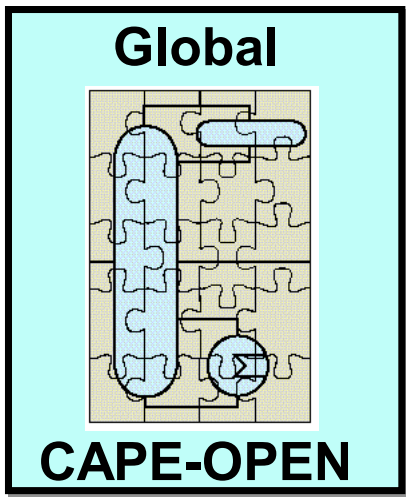


Global CAPE-OPEN

Delivering the power of component software
and open standard interfaces
in computer-aided process engineering



*Update on Types, Interfaces Naming
and Undefined Values*

Jean-Pierre Belaud, INPT

GCO-M&T-Update

15 June 2001

Archival Information

Reference	
Filename (if different)	
Authors	Jean-Pierre Belaud, INPT
Date	15 June 2001
Number of Pages	12
Version	2
Reviewed by (date)	Reviewed by (name) (greyed out means not yet) M&T group
Distribution	GCO Project
Additional Material	
Location on BSCW	GCO/ WORK PACKAGES /MANAGEMENT /M&T/02 M&T Technical Documents

Important Notices

Disclaimer of Warranty

Global CAPE-OPEN documents and publications include software in the form of *sample code*. Any such software described or provided by Global CAPE-OPEN --- in whatever form --- is provided "as-is" without warranty of any kind. Global CAPE-OPEN and its partners and suppliers disclaim any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by the Global CAPE-OPEN project --- remains with you.

Copyright © 2000 Global CAPE-OPEN and project partners and/or suppliers.
All rights are reserved unless specifically stated otherwise.

Global CAPE-OPEN is a collaborative research project established under BE 3512 "Industrial and Materials Technologies" (Brite-EuRam III), under contract BPR-CT98-9005

Summary

This document gives new updated guidelines for elementary types, interfaces naming and undefined values.

Contents

1. INTERFACE NAMING	4
2. ELEMENTARY TYPES.....	5
2.1 NEW TYPES	5
2.2 OTHER TYPES.....	5
2.3 ARRAY VERSUS SEQUENCE	5
3. UNDEFINEDVALUES	7
4. METHODS & PROPERTIES NAMING.....	9

1. Interface Naming

Why are we repeating Cape and the module name in each interface name?

CapeOpen::Unit::IPortCollection would be more logical than CapeOpen::Unit::ICapeUnitPortCollection for the CORBA naming rule. Thus the redundant information would be removed.

The idea is to suppress "Cape" and "Work Package name" within the interface name.

Solutions:

- ✓ To keep the interface naming even if this typing is not well adapted to CORBA
- ✓ To modify the interface naming, and detail a mapping rule between COM & CORBA
- ✓ To have one solution for COM, one for CORBA

According to Alexander, and with an academic point of view, he prefers, for the coming standard, to modify the interface naming and have a strict mapping rule between CORBA and COM. Concerning the existing standard, we can allow use of the old name and have a mapping between the old standard to the new one. This may be a little bit complex but this is easy to implement.

Due to the consequences resulted from a modification on interface naming, the M&T group decided not to modify the current rule even if the interface naming doesn't suit CORBA naming rule. So the first solution is decided.

There is no modification on interface naming. The current rule remains valid.

2. Elementary types

2.1 New types

Additionally to the list of existing elementary types, new elementary types are defined. Here is the updated list :

CAPE-OPEN	Analysis	COM	CORBA
CapeLong	long	long	long
CapeShort	short	short	short
CapeDouble	double	double	double
CapeFloat	float	float	float
CapeBoolean	boolean	VARIANT_BOOL	boolean
CapeChar	char	?	char
CapeString	string	BSTR	string
CapeDate	string date	DATE	string
CapeURL	URL string	BSTR	string
CapeVariant	void	VARIANT	any
CapeInterface	CO interface	LPDISPATCH	Object
CapeArrayT	array of T	VARIANT	sequence<T>

The libraries (COM TBL and CORBA IDL) include these elementary types.

2.2 Other types

It is may be useful to add some missing types among the 18 CORBA elementary types.

For example we could add:

```
long long    CapeLongLong
long double  CapeLongDouble
wchar        CapeWChar
wstring      CapeWString
```

As these types are not available for all platforms we choose not to add them at this time.

