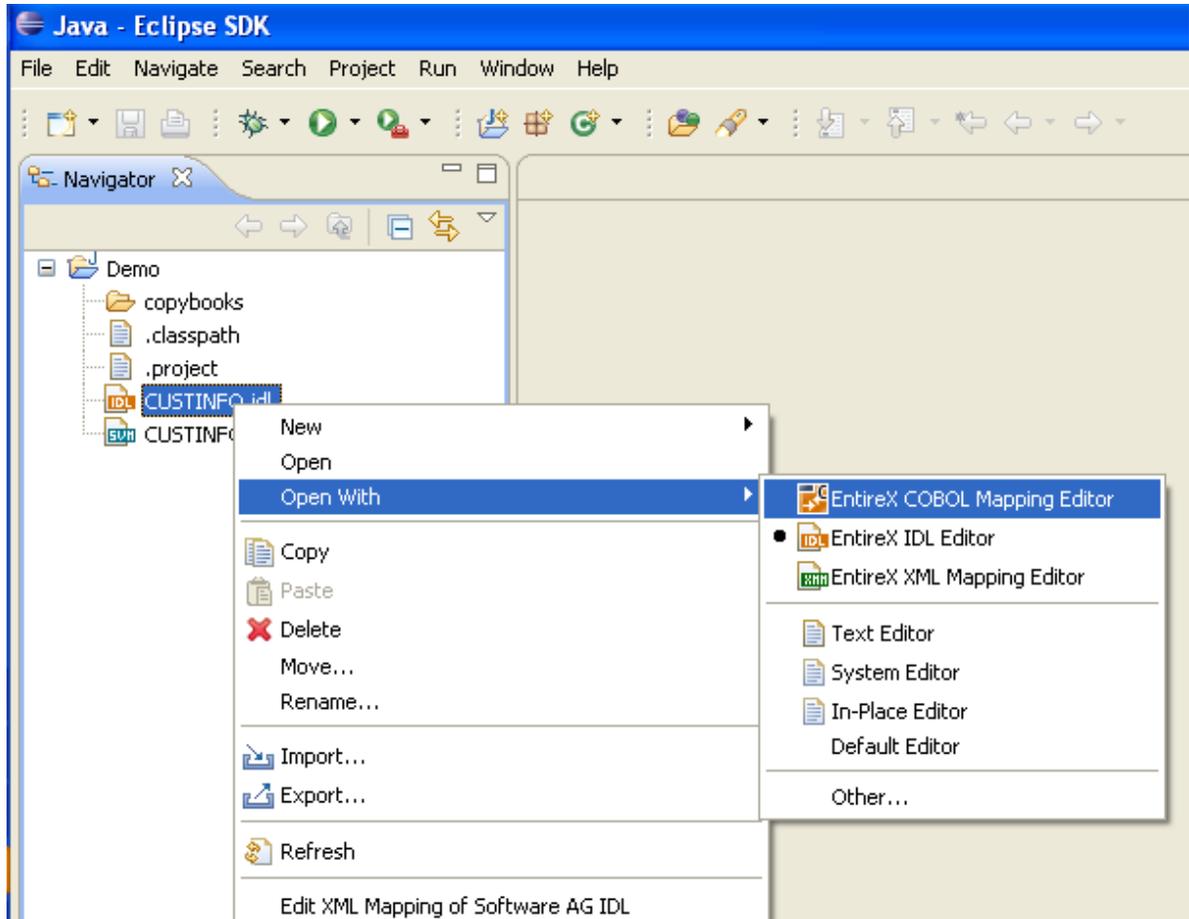
Continue with *Step 2: Select to Modify Existing IDL and SVM Files* below.

Step 2: Select to Modify Existing IDL and SVM Files

The following dialog appears. There are two options to choose from:

- Extract additional COBOL program and append to IDL and SVM files
- Modify existing IDL and SVM files
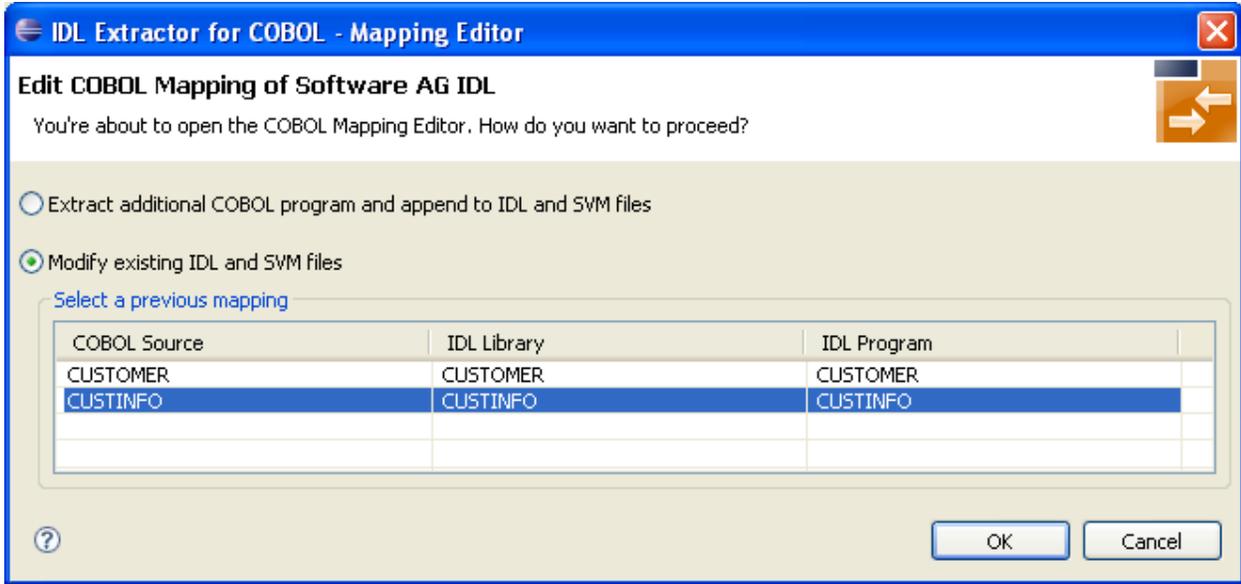
▶ **To open the COBOL Mapping Editor and modify existing IDL and SVM files**

1   Check the option **Modify existing IDL and SVM files**.

2   Select a previous mapping from the table and choose **OK**.

Continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor* as described under *Scenario I: Create New IDL and SVM*.

III

5 COBOL Parameter Selection

This chapter describes the *COBOL Parameter Selection User Interface*, its toolbars, context menus and panes. It also describes briefly the purpose of the *COBOL Parameter Selection*. It also provides background information on parameter selection.

COBOL Parameter Selection Purpose

The purpose of the COBOL parameter selection page is to select the COBOL data items that match the COBOL server interface. This allows you to perform the following tasks:

- Extract from a COBOL CICS server. The interface definition `DFHCOMMAREA` or channel container in many cases has to be selected manually because of various different programming styles and techniques for accessing the CICS interfaces.

- Extract from COBOL IMS MPP message interface (IMS Connect) server where the interface definition is within the working storage section and not given by the parameters provided in the `PROCEDURE DIVISION` header.

- Manually check and correct the COBOL server parameter suggestion given by the extractor wizard. Take an extra look at `REDEFINE` data items as parameters (top-level) for the following reasons:

  - Because the wizard always suggests the first `REDFINE` path. In many cases this will not be the `REDEFINE` path you want, so you can manually select a different one.

  - If your COBOL server implements multiple business functions and you want to map them to Web service operations or methods using a wrapper with a class/method concept, such as the *Java Wrapper* or *.NET Wrapper*, the complete `REDEFINE` data item including all `REDEFINE` paths can be selected. Later in the COBOL Mapping Editor, use the feature described under *Mapping COBOL Functions to Multiple IDL Programs* to design your IDL.

This page is used in *Step 5: Select the COBOL Parameters* of *Scenario I: Create New IDL and SVM*.

COBOL Parameter Selection User Interface

The following page is provided for selecting the COBOL data items as parameters of your COBOL server:
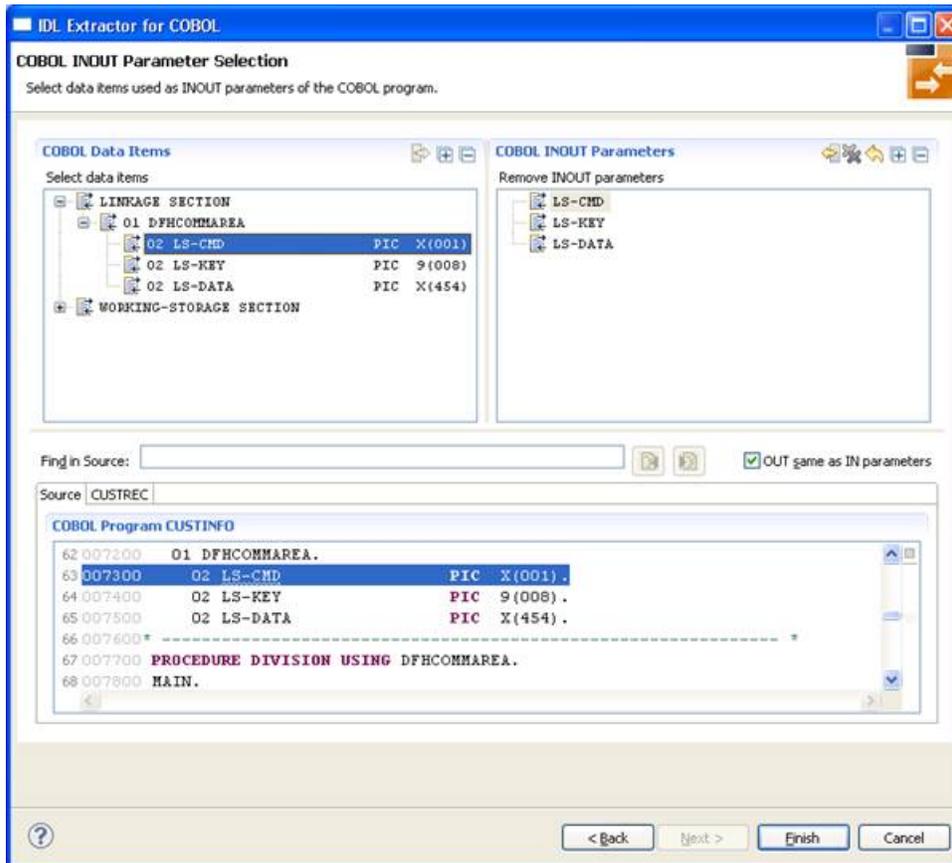
The page consists of the following main panes:

- COBOL Data Items
- COBOL INOUT Parameters
- OUT Same as IN Parameters
- Find in Source

- COBOL Source

COBOL Data Items

In the **COBOL Data Items** pane, all COBOL data items contained in the `LINKAGE` and `WORKING-STORAGE SECTION` are offered in a tree view. Redefine units (all redefine paths addressing the same storage location) are marked with special icons (). By default, the `LINKAGE SECTION` is expanded and the `WORKING-STORAGE SECTION` is not.

Context Menu

A *context menu* is available on the node items of the **COBOL Data Items** tree for selecting parameters. See toolbar description below for explanation.

Toolbar

The *toolbar* in the **COBOL Data Items** pane provides the following actions:

Action	Description
	Select one or more unselected items and move them to the right section. The same action is provided from the context menu and with a drag-and-drop operation.
	Expand the full tree.
	Collapse the full tree.

COBOL INOUT Parameters

In the **COBOL INOUT Parameters** pane, the currently selected parameters are presented in a tree view, representing the interface of your COBOL program.

Context Menu

A context menu is available on the node items of the **COBOL INOUT Parameters** tree for deselecting parameters. See toolbar description below for explanation.

Toolbar

The toolbar in the **COBOL INOUT Parameters** pane provides the following actions:

Action	Description	Restrictions
	Deselect one or more items and remove them from the right section. The same action is provided from the context menu and by a drag-and-drop operation.	Deselection is only allowed on the first level, except of interface type CICS, where items on the level following the DFHCOMMAREA level can be selected.
	Change the parameter order: move the selected item up.	Only for Interface types BATCH and IMS BMP.
	Change the parameter order: move the selected item down.	Only for Interface types BATCH and IMS BMP.
	Deselect all items and clear the section.	
	Restore the default selection.	Only on the INOUT or IN parameter selection page.
	Expand the full tree.	
	Collapse the full tree.	

