NOKIA

3GPP CORBA Bulk CM Northbound Interface Principles

Contents

List of tables

List of figures

1 About this document

This document describes the Nokia NetAct implementation of the 3GPP CORBA Bulk CM Northbound Interface as well as the Interface-N object model mapping between the Nokia object model and the 3GPP R6 object model.

The document is aimed at people integrating the Nokia NetAct 3GPP CORBA Bulk CM Northbound Interface.

1.1 NetAct compatibility and capacity information

For information on the NetAct system and capacity, and the compatibility between NetAct and network element releases, see the *NetAct Compatibility and Capacity Information* document.

1.2 Terms

For general information on terms, see Glossary, DN9763965.

Table 1. Terms

Term	Explanation
3GPP	Third Generation Partnership Project
Common	The term refers to 3GPP R6 specifications. For instance, “common parameters” equals parameters defined in 3GPP R6 bulk CM specifications
DN	Distinguished name, a name of a managed object which consists of a sequence of the relative distinguished names of its superiors in the naming tree, starting at the root and working to the managed object to be identified.
FBTS	Foreign BTS

Table 1. Terms (Continued)

Term	Explanation
Interface -N	An interface between a global Network Management System and a regional Element Management System
Main object, Related object	Two 3GPP objects that refer to the same Nokia object can be called a main object and a related object. The parent 3GPP object is the main object and the child object is the related object. For example, ManagedElement:RNC is a main object and its child, RncFunction, is a related object, because they both refer to the same RNC.
MO	Managed object
NM	Network Manager
NEM	Network element manager
NRM	Network resource model
NOM	Nokia Object Model (OSS4)
Non-network parameters	Parameters visible only in the NEM level, not in the network interface
OMG	Object Management Group
R6	Release 6 (of the 3GPP specifications)
R6OM	3GPP R6 Object Model
RRAC	Regional Radio Access Configurator
VSD	Vendor-specific data
IRP	Integration reference point, a collection of related integration interfaces that form a connection point between subsystems
IRPAgent	A subset of network (element) functions in a region. For instance, Regional Radio Access Configurator (RRAC).
IRPManager	A system that can act as a global-level network management system.

2 Introduction to 3GPP

The 3rd Generation Partnership Project (3GPP) is a collaboration agreement that brings together a number of telecommunications standard bodies. 3GPP has specified the interface-N which can be used to transfer network configuration data between:

- one global network management system and
- several regional network element managers (by different network vendors).

Parameters specified in the 3GPP specifications are the same for all vendors. In other words, all vendors have to support them to allow centralised base network configuration management.

3GPP CORBA Bulk CM Northbound Interface provides a network management interface for integration of the Radio Access Configurator regional cluster into a 3GPP-compliant network-wide configuration management system. The interface is provided for all networks (however, the current release supports only UMTS networks) and it covers 3GPP common objects and parameters as well as Nokia-specific objects and parameters. For the data exchange standard, 3GPP Bulk CM file format is used including vendor-specific data extensions to cover Nokia-specific objects and parameters.

2.1 Overview

The following graphic illustrates the basic idea in global NM and the position of the 3GPP CORBA Bulk CM Northbound Interface.

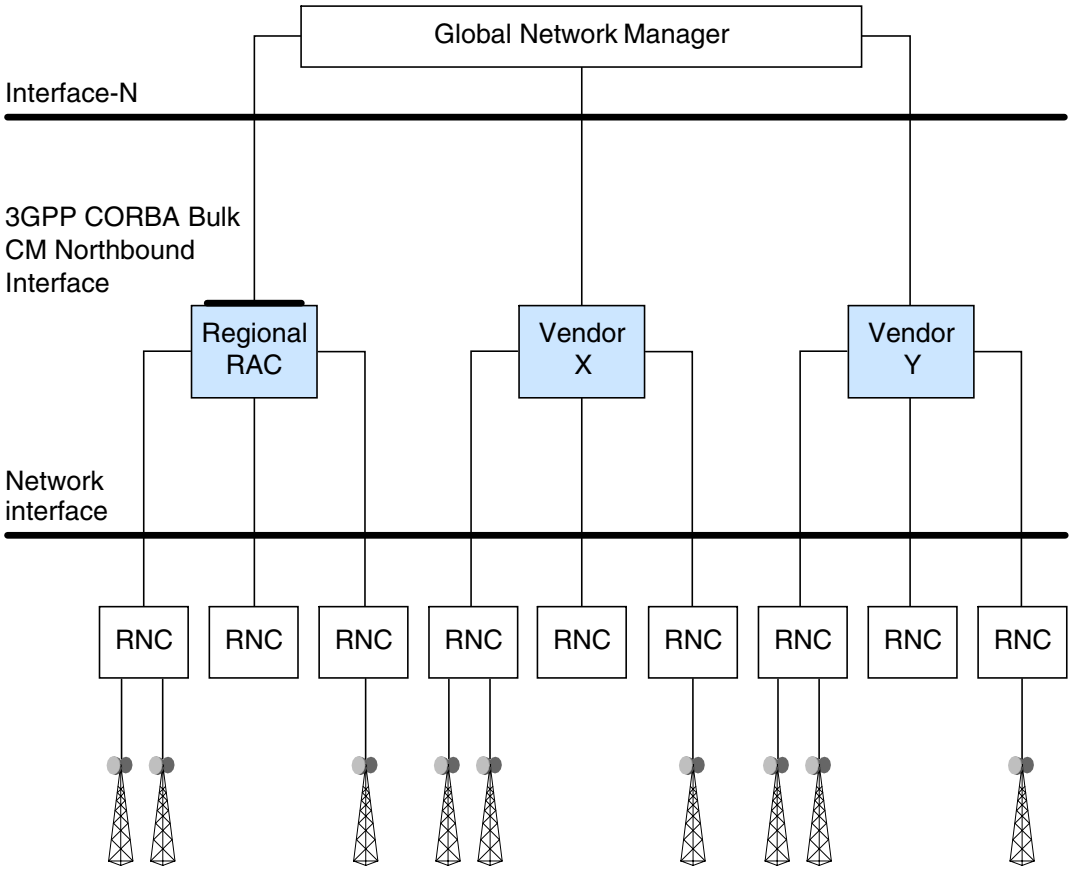

Figure 1. Overview of the global Network Manager

3 3GPP CORBA Bulk CM Northbound Interface

This section presents the 3GPP CORBA Bulk CM Northbound Interface.

3.1 IDL files

This section lists all IDL files that are required for 3GPP Bulk Configuration Management Northbound interfaces.

3.1.1 3GPP IDL files

These files are included in this document as appendixes.

- Appendix A BulkCmIRPConstDefs.idl
- Appendix B BulkCmIRPSystem.idl
- Appendix C BulkCMIRPNotifications.idl

3.1.2 Object Management Group IDL files

These IDL files can be downloaded from the Object Management Group Website at http://www.omg.org.

- `TimeBase.idl`
- `CosNotifyFilter.idl`
- `CosNotifyComm.idl`
- `CosNotifyChannelAdmin.idl`
- `CosNotification.idl`
- `CosEventComm.idl`
- `CosEventChannelAdmin.idl`

3.2 Supported network resource models

The NetAct 3GPP CORBA Bulk CM interface supports the 3GPP network resource model specified in the following 3GPP specifications:

- TS 32.622-6.4.0 (2005-03)
- TS 32.642-6.4.0 (2005-03)
- TS 32.652-6.4.0 (2005-03)

For these specifications, refer to 3GPP web pages in http://www.3gpp.org/.

