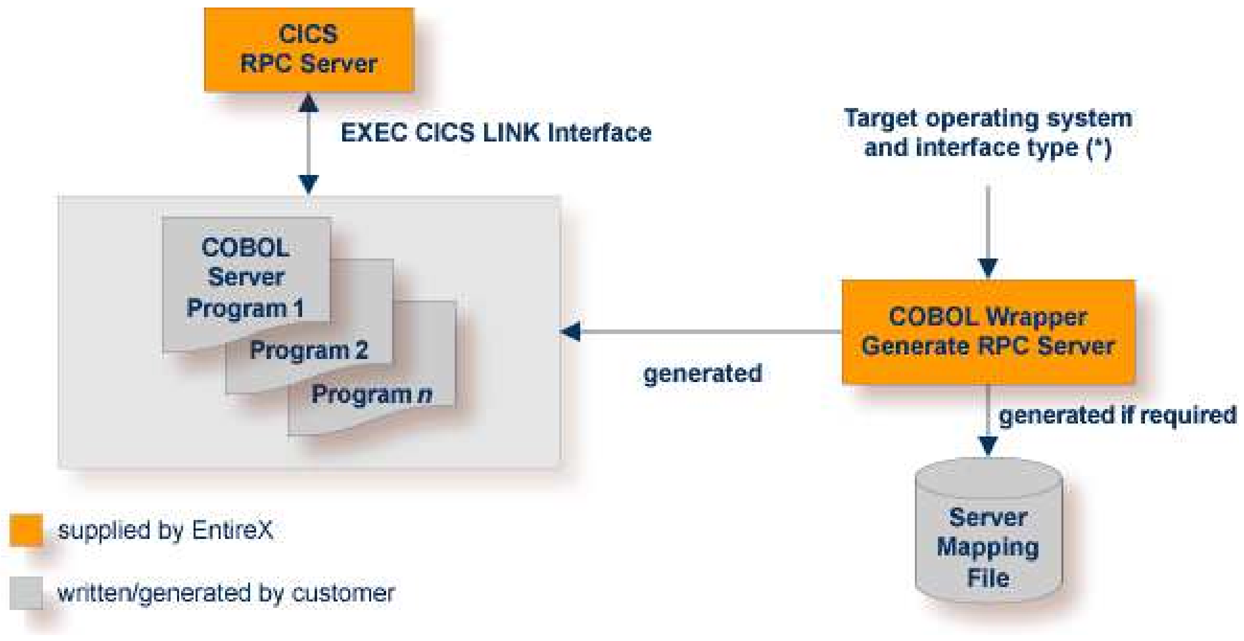
Using the COBOL Wrapper for the Server Side

The COBOL Wrapper provides access to RPC-based components from COBOL applications and enables users to develop both clients and server. This section introduces the various possibilities for RPC-based server applications written in COBOL and covers the following sections:

- Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)
- Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)
- Using the COBOL Wrapper for CICS with DFHCOMMAREA Large Buffer Interface (z/OS and z/VSE)
- Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)
- Using the COBOL Wrapper for IMS BMP (z/OS)
- Using the COBOL Wrapper for Micro Focus (UNIX and Windows)

Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)

This mode applies to z/OS and z/VSE. See also *COBOL Scenarios* under z/OS | z/VSE | CICS ECI in the CICS RPC Server documentation.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all of your server's parameters dynamically in the format required. Your server is called using `EXEC CICS LINK`.

Use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention if

- you want to have a standard `EXEC CICS LINK` DFHCOMMAREA interface to your server
- you require a distributed program link (CICS DPL) to your server
- the DFHCOMMAREA length restriction (31 KB) suits your needs, otherwise consider the following interface types:
  - *Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)*
  - *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Large Buffer Interface (z/OS and z/VSE)*

To use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention

1. Generate the server (skeleton) for the target operating system, for example "z/OS", and use interface type "CICS with DFHCOMMAREA calling convention". See *Generating COBOL Source Files from Software AG IDL Files*.

2. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* (z/OS | z/VSE | CICS ECI) in the RPC server documentation, except for CICS ECI connections with the webMethods EntireX Adapter, where you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping* and *Aborting RPC Server Customer Code and Returning Error to RPC Client* (z/OS | z/VSE) in the CICS RPC Server documentation.

5. Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6. Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file. See `program-definition` under *Software AG IDL Grammar*.

7. Provide your server(s) to the CICS RPC server, EntireX Adapter, or CICS ECI RPC server:

   - Install your server(s) as separate CICS program(s).