OUT Same as IN Parameters

The **OUT same as IN parameters** check box is available for interface types *CICS with DFHCOMMAREA Calling Convention* and *CICS with DFHCOMMAREA Large Buffer Interface*, and allows you to extract from a COBOL server with the same COBOL data structures on input and output, or with different data structures on input and output (i.e. the output overlays the input). See DFHCOMMAREA examples *1*, *2*, *3* and the *DFHCOMMAREA Example 2* for CICS COBOL server with different data structures on input and output.

Find in Source

With the **Find in Source** feature you can search in the COBOL source and locate text. This is useful if there is a large linking or working storage section and if you know the name of the COBOL data item you are looking for at least in parts.

COBOL Source

The **COBOL source** pane in the lower part of the page is headed by a list of (selectable) tabs. If **Source** is selected, the source code you extracted is displayed. Additional tabs are related to the copybooks, CA Librarian (`-INC` control statement) or CA Panvalet (`++INCLUDE` control statement) members that are referenced in your sources. This way, you can easily switch between the source, the copybooks and members.

> 📄 **Notes:**

1. There are no additional tabs if there are no references to copybooks, CA Librarian (`-INC` control statement) or CA Panvalet (`++INCLUDE` control statement) members.

2. Each single-parameter selection done in the **COBOL Data Items** pane is highlighted in the corresponding source or copybook. This makes it easier to monitor the interface composition.

It is possible to resize the page and also to change the size of all these panes within the lines between the sections to handle large COBOL sources. Selections in the tree will reflect selections in the opposite tree and always in the COBOL source where you find all references to the selected parameter.

How to Select COBOL Parameters

Selecting the COBOL parameters differs depending on the interface type:

Interface Type	Proceed with ...
CICS with `DFHCOMMAREA` calling convention	*CICS DFHCOMMAREA*
CICS with channel container calling convention	*CICS Channel Container*
CICS with `DFHCOMMAREA` large buffer interface	*CICS DFHCOMMAREA Large Buffer*
Batch with standard linkage calling convention	*Standard Linkage*
Micro Focus with standard linkage calling convention	*Standard Linkage*
IMS MPP message interface (IMS Connect)	*IMS MPP Message Interface (IMS Connect)*
IMS BMP with standard linkage calling convention	*IMS BMP Standard Linkage*

CICS DFHCOMMAREA

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.

▶ **To extract from a CICS** `DFHCOMMAREA` **program**

1. Clarify whether the `OUT` and `IN` parameters of your COBOL servers are the same or different. Different `OUT` and `IN` parameters mean that parameter formats (usage clauses) for the input differ from those of the output. The following picture illustrates this:
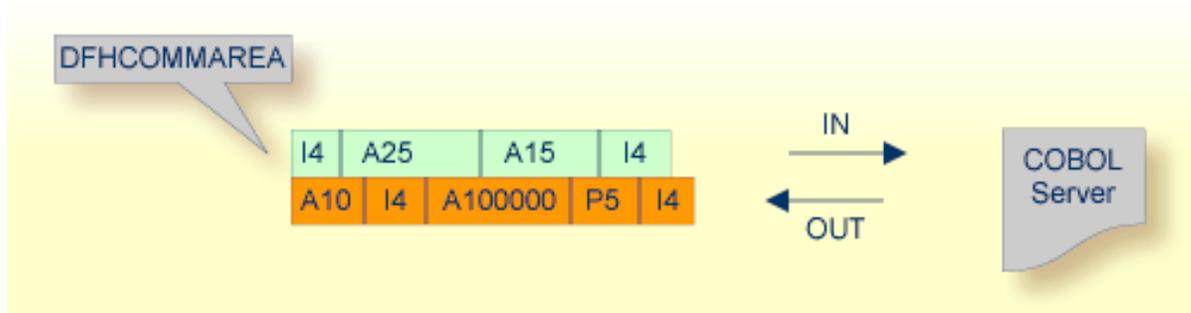
There are various possibilities for programming/coding such a COBOL server with different `OUT` and `IN` parameters, for example:

- `REDEFINES`
  The output data is described with a `REDEFINE` that overlays the input data as in the following example:

`DFHCOMMAREA` **Example 1**

```
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.
```

■ **a 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 parameters are described within the working storage section.

`DFHCOMMAREA` **Example 2**

```
LINKAGE SECTION.
01 DFHCOMMAREA.
   02 IO-BUFFER                      PIC X(9).
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.
   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.
```

■ **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.

DFHCOMMAREA **Example 3**

```
 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.
```

If your COBOL server has the

- **same** OUT **and** IN **parameters**
  check "OUT same as IN" (if not already checked). In this case, IDL directions are defined
  later in the mapping editor and not here. See *Providing IDL Directions (IN OUT INOUT)*.

- **different** OUT **and** IN **parameters**
  that is, the DFHCOMMAREA on output is overlayed with a data structure as illustrated in the
  examples above. Clear "OUT same as IN" (if not already cleared). As a result, the screen
  title **INOUT Parameters** is changed to **IN Parameters** and the **Next** button becomes enabled.
  In this case, IDL directions are implicitly defined here, and it is not possible to set them in
  the mapping editor. Proceed with step 4 below.

2   Select the INOUT parameters (of your COBOL server) by using the context menu or toolbars
    available in **COBOL Data Items** and **COBOL INOUT Parameters**. Even if the data structure
    remains the same for input and output, the techniques explained for the DFHCOMMAREA examples
    1, *2*, *3* above and others can be used to address an in-out structure. See the notes below.

3   Choose Finish and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping
    Editor*.

4   Select the IN parameters in correct sequence and number by using the context menu or toolbars
    available in **COBOL Data Items and COBOL IN Parameter**. Consider the techniques explained
    for the DFHCOMMAREA examples *1*, *2*, *3* above and others to address the input structure. See also
    the notes below.

5   Choose **Next**.

6   Select the OUT parameters of your COBOL server by using the context menu and toolbars
    available in the **COBOL Data Items and COBOL OUT Parameter** pane. Consider the tech-
    niques explained for the DFHCOMMAREA examples *1*, *2*, *3* above to address the output structure.
    See also the notes below.

7   Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

Notes

■ Your `DFHCOMMAREA` COBOL server must be DPL-enabled to be directly supported by EntireX. 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*.

■ Only level-one parameters can be selected, or if a `DFHCOMMAREA` is present, the parameters directly below the `DFHCOMMAREA`.

■ 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

   ■ sequence of formats (COBOL usage clause).

■ If your COBOL server contains `REDEFINE`s as top-level parameters (redefines on lower levels are mapped in the mapping editor, see *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*), the first `REDEFINE` path is offered by default. Check manually whether this is the one you want. If not, correct it. You can select here:

   ■ any other `REDEFINE` path

   ■ the complete `REDEFINE` data item, consisting of all `REDEFINE` paths. This is useful if your COBOL server implements multiple business functions and you want to map them to Web service operations or methods using a wrapper with a class/method concept, such as the *Java Wrapper* or *.NET Wrapper*. For this purpose, select the complete `REDEFINE` data item including all `REDEFINE` paths. Later in the COBOL Mapping Editor, use the feature described under *Mapping COBOL Functions to Multiple IDL Programs* to design your IDL.

CICS Channel Container

Modern CICS program 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.

The page for selecting the COBOL data items that describe the container layouts contains a table on the right that provides an overview of the selected containers and their names.
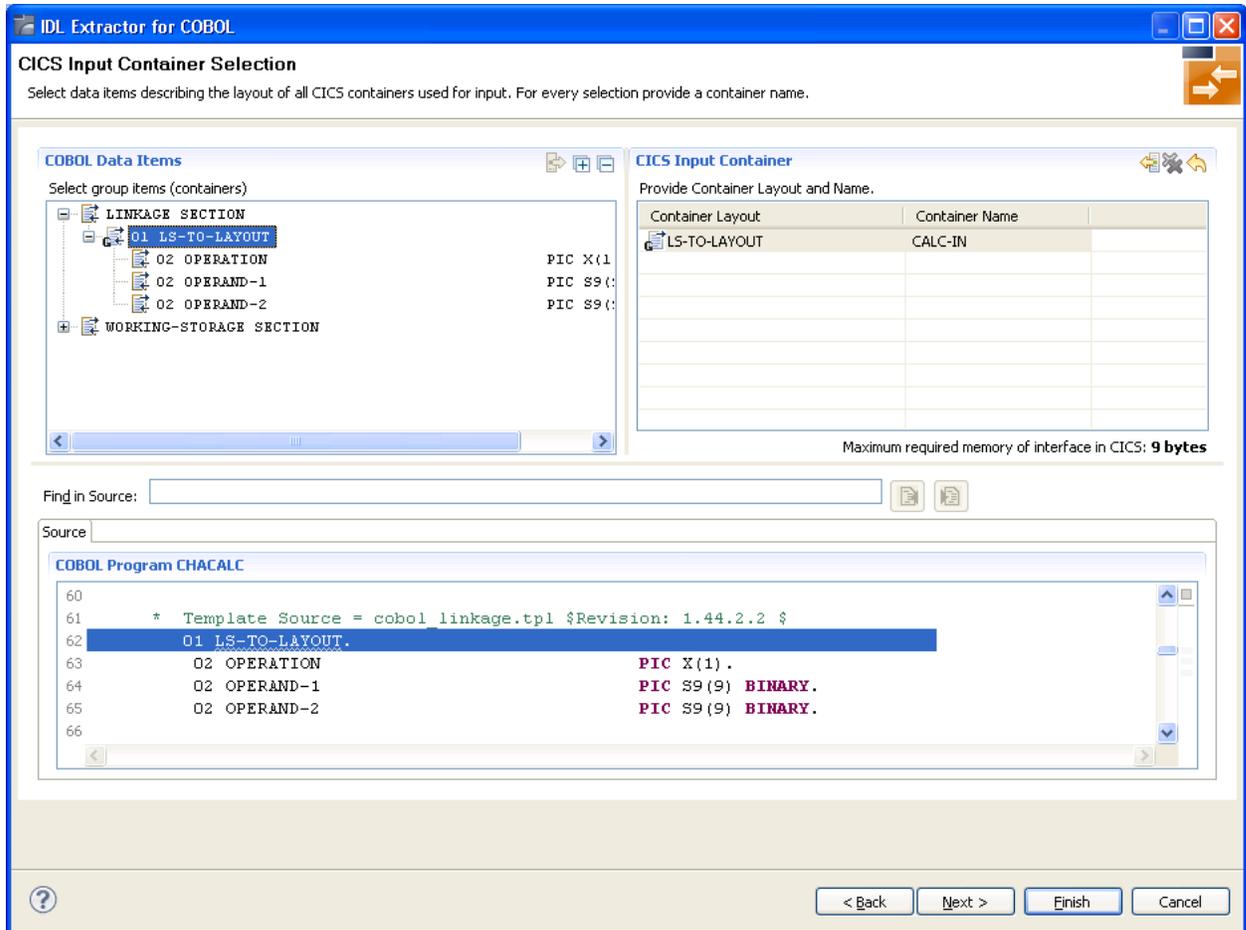
▶ **To extract from a CICS Channel Container program**

1   Define the CICS Input Container by selecting the Source tab and entering "`EXEC CICS`" in **Find in Source** to look for a `GET` call containing "`EXEC CICS GET`", function "`CONTAINER`" and the "`CHANNEL`", example:

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

See *Find in Source*.

Select the corresponding `<container>` in **COBOL Data Items**. Enter the container name, found in the value of `<container name constant>`. You can select multiple CICS input containers.