The implementation supports 3G. For 2G, only `GsmRelation` and `ExternalGsmCell` are supported.

3.3 Used XML schemas

XML schemas should be used by the Global Network Manager system to validate the contents of configuration files before downloading them and after uploading them.

For the XML file format used in the Interface-N see http://www.3gpp.org/ftp/Specs/

- 3GPP TS 32.615-6.20 Bulk Configuration Management IRP: XML File Format Definition
- Parameter names defined in 3GPP TS 32.643-6.2.0 are used in parameter mapping (network resource models define some parameter names differently)

The names of the vendor specific schemas are formed from the following parts:

- object type
- RAN release
- NetAct release (or CD)

For example, the vendor specific schema file for WCEL RAN RN152ED2 in CD279 is `nokia_wcel_152.2_cd0279.xsd`, and the corresponding schema file name for RAN RN2.0 is `nokia_wcel_rn2.0_cd0279.xsd`. EWCE (`nokia_ewce_1.0_cd0279.xsd`) and EGCE (`nokia_egce_1.0_cd0279.xsd`) make an exception because they exist only in the NetAct database but not in the network.

The latest versions of XML schemas provided by Nokia for generic and vendor-specific data are available via Nokia Online Services, NOLS, at http://www.online.nokia.com. Authentication to access NOLS is required.

3.4 Functionality overview

This section gives an overview of the 3GPP CORBA Bulk CM Northbound interface functionality.

For operations between the global NM and the regional RAC, as well as operations between the regional RAC and the network, see Radio Access Configurator Principles.

3.4.1 System roles

The system roles in the interface-N CORBA communication are:

- The regional RAC system acts as a 3GPP CORBA IRPAgent.
- The global NM system acts as a 3GPP CORBA IRPManager.

3.4.2 IRP-related functionality

This section presents the IRP-related functionality of the 3GPP CORBA Bulk CM Northbound Interface.

The Bulk Configuration Management IRP-related functionality

Table 2. The Bulk Configuration Management IRP-related functionality

Information Service operation / notification in 3GPP TS 32.612	Solution Set method in TS 32.613
startSession	start_session
endSession	end_session
upload	upload
download	download
validate [1]	validate
preactivate [1]	preactivate
activate [2]	activate
fallback [2]	fallback

Table 2. The Bulk Configuration Management IRP-related functionality

Information Service operation / notification in 3GPP TS 32.612	Solution Set method in TS 32.613
abortSessionOperation	abort_session_operation
getSessionIds	get_session_ids
getSessionStatus	get_session_status
getSessionLog	get_session_log
getBulkCmIRPVersion	get_bulk_CM_IRP_versions
getNotificationProfile[3]	get_notifications_profile
getOperationProfile[3]	get_operation_profile

[1] The validate and pre-activate operations are optional in 3GPP and are not supported in NetAct.

[2] The activate and fallback operations cannot be aborted.

[3] The get_notifications_profile and get_IRP_operation_profile operations are optional and not supported in NetAct.

3.5 Getting started

A 3GPP IRPManager that needs to communicate with the IRPAgent through the 3GPP CORBA Bulk Configuration Management Interface will first have to discover the agent, i.e. get the object reference for the NetAct CORBA servers implementing that interface.

Entry Point IRP

To get these object references, the 3GPP IRPManager has to connect to Entry Point IRP (EP IRP). The CORBA Naming Service Context of EP IRP can be uploaded via HTTP.

This can be achieved by opening a Telnet session to the Nokia NetAct osscore2 package on port 10115, or by opening a socket connection to the Nokia NetAct osscore2 package on port 10115 and then sending the following request:

```
GET /cgi-bin/get_nbif_ior
```

The 3GPP IRPManager retrieves and reads the IOR using methods described above. It then uses this IOR string to construct an object of type EPIRP by resolving naming context `rel_6.EPIRP` and then narrowing the object reference returned by resolver.

CM Bulk Data IRP

After the EPIRP object is constructed, the `get_IRP_reference()` method of EPIRP object is used to retrieve the IOR of desired interface. This method requires an irpId parameter which is the naming context of the interface. The naming context for CM Bulk Data is `BulkCmIRP`. Parameter `managerIdentifier` in method get_IRP_reference is an identifier of the IRP Manager. Any string can be used for this parameter. Parameter `systemDN` is the systemDN of the IRP Agent. An empty string can be used as the systemDN.

The returned IOR is placed in an inout parameter of the method call. This stringified IOR can then be converted to a CORBA object using the `string_to_object()` method of the ORB. Then this CORBA object is narrowed to the BulkCmIRP object.

The BulkCmIRP uses port 10200.

Notification IRP

CM Bulk Data IRP uses Notification IRP to notify the IRP Manager about events such as session state changes.

Notification IRP is retrieved through EPIRP in a similar fashion. The Naming Context is `NotificationIRP`.

After retrieving `Notification IRP`, IRPManager can subscribe to notifications. This is done by calling `attach_push()` method on the Notification IRP interface. This method requires following parameters:

Table 3. subscribe

Information Service operation parameter	Solution Set method parameter
managerReference	input parameter `manager_reference`, type `string`
timeTick	input parameter `time_tick`, type `long`
notification Categories	input parameter `notification_category_set`, type `NotificationIRPConstDefs::NotificationCategorySet`

Table 3. subscribe (Continued)

Information Service operation parameter	Solution Set method parameter
filter	input parameter `filter`, type `string`
subscriptionId	NotificationIRPConstDefs::SubscriptionId
	exceptions: Attach ParameterNotSupported InvalidParameter AlreadySubscribed AtLeastOneNotificationCategoryNotSupported

The parameter notification_categories defines which category notifications are subscribed. The category for CM Bulk Data notifications is `BulkCmIRP`.

Method call will return a reference value, which needs to be used when notifications are unsubscribed. This is done by calling method `detach_push()` on the Notification IRP interface.

Table 4. unsubscribe

Information Service operation parameter	Solution Set method parameter
managerReference	input parameter `manager_reference`, type `string`
subscriptionId	input parameter `subscription_id`, type `NotificationIRPConstDefs::SubscriptionId`
	exceptions: Detach InvalidParameter

3.6 Supported operations

The following operations are supported by Nokia:

- start_session
- upload

- download
- activate
- fallback
- abort_session_operation
- end_session
- get_session_ids
- get_session_status
- get_session_log
- get_bulk_CM_IRP_versions

For more information on the operations, refer to the 3GPP specification TS 32.612 in 3GPP web pages at http://www.3gpp.org/.

All operations listed in the following tables are supported by Nokia NetAct, unless otherwise stated in a footnote.

3.6.1 startSession

Note

This operation must be done before any of the other operations can be performed.

Table 5. startSession

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
	exceptions: StartSessionException SessionIdInUseException MaxSessionReachedException InvalidParameter

3.6.2 endSession

Table 6. endSession

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
	exceptions: EndSessionException UnknownSessionIdException NotValidInCurrentStateException InvalidParameter

endSession operation description

After the session is closed, all information related to the session is removed from the regional database. If a download operation was performed during the session, also the plan is removed from the database. The local copy of the session log is deleted.