   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding program can be found, the access will fail.
   - If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize Automatically Generated Server Names*) is executed. If no corresponding program can be found, the access will fail.

- If neither a server-side nor client-side mapping file is used - for example it is not required or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition`) given in the IDL is ignored.

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

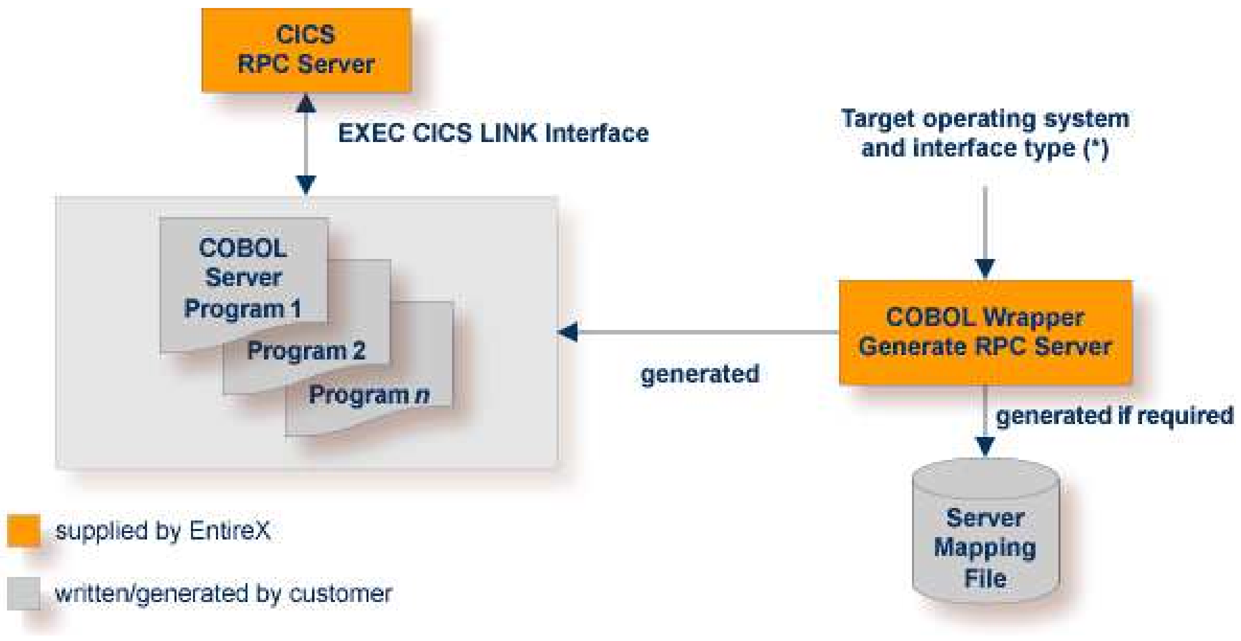
Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)

This section covers the following topics:

- Introduction
- CICS Channel Container IDL Rules
- Restrictions
- Example 1: Same Container for Direction In and Out
- Example 2: Different Container for Direction In and Out
- Example 3: Multiple Containers
- Example 4: Variable Number of Containers (Direction Out Only)
- Steps

Introduction

This mode applies to z/OS. See also *COBOL Scenarios* in the CICS RPC Server documentation.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all of your server’s parameters dynamically in the format required. Your server is called using `EXEC CICS LINK` passing the container(s) in the defined channel to your server. See *Channel Name*.

Use the COBOL Wrapper for CICS with channel container calling convention if

- you require more than 31 KB of data to transfer to your server
- your IDL complies with CICS channel container IDL rules (see below). If your IDL does not match these rules, consider the interface type *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Large Buffer Interface (z/OS and z/VSE)* to implement your server.
- you want to have a standard CICS channel container interface to your server
- you require a distributed program link (CICS DPL) to your server.

CICS Channel Container IDL Rules

The following rules apply to CICS channel container IDL:

- A container is described with an IDL structure. See `structure-definition`.
- The container name is the name of the IDL structure. A maximum of 16 characters are allowed by CICS for container names.
- IDL programs reference IDL structures only. No other parameters may be referenced.
- Multiple containers can be defined, see *Example 3: Multiple Containers*.
- A variable number of containers can be defined using one-dimensional IDL unbounded arrays with maximum (see *`array-definition`* under *Software AG IDL Grammar* in the IDL Editor documentation). See also *Example 4: Variable Number of Containers (Direction Out Only)*.