2   Choose **Next**.

3   Define the CICS Output Container 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
```

Select the corresponding `<container>` in **COBOL Data Items**. 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.

4   Choose Finish and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

> **Notes:**

1. The container name length is restricted to 16 characters.

2. Arrays (direction `OUT`) will enlarge the container name, because the number of occurrence will be added to the name (max. 16 characters), for example MYCONTAINER00001.

CICS DFHCOMMAREA Large Buffer

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.

Large Buffer Example 1

```
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.
```

▶ **To extract from a CICS** `DFHCOMMAREA` **Large Buffer program**

1   Clarify whether the `OUT` and `IN` parameters of your COBOL server are the same or different. Different `OUT` and `IN` parameters mean that parameter formats (usage clauses) for the input differ from those of the output. The following picture illustrates this:
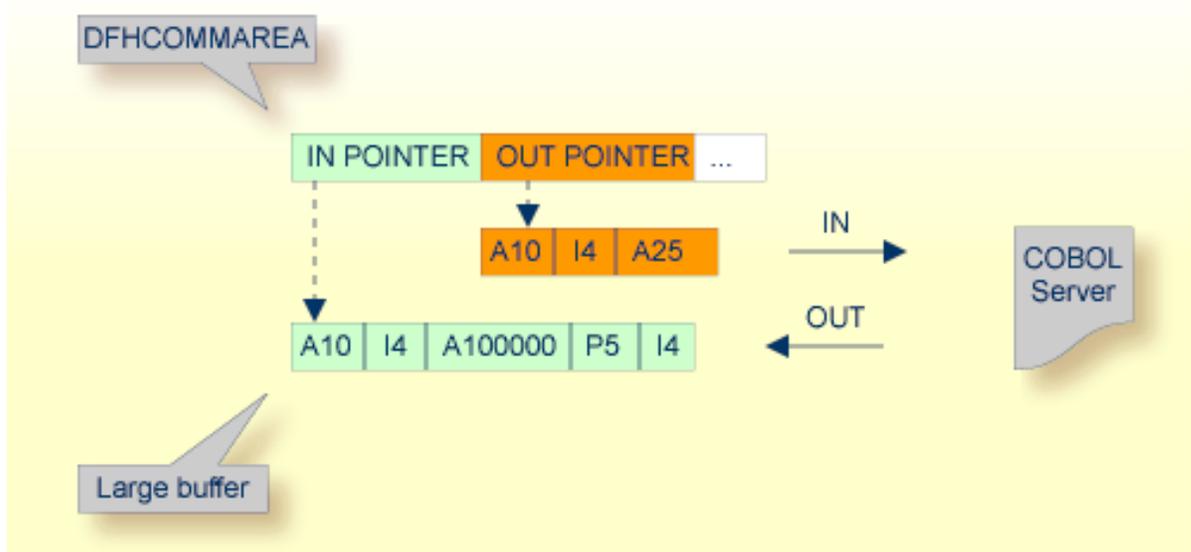
There are various possibilities for programming/coding such a COBOL server with different `OUT` and `IN` parameters. Example 2 below illustrates one of the possibilities:

Large Buffer Example 2

```
 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 your COBOL server has

- **the same** OUT **and** IN **parameters**
  that is, WM-LCB-OUTPUT-BUFFER is set to the same address as WM-LCB-INPUT-BUFFER (as in the DFHCOMMAREA large buffer example *1* above):

  Check **OUT same as IN** (if not already checked). In this case, IDL directories are defined later in the mapping editor and not here. See *Providing IDL Directions (IN OUT INOUT)*. Proceed with step 2 below.

- **different** OUT **and** IN **parameters**
  that is, the WM-LCB-OUTPUT-BUFFER (as in the large buffer example *2* above) is set to a different address from WM-LCB-INPUT-BUFFER:

  Clear **OUT same as IN** (if not already cleared). As a result, the screen title **INOUT Parameters** is changed to **IN Parameters** and the **Next** button becomes enabled. In this case, IDL directions are defined implicitly here and it is not possible to set them in the mapping editor. Proceed with step 4 below.

2 Select the INOUT parameters (of your COBOL server) by using the context menu or toolbars available in the **COBOL Data Items and COBOL INOUT Parameters** pane. To do this, locate in the PROCEDURE DIVISION the SET ADDRESS OF <x> TO <y> statements, where <y> is WM-LCB-INPUT-BUFFER and WM-LCB-OUTPUT-BUFFER. Both statements should point to the same <x> as in the example above. The COBOL data item <x> - INOUT-BUFFER in our example - is the INOUT parameter you are looking for. Consider the notes below.

3 Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

4 Select the IN parameters (of your COBOL server) by using the context menu or toolbars available in **COBOL Data Items and COBOL IN Parameter**. To do this, locate in the PROCEDURE DIVISION the SET ADDRESS OF <x> TO <y> statement, where <y> is WM-LCB-INPUT-BUFFER. The COBOL data item <x> is the IN parameter you are looking for. Consider the notes below.

5 Choose **Next**.

6   Select the `OUT` parameters (of your COBOL server) by using the context menu and toolbars available in the **COBOL Data Items and COBOL OUT Parameter** pane. To do this, locate in the `PROCEDURE DIVISION` the `SET ADDRESS OF <x> TO <y>` statement, where `<y>` is `WM-LCB-OUTPUT-BUFFER`. The COBOL data item `<x>` is the `OUT` parameter you are looking for. Consider the notes below.

7   Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

Notes

- Only level-one parameters can be selected.

- 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`.

- 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

  - sequence of formats (COBOL usage clause)

- If your COBOL server contain `REDEFINE`s as top-level parameters (redefines on lower levels are mapped in the mapping editor, see *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*), the first `REDEFINE` path is offered by default. Check manually whether this is the one you want. If not, correct it. You can select here:

  - any other `REDEFINE` path

  - the complete `REDEFINE` data item, consisting of all `REDEFINE` paths. This is useful if your COBOL server implements multiple business functions, and you want to map them to Web service operations or methods using a wrapper with a class/method concept, such as the *Java Wrapper* or *.NET Wrapper* For this purpose, select the complete `REDEFINE` data item including all `REDEFINE` paths. Later in the COBOL Mapping Editor, use the feature described under *Mapping COBOL Functions to Multiple IDL Programs* to design your IDL.

Standard Linkage

Because COBOL servers with standard linkage always contain a `PROCEDURE DIVISION` header (see *`PROCEDURE DIVISION` Mapping*) with all parameters, the COBOL parameters can be evaluated by the IDL Extractor for COBOL and are already offered by the wizard. In most cases the offered COBOL parameters will be correct, but you should always check them manually.

▶ **To extract from a Standard Linkage program**

1   Select the `INOUT` parameters (of your COBOL server) by using the *Context Menu* or *Toolbar* available in the **COBOL Data Items** and **COBOL INOUT Parameters** pane.

    Consider the following:

- Only level-one parameters can be selected.

- 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 (if the number of parameters does not match the number of `INOUT` parameters defined in the list of `PROCEDURE DIVISION USING`, an error message will be displayed without giving the chance to finish the dialog)

  - sequence of formats (COBOL usage clause)

- If your COBOL server contain `REDEFINE`s as top-level parameters (redefines on lower levels are mapped in the mapping editor, see *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*), the first `REDEFINE` path is offered by default. Check manually whether this is the one you want. If not, correct it. You can select here:

  - any other `REDEFINE` path

  - the complete `REDEFINE` data item, consisting of all `REDEFINE` paths.This is useful if your COBOL server implements multiple business functions and you want to map them to Web service operations or methods using a wrapper with a class/method concept, such as the *Java Wrapper* or *.NET Wrapper*. For this purpose, select the complete `REDEFINE` data item including all `REDEFINE` paths. Later in the COBOL Mapping Editor, use the feature described under *Mapping COBOL Functions to Multiple IDL Programs* to design your IDL.

2   Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

IMS MPP Message Interface (IMS Connect)

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:

- Instead of the IOPCB pointer, CALL 'CBLTDLI' statements are also used with database PCB pointers to access IMS databases.

- 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).
    . . .
```

▶ **To extract from an IMS MPP message interface (IMS Connect) program**

1   In **Find 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>
```

See *Find in Source*.

Select the corresponding <*input-message*> in **COBOL Data Items**. See also *IMS MPP Message Interface (IMS Connect)*.

2   In **COBOL Data Items**, select the IN parameters (of your COBOL server) by using the *Context Menu* or *Toolbar* available. They 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 the notes below.

3   Choose **Next**.

4   Similar to step 1, enter "CBLTDLI" in *Find 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 Data Items**.

5   Select the OUT parameters of your COBOL server by using the *Context Menu* and *Toolbar* available in the **COBOL Data Items** and **COBOL OUT Parameter** pane. These are the fields after the technical fields LL (length of message) and ZZ. Also do not select LL and ZZ here.

6   Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

Notes:

■ It is very important to select the right COBOL data items describing the interface of the COBOL server correctly. COBOL data items used as parameters must match in

   ■ number

   ■ sequence of formats (COBOL usage clause)

■ REDEFINE paths on lower levels are mapped in the COBOL Mapping Editor, see *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

■ If your COBOL server contains REDEFINE paths as top-level parameters, the first REDEFINE path is offered by default. Check manually if this is the one you want. If not, correct it. You can select

   ■ any other REDEFINE path

   ■ the complete REDEFINE data item consisting of all REDEFINE paths. This is useful if your COBOL server implements multiple business functions and if you want to map them to Web service operations or methods using a wrapper with a class/method concept such as the *Java Wrapper* or *.NET Wrapper*. For this purpose, select the complete REDEFINE data item. See *Mapping COBOL Functions to Multiple IDL Programs* when you design your IDL file in the COBOL Mapping Editor later.

IMS BMP Standard Linkage

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 parameters (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 parameters will be correct, but you should always check them manually.

▶ **To extract from a IMS BMP Standard Linkage program**

■ Select the `INOUT` parameters (of your COBOL server) by using the *Context Menu* or *Toolbar* available in the **COBOL Data Items** and **COBOL INOUT Parameters** pane. Choose **Finish** and continue with *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*.

Consider the following:

▪ Only level-one parameters can be selected.

▪ 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 (if the number of parameters does not match the number of `INOUT` parameters defined in the list of `PROCEDURE DIVISION USING`, an error message will be displayed without giving the chance to finish the dialog)

▪ sequence of formats (COBOL usage clause)

▪ If your COBOL server contains `REDEFINE`s as top-level parameters (redefines on lower levels are mapped in the mapping editor, see *Step 6: Map the COBOL Interface to IDL with the Mapping Editor*), the first `REDEFINE` path is offered by default. Check manually whether this is the one you want. If not, correct it. You can select here:

▪ any other `REDEFINE` path

▪ the complete `REDEFINE` data item, consisting of all `REDEFINE` paths. This is useful if your COBOL server implements multiple business functions and you want to map them to Web service operations or methods using a wrapper with a class/method concept, such as the *Java Wrapper* or *.NET Wrapper*. For this purpose, select the complete `REDEFINE` data item including all `REDEFINE` paths. Later in the COBOL Mapping Editor, use the feature described under *Mapping COBOL Functions to Multiple IDL Programs* to design your IDL.

▪ Make sure the PCB pointers are also selected at the correct position.

6 COBOL Mapping Editor

This chapter describes the purpose and features of the COBOL Mapping Editor, its context menus, toolbars, main panes and related views.

COBOL Mapping Editor Purpose

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 selected (see *COBOL Parameter Selection*) to Software AG IDL. With the mapping editor, you define:

■ which COBOL server parameters are mapped to IDL (**Select REDEFINE paths**, **Suppress or hide unneeded fields**)

■ the direction of the COBOL server parameters are mapped to IDL (**Map to IN, OUT, INOUT**)

■ field values for COBOL server parameters that are not sent by clients to the COBOL server (**Map to Constants**)

■ which COBOL functions are mapped to IDL programs (**Map to Operations**)

See *Step 6: Map the COBOL Interface to IDL with the Mapping Editor* of *Scenario I: Create New IDL and SVM* for information on how to do this.