3.6.3 upload

Table 7. upload

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
uploadDataFileReference	BulkCmIRPConstDefs::FileDestination sink

Table 7. upload (Continued)

Information Service operation parameter	Solution Set method parameter
baseObjectInstance	BulkCmIRPConstDefs::DistinguishedName base_object
scope, filter	BulkCmIRPConstDefs::SearchControl search_control
	exceptions: UploadException UnknownSessionIdException MaxSessionReachedException NotValidInCurrentStateException ConcurrencyException IllegalDNFormatException IllegalFilterFormatException IllegalScopeTypeException IllegalScopeLevelException IllegalURLFormatException InvalidParameter

Upload operation description

The upload operation consists of the following phases:

1. The global NM user initiates an upload operation.
2. The Regional RAC (RRAC) receives the upload operation request and reads the parameter settings in the upload operation request.
3. The RRAC reads the actual configuration from the regional database and converts it from Nokia object model to 3GPP NRM and writes the data to an XML file to a local working directory.
4. The generated XML file is transferred via FTP to the location specified by the IRPManager. After the transfer is completed, the local copy of the 3GPP XML file is deleted.
5. The RRAC sends a notification, `notifySessionStateChanged`, about a new session state.

Upload operation scope determination

Starting from the baseObjectInstance, the scope (that is, scopeType and level) is applied.

If you require more detailed control over the object classes included in the upload file, you can use a filter. The filter is a list of 3GPP object classes (separated with spaces) that should be included in the upload.

The filter overrides the ScopeType and level parameters; only the object classes specified in the filter are included in the upload. If you do not need a filter, you should give an empty filter string.

Examples:

- Uploading the whole Subnetwork:

```
baseObjectInstance = "SubNetwork=Nokia-1"
scope = ScopeType.BaseAll, level = 5
filter = ""
```

- Uploading only UtranCells,

  ExternalUtranCells and ExternalGsmCells:

```
baseObjectInstance = "SubNetwork=Nokia-1"
filter = "UtranCell ExternalUtranCell ExternalGsmCell"
```

Error situations in the upload operation

- The following errors cause the upload to fail:
  - An unrecoverable error happened when reading from the database.
  - An OMC object is missing from the database.
- In the following situations, some parameters are written to the XML file as empty:
  - If a target cell of an adjacency is not found from the database: parameter adjacentCell is left empty (for UtranRelation and GsmRelation)
  - If there is no COCO object connected to a WBTS, the following parameters are left empty:
    - NodeBFunction: nodeBFunction-iubLink
    - IubLink: iubLink-NodeBFunction
    - UtranCell: utranCell-IubLink
  - If there is no Site object attached to a Nokia managed object, the parameter locationName (for ManagedElement) is left empty.
  - If there is no Site object attached to the OMC object, the parameter locationName (for ManagementNode) is left empty.

Note

Additionally, if the parameters of some object cannot be read from the database, but that does not prevent any other objects from being uploaded, the upload is not aborted. The parameters of the object in question are simply skipped, and its child objects can still be uploaded.

3.6.4 download

Table 8. download

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
downloadDataFileReference	BulkCmIRPConstDefs::FileDestination sink
	exceptions: DownloadException UnknownSessionIdException MaxSessionReachedException NotValidInCurrentStateException IllegalURLFormatException InvalidParameter

Download operation description

To ensure data correctness and consistency and avoid problems with file handling in a region, the global NM tool (IRPManager) has to validate the XML files against the generic 3GPP and the vendor specific data schemas provided by Nokia before performing a download operation.

Note

If a plan with the same name already exists in the regional data repository when starting a download operation, the previous plan is deleted and a new plan is created.

The download operation consists of the following phases:

1. The global NM user initiates a download operation for plan file(s) validated against XML schemas.
2. The RRAC receives the download operation request and reads the parameter settings from the operation request.
3. The RRAC transfers the given 3GPP XML file to the region via FTP.
4. The RRAC converts the configuration data defined in the file from the 3GPP network resource model to the Nokia object model, and saves it as a plan in the regional database.

5. Possible errors in the file are written to the session log. Only fatal errors (for instance, database not available) cause the session state to be set to DOWNLOAD_FAILED.

   We recommend that you check the session log before proceeding with the activation, even if the state was DOWNLOAD_COMPLETED, to see that there were no problems when creating the plan in the regional database.

6. The name of the plan is written to the session log. The name is of the following format: GLOBAL _ <session ID>. For example, if the session ID is 1234, the plan name will be GLOBAL_1234.

7. After the file transfer, mapping and data storing are completed, the system sends a notification, `notifySessionStateChanged`, about a new session state.

After the operation is completed, the plan can be edited in the region with CM Editor, if changes are needed before activating the plan in the network. For more information on CM Editor, see CM Editor Help.

Error situations in the download operation

All error situations described below are written to the session log. The log also contains information on:

- the new plan name
- the objects that are successfully saved to the plan.

Error situations:

- Reading the XML file is stopped and the status of the download operation is set to DOWNLOAD_FAILED in the following situations:
  - The file contains some non-relevant, excess information that cannot be read by the XML reader.
  - The structure of the file is erroneous. For instance, `endConfig` data is called before all objects have been closed with `endManagedObject/endVSD`.
  - An unrecoverable error occurred when reading/writing into the database.
  - The vendor name in the file header is something else than Nokia.
  - The file format version in the file header is invalid.
  - The vendor specific data format version in any of the `VsDataContainers` is invalid.
  - Invalid type of root object (not SubNetwork).
  - Site reading, creation or attachment failed when creating a WBTS.
  - Reading templates failed when saving a creation operation.
  - Inconsistent modifiers with the main objects and related objects.
  - Inconsistent referenced objects with the main objects and related objects.

- Create operation for OMC (ManagementNode) or RNC. They can only be updated via the Interface-N.
- The object ID is not in the right form or it is out of range.
- The Nokia controller object is not found in the regional data repository (controller object holds the network element version information needed in creating a plan object).
- In creation, the object ID already exists in the Nokia SubNetwork.

- Reading the XML file continues and the status of the download operation is set to DOWNLOAD_COMPLETED in the following situations. The erroneous object and its children are skipped. That is, they are not written in to the regional plan.

  - The object is in a wrong place. For instance, UtranCell is under SubNetwork.
  - Unknown object type
  - Unknown modifier

- In the following situations, only the object is skipped, but its children are handled normally:

  - An unknown parameter name
  - Wrong type of a parameter value
  - A parameter value out of range
  - A dependant parameter could not be checked
  - An attempt to update read-only common parameters is made
  - There is more than one operation to the same 3GPP object in the download file.

3.6.5 activate

Table 9. activate

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
saveFallback	boolean fallback
	exceptions: ActivateException UnknownSessionIdException NotValidInCurrentStateException ConcurrencyException IllegalActivationModeException ParameterNotSupported InvalidParameter

Activate operation description

The system performs the following steps during the activation:

1. The global NM user initiates the activation operation.
2. The regional RAC (RRAC) receives the activation operation and reads the parameter settings in the operation (saveFallback) and saves them.
3. The RRAC starts provisioning (NWI3 download) to the mediators (RNCs) in the network. All RNCs included in the plan are activated in parallel.
4. After each RNC has received a plan and completion information is received, the system automatically requests NWI3 activation.
5. When the mediators (RNCs) receive the NWI3 activation operation, they first take a backup copy of the original network configuration, and then start the plan activation in the network.
6. Non-network parameters from the plan are merged to the actual configuration in the RRAC database.
7. When the activation is completed, the mediators send a notification to the regional RAC about completing the activation operation.

