Software AG IDL to C Mapping

This chapter describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the C programming language. Please note also the remarks and hints on the IDL data types valid for all language bindings found in the IDL file.

This chapter covers the following topics:

- Mapping IDL Data Types to C Data Types
- Mapping Library Name and Alias
- Mapping Program Name and Alias
- Mapping Parameter Names
- Mapping Fixed and Unbounded Arrays
- Mapping Groups and Periodic Groups
- Mapping Structures
- Mapping the Direction Attributes In, Out, InOut
- Mapping the ALIGNED Attribute
- Calling Servers as Procedures or Functions

Mapping IDL Data Types to C 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 in some cases *number1*.*number2*) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	C Data Type	Note
A*number*	Alphanumeric	`unsigned char [`*number*`]`	1,3
		`char [`*number* `+ 1]`	1,3
AV	Alphanumeric variable length	`ERX_HVDATA`	4
AV*number*	Alphanumeric variable length with maximum length	`ERX_HVDATA`	3,4
B*number*	Binary	`unsigned char [`*number*`]`	3
BV	Binary variable length	`ERX_HVDATA`	4

Software AG IDL	Description	C Data Type	Note
BV*number*	Binary variable length with maximum length	ERX_HVDATA	4
D	Date	unsigned char [ERX_GET_PACKED_LEN(7)]	3,5,8,12,14,15
F4	Floating point (small)	float	11
F8	Floating point (large)	double	11
I1	Integer (small)	signed char	
I2	Integer (medium)	signed short	
I4	Integer (large)	signed long	
K*number*	Kanji	unsigned char [*number*]	3
KV	Kanji variable length	ERX_HVDATA	4
KV*number*	Kanji variable length with maximum length	ERX_HVDATA	3,4
L	Logical	unsigned char	6
N*number1*[.*number2*]	Unpacked decimal	double	9
		unsigned char[*number1* + *number2*]	7,9
NU*number1*[.*number2*]	Unpacked decimal unsigned	double	9
		unsigned char[*number1* + *number2*]	7,9
P*number1*[.*number2*]	Packed decimal	double	10
		unsigned char[ERX_GET_PACKED_LEN(*number1* + *number2*)]	8,10
PU*number1*[.*number2*]	Packed decimal unsigned	double	10
		unsigned char[ERX_GET_PACKED_LEN(*number1* + *number2*)]	8,10
T	Time	unsigned char[ERX_GET_PACKED_LEN(13)]	5,8,13,14,16
U*number*	Unicode	wchar_t[number]	2,17,18
		wchar [number + 1]	2,17,18
UV	Unicode variable length	ERX_HVDATA	4,17
UV*number*	Unicode variable length with maximum length	ERX_HVDATA	4,17,19

See also the hints and restrictions valid for all language bindings under *IDL Data Types*.

Notes:

1. It is possible to map the data type to the C data type `unsigned char[..]` without null termination (mainframe Natural, COBOL, PL/I style) or C style string (C data type `char[.. + 1]` with null termination). The mapping is controlled with the *Mapping Options* when you generate

source files from IDL. See *Generate C Source Files from Software AG IDL Files*.

2. It is possible to map the data type to the C data type `wchar_t[..]` without null termination (mainframe Natural, COBOL, PL/I style) or to a C style wide character string (C data type `wchar_t[.. + 1]` with null termination). The mapping is controlled with the *Mapping Options* when you generate source files from IDL. See *Generate C Source Files from Software AG IDL Files*.
3. The field length is given in bytes.
4. The data type `ERX_HVDATA` is an RPC-specific data type in the header file *erxVData.h* for handling variable-length data types. See *API Function Descriptions for Variable-length Data Types AV, BV, KV and UV*.
5. Date and Time values are mapped in mainframe-style packed format.
6. The binary value zero is treated as `FALSE`. Contents other than a binary zero are treated as `TRUE`.
7. Unpacked format is an internal representation of numbers with a specified number of digits used in mainframe environments (Natural, COBOL, PL/I).
8. Packed format is an internal representation of numbers with a specified number of digits used in mainframe environments (Natural, COBOL, PL/I). `ERX_GET_PACKED_LENGTH` is a macro within the header file *erx.h* used to evaluate the length of a field in bytes with packed contents.
9. It is possible to map the data type to double or unpacked. The mapping is controlled with the *Mapping Options* when you generate source files from IDL. See *Generate C Source Files from Software AG IDL Files*.

   - For unpacked, the total number of digits (*number1*+*number2*) is 99.
   - For double, it depends on your target environment how many significant digits are supported. In most environments 15 - 16 digits, which means for exact results the total number of digits (*number1*+*number2*) should be less than 16.

   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.

10. It is possible to map the data type to `double` or `packed`. The mapping is controlled with the *Mapping Options* when you generate source files from IDL. See *Generate C Source Files from Software AG IDL Files*.

    - For unpacked the total number of digits (*number1*+*number2*) is 99.
    - For double, it depends on your target environment how many significant digits are supported. In most environments 15 - 16 digits, which means for exact results the total number of digits (*number1*+*number2*) should be less than 16.