Mapping Editor User Interface

The mapping editor is a multi-page Eclipse editor with the following user interface:
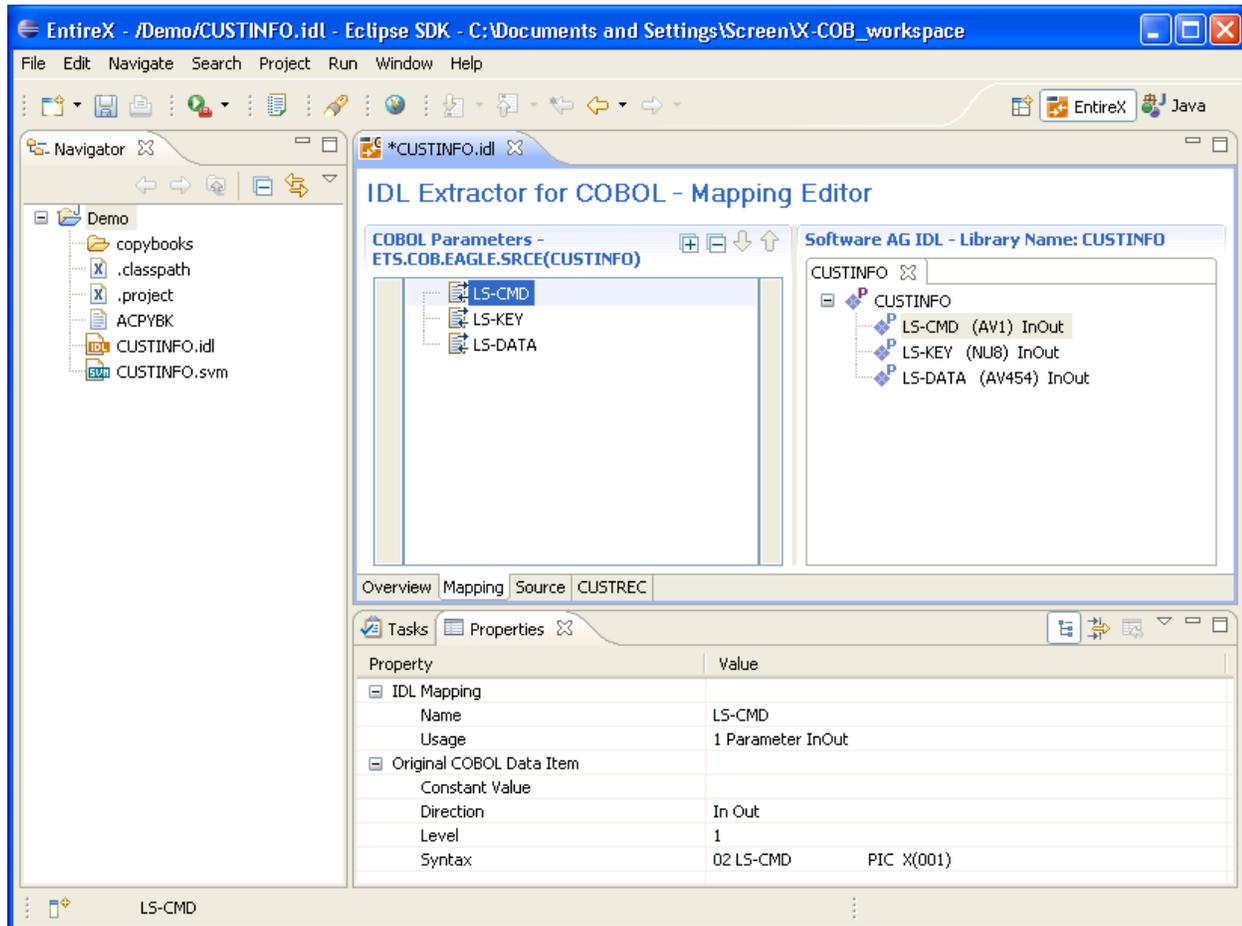
The mapping editor provides four different types of pages which can be selected by the tabs in the middle of the editor: one for **Overview**, **Mapping** and **Source**, and for each copybook, CA Librarian or CA Panvalet member referenced (one tab for each copybook or member) a **Copybook** page. The Eclipse **Properties** are displayed in the lower pane. This section covers the following topics:

- Overview Page
- Mapping Page
- The Source Page

- The Copybook Pages

Overview Page

The **Overview** page summarizes all settings of your extraction from the COBOL source, the IDL file that is being generated and the COBOL-to-IDL mapping defaults used.

Mapping Page

The **Mapping** page consists of two main panes with tree views. On the left there is the COBOL parameter tree and on the right there is the Software AG IDL tree.

💡 **Tip:** The panes can be resized with the vertical line between them, so you can also handle larger parameter lists. The relation between the mapped parameters is established by selection in both trees.

- The COBOL Parameters Pane
- The Software AG IDL Tree Pane

The COBOL Parameters Pane

The **COBOL Parameters** pane, shown in a tree view your selected COBOL parameters). A *Context Menu* is available on those COBOL parameters that provides mapping and other functions. On some COBOL parameters, *Decision Icons* indicate where particular attention is needed, including *Mapping Icons* to visualize the COBOL data type and your current mapping.

COBOL Parameter Names
The *COBOL parameter names* are derived from the COBOL source from which they were extracted. The COBOL names are appended in the following situations:

- If you extract from an interface type IMS BMP with standard linkage calling convention (see *Step 4: Define the Extraction Settings and Start Extraction*), the PCB pointers are prefixed with `[PCB]`.

- If your COBOL interface contains parameters without a name, that is the keyword `FILLER` is used, those COBOL parameters are shown as `[FILLER]`. See *`FILLER` Pseudo-Parameter*.

Context Menu
The **Context Menu** on COBOL parameter nodes provides mapping and other functions. The functions available depend on the setting of the *`OUT` Same as `IN` Parameters* (see *COBOL Parameter Selection*), the COBOL data item type, the COBOL level and the current mapping. The full list of functions is as follows:

Function	Description
Map to `INOUT`	See *Providing IDL Directions (IN OUT INOUT)*.
Map to `IN`	
Map to `OUT`	
Map to Operations	See *Mapping COBOL Functions to Multiple IDL Programs*.
Map to Constants	See *Mapping COBOL Data Items to Constants*.
Suppress	See *Suppressing or Hiding Unneeded Fields of the COBOL Server*.
Properties	Properties of selected COBOL parameters are connected to the Eclipse property view, which enables you to select more details. See *Properties View*.

> **Note:** If you extract from an interface type IMS BMP with standard linkage calling convention (see *Step 4: Define the Extraction Settings and Start Extraction*), the PCB pointers are always suppressed in the IDL, so the only function available is to view the properties.

Decision Icons

The *decision icons* are presented in the first column of the COBOL parameter tree. These markers are set on COBOL data items where particular attention is needed:

Icon	Description
	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".
	This icon visualizes a COBOL level-88 data item where **Map to Operation** is a further mapping offered. See *Mapping COBOL Functions to Multiple IDL Programs*.

With the *Toolbar* you can jump from one decision icon to the next or previous decision icon. In this way you can reach quickly all COBOL data items that require particular attention.

The Mapping Icons

The *mapping icons* on the COBOL parameter nodes indicate the COBOL data type and your current mapping. They are as follows:

Icon	Description
	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`.
	Level-88 data item.

Icon	Description
	Parameter mapped to "Operation".
	Parameter mapped to "Constant".

The Toolbar

The *toolbar* of the COBOL parameters pane provides the following actions:

Action	Description
	Expand the full tree.
	Collapse the full tree.
	Go to the previous COBOL data item with a decision icon. See *Decision Icons*.
	Go to the next COBOL data item with a decision icon. See *Decision Icons*.

The Software AG IDL Tree Pane

The Software AG IDL file, organized in a tree view, represents a tab for each IDL program, and each program is shown as a tree as well. The programs can be selected by choosing the tabs, and they can be deleted by closing the tabs with their close buttons. The last IDL program must remain, thus the tab cannot be closed and deleted. In the Software AG IDL tree view, a context menu is also available with the following possibilities:

■ Rename IDL program

■ Rename IDL parameters

■ *Properties* of selected IDL parameters are connected to the Eclipse property view so you can select more details

The Source Page

The **Source** page provides a syntax-highlighted browser to the COBOL source. Press key F3 on the mapping page of a single parameter selection in the tree to highlight the parameter definition in the source.

The Copybook Pages

The **Copybook** pages provide a syntax-highlighted browser to the copybooks, CA Librarian or CA Panvalet members referenced by the COBOL source extracted. Every copybook or member referenced is displayed in a separate page.

Properties View

The **Properties** view shows your chosen IDL mapping together with the information about the original COBOL data item. The property view can be called from the *Context Menu* on the COBOL parameters in the **COBOL Parameters** pane and the context menu in the *The Software AG IDL Tree Pane*.

7 Server Mapping Deployment

A server mapping file (SVM) enables the RPC server to correctly support special COBOL syntax such as `REDEFINE`s, `JUSTIFIED`, `SYNCHRONIZE` and `OCCURS DEPENDING ON` clauses, `LEVEL-88` fields, etc. If one of these elements is used, the EntireX Workbench automatically extracts an SVM file in addition to the IDL (interface definition language), or an SVM file is generated by the COBOL Wrapper for a server skeleton. The SVM file is used at runtime to marshal and unmarshal the RPC data stream.

To make an SVM file available at runtime for the RPC server used, it has to be deployed. See *Handling SVM Files* in the respective sections of the documentation. As another prerequisite, the RPC server or EntireX Adapter must support the interface type of the COBOL server. This chapter covers the following topics:

Compatibility between Interface Type and RPC Server

To call a server successfully, the RPC server used must support the interface type of the COBOL server. In this context, there are two scenarios, where you need to select an appropriate interface type:

■ Scenario I: Calling an existing COBOL server (BS2000/OSD | Micro Focus):

   within extraction stettings, see *Step 4: Define the Extraction Settings and Start Extraction*.

■ Scenario II: Writing a new COBOL server (BS2000/OSD | Micro Focus):

   within the COBOL Wrapper IDL properties, see *Server Interface Types* under *Generate COBOL Source Files from Software AG IDL Files*.

The table below gives an overview of possible combinations of an interface type and a supporting RPC server:

Interface Type	Wrapper	Extractor	Supported by								
			RPC Server								
			z/OS			UNIX/Windows			BS2000/OSD	z/VSE	
			CICS	Batch	IMS	ECI	Micro Focus	IMS Connect	Batch	CICS	Batch
CICS with DFHCOMMAREA calling convention	yes	yes	x			x				x	
CICS with DFHCOMMAREA large buffer interface	yes	yes	x							x	
CICS with Channel Container calling convention	yes	yes	x								
Batch with standard linkage calling convention	yes	yes		x	x				x		x
Micro Focus with standard linkage calling convention	yes	yes					x				
IMS BMP with standard linkage calling convention	yes	yes			x						
IMS MPP message interface (IMS Connect)	no	yes						x			

Deploying a Server Mapping File

To deploy a server mapping, you can either use a wizard or copy the file manually to a target container. The approach depends on the RPC server used. For instance, not all RPC servers support both deployment by using a wizard and manual deployment. The table below gives an overview of supported approaches for available RPC servers and the webMethods EntireX Adapter for Integration Server, and provides further instructions for various forms of manual deployment:

RPC Server	Wizard	Manual Deployment
CICS ECI RPC server (Windows and UNIX)	No	Copy the SVM file to the folder specified by `cics.mapping.folder`. See *Configuring the CICS ECI Side*.
CICS RPC server (z/OS)	**Yes**	Use FTP and IDCAMS. See *Deploying a Server Mapping File*.
Batch RPC server (z/OS)	**Yes**	Use FTP and IDCAMS. See *Deploying a Server Mapping File*.
Batch RPC server (BS2000/OSD)	**Yes**	No
Micro Focus RPC server (Windows and UNIX)	**Yes**	No
IMS Connect RPC server (Windows and UNIX)	No	Copy the SVM file to the folder specified by `property ims.mapping.folder`. See *Configuring the IMS Connect Side* in the IMS Connect RPC Server documentation.
IMS RPC server (z/OS)	**Yes**	Use FTP and IDCAMS. See *Deploying a Server Mapping File*.
webMethods EntireX Adapter for Integration Server	No	No. Do not change the location of the generated SVM file. It has to be kept in the same folder as the IDL file and will be picked up automatically together with the IDL file when an adapter connection for IMS Connect or CICS ECI is generated. For more information, see the EntireX Adapter documentation under *webMethods > Mainframe Integration* on the **Software AG Product Documentation** website.