8. In RNCs version RN1.5.2ED2 and older, the regional RAC automatically performs a network upload to update the regional RAC actual data configuration according to the current network situation after the plan activation. The automatic network upload can be turned off in the 3GPP CM Interface -N configuration file.

   In RNCs version RN2.0, events from RNC take care of updating NetAct database so there is no need for an automatic network upload. However, the automatic network upload can be turned on in the 3GPP CM Interface -N configuration file.

9. When the network upload is completed, the regional RAC sends a notification, `notifySessionStateChanged`, to the global NM (IRPManager) about a new session state.

3.6.6 fallback

Table 10. fallback

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
	exceptions: FallbackException UnknownSessionIdException NoFallbackException NotValidInCurrentStateException ConcurrencyException InvalidParameter

Fallback operation description

As with the activate operation, the regional RAC automatically performs a network upload to update the regional RAC actual data configuration according to the current network situation after the fallback in RNCs version RN1.5.2ED2 and older.

In RNCs version RN2.0, events from RNC take care of updating NetAct database so there is no need for an automatic network upload.

The RNC supports only one fallback area, which implies that the fallback operation only applies to the latest activation operation performed on a given RNC. If another activation operation is performed after your activation, either from the global NM tool or from the region, it overrides your activation operation and you cannot perform fallback any more.

Note

Fallback is not supported for non-network parameters nor external GSM cells. For more information on non-network parameters, see *Interface-N: WCDMA (3G) Parameter Mapping*, DN03373371.

3.6.7 abortSessionOperation

Table 11. abortSessionOperation

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
	exceptions: AbortSessionOperationException UnknownSessionIdException NotValidInCurrentStateException InvalidParameter

The following operations can be aborted:

- upload
- download

The abort operation is not supported with Bulk CM operations activate and fallback. This is due to constraints from the network element side.

3.6.8 getSessionIds

Table 12. getSessionIds

Information Service operation parameter	Solution Set method parameter
sessionIdList	BulkCmIRPConstDefs::SessionIdList
	No error condition identified.

3.6.9 getSessionStatus

Table 13. getSessionStatus

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id
sessionState	BulkCmIRPConstDefs::SessionState
`status`	BulkCmIRPConstDefs::ErrorInformation error_information
	Exceptions: GetSessionStatusException UnknownSessionIdException InvalidParameter

3.6.10 getSessionLog

Table 14. getSessionLog

Information Service operation parameter	Solution Set method parameter
sessionId	BulkCmIRPConstDefs::SessionId session_id

Table 14. getSessionLog (Continued)

Information Service operation parameter	Solution Set method parameter
logFileReference	BulkCmIRPConstDefs::FileDestination sink
contentType	boolean `only_error_info` [1]
	exceptions: GetSessionLogException UnknownSessionIdException IllegalURLFormatException InvalidParameter

[1] Not supported by Nokia NetAct.

3.6.11 getBulkCmIRPVersion

Table 15. getBulkCmIRPVersion

Information Service operation parameter	Solution Set method parameter
versionNumberList	ManagedGenericIRPConstDefs::VersionNumberSet
	No error condition identified.

3.7 Supported notifications

3.7.1 Push and Pull style

Object Management Group (OMG) Notification Service defines two styles of interaction:

- push style
- pull style

Nokia NetAct only supports the push style.

3.7.2 notifySessionStateChanged

Table 16. notifySessionStateChanged

Solution Set Attribute	OMG CORBA Structured Event Attribute	Supported by NetAct	Comment
There is no corresponding Information Service attribute.	domain_name	Yes	Carries the IRP document version number string. Indicates the syntax and semantics of the Structured Event as defined by this document.
notificationType	type_name	Yes	Carries the string NOTIFY_SESSION_STATE_CHANGED. In the Information Service, this is defined as a Header attribute.
sessionState	event_name	Yes	Carries one of the following: UPLOAD_FAILED UPLOAD_COMPLETED DOWNLOAD_FAILED DOWNLOAD_COMPLETED ACTIVATION_FAILED ACTIVATION_PARTLY_REALISED ACTIVATION_COMPLETED FALLBACK_FAILED FALLBACK_PARTLY_REALISED FALLBACK_COMPLETED In the case of XXX_FAILED and XXX_PARTLY_REALISED, the name-value pair indicating ERROR_INFORMATION may be present
There is no corresponding Information Service attribute.	Variable Header		
managedObjectClass, managedObjectInstance	One name-value pair of filterable_body_fields	No	The order of the name-value pairs is not significant. The name and the value of a name-value pair are of the type string. The name of a name-value pair is the MANAGED_OBJECT_INSTANCE of interface AttributeNameValue of module NotificationIRPConstDefs. See encoding of this string in [5]. These are attributes of Header defined in the Information Service.

Table 16. notifySessionStateChanged (Continued)

Solution Set Attribute	OMG CORBA Structured Event Attribute	Supported by NetAct	Comment
notificationId	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the NOTIFICATION_ID of interface AttributeNameValue of module NotificationIRPConstDefs. The value of a name-value pair is of the type long. In the Information Service, this is defined as a Header attribute.
eventTime	One name-value pair of filterable_body_fields	Yes	The name of the name-value pair is the EVENT_TIME of interface AttributeNameValue of module NotificationIRPConstDefs. The value of the name-value pair is IRPTime. In the Information Service, this is defined as a Header attribute.
systemDN	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the SYSTEM_DN of interface AttributeNameValue of module NotificationIRPConstDefs. The value of the name-value pair is a string. In the Information Service, this is defined as a Header attribute.
sessionId	One name-value pair of filterable_body_fields	Yes	The name of the name-value pair is the SESSION_ID of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.
sourceIndicator	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the SOURCE_INDICATOR of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.
There is no corresponding Information Service attribute.	One name-value pair of filterable_body_fields		The name of the name-value pair is the ERROR_INFORMATION of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.

3.7.3 notifyGetSessionLogEnded

Table 17. notifyGetSessionLogEnded

Solution Set Attribute	OMG CORBA Structured Event Attribute	Supported by NetAct	Comment
There is no corresponding Information Service attribute.	domain_name	Yes	Carries the IRP document version number string. Indicates the syntax and semantics of the Structured Event as defined by this document.
notificationType	type_name	Yes	Carries the string NOTIFY_GET_SESSION_LOG_ENDED.
sessionLogStatus	event_name	Yes	Carries either the string GET_SESSION_LOG_COMPLETED_SUCCESSFULLY or GET_SESSION_LOG_COMPLETED_UNSUCCESSFULLY . In the case of the latter, the name-value pair indicating ERROR_INFORMATION may be present.
There is no corresponding Information Service attribute	Variable Header		
managedObjectClass, managedObjectInstance	One name-value pair of filterable_body_fields	No	The order of the name-value pairs is not significant. The name and the value of a name-value pair are of the type string. The name of an name-value pair is the MANAGED_OBJECT_INSTANCE of interface AttributeNameValue of module NotificationIRPConstDefs. See encoding of this string in [5]. These are attributes of Header defined in the Information Service.
notificationId	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the NOTIFICATION_ID of interface AttributeNameValue of module NotificationIRPConstDefs. The value of a name-value pair is of the type long. In the Information Service, this is defined as a Header attribute.
eventTime	One name-value pair of filterable_body_fields	Yes	The name of the name-value pair is the EVENT_TIME of interface AttributeNameValue of module NotificationIRPConstDefs. The value of the name-value pair is IRPTime. In the Information Service, this is defined as a Header attribute.

