Software AG IDL to .NET Mapping

This chapter covers the following topics:

Mapping IDL Data Types to .NET Data Types

The table below lists the metasymbols and informal terms that are used for the Software AG IDL.

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

Software AG IDL	Description	.NET Data Types	Note
A1	Alphanumeric	char or String/StringBuilder	1, 5
A*number*	Alphanumeric	String/StringBuilder	1
AV	Alphanumeric variable length	String/StringBuilder	1
AV[*number*]	Alphanumeric variable length with maximum length	String/StringBuilder	1
B1	Binary	byte or byte[]	6
B*number*	Binary	byte[]	
BV	Binary variable length	byte[]	2
BV[*number*]	Binary variable length with maximum length	byte[]	
D	Date	DateTime	3, 7
F4	Floating point (small)	float	
F8	Floating point (large)	double	
I1	Integer (small)	sbyte	
I2	Integer (medium)	short	
I4	Integer (large)	int	
K*number*	Kanji	String/StringBuilder	1
KV	Kanji variable length	String/StringBuilder	1
KV[*number*]	Kanji variable length with maximum length	String/StringBuilder	1
L	Logical	bool	
N*number1*[.*number2*]	Unpacked decimal	BigNumeric	9,10
		decimal	8,10
NU*number1*[.*number2*]	Unpacked decimal unsigned	BigNumeric	9,10
		decimal	8,10
P*number1*[.*number2*]	Packed decimal	BigNumeric	9,10
		decimal	8,10
PU*number1*[.*number2*]	Packed decimal unsigned	BigNumeric	9,10
		decimal	8,10
T	Time	DateTime	4,7

Notes:

1. `System.String` for direction `in`, otherwise `System.Text.StringBuilder` if `Default` is used for parameter `ATOSTRING`. If `String` is used for `ATOSTRING`, `System.String` is used everywhere, and if `StringBuilder` is used for `ATOSTRING`, `System.Text.StringBuilder` is used everywhere. See *Using the .NET Wrapper*.

2. Unsigned integer ranging from 0 to 255.

3. Count of days AD (anno domini, after the birth of Christ). The valid range is from 1.1.0001 up to 28.11.2737 (only the date part of `DateTime` is used).

4. Count of tenths of a second AD (Anno Domini, after the birth of Christ). The valid range is from 1.1.0001 00:00:00.0 up to 16.11.3168 09:46:39 plus 0.9 seconds.

5. If `-D A1TOCHAR=1` is defined in the `erxidl` call, `A1` is mapped to `char`, otherwise to `String/StringBuilder`.

6. If `-D B1TOBYTE=1` is defined in the `erxidl` call, `B1` is mapped to `byte`, otherwise to `byte[]`.

7. The Natural `DATE` type allows for the value 01.01.0000 to denote an undefined date. In order to avoid the .NET runtime throwing an exception when attempting to assign the invalid date value 01.01.0000 to a .NET `DateTime` variable, the .NET runtime converts an incoming neutral date/time value 01.01.0000 00:00:00.0 into the special .NET `DateTime` value `DateTime.MaxValue` - 1 tick (that is 31.12.9999:23:59:59.9999998). When this value is passed to the EntireX runtime to be sent to an EntireX RPC service, it is converted back into the neutral RPC date/time value 01.01.0000 00:00:00.0.

8. If the total number of digits (*`number1`*+*`number2`*) is equal to or lower than 28, mapping is to the .NET data type decimal.
9. If the total number of digits (*`number1`*+*`number2`*) is greater than 28, mapping is to the .NET class `BigNumeric`. See *`BigNumeric`* under *.NET Wrapper Reference*.

10. 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.

Please also note the hints and restrictions on the IDL data types valid for all programming language bindings as described under *IDL Data Types*.

Mapping Library Name and Alias

The library name as specified in the IDL file is sent from a client to the server. Special characters are not replaced. The library alias is not sent to the server.

In the RPC server, the IDL library name sent may be used to locate the target server. See *Locating and Calling the Target Server* under z/OS (CICS, Batch, IMS) | UNIX | Windows | Micro Focus | BS2000/OSD | z/VSE (CICS, Batch) | IBM i.

The name of the .NET server assembly must match the library name.

The library name as given in the IDL file is used to compose the names of the generated output files. See `library-definition` under *Software AG IDL Grammar*. Therefore the allowed characters are restricted by the underlying file system. The name is composed from `<library-name>.idl` to `<library-name>.cs` as default. The name of the client stub file can be changed by using the `-F` option of the `erxidl` command. See *Using the .NET Wrapper in IDL Compiler Command-line Mode*.

