TSGS#25(04)0559

Source: SA5 (Telecom Management)

Title: Rel-6 CR 32.150 Add Style Guide for CORBA SS IDL

Document for: Approval

Agenda Item: 7.5.3

Doc-1st-Level	Doc-2nd-Level	Spec	CR	Rev	Phase	Subject	Cat	Ver-Cur	Wi
SP-040559	S5-046684	32.150	001	--	Rel-6	Add Style Guide for CORBA SS IDL	B	6.0.0	OAM-NIM

CR-Form-v7

CHANGE REQUEST

⌘ 32.150 CR 001 ⌘ rev - ⌘ Current version: 6.0.0 ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: | UICC apps⌘ ☐ ME ☐ Radio Access Network X Core Network X

Title:	⌘	Add Style Guide for CORBA SS IDL
Source:	⌘	**SA5** (edwin.tse@ericsson.com)
Work item code:	⌘	OAM-NIM
Date:	⌘	02/07/2004
Category:	⌘	**B**
Release:	⌘	Rel-6

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change:	⌘	Since SA5 in Release 6 will be producing some 20 Interface and NRM IRP CORBA SS specifications, it would be beneficial for the readers/users/implementers that the IDL statements of these specifications be structured in one logical pattern.
Summary of change:	⌘	Add correct reference to OMG specification. Add new informative Annex D.
Consequences if not approved:	⌘	Different Release 6 CORBA SS will use different and ad-hoc styles for specifying its IDL statements that can result in loss of readability.

Clauses affected:	⌘	2, B.2, B.4, New Annex D.

Other specs affected: ⌘	Y	N	
		X	Other core specifications
		X	Test specifications
		X	O&M Specifications

Other comments: ⌘

How to create CRs using this form:

Change in Clause 2

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[1] 3GPP TS 32.101: "Telecommunication management; Principles and high level requirements".

[2] 3GPP TS 32.102: "Telecommunication management; Architecture".

[3] 3GPP TS 32.151: "Telecommunication management; Integration Reference Point (IRP) Information Service (IS) template".

[4] 3GPP TS 32.152: "Telecommunication management; Integration Reference Point (IRP) Information Service (IS) Unified Modelling Language (UML) repertoire".

[5] ITU-T Recommendation M.3020: "TMN Interface Specification Methodology".

[6] OMG IDL Style Guide, ab/98-06-03, June 17, 1998

End of change in clause 2

Change in B.2

B.2 Rules for specification of CORBA Solution Sets

B.2.1 Introduction

This subclause identifies rules for specification of CORBA SSs. This subclause is mainly for 3GPP-internal use. It is only for information for the implementer of a CORBA SS.

B.2.2 Pragma prefix

All IDL-code shall define the pragma prefix using the following statement:

#pragma prefix "3gppsa5.org"

See D.1.4.3 for information of this `#pragma` statement in relation to other IDL statements.

End of change in B.2

Change (removal) of Clause B.4

~~B.4 IDL file name rule~~

~~CORBA IDL uses "#include "X"" statement where X is a name of a file containing IDL statements. In the CORBA Solution Set, IDL statements are specified.~~

~~The rule defined here specifies:~~

~~a) the IDL statements that shall belong to one file; and~~

~~b) the name of the file.~~

~~**Rule:**~~

~~In the annex where IDL statements are defined, use a special marker to indicate that a set of IDL statements shall be contained in one file. The name of the file shall be the name of the first IDL module of that set (of IDL statements). Multiple markers can be used.~~

End of change in Clause B.4

Change in (add a new) Annex D

Annex D (Informative): Style Guide for CORBA SS IDL

This clause is the style guide for writing IDL statements for Interface IRP and NRM IRP. The guidelines are largely based on the OMG IDL Style Guide (OMG document: ab/98-06-03) [6] with extensions for 3GPP IRP use.

The guide sets out consistent naming, structural conventions and usage of SS interface for the IDL in 3GPP IRP CORBA SS specifications.

D.1 Modules and File

D.1.1 Use of Modules

All declarations of IDL shall be contained in modules. No declarations of interfaces and definitions shall appear in the global scope.

Nesting modules is a useful technique when dealing with large namespaces to avoid name clashes and clarify relationships. A module nested within another module shall not have the same name as a top-level module in any other IRP CORBA SS specification.

D.1.2 File Names

CORBA SS specifications contain IDL statements.

The rule defined below specifies:

a) How to partition/extract these IDL statements to be placed in a file; and

b) How to name the file.

Note that IDL uses "`#include "X"`" statement where `X` is a name of a file containing IDL statements.

Rule:

In the annex where IDL statements are defined, use a special marker to indicate that a set of IDL statements shall be contained in one file. The name of the file shall be the name of the first IDL module, concatenated with four characters ì`.idl`î. Within a CORBA SS, multiple markers (implying multiple files), can be used.

It is not allowed to have an IDL module split into multiple files.