Restrictions

- IDL unbounded arrays (i.e. variable containers) for direction In and `INOUT` are not supported.
- Two and three-dimensional IDL unbounded arrays are not supported.

Example 1: Same Container for Direction In and Out

This example uses the same container for input and output. The container name is "CALC".

```
Library 'EXAMPLE' Is
  Program 'CONCALC' Is
   Define Data Parameter
    1 Container       ('CALC')        InOut
   End-Define

  Struct 'CALC' Is
   Define Data Parameter
    1 Operation       (A1)
    1 Operand_1       (I4)
    1 Operand_2       (I4)
    1 Function_Result (I4)
   End-Define
```

Example 2: Different Container for Direction In and Out

This example uses separate containers for input and output.

```
Library 'DFHCON' Is
 Program 'TWOC' Is /* Two Container - Separate for Input and Output
   Define Data Parameter
    1 ContainerIn   ('CONTAINER1')  In
    1 ContainerOut  ('CONTAINER2')  Out
   End-Define
Struct 'CONTAINER1' Is
  Define Data Parameter
   1 Just-Occupied-Space (A39000) /* 39K
   1 Request             (A1000/5) /* 5K
  End-Define
Struct 'CONTAINER2' Is
  Define Data Parameter
    1 Just-Occupied-Space (A49000) /* 49K
    1 Reply               (A250)
  End-Define
```

See IDL program `TWOC` under *Advanced CICS Channel Container RPC Server Example - `DFHCON`*.

Example 3: Multiple Containers

This example shows how more than one container is used per direction. Each container has its own structure layout.

```
Library 'DFHCON' Is
  Program 'MULTIC' Is
   Define Data Parameter
    1 InContainer1     ('INCONTAINER1')  In
    1 InContainer2     ('INCONTAINER2')  In
    1 InContainer3     ('INCONTAINER3')  In
    ...

    1 OutContainer1    ('OUTCONTAINER1') Out
    1 OutContainer2    ('OUTCONTAINER2') Out
    1 OutContainer3    ('OUTCONTAINER3') Out
    ...

   End-Define

  Struct 'INCONTAINER1' Is ...
  Struct 'INCONTAINER2' Is ...
  Struct 'INCONTAINER3' Is ...
  ...

  Struct 'OUTCONTAINER1' Is ...
  Struct 'OUTCONTAINER1' Is ...
  Struct 'OUTCONTAINER1' Is ...
  ...
```

Example 4: Variable Number of Containers (Direction Out Only)

This example shows how to specify a range of containers. At runtime, the called RPC server creates a variable number of containers from this range. Each container created has the same structure layout and a container name that is formed from the structure name as prefix and the structure index as suffix. In this example:

- `MULTIPLE` container names are `MULTIPLE0001` thru `MULTIPLE9999`.
- `OPTIONAL` container name is `OPTIONAL1`.

Note:
Make sure IDL observes the 16-character length restriction for container names given by CICS.

```
Library 'DFHCON' Is
  Program 'VARC' Is
   Define Data Parameter
    1 Input            ('INPUT')          In
    1 Multiple         ('MULTIPLE'/V9999) Out /* 0 thru 9999 times
    1 Optional         ('OPTIONAL'/V1)    Out /* 0 or 1 times
   End-Define

  Struct 'INPUT' Is ...
  Struct 'MULTIPLE' Is ...
  Struct 'OPTIONAL' Is ...
```

Steps

To use the COBOL Wrapper for CICS with channel container calling convention

1. Generate the server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "CICS with channel container calling convention". See *Generating COBOL Source Files from Software AG IDL Files*.
2. The generated server mapping file has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* (z/OS | z/VSE) in the CICS RPC Server documentation.
   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.
3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.
4. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping* and *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server documentation.
5. Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6. Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file (see `program-definition`).