Table 17. notifyGetSessionLogEnded (Continued)

Solution Set Attribute	OMG CORBA Structured Event Attribute	Supported by NetAct	Comment
systemDN	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the SYSTEM_DN of interface AttributeNameValue of module NotificationIRPConstDefs. The value of the name-value pair is a string. In the Information Service, this is defined as a Header attribute.
sessionId	One name-value pair of filterable_body_fields	Yes	The name of the name-value pair is the SESSION_ID of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.
sourceIndicator	One name-value pair of filterable_body_fields	No	The name of the name-value pair is the SOURCE_INDICATOR of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.
There is no corresponding Information Service attribute	One name-value pair of filterable_body_fields		The name of the name-value pair is the ERROR_INFORMATION of interface AttributeNameValue of module BulkCMIRPConstDefs. The value of the name-value pair is a string.

4 Nokia Object Model RAN data

This sections presents some basic information on Nokia Object Model data.

For the allowed object identifiers, see Managed Object Reference.

For the associations between objects and data types, see WCDMA RAN RNW Parameters.

4.1 Naming conventions

The regional interface-N adaptation supports the naming conventions defined in the 3GPP specification *3GPP: 3G TS 32.300-400 (06/2001), Name Convention for Managed Objects*.

In the interface-N, a distinguished name (DN) is used to identify objects.

Nokia specific object names and parameter names for common parameters are used in Nokia Radio Access Configurator GUI. The conversion between R6OM and NOM is performed in the interface-N adaptation.

4.2 Associations between object models

The graphics in this section present the associations between the 3GPP object model and the Nokia object model.