D.1.3 Include Conventions

All included IDL files shall be specified using the "…" form of `#include`. For example:

```
#include "ManagedGenericIRPConstDefs.idl"
```

D.1.4 File Structure

D.1.4.1 File Internal Identification

The first line of the IDL file shall contain ì`//File:`î followed by a single space followed by the name of the file. For example,

```
//File: ExampleIRPConstDefs.idl
```

D.1.4.2 File Guard

An IDL file shall use a *guard* (consisting of three preprocessor lines) to avoid multiple definition errors. An example of a guard for the file called `TestManagementIRPConstDefs.idl` is:

```
#ifndef _TESTMANAGEMENTIRPCONSTDEFS_IDL_
#define _TESTMANAGEMENTIRPCONSTDEFS_IDL_

...remainder of the IDL

#endif // _TESTMANAGEMENTIRPCONSTDEFS_IDL_
```

D.1.4.3 Required Contents

If any other files are to be included, the `#include` statements come after the guard.

After `#include` lines, if any, and immediately before the `module` statement, the following line shall appear:

```
#pragma prefix "3gppsa5.org"
```

D.1.4.4 Example illustrating a File Structure

```
//File: ExampleIRPConstDefs.idl
#ifndef _EXAMPLE_IRP_CONST_DEFS_IDL_
#define _EXAMPLE_IRP_CONST_DEFS_IDL_

// This module describes/is part of…
#include "ExampleIncludeOne.idl"
#include "ExampleIncludeTwo.idl"

#pragma prefix "3gppsa5.org"
module ExampleIRPConstDefs {

// IDL Definitions here

};
#endif // _EXAMPLE_IRP_CONST_DEFS_IDL_
```

D.2 Identifiers

D.2.1 Mixed Case, Beginning Upper, No Underscores

The following categories of identifiers follow the *Mixed Case, Beginning Upper, No Underscores* rules:

- `module`
- `interface`
- `typedef`
- Constructed types (`struct, union, enum`)
- `exception`

The ìNo underscoresî rule is also applicable to all words that begin with an upper case letter with the remaining letters being lower case.

As a further note on naming, it is not necessary to append the value ì*Type*î to an identifier. The fact that it is a type is obvious from the consistent application of this naming convention.

Examples:

```
module PMIRPConstDefs(…);
interface AttributeNameValue(…);
```

D.2.2 Lower Case with Underscores

The following categories of identifiers follow the *Lower Case with Underscores* rules. All letters are lower case and words (if more than one) are separated with underscores.

- Operation name and notification name
- Attribute name
- Parameter name
- Structure member name

Examples:

```
get_notification_categories(…);
string comment_text;
void get_alarm_count (…, out unsigned long critical_count,..);
struct Comment {…; string user_id; string system_id;..};
```

D.2.3 Upper Case with Underscores

The following categories of identifiers follow *Upper Case with Underscores* rules. All letters are in upper case and words have an underscore separating them.

- Enum value
- Constant

Examples:

```
enum SubscriptionState {ACTIVE, SUSPENDED, INVALID};
const string JOB_ID = "JOB_ID";
```

D.2.4 Naming IDL Sequence Types

Typically a new type declared as an IDL sequence of another type will have the text ìListî appended to the name of the base type. Another convention is to declare such types as unordered sequences or ordered sets for consistency with ASN.1 notation. In this case they should have the ìSeqî or ìSetî (instead of ìListî) appended respectively.

Example of an ìordered setî:

```
typedef sequence <SubscriptionId> SubscriptionIdSet;
```

D.3 Interface IRP

Every Interface IRP should have 3 IDL modules (each specified in a separate IDL file):

```
module YyyIRPConstDefs {…};  // no change from Rel-5 practice.

module YyyIRPSystem {…}; // no change from Rel-5 practice.

module YyyIRPNotifications {…}; // new compared to Rel-5 practice
```

The first module defines all necessary IDL constructs, such as constant strings and type definitions, for the methods and notifications. The second module defines the methods. The third module defines the notifications.

D.3.1. Constant String and Type Definitions

This first module defines all necessary IDL constructs used by the methods (defined in the second module) and notifications (defined in the third module). The name of this module is `YyyIRPConstDefs` where `Xxx` is the name of the subject Interface IRP. An example is ì`PMIRPConstDefs`î.

Within this module, define data types used in the methods.

Also, define the data types of the attribute values used in the notifications.

CORBA SS authors should always check the generic types defined in ì`ManagedGenericIRPConstDefs`î before creating a new type.

For the attribute names of the structured notifications, define an interface `AttributeNameValue` that captures the string definitions. Make sure these definitions do not clash with those defined for the notification header, i.e. notification id, event time, system DN, managed object class and managed object instance (see `NotificationIRPNotification::Notify`).

An example from `PMIRPConstDefs`:

```
   /**
    * This block identifies attributes which are included as part of the
    * PMIRP. These attribute values should not
    * clash with those defined for the attributes of notification
    * header (see IDL of Notification IRP).
    */


   interface AttributeNameValue
   {
       const string JOB_ID = "JOB_ID";
       const string JOB_STATUS = "JOB_STATUS";
       const string REASON = "REASON";
       const string MONITOR_ID = "MONITOR_ID";
       const string MONITOR_STATUS = "MONITOR_STATUS";
   };
```

D.3.2 Operations

The second module defines the methods. The name of the module is `YyyIRPSystem` where `Yyy` is the name of the subject Interface IRP. An example is `AlarmIRPSystem`.