    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.

11. For `float` and `double`, rounding errors can occur, so that the values of senders and receivers might differ.
12. Count of days AD (anno domini, after the birth of Christ). The valid range is from 1.1.0001 up to 28.11.2737. Mapping of the number to the date in the complete range from 1.1.0001 on, follows the Julian and Gregorian calendar, taking into consideration the following rules:

1. Years that are evenly divisible by 4 are leap years.
2. Years that are evenly divisible by 100 are not leap years unless rule 3, below, is true.
3. Years that are evenly divisible by 400 are leap years.
4. Before the year 1582 AD, rule 1 from the Julian calendar is used. After the year 1582 AD, rules 1, 2 and 3 of the Gregorian calendar are used.

See the following table for the relation of the packed number to a real date:

Date / Range of Dates	Value / Range of Values
1.1.0000	0 (special value - no date)
undefined dates	1 - 364 (do not use)
1.1.0001	365
1.1.1970	719527 (start of C-time functions)
28.11.2737	999999 (maximum date)

13. Count of tenth of seconds 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 9:46:39 plus 0.9 seconds. See the following table for the relation of the packed number to a real time:

Time / Range of Times	Value / Range of Values
1.1.0000 00:00:00.0	0 (special value - no time)
undefined times	1 - 315359999
1.1.0001 00:00:00.0	315360000
1.1.1970 00:00:00.0	621671328000 (start of C-time functions)

14. The relation between the packed number of a Date and Time data type is as follows:

tenths of a second per day	=24*60*60*10 = 864000		
number of time	= number of date	* 864000	
315360000	= 365	* 864000	1.1.0001 00:00:00.0
621671328000	= 719527	* 864000	1.1.1970 00:00:00.0
number of date	= number of time	/ 864000	
365	= 315360000	/ 864000	1.1.0001
719527	= 621671328000	/ 864000	1.1.1970

15. The `no date` value is the internal state of a `#DATE` (Natural type D) variable after a `RESET #DATE` is executed within Natural programs. This internal state is not a valid date. The `no date` value can be transferred as the invalid date 1.1.0 from RPC clients to servers and vice versa. C Wrapper supports Natural `no date` value. C Wrapper passes 0 to the C application when `no date` is received. With the same value of 0, the C application can send `no date` to its partner (client or

server).

16. The `no time` value is the internal state of a `#TIME` (Natural type T) variable after a `RESET #TIME` is executed within Natural programs. This internal state is not a valid time. The `no time` value can be transferred as the invalid time 1.1.0 0:00:00.0 from RPC clients to servers and vice versa. C Wrapper supports Natural `no time` value. C Wrapper passes 0 to the C application when `no time` is received. With the same value of 0, the C application can send `no time` to its partner (client or server).
17. The Unicode encoding provided (on receive) or expected (on send) on the API depends on the width of the `wchar_t` data type in your environment:

Size of `wchar_t`	**UTF Version Used**
2 bytes (Windows and some UNIX environments)	UTF-16
4 bytes (some other UNIX environments)	UTF-32

    The endianness (big endian or little endian) of the Unicode encoding that is used (UTF-16-LE, UTF-16-BE, UTF32-LE or UTF32-BE) is the same as the hardware architecture of the machine.

18. In environments where `sizeof(wchar_t)` is 4 bytes, only Unicode characters from the Basic Multilingual Plane are supported (code points U+0000 to U+FFFF except U+D800 to U+DFFF reserved for leading and trailing surrogates). Unicode characters from Supplementary Planes (code points U+10000 to U+10FFFF) are not supported.
19. In environments where `sizeof(wchar_t)` is 4 bytes, the number of UTF-16 Unicode code points (after conversion from UTF-32 to UTF-16) must be less than or equal to *`number`*. Please note the following:

    - Unicode characters from the Basic Multilingual Plane (code points U+0000 to U+FFFF except U+D800 to U+DFFF, reserved for leading and trailing surrogates) require one code point in UTF-16
    - Unicode characters from Supplementary Planes (code points U+10000 to U+10FFFF) require two code points in UTF-16

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 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.

For the server interface object, the name is composed with a prefix D as `D`*`library-name`*`.c` and for the server as *`library-name`*`.c`. For the client interface object the name is composed with a prefix C as `C`*`library name`*`.c`. For both interface objects the same header file `C`*`library-name`*`.h` is also used. When the name of the generated sources is built, lower and uppercase characters are considered and the special characters '#', '$', '&', '+', '-', '.', '/' and '@' used in the name for libraries are replaced by the character underscore '_'. Other special characters used in the library name are not changed and may lead to problems with your underlying file system.

Aliases for the library name in the IDL file are not supported in C Wrapper language binding. See `library-definition`.

Examples:

- A library name of `#HU$GO.` results in `C_HU_GO_.c` and `C_HU_GO_.h` as the client interface object file name for the generated source.
- A library name of `#HU$GO.` results in `D_HU_GO_.c` and `C_HU_GO_.h` as the server interface object file name for the generated source.