In accordance with the C# conventions, the class name is built as follows with the default setting `-PSANITIZE`:

- The initial character and characters following one of the special characters ’#’, ’$’, ’&’, ’+’, ’-’, ’_’, ’.’, ’/’ and ’@’ are converted to uppercase.
- All other characters are converted to lowercase.
- The special characters ’#’, ’$’, ’&’, ’+’, ’-’, ’_’, ’.’, ’/’ and ’@’ are removed.

Other special characters used in the library name are not changed and may lead to problems with your underlying file system and to compile errors.

If there is an alias for the library name in the `library-definition`, this alias is used "as is" to form the class name. Therefore, this alias must be a valid C# class name. To fully control the output, use alias names and do not use `SANITIZE`.

Examples:

`MY-CLASS` to `MyClass (class)`

`MY-CLASS alias YOUR_CLASS` to `YOUR_CLASS(class)`

Mapping Program Name and Alias

The program name is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server.

In the RPC server, the IDL program name sent is used to locate the target server. See *Locating and Calling the Target Server* under z/OS (CICS, Batch, IMS) | UNIX | Windows | Micro Focus | BS2000/OSD | z/VSE (CICS, Batch) | IBM i.

The program names as given in the IDL file are mapped to methods within the generated C# sources. See `program-definition` under *Software AG IDL Grammar*.

In accordance with the C# conventions method names are built as follows with the default setting `-PSANITIZE`:

- Characters are converted to lowercase with the following exceptions
  - The special characters ’#’, ’$’, ’&’, ’+’, ’-’, ’_’, ’.’, ’/’ and ’@’ are removed
  - The character following one of the special characters is converted to uppercase.

Other special characters used in the program name are not changed and may lead to compile errors.

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 C# method name. To fully control the output, use alias names and do not use `SANITIZE`.

Examples:

`MY-PROGRAM` to `MyProgram (method)`.

`MY-PROGRAM alias YOUR_PROGRAM` to `YOUR_PROGRAM(method)`.

Mapping Parameter Names

The parameter names as given in the `parameter-data-definition` of the IDL file are mapped to parameters of the generated C# methods.

In accordance with the C# conventions the parameter names are built as follows with the default setting `-PSANITIZE`:

- Characters are converted to lowercase except
  - The special characters ’#’, ’$’, ’&’, ’+’, ’-’, ’_’, ’.’, ’/’ and ’@’ are removed
  - The character following one of those special characters is converted to uppercase.

IDL files that use C# keywords (e.g. `string` or `float`) as parameter names are not supported. Do not use C# keywords such as `string` or `float` as parameter names. Modify your IDL file accordingly.

To fully control the output do not use `SANITIZE`.

Example:

`MY-PARAM` to `myParam (parameter)`

Mapping Fixed and Unbounded Arrays

Arrays in the IDL file are mapped to C# arrays. If an array value does not have the correct number of dimensions or elements, this will result in an exception. If the value `null` (null pointer) is used as an input parameter (for `IN` and `INOUT` parameters), an array will be instantiated by the runtime.

Mapping Groups and Periodic Groups

Groups in the IDL file are mapped to C# classes.

The namespace for group classes is `SoftwareAG.EntireX.NETWrapper.Generated.`*`filename`*`.Groups` on the client side, and `SoftwareAG.EntireX.NETWrapper.Server.`*`libraryname`*`.Groups` on the server side.

Mapping Structures

Structures in the IDL file are mapped to C# classes.

The namespace for structure classes is `SoftwareAG.EntireX.NETWrapper.Generated.`*`filename`*`.Structs` on the client side, and `SoftwareAG.EntireX.NETWrapper.Server.`*`libraryname`*`.Structs` on the server side.

See *Mapping Groups and Periodic Groups*.

Mapping the Direction Attributes In, Out, InOut

- `IN` parameters are implemented as normal parameters of the generated C# class method.
- `OUT` parameters are implemented as out parameters of the generated C# class method.
- `INOUT` parameters are implemented as ref parameters of the generated method.

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

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

Mapping the ALIGNED Attribute

Not supported.

Calling Servers as Procedures or Functions

The IDL syntax allows definitions of procedures only. It does not have the concept of a function. A function is a procedure which, in addition to the parameters, returns a value. Procedures and functions are transparent between clients and server, i.e. a client using a function can call a server implemented as a procedure and vice versa.

In C# a procedure corresponds to a method with result type void, a function returns a value of some type.

It is possible to treat an `OUT` parameter of a procedure as the return value of a function. The .NET Wrapper generates a method with a non-void result type when the following two conditions are met:

- The last parameter of the procedure definition is of type `OUT`;
- This last parameter of the procedure definition has the name `Function_Result`.

In this case no function parameter is generated for this `OUT` parameter.

See the .NET Wrapper example that comes with EntireX.