At the beginning of this module, define all required exceptions. Naming conventions for `exception` are covered in D.2.1 above. CORBA SS authors should always check if the generic exceptions defined in the `ManagedGenericIRPSystem` can be reused before declaring new `exception` types.

Then define one interface called `YyyIRP` encapsulating all methods of the subject `Yyy` Interface IRP. If the subject Interface IRP IS specifies that its `YyyIRP` inherits from `XxxIRP`, then reflect the inheritance relation in the interface definition. The following is an example of `AlarmIRP` that inherits from `ManagedGenericIRP.`

```
module AlarmIRPSystem
{
…
…
interface AlarmIRP : ManagedGenericIRPSystem:: ManagedGenericIRP {…};
…
};
```

Naming conventions for operations are covered in D.2.2 above.

D.3.3 Notifications

Use a separate module to define the notifications. The name the module is `YyyIRPNotifications` where `Yyy` is the name of the subject Interface IRP. Examples are `KernelCMIRPNotifications` and `PMIRPNotifications`.

For `NotificationIRPNotifications`, do:

- Define one IDL interface `Notify`. Capture the four constant strings that are the names of the four NV (name value) pairs of `filterable_body_field` of the CORBA structured event. These four CORBA NV pairs are mapped from the five notification header attributes (defined by the Notification IRP IS), i.e. the `objectClass, objectInstance, notificationId`, `eventTime` and `systemDN`.

For `YyyIRPNotifications` where `Yyy` is not `Notification`, do:

- At the beginning of this module, define the const strings for the notification types that correspond to the set of notifications specified by (and not inherited by and not imported by) the subject Interface IRP.

- Then define a number of IDL interfaces corresponding to notifications specified in the subject Interface IRP. These interfaces should inherit from `NotificationIRPNotifications`::`Notify`. Within each interface, the first IDL statement defines the notification type (that is used as the second field of the fixed header of the structured notification). The second and subsequent IDL statements define the attribute names of this notification type, excepting those already defined by `NotificationIRPNotifications`::`Notify`. The data type of the attribute value, which is defined in `YyyIRPConstDefs`, should be mentioned in the comment block of this IDL statement.

- Then define a number of IDL interfaces corresponding to notifications imported, if any. These interfaces should inherit from the imported interface. An example is `interface NotifyObjectCreation : KernelCMIRPNotifications:: NotifyObjectCreation`. Within this interface, define all necessary IDL constructs, if any, which are not defined in the imported interface. This interface may contain no IDL

statement if the IDL constructs defined in the imported interface are sufficient. For each interface imported, insert a comment ìThe first field of this notification carries the IRPVersion of this CORBA SS.î

- There is no need to re-define interfaces for notifications that are already specified in other Interface IRP, and from which the subject IRP inherits.

The following is an extract from `PMIRPNotifications`.

```
module PMIRPNotifications
{

    const string ET_MEASUREMENT_JOB_STATUS_CHANGED = "notifyMeasurementJobStatusChanged";
    const string ET_THRESHOLD_MONITOR_STATUS_CHANGED = "notifyThresholdMonitorStatusChanged";

    interface NotifyMeasurementJobStatusChanged: NotificationIRPNotifications::Notify
    {
      const string EVENT_TYPE = ET_MEASUREMENT_JOB_STATUS_CHANGED;

      /**
      * This constant defines the name of the jobId property,
      * which is transported in the filterable_body fields.
      * The data type for the value of this property
      * is PMIRPConstDefs::JobIdType.
      */
      const string JOB_ID = PMIRPConstDefs::AttributeNameValue::JOB_ID;

      …
      …
    };

    interface NotifyXXX : NotificationIRPNotifications::Notify
    {
        …
    };

    …
};
```

D.4 NRM IRP

Use one module to define the IDL constructs for the managed object classes. The name of this module is `XxxNRIRPConstDefs` where `Xxx` is the name of the subject NRM IRP.

An example is `UtranNRIRPConstDefs`.

Within the module, define a set of IDL interfaces each of which corresponds to a managed object class specified. The interface definition respects the inheritance relation specified. An example of managed object class `RncFunction`, which inherits from `GenericNRIRPConstDefs::ManagedFunction`, is shown below.

```
module UtranNRIRPConstDefs
{
      Ö

      /**
       *  Definitions for MO class RncFunction
       */
      interface RncFunction : GenericNRIRPConstDefs::ManagedFunction
      {
         const string CLASS = "RncFunction";

         // Attribute Names
         //
         const string rncFunctionId = "rncFunctionId";

         const string mcc= "mcc";
         const string mnc= "mnc";
         const string rncId= "rncId";
      };
…
};
```

Ö

End of Change in Annex D
End of Document

Annex D (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Dec 2003	S_22	SP-030613	--	--	Submitted to TSG SA#22 for Information	1.0.0	
Mar 2004	S_23	SP-040113	--	--	Submitted to TSG SA#23 for Approval	2.0.0	6.0.0