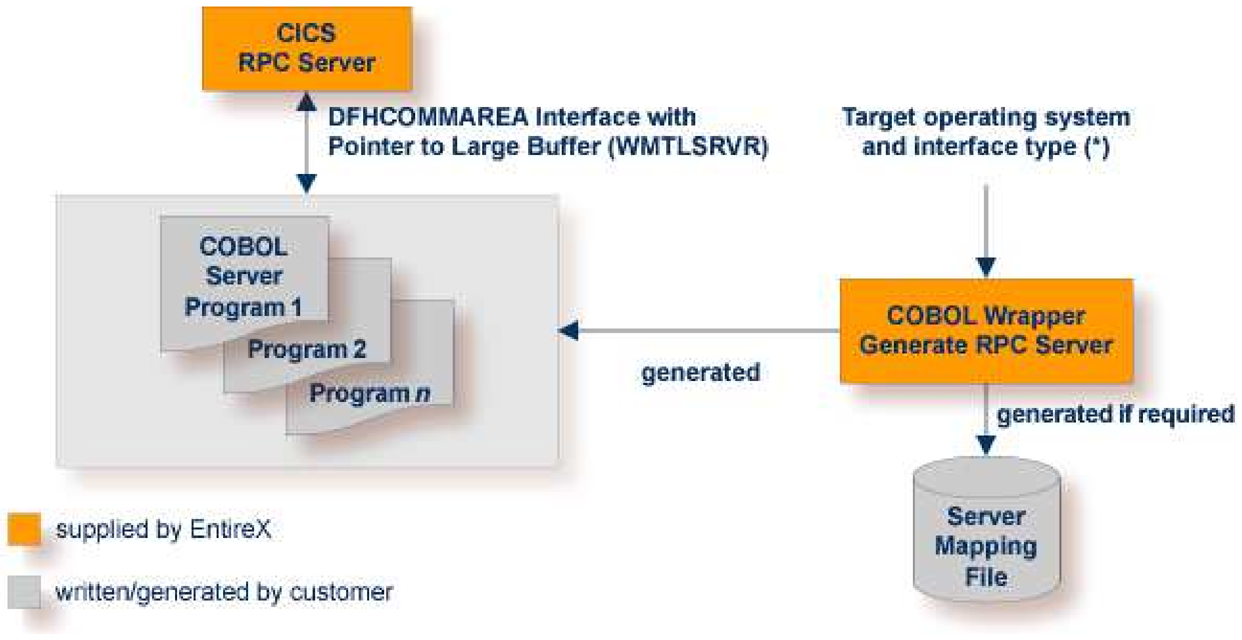
7. Provide your server(s) to the CICS RPC server.

   - Install your server(s) as separate CICS program(s).

   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding program can be found, the access will fail.
   - If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize Automatically Generated Server Names*) is executed. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for CICS with DFHCOMMAREA Large Buffer Interface (z/OS and z/VSE)

This mode applies to z/OS and z/VSE. See also *COBOL Scenarios* under z/OS | z/VSE | CICS ECI in the CICS RPC Server documentation.


CICS RPC Server
DFHCOMMAREA Interface with Pointer to Large Buffer (WMTLSRVR)
Target operating system and interface type (*)
COBOL Server Program 1
Program 2
Program *n*
COBOL Wrapper Generate RPC Server
generated
generated if required
Server Mapping File
supplied by EntireX
written/generated by customer

(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all your server's parameters dynamically in the format required. Your server is called by `EXEC CICS LINK`. Within the DFHCOMMAREA, pointers are passed to a large input/output buffer.

Use the COBOL Wrapper for CICS with DFHCOMMAREA large buffer interface in the following situations:

- You need to migrate COBOL programs implemented with webMethods WMTLSRVR interface to the CICS RPC server.
- You require more than 31 KB of data to transfer to your server.
- You cannot use the channel container calling convention because your IDL does not match the applicable rules; see *CICS Channel Container IDL Rules* under Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS). There are no IDL restrictions for this interface type - every IDL can be used.
- You prefer this interface type rather than the channel container interface type.
- You do *not* require a distributed program link (CICS DPL) to your server.

To use the COBOL Wrapper for CICS with large buffer interface

1. Generate the server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "CICS with DFHCOMMAREA large buffer interface". See *Generating COBOL Source Files from Software AG IDL Files*.

2. The generated server mapping file has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* (z/OS | z/VSE) in the CICS RPC Server documentation.

   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping* and *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server documentation.

5. Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6. Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file (see `program-definition`).

