Software AG IDL to Natural Mapping

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

This chapter covers the following topics:

- Mapping IDL Data Types to Natural Data Formats
- 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 Natural Data Formats

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	Natural Data Format	Notes
A*number*	Alphanumeric	A*number*	
AV	Alphanumeric variable length	A DYNAMIC	
AV*number*	Alphanumeric variable length with maximum length	A DYNAMIC	
B*number*	Binary	B*number*	
BV	Binary variable length	B DYNAMIC	
BV*number*	Binary variable length with maximum length	B DYNAMIC	
D	Date	D	3,5
F4	Floating point (small)	F4	2
F8	Floating point (large)	F8	2
I1	Integer (small)	I1	
I2	Integer (medium)	I2	
I4	Integer (large)	I4	
K*number*	Kanji	A*number*	1
KV	Kanji variable length	A DYNAMIC	1
KV*number*	Kanji variable length with maximum length	A DYNAMIC	1
L	Logical	L	
N*number1*[.*number2*]	Unpacked decimal	N*number1*[.*number2*]	6
NU*number1*[.*number2*]	Unpacked decimal unsigned	N*number1*[.*number2*]	6
P*number1*[.*number2*]	Packed decimal	P*number1*[.*number2*]	6
PU*number1*[.*number2*]	Packed decimal unsigned	P*number1*[.*number2*]	6
T	Time	T	4,5
U*number*	Unicode	U*number*	
UV	Unicode variable length	U DYNAMIC	
UV*number*	Unicode variable length with maximum length	U DYNAMIC	

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

Notes

1. Data type K is an RPC-specific data format that is not part of the Natural language.

2. When floating-point data types are used, rounding errors can occur, so that the values of senders and receivers might differ slightly. This is especially true if client and server use different representations for floating point data (IEEE, HFP).
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. 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)

4. Count of tenths 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)

5. 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

6. For Natural, the total number of digits (*number1*+*number2*) is 29. The supported number of digits after the decimal point (*number2*) depends on the target platform:

   - On UNIX or Windows it is 7.
   - On mainframes there is no additional restriction.

   In either case this is lower than the maximum of 99 supported by EntireX. 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.

Mapping Library Name and Alias

Client Side

If you are using client interface objects (recommended) generated for the client side (see *Using the Natural Wrapper for the Client Side*), the IDL library name as specified in the IDL file (there is no 8 character limitation) is sent from a Natural client to the server. Special characters are not replaced. The IDL library alias is neither sent to the server nor used for other purposes on the Natural client side.

If you are using instead so-called stubs generated with SYSRPC (not recommended) or stubless Natural RPC (also not recommended), the IDL library name as specified in the IDL file is not supported by Natural. By default, a Natural client sends the library name SYSTEM to the server. To send a library name other than SYSTEM from a Natural client to a server, the following steps are required for the client:

- Turn on the logon option.
- Call application programming interface USR4008N to specify the name of the library, otherwise the name of the current library is sent. The length of the library name sent is limited to 8 characters.

Server Side

A Natural RPC Server considers the library name (up to 8 characters) received from an RPC client only if the Natural Logon option is set by the sending RPC client.

- If the RPC client *does not* set its Natural Logon option, the Library name is ignored by the Natural RPC Server and the target RPC server (Natural subprogram, extension .NSN) is searched in the startup Natural library and defined steplibs of the Natural RPC server.
- If the RPC client *does* set its Natural Logon option, the target RPC server (Natural subprogram, extension .NSN) must reside in the Natural library, specified as IDL library name in the IDL file.

EntireX RPC client components provide a possibility to set the Natural Logon option. See *Natural Logon or Changing the Library Name* and the documentation of the EntireX RPC client component.

Mapping Program Name and Alias

Client Side

If you are using client interface objects (recommended) generated with the Natural Wrapper for the client side (see *Using the Natural Wrapper for the Client Side*), the IDL program name as specified in the IDL file (there is no 8-character limitation) is sent from a Natural client to the server. Special characters are not replaced. The IDL program alias is not sent to the server, but it is used to derive the suggestion for the source file names of the client interface objects (NSN, PDA, PGM) instead using the IDL program names. See *Step 2: Customize Natural Client Names*.

If you are using instead so-called stubs generated with SYSRPC (not recommended) or stubless Natural RPC (also not recommended) the IDL program name as specified in the IDL must match the name in your CALLNAT statement. In this case the IDL program is limited to 8 characters. Example:

```
CALLNAT 'MYSRV' P1 P2 P3
```

Server Side

If you are using a server mapping file (see *Server Mapping Files for Natural*), the target RPC server (Natural subprogram, extension .NSN) is located with the help of this file. This has the following advantages:

- IDL program names do not have to match the target RPC server (Natural subprogram, extension .NSN) names.
- Target RPC server names (Natural subprogram, extension .NSN) can be customized during wrapping (see *Step 2: Customize Natural Server Names*) or during extraction (see *Step 6: Redesign the Interface for Natural Subprograms (Optional)*).
- IDL program names are not limited to 8 characters.

The server mapping file is generated either during wrapping (see *Using the Natural Wrapper for the Server Side*) or during extraction (see *IDL Extractor for Natural*). It is wrapped into the RPC client components and the relevant information is sent from a client to the server. Therefore it is important to generate or extract the target Natural RPC (Natural subprogram, extension .NSN) server first, before creating any RPC client component.

If you are *not* using a server mapping file, the target RPC server (Natural subprogram, extension .NSN) must match the IDL program name. In this case the length of the IDL program name is limited to 8 characters.

Mapping Parameter Names

The parameter names as given in the IDL file are replaced by artificial names in the generated Natural interface object (stub subprogram). See `parameter-data-definition` under *Software AG IDL Grammar*.

Mapping Fixed and Unbounded Arrays

- Fixed arrays within the IDL file are mapped to fixed Natural arrays. The lower bound is set to 1 and the upper bound is set to the upper bound given in the IDL file.

  See *`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 Natural X-arrays. The lower bound is always fixed and set to 1.

  See `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 Natural groups. 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 Natural groups. See `structure-definition` under *Software AG IDL Grammar* 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 by Natural as follows:

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

- Parameters with the `IN` 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 OUT` attribute are sent from the RPC client to the RPC server and then back to the RPC client.
- 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` under *Software AG IDL Grammar* for the syntax on how to describe attributes within the IDL file and refer to `direction` attribute.

Note:
If you define an interface object layout in the Natural application SYSRPC, the meaning of the direction attributes `IN` and `OUT` are reversed compared to the IDL:

- `IN` in SYSRPC is `OUT` in IDL
- `OUT` in SYSRPC is `IN` in IDL

Mapping the ALIGNED Attribute

The `ALIGNED` attribute is not relevant for the programming language Natural. However, a Natural client can send the `ALIGNED` attribute to an RPC server where it might be needed. To do this you need a Natural interface object (stub subprogram) that has been generated from an IDL file.

See the `attribute-list` under *Software AG IDL Grammar* 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.

Client and Server Side

The Natural RPC does not support functions.