Associations for Nokia PLMN managed objects

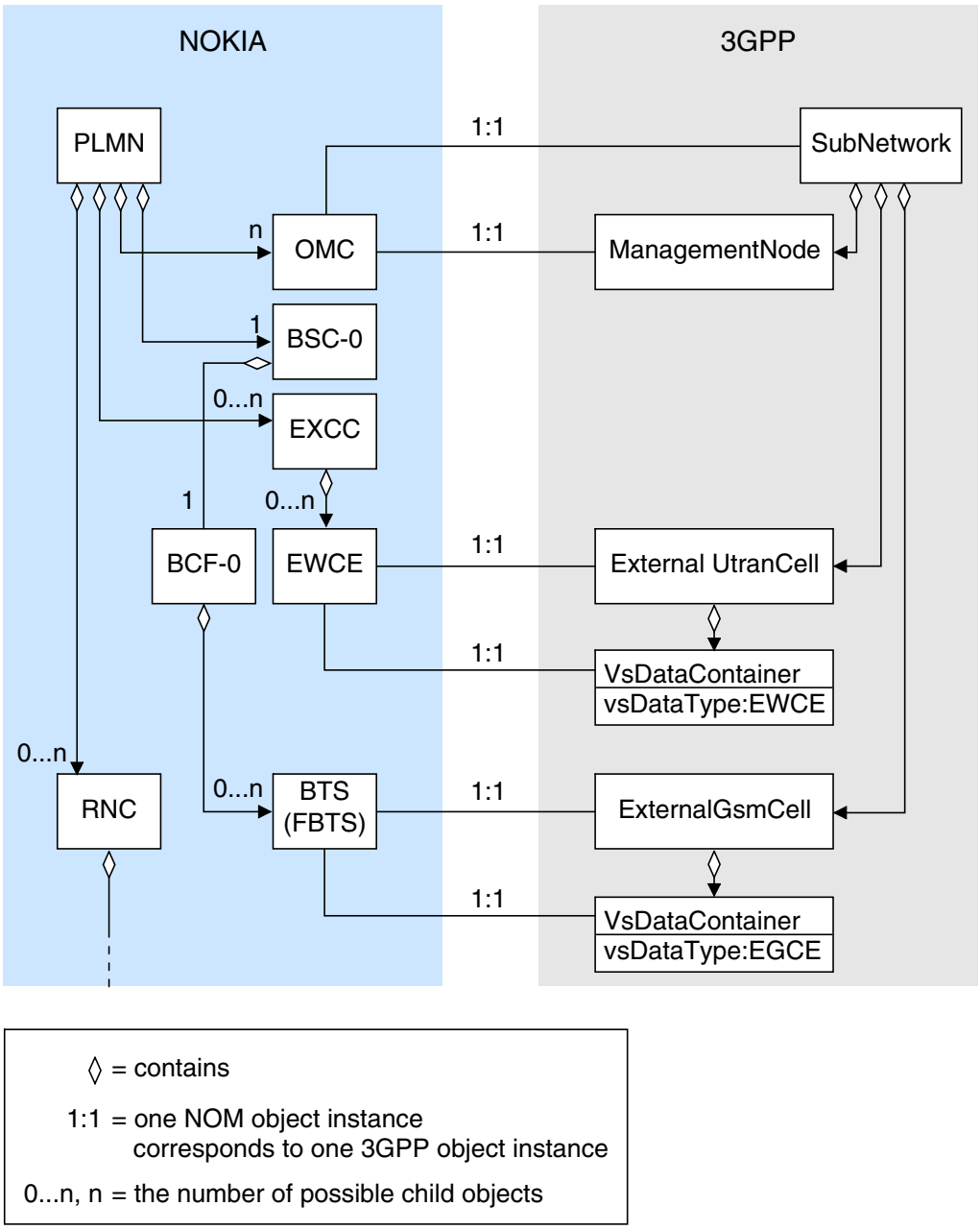

Figure 2. PLMN association

Associations for Nokia RNC managed objects

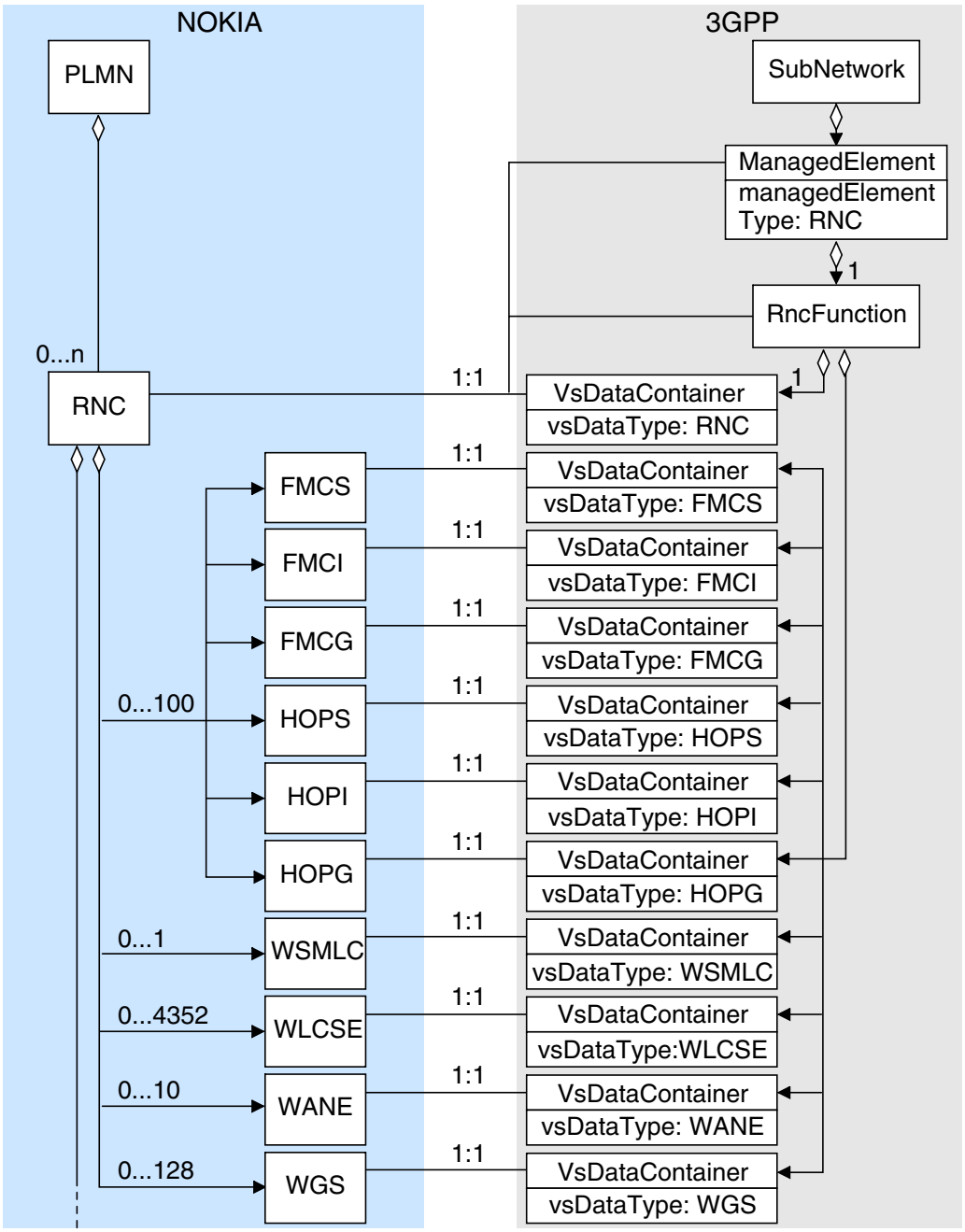

Figure 3. RNC associations

Associations for the IurItem parameter in the Nokia RNC

Note

The Nokia parameter IurItem (under the RNC) maps to the 3GPP class.

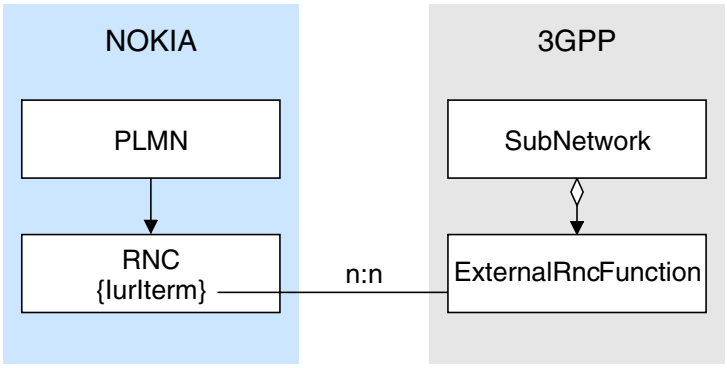

Figure 4. IurItem parameter association

The IurItem parameters under the RNCs map to the ExternalRncFunction with any duplicate parameter information removed.

Associations for Nokia WBTS managed objects

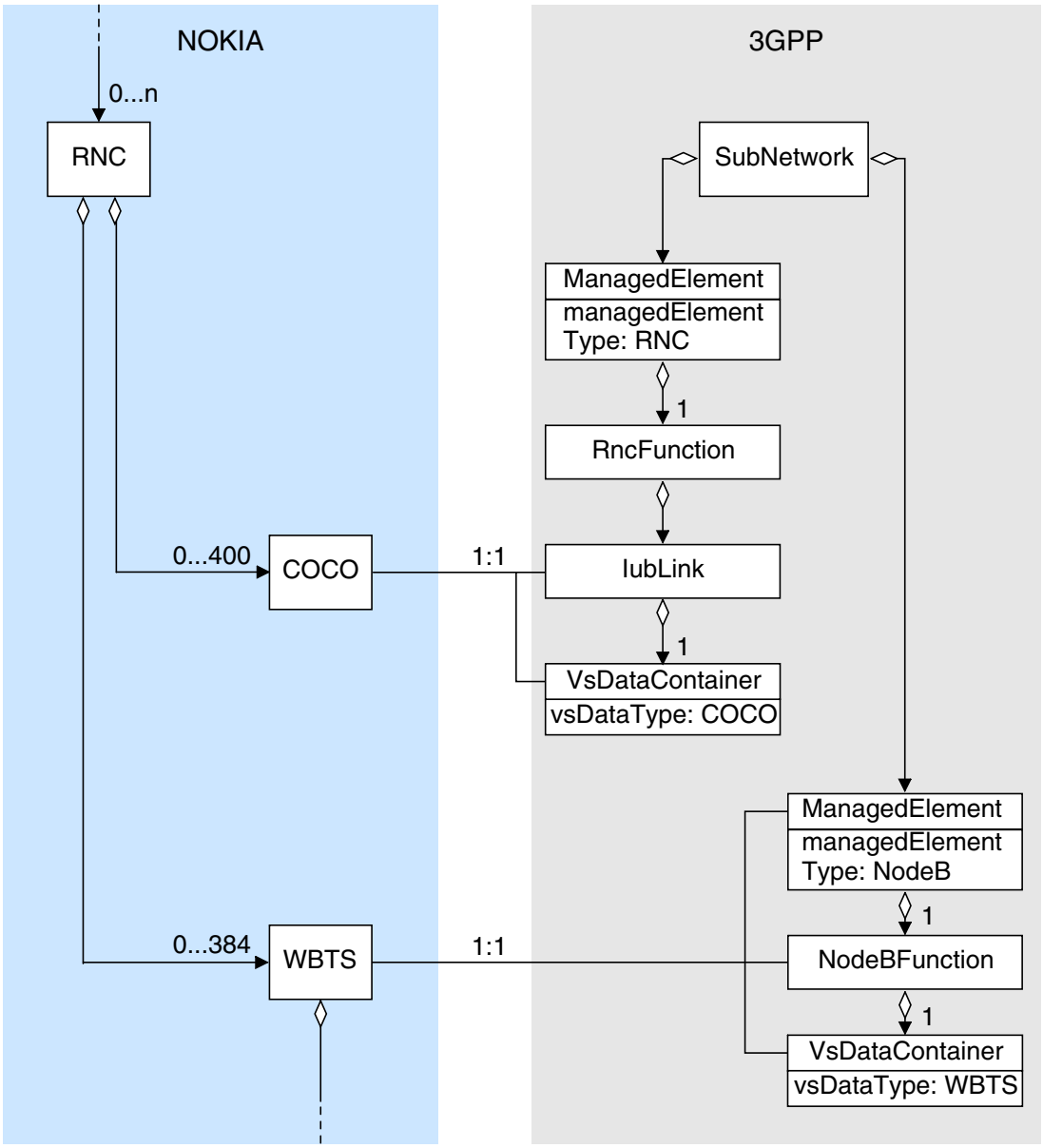

Figure 5. WBTS associations

Note

The total number of Nokia child managed objects per one RNC are defined in figure Data amount example, column "Objects total".