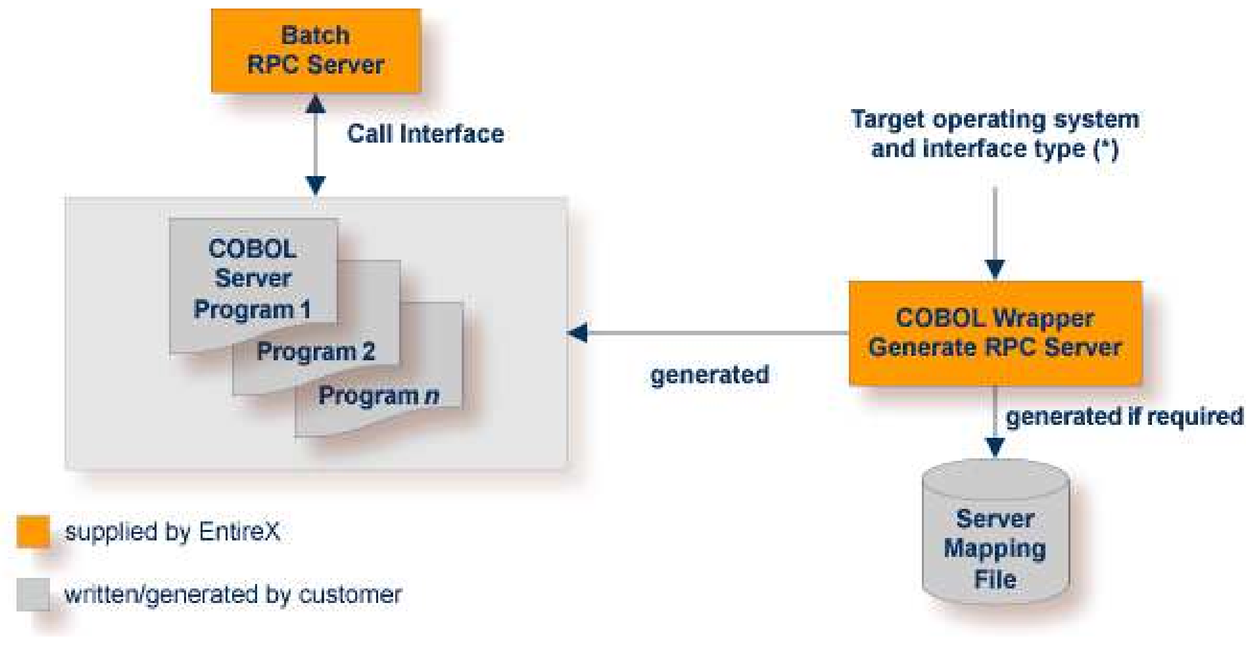
7. Provide your server(s) to the CICS RPC server.

   - Install your server(s) as separate CICS program(s).

   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding program can be found, the access will fail.
   - If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize Automatically Generated Server Names*) is executed. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)

This mode applies to z/OS, BS2000/OSD, z/VSE and IBM i. See also *COBOL Scenarios* (z/OS | BS2000/OSD | z/VSE) in the Batch RPC Server documentation.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In batch mode, the RPC server sets up all of your server's parameters dynamically in the format required. Your server is called dynamically using standard call interfaces.

Use the COBOL Wrapper for batch to build servers for the Batch RPC server.

To use the COBOL Wrapper for batch

1. Generate a server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "Batch with standard linkage calling convention". See *Generating COBOL Source Files from Software AG IDL Files* for details.

2. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* for z/OS (Batch, IMS) | BS2000/OSD | z/VSE in the RPC server documentation.

   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select* *the Connection Type* in the Integration Server Wrapper documentation.

See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.
4. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping*.

   Under **z/OS**

   - See *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the Batch RPC Server documentation.

   Under **IBM i**, consider multithreading issues:

   - Your server has to be implemented as an ILE COBOL program of type *PGM.
   - The RPC server is running in a multithreaded environment. Therefore your server must be thread-safe. This implies that all commands and subprograms accessed in your servers must allow multithreads.
   - Please note that some COBOL statements do not support multithreads. Using statements that are not thread-safe (e.g. `STOP RUN`) can result in the RPC server ending abnormally. Therefore the server programs have to be terminated with a thread-safe statement, for example `EXIT PROGRAM`. For details, see the IBM documentation *Language Restrictions under THREAD and Preparing ILE COBOL Programs for Multithreading*.
5. Use a COBOL compiler supported by the COBOL Wrapper to compile your server.

   Under **BS2000/OSD,**

   - the IDL types U or UV require a compiler that supports COBOL data type NATIONAL. See *BS2000/OSD Prerequisites* for more information on supported compilers.
   - compile them as OM or LLM modules.

   Under **IBM i,**

   - use the IBM i command `CRTCBLMOD` (create bound COBOL module).
   - as an alternative, you can compile and bind in one step, see the next step below.