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 functions within the generated C sources. See `program-definition` under *Software AG IDL Grammar*. When building function names, lower and uppercase characters are considered and the special characters '#', '$', '&', '+', '-', '.', '/' and '@' are replaced by the character underscore '_' valid for C names. Other special characters used in the program name are not changed and may lead to compilation errors when compiling the generated sources.

Aliases for the program name in the `program-definition` are not supported in C Wrapper language binding.

Example

- A parameter name of `#HU$GO.` results in `_HU_GO_` as the function name for the C programming language.

Mapping Parameter Names

The parameter names as given in the IDL file are mapped to parameters of the generated C functions. See `parameter-data-definition` under *Software AG IDL Grammar*. When building parameters, lower and uppercase characters are considered and the special characters '#', '$', '&', '+', '-', '.', '/' and '@' are replaced by the character underscore '_' valid for C names.

- **Examples:**
  A parameter name of `#HU$GO.` results in `_HU_GO_` as the parameter name for the C programming language.

Mapping Fixed and Unbounded Arrays

- Fixed arrays within the IDL file are mapped to fixed C arrays. The upper bound given in the IDL file is decremented by 1 because C arrays always start with the lower bound 0. For example the number `(I2/5)` in the IDL file will be mapped to `signed short number [4]`.

  See the *`array-definition`* under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe fixed arrays within the IDL file and refer to `fixed-bound-array-index`.

- Unbounded arrays within the IDL file are mapped to the `ERX_HARRAY` data type found in the header file `erxArray.h`. See *Using Unbounded Arrays* for more information.

  See the `array-definition` for the syntax of unbounded arrays within the IDL file and refer to `unbounded-array-index`.

Mapping Groups and Periodic Groups

Groups within the IDL file are mapped to the C data type `struct`. See the `group-parameter-definition` under *Software AG IDL Grammar* for the syntax on how to describe groups within the IDL file.

Mapping Structures

Structures within the IDL file are mapped to the C data type `struct` like groups. See `structure-definition` for the syntax on how to describe structures within the IDL file.

Mapping the Direction Attributes In, Out, InOut

The IDL syntax allows you to define parameters as `IN` parameters, `OUT` parameters, or `IN OUT` parameters (which is the default if nothing is specified). This direction specification is reflected in the generated C interface object as follows:

- Parameters with the `IN` attribute are sent from the RPC client to the RPC server. When the parameter is a simple parameter (that is, no fixed or unbounded array, no group and no structure) the parameter is provided with the method call by value. Complex parameters such as fixed and unbounded arrays, groups and structures are provided with the call by reference method.

- Parameters with the `OUT` attribute are sent from the RPC server to the RPC client. They are always provided with the call by reference method.

- Parameters with the `IN` and `OUT` attribute are sent from the RPC client to the RPC server and then back to the RPC client. They are always provided with the call by reference 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 the `attribute-list` for the syntax on how to describe attributes within the IDL file and refer to `direction` attribute.

Mapping the ALIGNED Attribute

The ALIGNED Attribute is not relevant for the programming language C. However, a C client can send the `ALIGNED` attribute to an RPC server where it might be needed.

See the `attribute-list` for the syntax of attributes in the IDL file and refer to the `aligned` attribute.

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. This means a client using a function can call a server implemented as a procedure and vice versa.

It is possible to call the remote procedure as a function and not as a procedure, if you prefer it and if it suits your interface.

Example

The function `float sin(float x)` will be called as a function - and not as a procedure - when defined in the IDL file as follows:

```
Library ... is
  Program 'sin' is
    Define Data Parameter
      1 x (F4) In
      1 Function_Result (F4) Out
    End-Define
```

it can be invoked as :

```
y = sin(x);
```

When you generate source files from IDL (see *Generate C Source Files from Software AG IDL Files*), a C function instead of a C procedure is generated if the following is true for the interface description in the IDL file:

- The last parameter's name is `function_result`. The name `function_result` is not case-sensitive.
- The last parameter's direction is `Out`. See `attribute-list`.
- The last parameter is a scalar variable, that is, not an array.

- The last parameter is of type and length:

Software AG IDL	Description	Note
`I1`	Integer (small)	2
`I2`	Integer (medium)	2
`I4`	Integer (large)	2
`A1`	Alphanumeric with length 1	2
`B1`	Binary with length 1	2
`L`	Logical	2
`N`*`number`* `[.`*`number`*`]`	Unpacked decimal	1, 2
`P`*`number`* `[.`*`number`*`]`	Packed decimal	1, 2
`NU`*`number`* `[.`*`number`*`]`	Unpacked decimal unsigned	1, 2
`PU`*`number`* `[.`*`number`*`]`	Packed decimal unsigned	1, 2

Notes:

1. The data types must be mapped to double. See *Mapping Options* when you generate C source files from Software AG IDL files.
2. The type of the function returned is defined by *Mapping IDL Data Types to C Data Types*.