Associations for Nokia WCEL managed objects

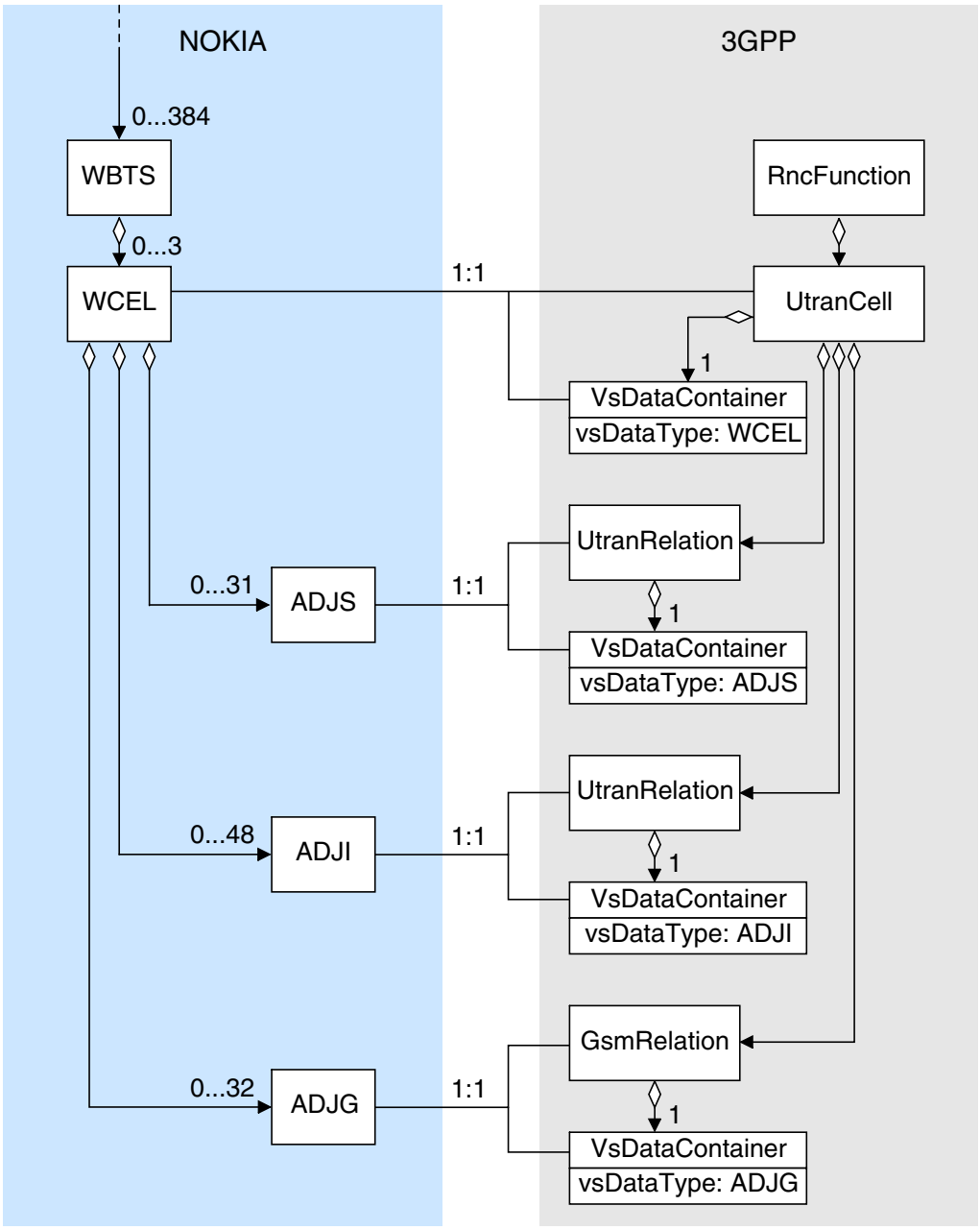

Figure 6. WCEL associations

Associations for Nokia ANTE managed objects

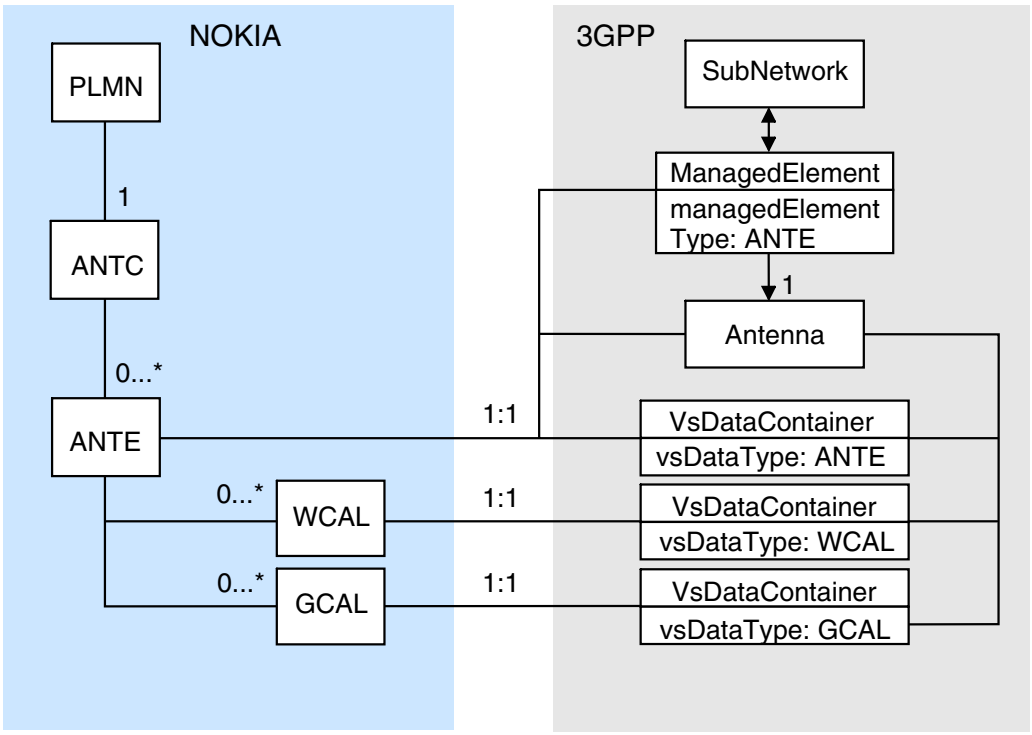

Figure 7. ANTE associations

4.3 Data amounts

The example case below presents data amounts for one RNC and the maximum number of child objects below it.

Note

External cells and other non-network objects are not included in this table.

Object	Number of objects / parent object	Objects total	Parameters / object	Parameters total
RNC	1	1	518	518
WBTS	384	384	49	18816
WCEL	3	1152	172	198144
ADJS	31	35712	17	607104
ADJI	48	55296	18	995328
ADJG	32	36864	15	552960
FMCS	100	100	23	2300
FMCI	100	100	24	2400
FMCG	100	100	24	2400
HOPS	100	100	18	1800
HOPI	100	100	17	1700
HOPG	100	100	10	1000
COCO	400	400	76	30400
WSMLC	1	1	5	5
WLCSE	4352	4352	34	147968
WANE	10	10	9	90
WSG	128	128	9	1152

Objects total:	134900

PARAMETERS TOTAL:	2564085

Lines in an XML file (one line per parameter):	2564085
Characters per line (average):	40
Bytes per line:	40

Total XML file size	102563400

File size in megabytes	103

Figure 8. Data amount example

5 Nokia OSS4 - 3GPP R6 object model mapping for WCDMA

This section describes the instance mappings between Nokia Object Model (NOM) and 3GPP R6 Object Model (R6OM).