   On all **other platforms,**

   - use the standard COBOL compiler of the target platform.
6. Link (bind) your server to an executable program. Give the resulting server program the same name as the `program-name` in the IDL file. See `program-definition` under *Software AG IDL Grammar*.

Under **BS2000/OSD**:

- There is no need to link the server modules with the BS2000/OSD Common Runtime Environment (CRTE). The CRTE is included in the server's BLSLIB chain and loaded dynamically. If this is needed for any reason, the CRTE must be linked as a subsystem. All entries must be hidden to prevent duplicates. Linking the CRTE statically will consume resources and slow down the load time of the server modules.

Under **IBM i**:

- Bind it as a dynamically callable program of type *PGM using the command `CRTPGM`.
- As an alternative to compiling with `CRTCBLMOD` (see step above) and binding with `CRTPGM` separately, you can compile and bind in one step with the command `CRTBNDCBL`.
- When linking/binding servers, the CRTPGM parameter `ACTGRP (*CALLER)` must be specified. This guarantees that the server application runs in the same activation group as the calling RPC server.

On all **other platforms**

- Use the standard linker (binder) of the target platform.

7. Provide your server to the Batch RPC Server.

   Under **IBM i**

   - Put the server into a library whose name corresponds to the library name in the IDL file (see `library-definition`).
   - If you put the server program into a library other than the library name given in the IDL (e.g. *MyLib*), you must tell this to the RPC server, using the server parameter `Library=Fix(MyLib)`. In this case, the library name sent with the client request is ignored.

     Example: If a client performs an RPC request that is based on the IDL program name CALC in the IDL library EXAMPLE, the remote RPC server will dynamically try to execute the ILE program CALC in the IBM i library EXAMPLE. If no corresponding program can be found, the access will fail.

   On all **other platforms**

   - Add the server to the Batch RPC Server STEPLIB chain.
   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding program can be found, the access will fail.
   - If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize*

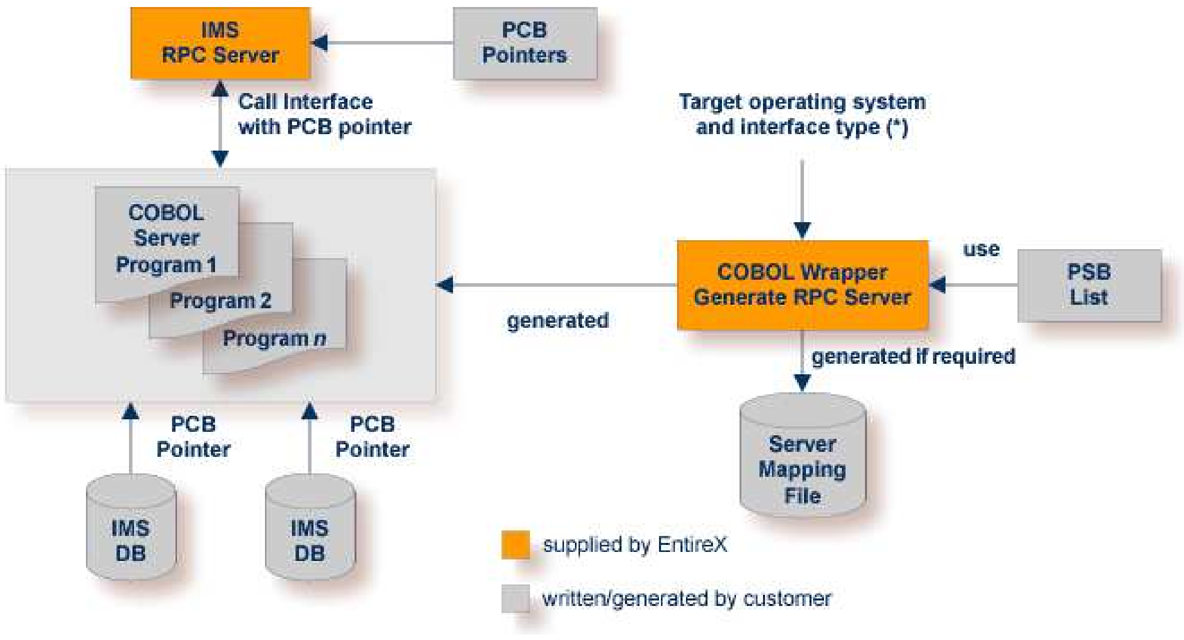
Automatically Generated Server Names) is executed. If no corresponding program can be found, the access will fail.