The remainder of this section focuses on the major approaches:

- Server Mapping Deployment Wizard
- Server Mapping Deployment Wizard Preferences
- Server Mapping Deployment in Command-line Mode

Server Mapping Deployment Wizard
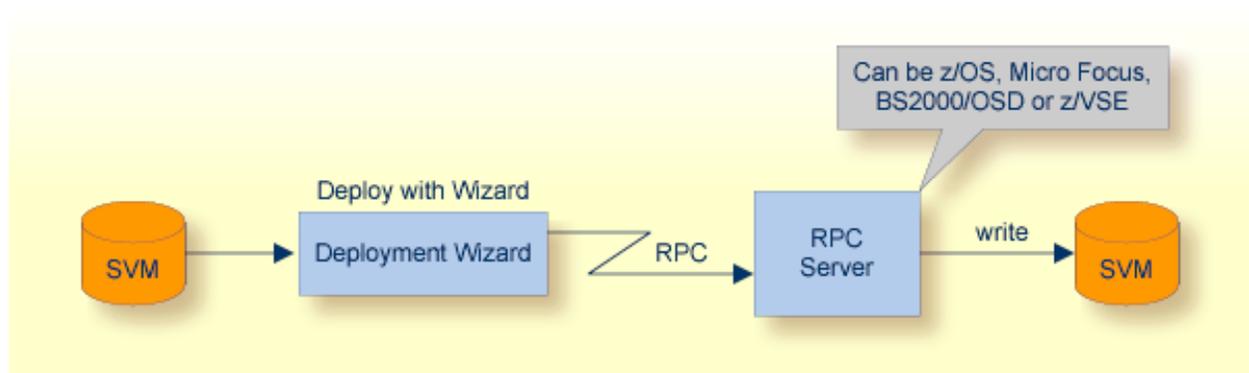
Deploying with the wizard requires an active RPC server. Also, the Deployment Service of the RPC server must be properly configured. See the platform-specific documentation for more information:

■ z/OS, see *Deployment Service* in the respective RPC server documentation.

■ UNIX or Windows for Micro Focus COBOL, see *Deployment Service* in the Micro Focus RPC Server .

■ BS2000/OSD, see *Deployment Service* in the BS2000/OSD Batch RPC Server documentation.

■ z/VSE, see the separate 7.2.3 documentation.

> **Caution:** The codepage (locale string) of the RPC server used during deployment must be the same as the one used with running RPC clients issuing RPC requests.

To deploy a server mapping with the wizard, follow the steps below:

■ Step 3: Select an Existing Deployment Environment and Deploy

Step 1: Start the Wizard

To start the server mapping deployment wizard, select an SVM file and choose **Deploy/Synchronize Server Mapping...** from the context menu.
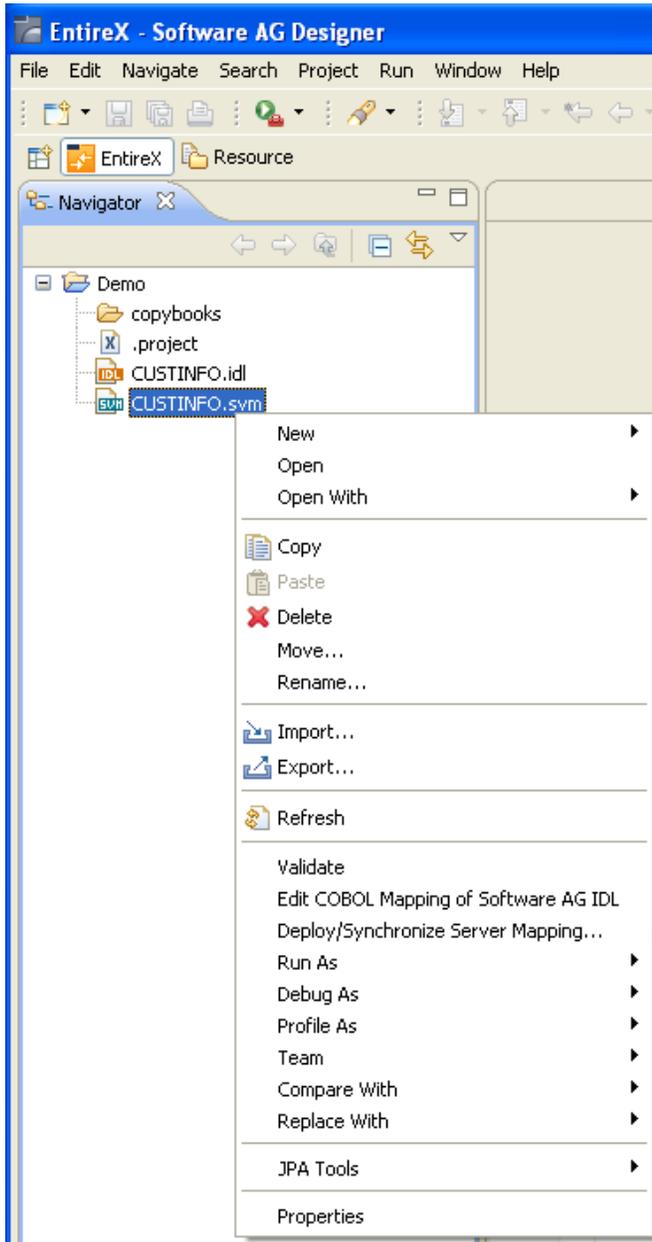
To continue, press **Next** together with one of the following choices:

- If you enter the server mapping deployment wizard the first time with no predefined deployment environment preferences (see *IDL Extractor for COBOL Preferences*), continue with *Step 2a: Create a New Deployment Environment* below.

- If deployment environments are already defined, you may also continue with *Step 3: Select an Existing Deployment Environment and Deploy* below.

Step 2a: Create a New Deployment Environment

If no deployment environments are defined, you only have the option to create a new deployment environment.
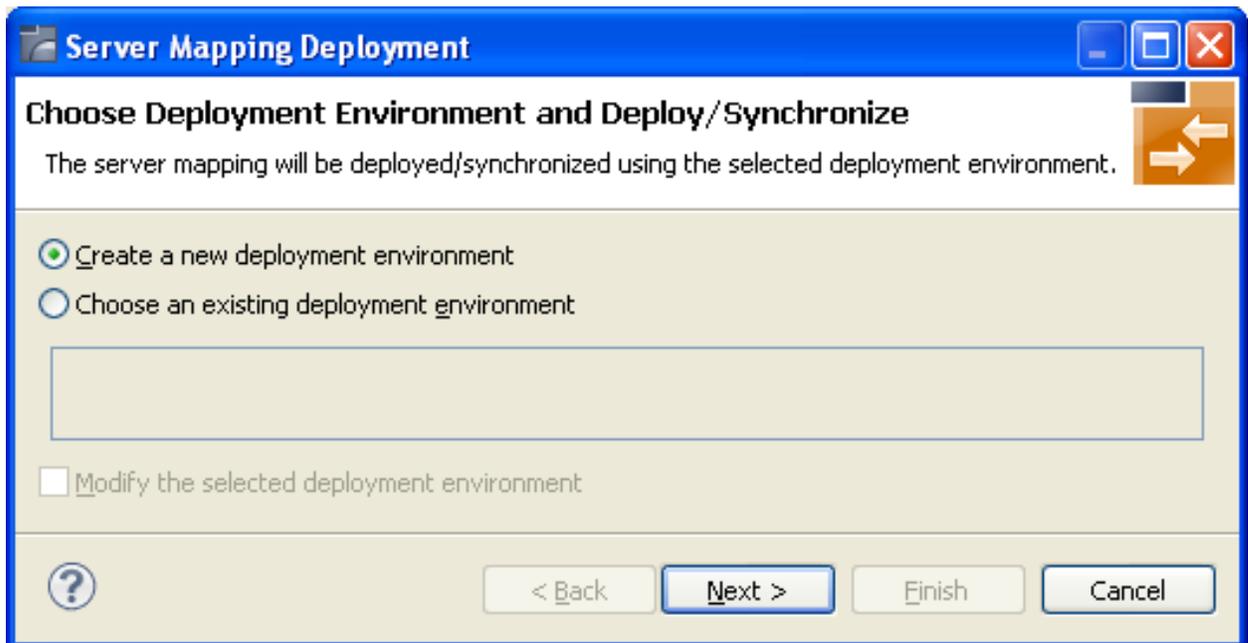
To proceed:

1. Select **Create a new deployment environment**.

2. Press **Next** and continue with *Step 2b: Define the Connection to the Deployment Service and Deploy*.

Step 2b: Define the Connection to the Deployment Service and Deploy

Use this page to define a connection to the deployment service of the RPC server.

Enter the required fields:

1. **Broker Parameters** Broker ID and Server Address, which will have the default format. The last part (broker service) of the server address must always be "DEPLOYMENT".

2. The **EntireX Authentication** parameters describe the settings for the broker. These parameters apply if the *broker* is running with EntireX Security. See *Which EntireX Security Solution*.

3. The **RPC Server Authentication** parameters describe the settings for the RPC server. These parameters apply if the *RPC server* is running with security, for example a Natural RPC Server running with Natural Security.

4. The given Timeout value must be in the range from 1 to 9999 seconds (default: 60).

Press **Finish** to deploy. Deployment of the server mapping is successful if the wizard ends. No confirmation message is given.

Step 3: Select an Existing Deployment Environment and Deploy

Use this page to select the deployment environment (i.e. the RPC server) to which you want to deploy.
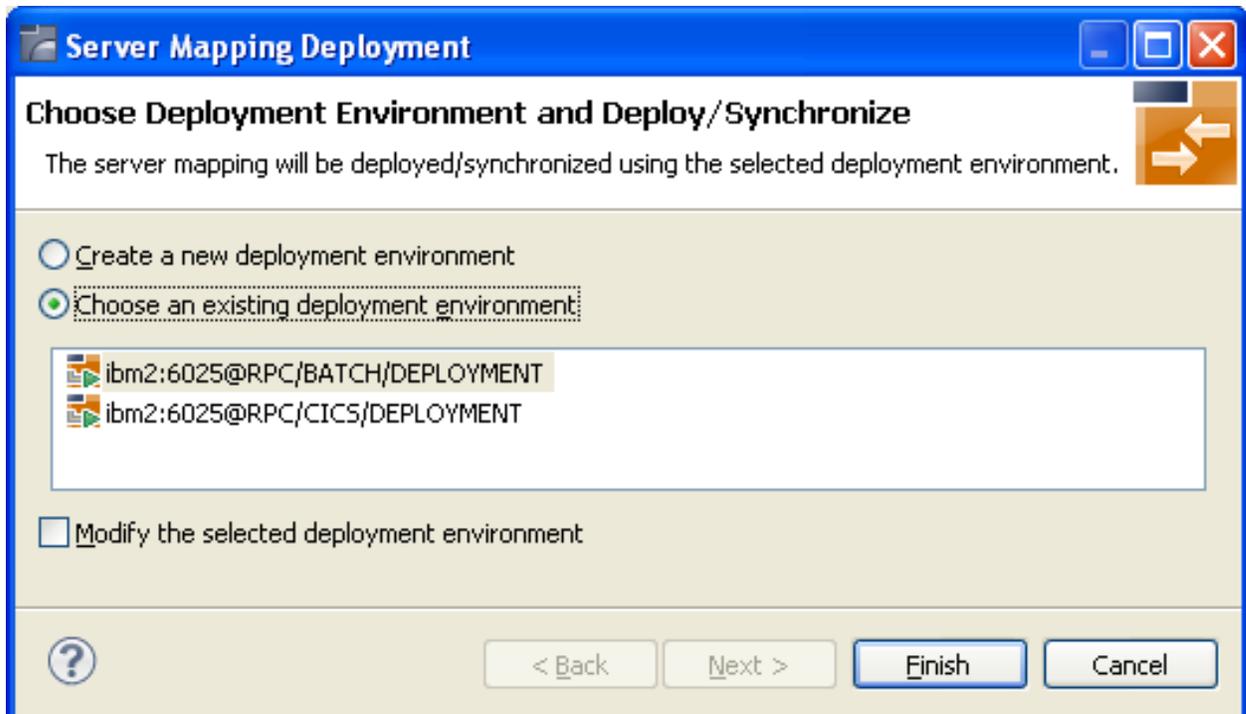
Check the option **Choose an existing deployment environment** and select a deployment environment from the list. Press **Finish** to deploy. Deployment is successful if the wizard ends. No confirmation message is given.