2.3 Array versus sequence

Note that we currently use two different naming conventions for a list:

In COM: the rule is *CapeArrayType*

i.e. *CapeArrayLong*

In CORBA: the rule is *CapeTypeSequence*

i.e. `typedef sequence<CapeLong> CapeLongSequence`

It was decided that a common naming convention has to be used. Therefore the term for the list of something will be the same for the analysis view and the implementation specification view (COM and CORBA).

We keep the COM rule. For CORBA we have now: `typedef sequence<Type> CapeArrayType`.

Note that we always use the *sequence* CORBA type and not the *array* CORBA type. The CORBA IDL library will be updated with respect to this new sequence naming.

3. Undefined Values

The PPDB Designer Team proposed the following empty elementary type definition:

```
#define      CapeFloatEmpty          - FLT_MAX

//This is the smallest floating point number("-" means "minus"), e.g. -1.7E+38

#define      CapeDoubleEmpty         - DBL_MAX
#define      CapeLongEmpty           LONG_MIN
#define      CapeShortEmpty          INT_MIN
#define      CapeStringEmpty         ""
#define      CapeArrayDoubleEmpty    NULL
#define      CapeArrayFloatEmpty     NULL
#define      CapeArrayLongEmpty      NULL
#define      CapeArrayShortEmpty     NULL
#define      CapeArrayStringEmpty    NULL
#define      CapeArrayEmpty          NULL
#define      sequenceEmpty           NULL
```

We list some drawbacks to this definition:

- ✓ From a conceptual view, the fact that we always returned a “real” value in order to say that the asked value does not exist is a source of Error. As a consequence of this choice, for each call, we should test the returned value to make sure that the value is not the empty one.
- ✓ The line directive from pre-processor #define is used. This definition could be replaced by basic comments (in fact as a requirement for empty values within the specification document; through a new annexe for instance) since the #define lines have no effect on the resulting preprocessed IDL file. Another solution would be to use definition of constants.
- ✓ This solution obliges us to fix values as conventions. Are we sure that these values will never be possible.
- ✓ It is said that "The values of these constants may be machine-dependent". Should these "constants" not be machine-dependant ? Indeed if the caller and the callee have different platform (32 and 64 bits for example), these values can be different. So the caller and the callee would have two different conventions for empty properties.

However in the CO specification, some information can be unavailable and that is not an abnormal process (CO error handling should not be applied). Therefore as a common specification for all interfaces the following undefined constants are defined :

COM & CORBA :

const	CapeFloat	CapeFloatUNDEFINED	=	NaN ¹
const	CapeDouble	CapeDoubleUNDEFINED	=	NaN ²
const	CapeLong	CapeLongUNDEFINED	=	-2 ³¹ ;
const	CapeShort	CapeShortUNDEFINED	=	-2 ¹⁵ ;
const	CapeChar	CapeCharUNDEFINED	=	'\0';
const	CapeArrayT	CapeArrayTUNDEFINED	=	NULL;

CORBA specific :

const	CapeString	CapeStringUNDEFINED	=	"";
const	CapeURL	CapeURLUNDEFINED	=	"";
const	CapeDate	CapeDateUNDEFINED	=	"";
const	CapeVariant	CapeVariantUNDEFINED	=	NULL;

COM specific :

const	CapeString	CapeStringUNDEFINED	=	NULL;
const	CapeURL	CapeURLUNDEFINED	=	NULL;
const	CapeDate	CapeDateUNDEFINED	=	NULL;
const	CapeVariant	CapeVariantUNDEFINED	=	VT_EMPTY;

This list of Undefined values will be included since version 0.9.4 of the Implementation Specification for COM and CORBA.

¹ NaN: Not a Number, defined by the IEEE single float format, e.g. 7fffffff (0111 1111 1000 000 0000 0000 0000 0000), Test for NaN of floating point number x: ((x & 78000000)>0)&&((x & 87fffffff)>0) returns true, if x is not a number; false otherwise. & is the bitwise AND, && the logical AND. See also function "isnan" of library libm.so of Solaris systems or equivalent libraries.

² NaN: Not a Number, defined by the IEEE double float format, e.g. 7fff ffff ffff ffff (0111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111), Test for NaN of floating point number x: ((x & 7ff0 0000 0000 0000)>0)&&((x & 800f ffff ffff ffff)>0) returns true, if x is not a number; false otherwise. & is the bitwise AND, && the logical AND. See also function "isnan" of library libm.so of Solaris systems or equivalent libraries.

4. Methods & Properties Naming

By and large, method names start with an Upper case, whereas property name start with an lower case. But there are some inconsistency in the naming for example, the properties get and set ComponentName and ComponentDescription start with an Upper Case.

All the names of CO operation should start with an upper case letter.