- If neither a server-side nor client-side mapping file is used - for example it is not required or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition`) given in the IDL is ignored.

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for IMS BMP (z/OS)

This mode applies to z/OS IMS mode BMP. See also *COBOL Scenarios* in the IMS RPC Server documentation.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In IMS BMP, the IMS RPC server sets up all of your server's parameters dynamically in the format required. Your server is called dynamically using standard call interfaces. IMS-specific PCB pointers can be provided as parameters in the linkage section.

Use the COBOL Wrapper for IMS BMP if you need to

- access IMS BMP programs with standard linkage calling convention
- access IMS databases through IMS PCB pointers and to pass them via parameters in the linkage section
- access the IMS PCB pointer IOPCB, for example to print data or to start an asynchronous transaction

- use the COBOL/ DLI interface module "CBLTDLI" which requires PCB pointers in its interface.

If PCB pointers have to be provided as parameters in the COBOL linkage section of your server, your IDL must comply with the IMS PCB Pointer IDL rules listed below. If no PCB pointers are required, the rules can be skipped.

IMS PCB Pointer IDL Rules

- An IMS PSB list contains the PCB pointers of your environment:
  - The IMS PSB list is a text file and can be created with any text editor.
  - Only one PCB pointer is listed per line.
  - The PCB pointer `IOPCB` is always the first pointer in the IMS PSB list.
  - The PCB pointers (except `IOPCB`) match the related PSB generation for your server.
  - The PCB pointers listed match the PCB pointers provided at runtime to the IMS RPC server (including `IOPCB`) in number and sequence.
  - The IMS PSB list is assigned in the IDL properties, see *Generating COBOL Source Files from Software AG IDL Files* or IDL *Generation Settings - Preferences*. Example:

    ```
    IOPCB
    DBPCB
    ```

- PCB pointers are described in the IDL as parameters. Thus they can be accessed in your server as any other parameter. Additionally, the following is required:
  - IDL parameters that are PCB pointers are marked with the attribute `IMS` (see `attribute-list` under *Software AG IDL Grammar*).
  - IDL parameters that are PCB pointers must match a PCB pointer listed in the IMS PSB list, otherwise the IMS RPC server does not pass them as PCB pointers at runtime. This results in unexpected behavior. Example:

    ```
    Library ’IMSDB’ Is
       Program ’ IMSDB’ Is
          Define Data Parameter
          1 IN-COMMAND          (A3)   IN /* ADD, DEL, DIS
          1 IO-DATA                    IN OUT
            2 IO-LAST-NAME      (A10)
            2 IO-FIRST-NAME     (A10)
            2 IO-EXTENSION      (A10)
            2 IO-ZIP-CODE       (A07)
          1 DBPCB                       IN IMS /* this is a PCB pointer
            2 DBNAME            (A8)
            2 SEG-LEVEL-NO      (A2)
            2 DBSTATUS          (A2)
            2 FILLER1           (A20)
          1 OUT-MESSAGE         (A40)  OUT
          End-Define
    ```

To use the COBOL Wrapper for IMS BMP

1. Generate the server (skeleton(s)) for the target operating system "z/OS", use interface type "IMS BMP with standard linkage calling convention". If PCB pointers should be provided as COBOL linkage section parameters for your server, set the IMS PSB list; otherwise omit the IMS PSB list. See *Generating COBOL Source Files from Software AG IDL Files*.

2. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server*.

   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4. Use the generated server (skeleton(s)) and complete it by applying your application logic. You can use the IMS-specific PCB pointers in your server as usual. Note the information given in *Software AG IDL to COBOL Mapping* and *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the IMS RPC Server documentation.

5. Using a COBOL compiler supported by the COBOL Wrapper, compile your server.

6. Link (bind) the server to an executable program, using the standard linker (binder) of the target program.

   - Give the resulting server program the same name as the program in the IDL file (see `program-definition` under *Software AG IDL Grammar*).

7. Provide the server to the IMS RPC server.

   - Add the server to the IMS RPC server STEPLIB chain.