Server Mapping Deployment Wizard Preferences

In the preferences for the server mapping deployment wizard you define deployment environments, a connection to the Deployment Service of the RPC server. See *Deployment Service* in the respective RPC server documentation. The following sections are offered:

- Create a new Deployment Environment
- Edit an Existing Deployment Environment
- Remove an Existing Deployment Environment

The deployment environment is managed from the deployment environment **Preferences** page. The deployment environments can be created, edited and removed. The deployment environment will be used for the selection lists in the *Server Mapping Deployment to z/OS, using FTP and ID-CAMS*. To manage these deployment environments, open the **Preferences** page:

Create a new Deployment Environment

▶ **To create a new deployment environment**

■   Press **Insert**.

Edit an Existing Deployment Environment

▶ **To edit an existing deployment environment**

■   Select the table row and press **Edit**. If multiple entries are selected, the first entry is used.

For more information on the input fields on this page, see *Step 2b: Define the Connection to the Deployment Service and Deploy* under *Server Mapping Deployment Wizard*.

Remove an Existing Deployment Environment

▶ **To remove an existing deployment environment**

■   Select the table row and press **Remove**. Multiple selections are possible.

Server Mapping Deployment in Command-line Mode

The command `-deploy:cobol` is provided to deploy server mapping files (SVMs) using the EntireX Workbench in command-line mode. See *Using the EntireX Workbench in Command-line Mode* for general information.

To undeploy previously deployed server mapping entries on the server side, remove the SVM file and execute the command `-deploy:cobol` with the IDL file only.

Command-line Options

Task	Command	Option	Description
Deploy SVM files.	`-deploy:cobol`	`-environment`	Target environment. Name of the COBOL deployment environment or an RPC server description.
		`-brokeruser`	User used for broker authentication (optional).
		`-brokerpassword`	Password used for broker authentication (optional).
		`-rpcuser`	User used for RPC server authentication (optional).
		`-rpcpassword`	Password used for RPC server authentication (optional).

> **Note:** Run the command from the directory containing the IDL file and the corresponding SVM file. If no SVM file is found, the previously deployed server mapping entries related to the IDL file will be removed on the server side (undeployed).

Example

```
-deploy:cobol /SVMDeployTests/idls/basicodo.idl  /SVMDeployTests/idls/basicdt.idl ↵
/SVMDeployTests/idls/basicarr.idl
 -environment ibm2:3980@RPC/RPCALL/DEPLOYMENT
 -brokeruser EXXUSR1
 -brokerpassword EXX$PWD1
```

Server Mapping Deployment to z/OS, using FTP and IDCAMS

This approach is available for z/OS only.
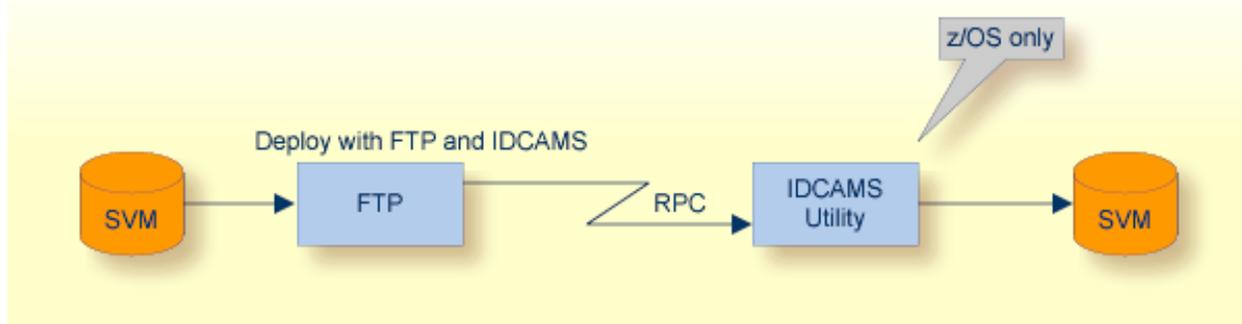
▶ **To deploy a server mapping using FTP and IDCAMS**

1   Allocate a target sequential file on mainframe with the following specifications for `DCB`:

```
DCB=(DSORG=PS,RECFM=V,LRECL=16384,BLKSIZE=16388)
```

2   Allow write access and usage of IDCAMS tools to the VSAM file mentioned above.

3   Transfer the SVM file to the target host, using FTP. You have to switch to text mode and the codepage of the FTP service must be the same as the codepage (locale string) of the RPC server used.

4   Install the server mapping into the VSAM SVM cluster with the IDCAMS job below.

> 📄 **Note:** If you omit the keyword REPLACE or define NOREPLACE in the SYSIN data stream instead, existing server mappings are not overwritten. This protects SVM records from being overwritten by duplicates.

```
//EXPSVMR  JOB (,,,999),ENTIREX,NOTIFY=&SYSUID,MSGLEVEL=(1,1),
//            CLASS=K,MSGCLASS=X,REGION=0M,COND=(0,LT)
//*----------------------------------------------------*
//* FILL THE SVM VSAM CLUSTER                           *
//*----------------------------------------------------*
//IMPORT   EXEC PGM=IDCAMS
//RECORDS  DD DISP=SHR,DSN=EXP.SVM.TARGET.SEQ.RECORDS
//SVM      DD DISP=SHR,DSN=EXP.SVM.KSDS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  REPRO -
    REPLACE -
    INFILE(RECORDS) -
    OUTFILE(SVM)
```

8 IDL Extractor for COBOL Preferences

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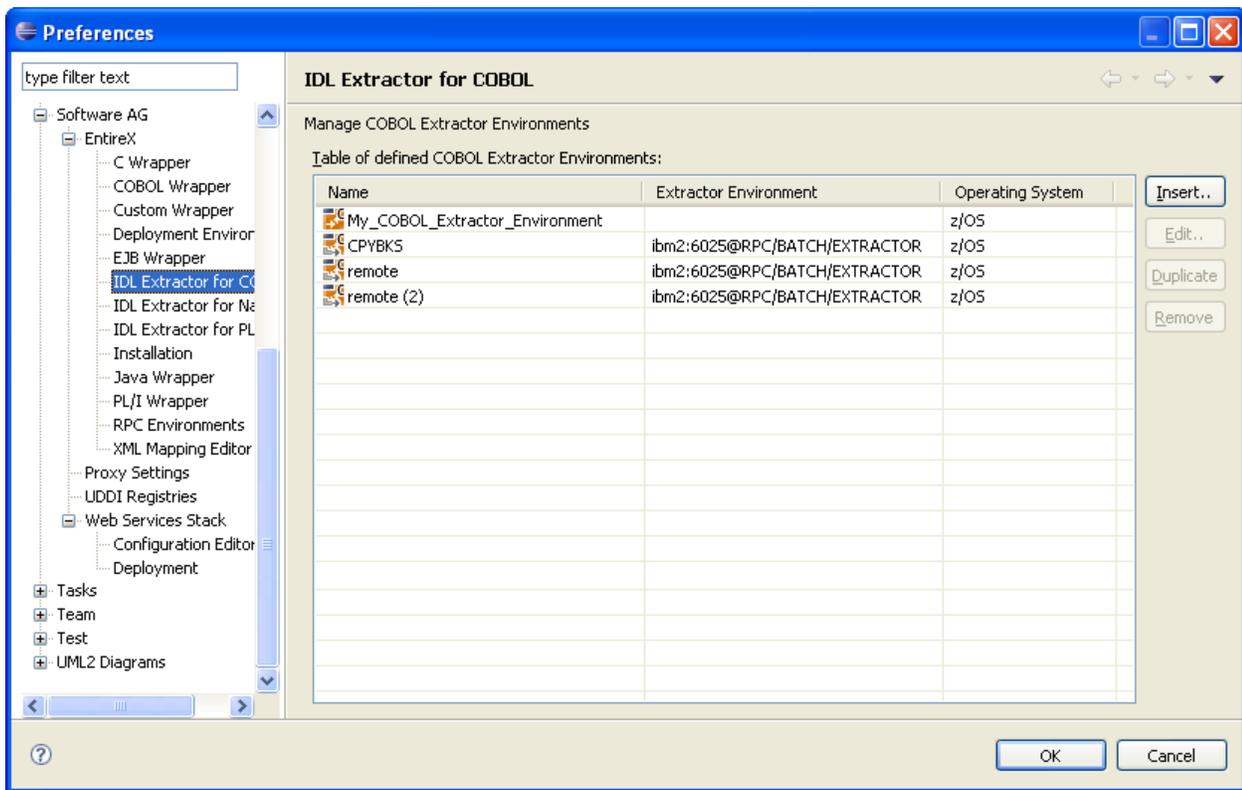
Create New Local Extractor Environment (z/OS, z/VSE, BS2000/OSD and IBM i)

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**.

▶ **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.

▶ **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.

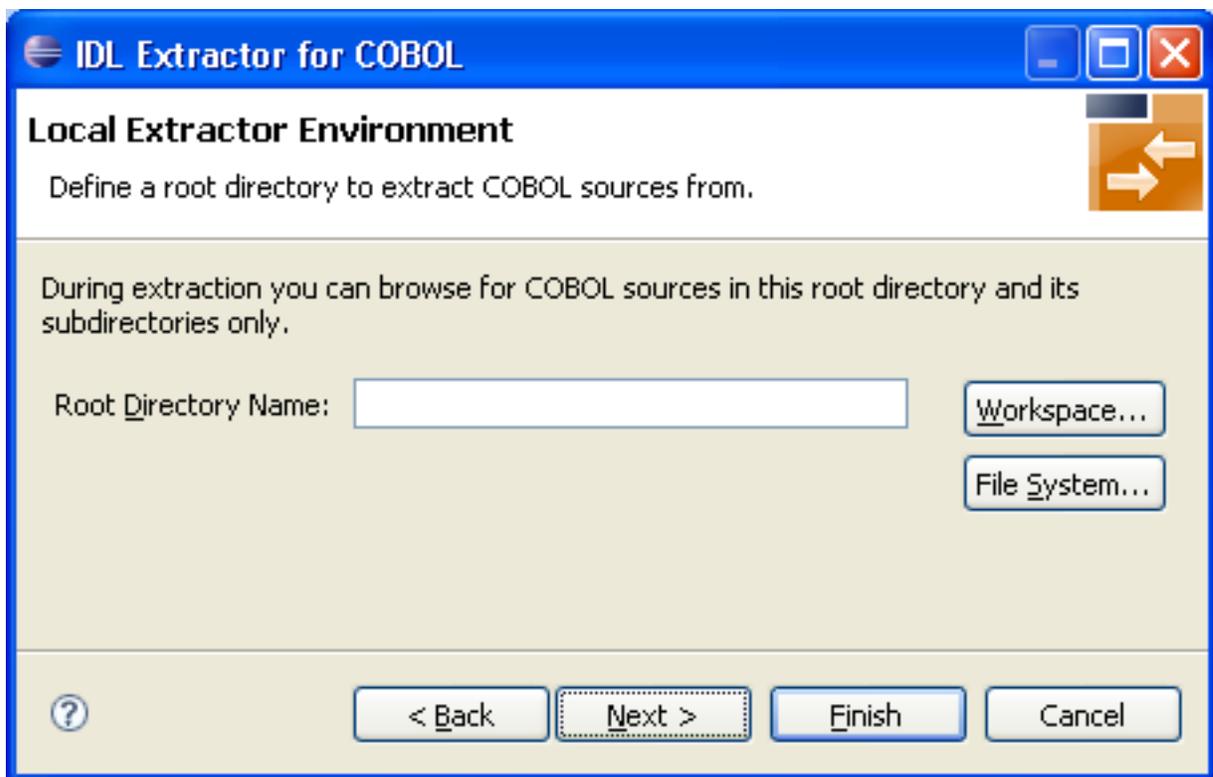- *IMS BMP with Standard Linkage Calling Convention*, you can set the default for **IMS PSB List**.

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.

