Software AG IDL to EJB Mapping

This chapter covers the following topics:

Mapping IDL Data Types to Java Data Types

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols [and] surround optional lexical entities.
- The informal term *`number`* (or *`number1`*`[.`*`number2`*`]`) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	Java Data Types	Note
`A`*`number`*	Alphanumeric	`String`	1, 3
`AV`	Alphanumeric variable length	`String`	
`AV[`*`number`*`]`	Alphanumeric variable length with maximum length	`String`	1
`B`*`number`*	Binary	`byte[]`	1, 6
`BV`	Binary variable length	`byte[]`	
`BV[`*`number`*`]`	Binary variable length with maximum length	`byte[]`	1
`D`	Date	`java.util.Date`	5
`F4`	Floating point (small)	`float`	2
`F8`	Floating point (large)	`double`	2
`I1`	Integer (small)	`byte`	
`I2`	Integer (medium)	`short`	
`I4`	Integer (large)	`int`	
`K`*`number`*	Kanji	`String`	1
`KV`	Kanji variable length	`String`	
`KV[`*`number`*`]`	Kanji variable length with maximum length	`String`	1
`L`	Logical	`boolean`	
`N`*`number1`*`[.`*`number2`*`]`	Unpacked decimal	`java.math.BigDecimal`	4
`NU`*`number1`*`[.`*`number2`*`]`	Unpacked decimal unsigned	`java.math.BigDecimal`	4
`P`*`number1`*`[.`*`number2`*`]`	Packed decimal	`java.math.BigDecimal`	4
`PU`*`number1`*`[.`*`number2`*`]`	Packed decimal unsigned	`java.math.BigDecimal`	4
`T`	Time	`java.util.Date`	5
`U`*`number`*	Unicode	`String`	7
`UV`	Unicode variable length	`String`	7
`UV`*`number`*	Unicode variable length with maximum length	`String`	7

Notes:

1. The field length is given in bytes.
2. If floating-point data types are used, rounding errors can occur. Therefore, the values of sender and receiver might differ slightly.
3. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for type A, a blank string will be used.
4. For Java, the total number of digits (`number1+number2`) is 99, which is the maximum that

EntireX supports. See *IDL Data Types*.

If you connect two endpoints, the total number of digits used must be lower or equal than the maxima of both endpoints. For the supported total number of digits for endpoints, see the notes under data types N, NU, P and PU in section *Mapping IDL Data Types* to target language environment C | CL | COBOL | DCOM | .NET | Java | Natural | PL/I | RPG | XML.

If you use the value null (null pointer) for direction `IN` (for `IN` and `INOUT` parameters), the value 0 (or 0.0) will be sent. See *Mapping the Direction Attributes In, Out, InOut* in the Java Wrapper documentation.

5. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for types D/T, the current date/time will be used. You change this with the property `entirex.marshal.date`. Setting `entirex.marshal.date=null` will map the value null to the invalid date 0000-01-01 of the RPC marshalling. This is the invalid date value in Natural, too. With this setting the invalid date as an output parameter will be mapped to null. The default is to map the invalid date to 0001-01-01.
6. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for type B, all binary values will be set to zero.
7. The length is given in 2-byte Unicode code units following the Unicode standard UTF-16. The maximum length is 805306367 code units.

Please note also hints and restrictions on the Software AG IDL data types valid for all programming language bindings. See *IDL Data Types*.

Mapping Library Name and Alias

The library name in the IDL file is mapped to the class name of the generated bean. See `library-definition` under *Software AG IDL Grammar*. For the bean, the names of the classes have the format *`library-name`*`Bean`.

The special characters ’#’ and ’-’ in the library name are replaced by the character ’_’.

If there is an alias for the library name in the `library-definition`, this alias is used "as is" to form the bean class name. Therefore, this alias must be a valid Java class name.

Example:

- library name `Hu#G-O` is converted to `EJBHu_g_oBean.class`

The library name is sent - without changes - to the server. The library alias is never sent to the server.

In the RPC server the library name sent may be used to locate the target server.

Mapping Program Name and Alias

The program name in the IDL file is mapped to method names within the generated Enterprise JavaBeans. See `program-definition` under *Software AG IDL Grammar*. To adhere to Java naming conventions, the program name is converted to lowercase.

The special characters '#' and '-' in the program name are replaced by the character '_'.

If there is an alias for the program name in the `program-definition`, this alias is used "as is" for the method name. Therefore, this alias must be a valid Java method name.

The program name is converted to uppercase before it is sent to the server. The program alias is never sent to the server.

The program name sent to the RPC server is used to locate the target server.

Mapping Parameter Names

The parameter names are mapped to fields inside the serializable input and output classes (see *Mapping the Direction Attribute In, Out, InOut*). The name of the input class is made up of the library name and the program name `<`*`library-name`*`><`*`program-name`*`>Input` and for the output class is `<`*`library-name`*`><`*`program-name`*`>Output`.

Example:

```
public class LibPgmInput implements Serializable {
 public String myInputString;
 public String myOutInString;
}
```

or

```
public class LibPgmOutput implements Serializable {
 public String myOutputString;
 public String myOutInString;
}
```

Mapping Fixed and Unbounded Arrays

Arrays in the IDL file are mapped to Java arrays. If an array value does not have the correct number of dimensions or elements, this will result in a `NullPointerException` or an `ArrayIndexOutOfBoundsException`. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters), an array will be instantiated.

Mapping Groups and Periodic Groups

Groups are mapped to serializable classes. The class name is the *<library name><program name><parameter name>*,

For example, the following Software AG IDL

```
Library 'Lib1130' Is
 Program 'X201G0' Is
  Define Data Parameter
   1 MyGroup
    2 MyLong (I4)
    2 MyFloat (F4)
   1 MyGroupAsString (A253)
   1 function_result (I4) Out
  End-Define
```

is mapped to the class

```
public class Lib1130X201g0MyGroup implements Serializable {
 public int mylong;
 public float myfloat;
}
```

Mapping Structures

Structures are mapped to serializable classes. The class name is the *<library name><struct name>*.

Mapping the Direction Attributes In, Out, InOut

IDL syntax allows you to define parameters as `IN` parameters, `OUT` parameters, or `INOUT` parameters (the default). This specification of the direction is reflected in the generated Enterprise JavaBeans as follows:

- IN parameters are sent from the RPC client to the RPC server. `IN` parameters are collected in the serializable input class (see *Mapping Parameter Names*).
- OUT parameters are sent from the RPC server to the RPC client. `OUT` parameters are collected in the serializable output class (see *Mapping Parameter Names*).
- INOUT parameters are sent from the RPC client to the RPC server and then back to the RPC client. `INOUT` are collected into both classes the serializable input class as well as into the serializable output class. (see *Mapping Parameter Names*).

Note:
Only the direction information of the top-level fields (level 1) is relevant. Group fields always inherit the specification from their parent. Any different specification is ignored.

See the `attribute-list` under *Software AG IDL Grammar* for the syntax on how to describe attributes in the Software AG IDL file and refer to the `direction` attribute.