   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding program can be found, the access will fail.
   - If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize*

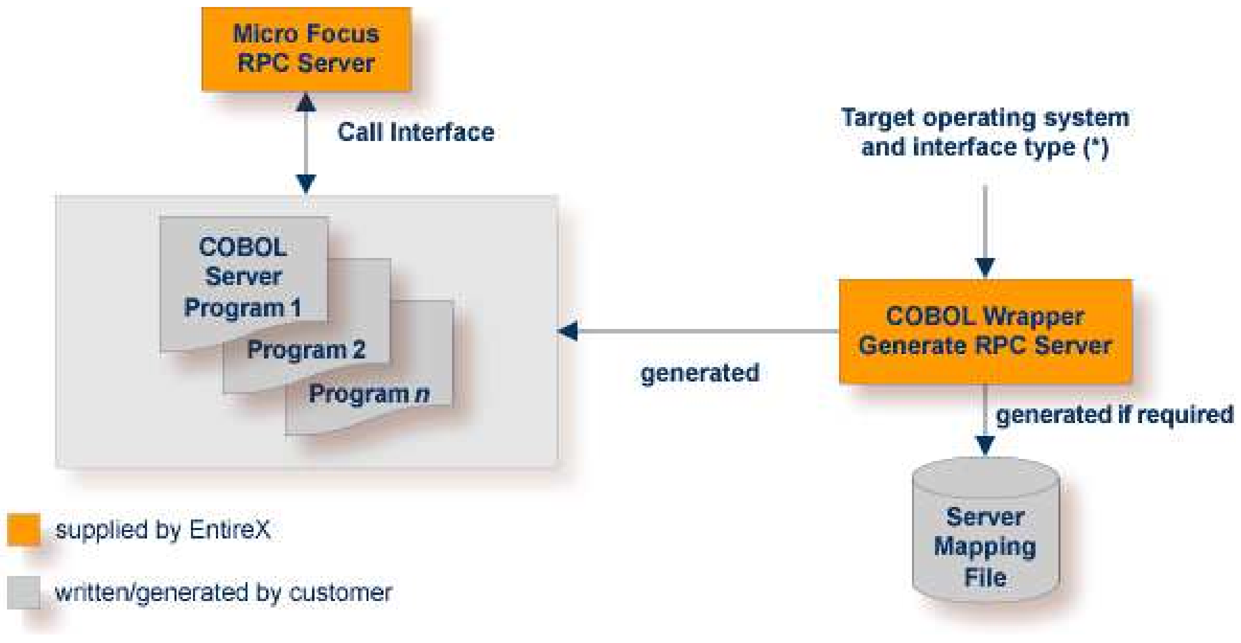
Automatically Generated Server Names) is executed. If no corresponding program can be found, the access will fail.

- If neither a server-side nor client-side mapping file is used - for example it is not required or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition`) given in the IDL is ignored.

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for Micro Focus (UNIX and Windows)

This mode applies to UNIX and Windows. See also *Scenarios and Programmer Information* in the Micro Focus RPC Server documentation.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

The Micro Focus RPC server sets up all of your server's parameters dynamically in the format required. Your server is called dynamically using standard call interfaces.

Use the COBOL Wrapperfor Micro Focus to build servers for the Micro Focus RPC server.

To use the COBOL Wrapper for Micro Focus

1. Generate a server (skeleton(s)) for the target operating system, for example "Windows", and use interface type "Micro Focus with standard linkage calling convention". See *Generating COBOL Source Files from Software AG IDL Files* for details.

2. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server*.
   - *Client*-side mapping files (.cvm): These are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4. Import the modules into your Micro Focus IDE.

5. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping*.

6. Compile and - if the format requires it - link (bind) and package your server(s) to one of the following formats:

   - Micro Focus intermediate code (int) or generated code (gnt). These formats can also be packaged into a Micro Focus library file (lbr). In this case the `program-name` (see `program-definition` under *Software AG IDL Grammar*) given in the IDL file must match the library file name. The `library-name` (`library-definition`) given in the IDL file is ignored and not used.
   - Under Windows to a DLL, and under UNIX to a shared library (so/sl). The `library-name` (`library-definition`) given in the IDL file must match the executables file name, and the `program-name` (see `program-definition`) given in the IDL file must match an entry point.

7. Provide your server to the Micro Focus RPC server.

   - Make sure your server(s) are accessible by the Micro Focus RPC server:
     - under UNIX, for example with the `LD_LIBRARY_PATH` environment variable
     - under Windows, for example with the `PATH` environment variable.
   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar*. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See *Customize Automatically Generated Server Names*. If no corresponding

program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from the RPC request and the program with the COBOL name defined in the server mapping, see *Customize Automatically Generated Server Names*) is executed. If no corresponding program can be found, the access will fail.

- If neither a server-side nor client-side mapping file is used - for example it is not required or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition`) given in the IDL is ignored.

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.