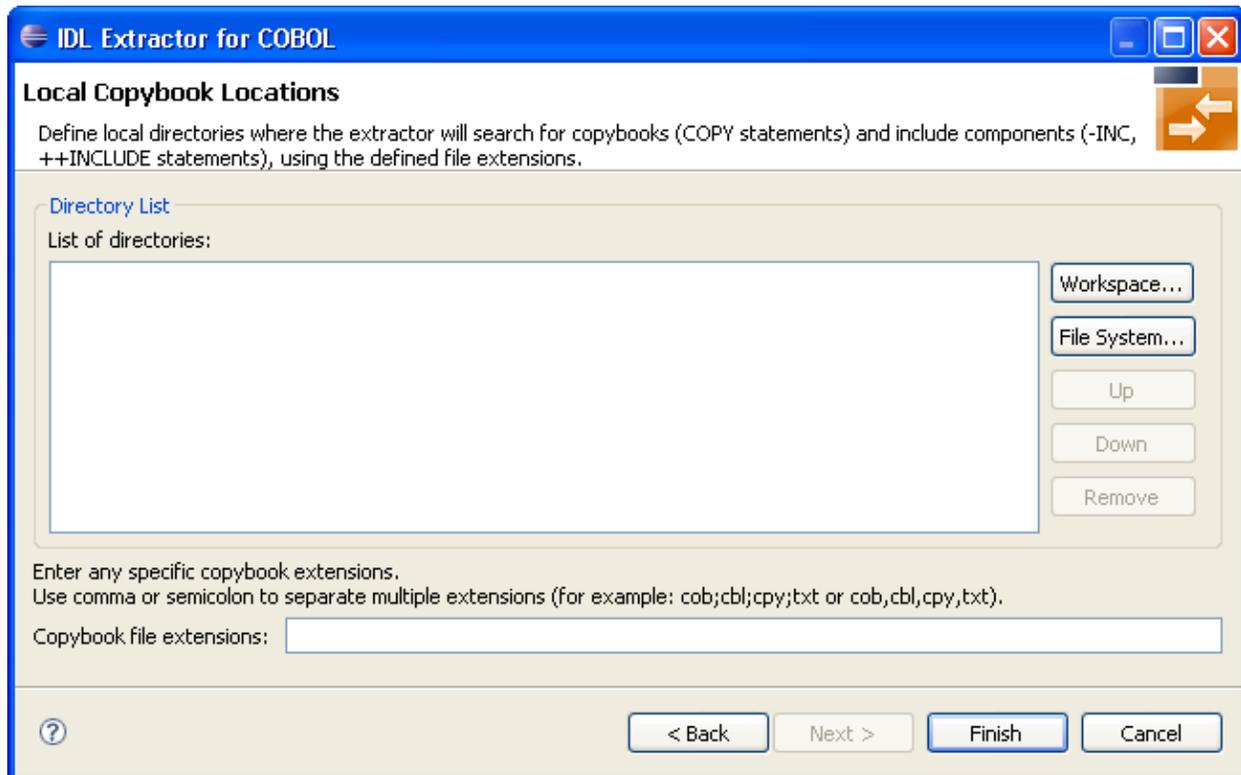### Step 3: Define the Local Extractor Environment

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

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 copy-books. Copybooks and members referenced with COPY statements, CA Librarian `-INC` statements and CA Panvalet `++INCLUDE` statements will be searched for in the defined local directories:

The file extensions for copybooks can also be entered. If no extensions are specified, the IDL Extractor for COBOL wizard will try to locate copybooks without any file extensions.

Press **Workspace...** or **File System...** to browse for a folder.
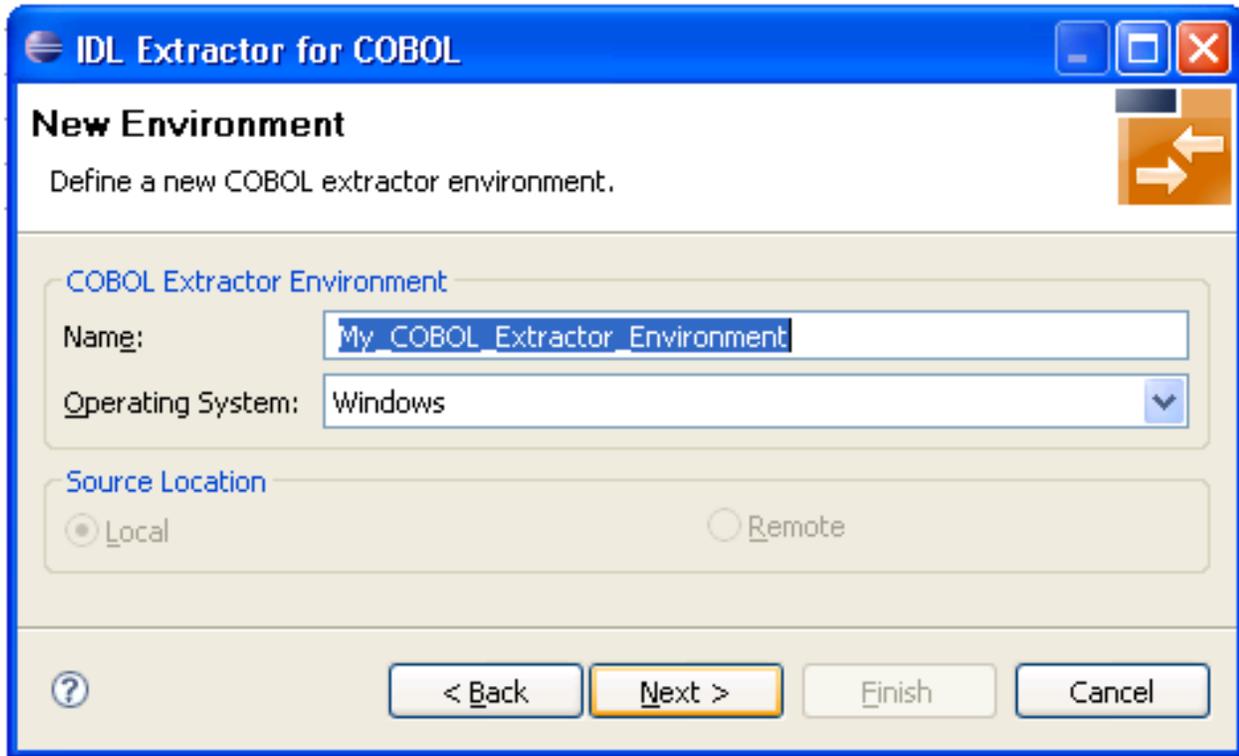
Press **Finish**.

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 1: Define the New Local Environment

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**.

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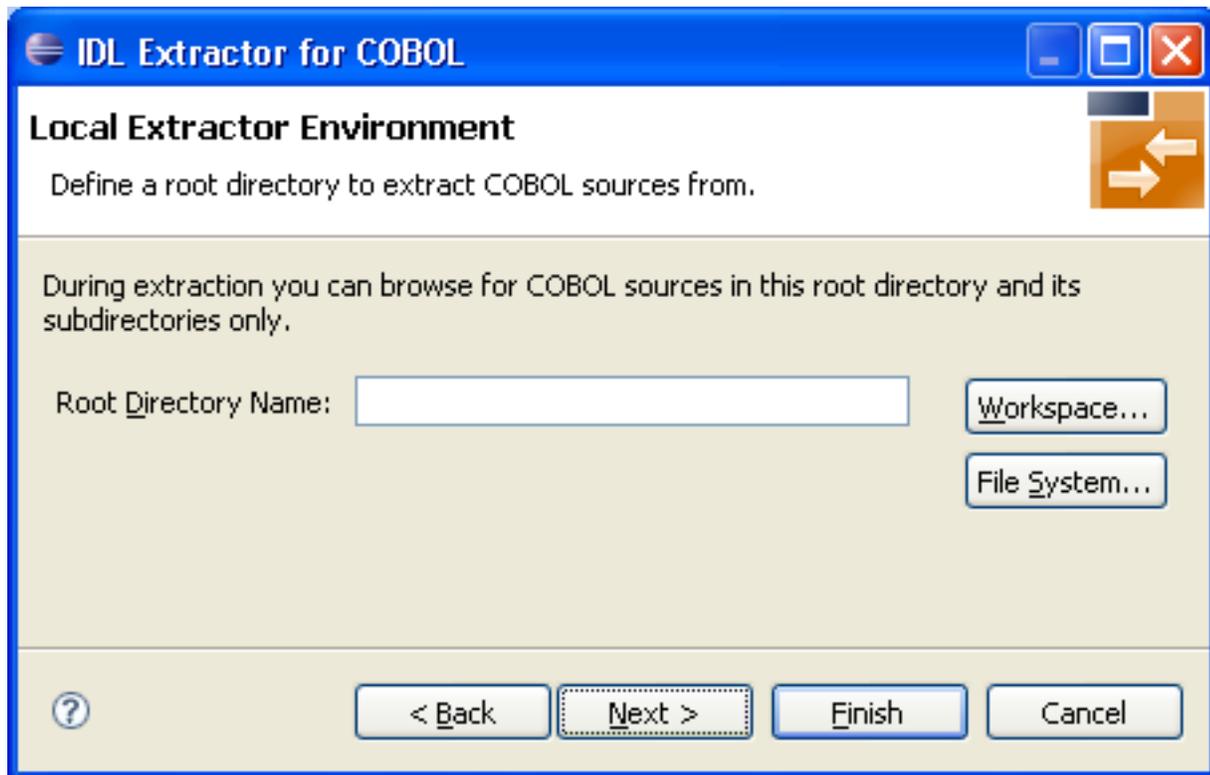
Refer to *Step 2: Define the Default Settings* for a local extractor environment for field descriptions.

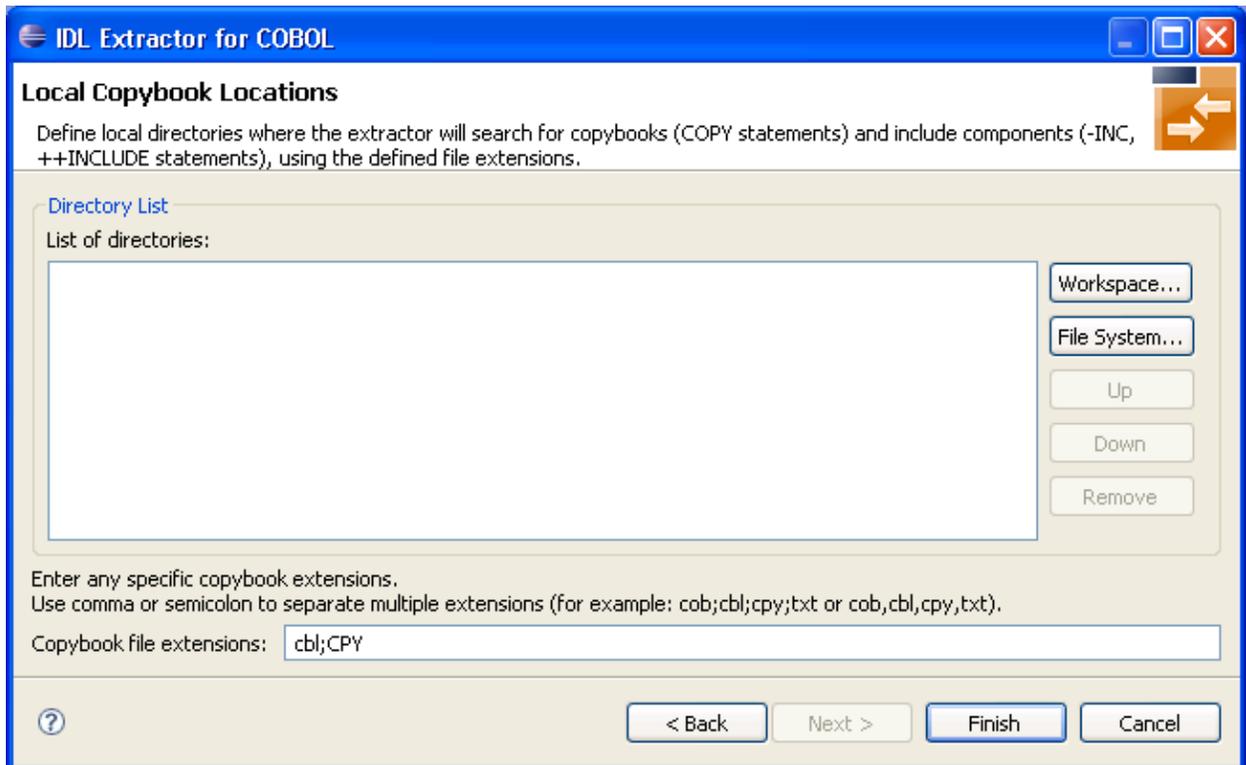### Step 3: Define the Local Extractor Environment

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

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:

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**.

▶ **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**.

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*.

▶ **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:
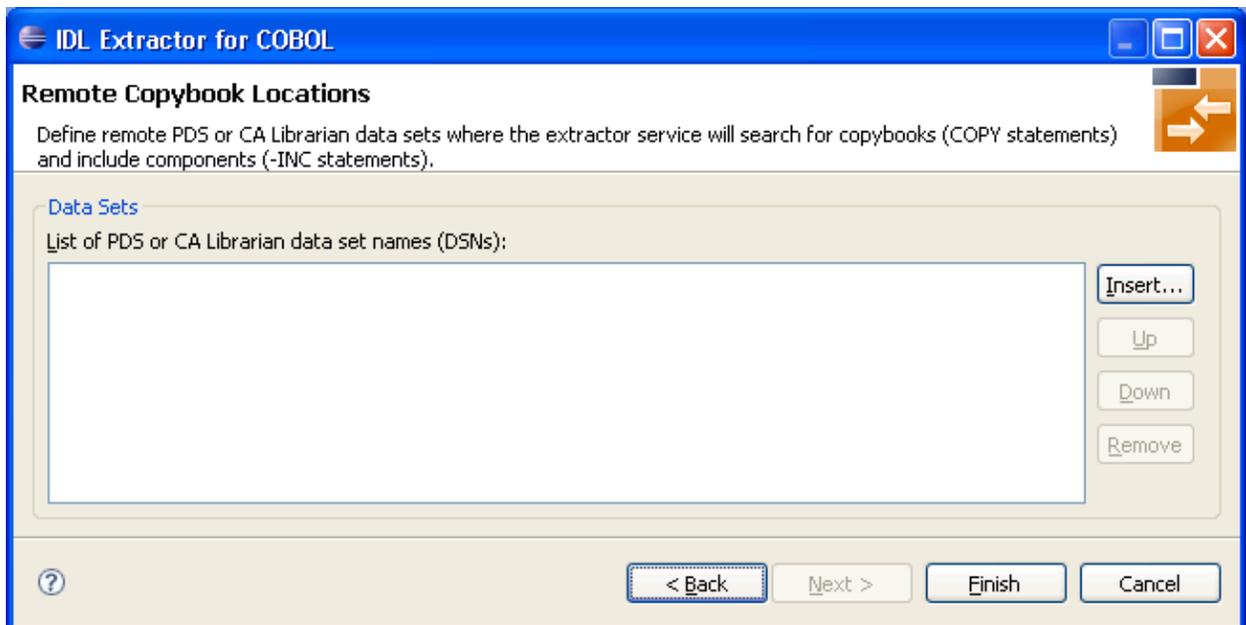
Press **Insert...** to add a new data set entry in the table. Use **Remove**, **Up** and **Down** to manage the data set list.

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**.

▶ **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**.

▶ **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.

▶ 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:
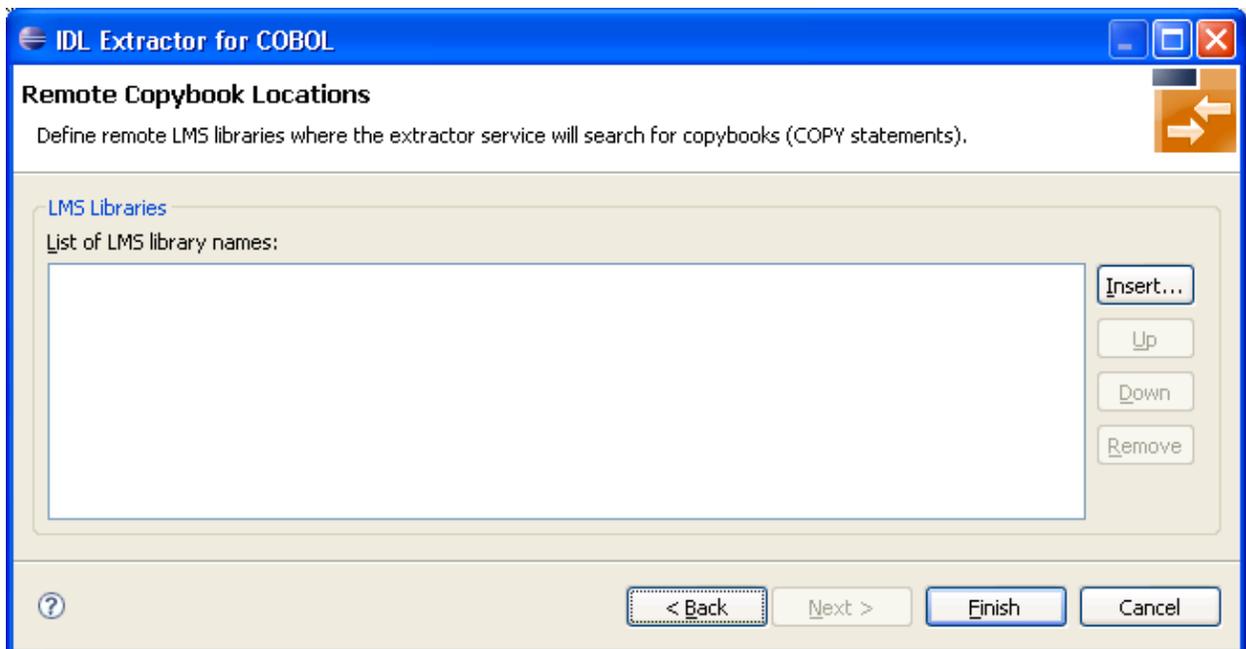
Press **Insert...** to add a new data set entry in the table. Use **Remove**, **Up** and **Down** to manage the list of LMS libraries.

Press **Finish**.

9 COBOL to IDL Mapping

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

COBOL Data Type to Software AG IDL Mapping

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

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

- The metasymbols "[" and "]" surround optional lexical entities.

- The informal terms *n* and *m* are sequences of numeric characters, for example 123.

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

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 syntax:

Format	COBOL Syntax
1	88 <condition-name> VALUE [IS] 'literal-1'
2	88 <condition-name> VALUE [IS] 'literal-1' [THRU | THROUGH] 'literal-2'
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, i.e. 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. See *Mapping COBOL Data Items to Constants*.

▪ **Function or Operation Codes**
If the level-88 values are function or operation codes, map the level-88 condition names to an operation. This will lead to an IDL program per operation code. See *Mapping COBOL Functions to Multiple IDL Programs*.

The through values of the level-88 condition names (see format 2 in the table above) are not considered by the IDL extractor for COBOL. The DATA DIVISION is parsed as without level-88 through values.

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

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

This will be mapped to Software AG IDL:

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

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

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

This will be mapped to Software AG IDL:

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

GLOBAL and EXTERNAL Clause

The `GLOBAL` clause

■ specifies that a data-name is available to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name.

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

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

The `EXTERNAL` clause

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

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

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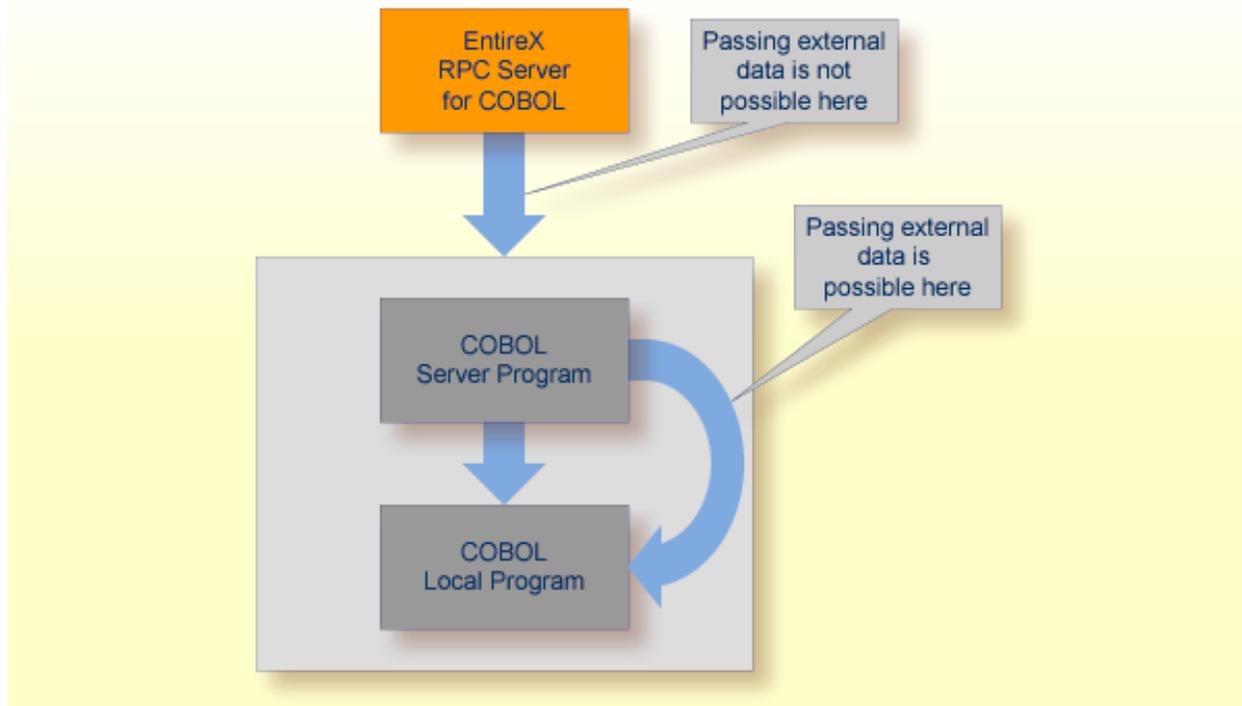
JUSTIFIED Clause

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

OBJECT REFERENCE Phrase

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

Parameter Names

Numbers in the first position of the parameter name are not allowed in Software AG IDL syntax (see *Software AG IDL Grammar* 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 *Server Mapping File (SVM)*. 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 Parameter Selection*
  - 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 Parameters* 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 (see *Mapping COBOL Functions to Multiple IDL Programs*), and where each operation interface is described by a level 1 `REDEFINE` path in the parameter selection window. See *Step 5: Select the COBOL Parameters* 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. See *Selecting `REDEFINE` Paths* 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. See *Selecting `REDEFINE` Paths* in the mapping editor.

■ See also *Step 6: Map the COBOL Interface to IDL with the Mapping Editor* in *Using the IDL Extractor for COBOL*.

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 *Server Mapping File (SVM)*.

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 sized 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 SVM 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 *Server Mapping File (SVM)*.

```
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\> (/Vm)
1 \<name\> OCCURS \<n\> TO \<m\> [TIMES] DEPENDING [ON] \<index\> [ASCENDING | DESCENDING [KEY] [IS] \<key-name\>]	1 \<name\> (/Vm)
1 \<name\> OCCURS \<n\> TO \<m\> [TIMES] DEPENDING [ON] \<index\> [INDEXED [BY] \<index-name\>]	1 \<name\> (/Vm)

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 parameter direction. This means that if the variable-size table is selected in the *COBOL Parameter Selection* page of the wizard as a

  - COBOL `INOUT` Parameter (see *Step 5: Select the COBOL Parameters*), 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, un-expected 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 in the *COBOL Parameter Selection*.

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 in the *COBOL Parameter Selection*. 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 Parameters* 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; see *Providing IDL Directions (`IN OUT INOUT`)*. See also *Step 6: Map the COBOL Interface to IDL with the Mapping Editor* in *Using the IDL Extractor for COBOL*.

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)==.
```