5.1 Examples of instance mappings

This section presents examples of instance mappings.

In the column titled "Mapping" of the following table, different fonts are used for the attributes of the `3GPP model` and the attributes of the *Nokia model*.

Table 18. Examples of instance mappings

Object class	Mapping	Nokia DN	3GPP DN	Notes
OMC	`subNetworkId` =*"Nokia"+ clusterId (NmsId)*	PLMN-PLMN/OMC-1	SubNetwork=Nokia-1	`dnPrefix` can be set to NULL, since in this case the `SubNetwork` is not the "root of the local MIB" (PLMN is the "root of the local MIB").
OMC	`managementNodeId` =*omcRDN*	PLMN-PLMN/OMC-1	SubNetwork=Nokia-1, ManagementNode=OMC-1	
RNC	`managedElementId` = `rncFunctionId` = *rncRDN*	PLMN-PLMN/RNC-3	SubNetwork=Nokia-1, ManagedElement=RNC-3, RncFunction=RNC-3	Attribute `rncId` shall be set to *rncRDN*.

Table 18. Examples of instance mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
WBTS	managedElementId = nodeBFunctionId = *rncRDN* + "*/*" + *wbtsRDN*	PLMN-PLMN/RNC-3/WBTS-4	SubNetwork=Nokia-1, ManagedElement=RNC-3/WBTS-4, NodeBFunction=RNC-3/WBTS-4	NodeBFunctionId must be unique subnetwork wide as it is not under RNC in the 3GPP model. The RNC identity needs to be carried in nodeB and `nodeBFunctionRDN`, otherwise the reverse mapping is not possible.
WCEL	`utranCellId` = *wbtsRDN* + "*/*" + *wcelRDN*	PLMN-PLMN/RNC-3/WBTS-4/WCEL-5	SubNetwork=Nokia-1, ManagedElement=RNC-3, RncFunction=RNC-3, UtranCell=WBTS-4/WCEL-5	The WBTS identity needs to be carried in UtranCell RDN, otherwise the reverse mapping is not possible. *wcelRDN* = WCEL-"*LcrId*" (*LcrId* = `localCellId`)
ADJS	`utranCellId` = *wbtsRDN* + "*/*" + *wcelRDN*, `utranRelationId` = *adjsRDN*	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, UtranRelation=ADJS-1	`utranCellId` must also carry the information of the parent WBTS as it is not under NodeBFunction in the 3GPP model. Object class name prefixes used in `UtranRelation` and `GsmRelation` identifiers in R6OM to separate the relation type in NOM.

Table 18. Examples of instance mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
ADJI	`utranCellId` = *wbtsRDN* +"*/*" + *wcelRDN*, `utranRelationId` = *adjiRDN*	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJI-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, UtranRelation=ADJI-1	`utranCellId` must also carry the information of the parent WBTS as it is not under `NodeBFunction` in the 3GPP model. Object class name prefixes used in `UtranRelation` and `GsmRelation` identifiers in R6OM to separate the relation type in NOM.
ADJG	`utranCellId` = *wbtsRDN* + "*/*" + *wcelRDN*, `gsmRelationId` = *adjgRDN*	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJG-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, GsmRelation=ADJG-1	`utranCellId` must also carry the information of the parent WBTS as it is not under `NodeBFunction` in 3GPP model. Object class name prefixes used in `UtranRelation` and `GsmRelation` identifiers in R6OM to separate the relation type in NOM.
COCO	`IubLinkId` = *cocoRDN*	PLMN-PLMN/RNC-1/COCO-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, IubLink=COCO-1	
EXCC		PLMN-PLMN/EXCC-1		No mapping, conceptual object only

Table 18. Examples of instance mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
EWCE	externalUtranCellId = ewceRDN	PLMN-PLMN/EXCC-1/EWCE-1234	SubNetwork=Nokia-1, ExternalUtranCell=EWCE-1234	It is not necessary to carry `ExccId` in the 3GPP DN. Operators can define the `EWCE` instance number but the requirement is that the identification has to be unique in the `subNetwork`. The maximum character number is 10.
BTS (FBTS)	`externalGsmCellId` = *egceRDN*	PLMN-PLMN/BSC-0/BCF-0/BTS-123 321	SubNetwork=Nokia-1, ExternalGsmCell=EGCE-123 321	It is not necessary to carry `ExccId` in the 3GPP DN. Note: currently FBTS (foreign BTS) is used but it will be replaced with EGCE in the future NetAct/RAC releases. The EGCE/BTS identification has to include real cell LAC (5 digits) and Cell ID (5 digits) information, that is, the first 5 digits are reserved for LAC and the next 5 digits for cell ID. (Empty spaces left after the LAC digits if all 5 digits are not used, for instance, EGCE-123 321)

Table 18. Examples of instance mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
ANTE	`managementElementId` = *AnteFunctionId = anteRDN*	PLMN-PLMN/ANTC-1/ANTE-1	SubNetwork=Nokia-1, ManagedElement=ANTC-1/ ANTE -1, Antenna=ANTE-1, vsDataContainer=ANTE-1	In IOC (32.642-3) 3GPP specifications the name is AntennaFunction, but in Schema (32.645) the name is Antenna.
--	`externalRncFunctionId` = *RNC-->IurItem.NRncId*	--	SubNetwork=Nokia-1, ExternalRncFunction=RNC-1	`ExternalRncFunction` does not map directly to any Nokia class. It is mapped to a parameter (*IurItem*) under RNC.
--	`irpAgentId` = *systemId of OMC*	--	SubNetwork=Nokia-1, ManagementNode=OMC-1, IRPAgent=Nokia-1	Nokia does not have a corresponding object.

Notes about the mappings

- Embedding objects Nokia distinguished names (DNs) into the object instances used in 3GPP DN: some operations like object creation require that knowledge of Nokia internal hierarchy is present in 3GPP DNs, otherwise the object's place in the Nokia hierarchy cannot be determined.
- PLMN is only a conceptual root object in the Nokia model and therefore has no mapping.
- In all mappings involving `ManagedElement`, its `dnPrefix` can be set to NULL.
- Objects like `MeContext` in R6OM is not taken into account in the mapping. It is not used in the Nokia model.

Notes about the attribute value formats (attributes that are affected by the mappings):

- The `ManagementNode` attribute `manages` is set to the 3GPP DN of the managed `ManagedElement(s)`. The content of 3GPP DN must include the Nokia topology.

  `SubNetwork=Nokia-1,ManagedElement=RNC-3`

  `SubNetwork=Nokia-1,ManagedElement=RNC-3/WBTS-4`

- The `ManagedElement` attribute `managedBy` is set to the 3GPP DN of the controlling OMC.

- The `NodeBFunction` attribute `nodeBFunctionIubLink` is set to the 3GPP DN of the relating IubLink.

- The `IubLink` attribute `IubLinkUtranCell` is set to the 3GPP DN of the relating UtranCell(s). The content of the 3GPP DN has to include the Nokia topology.

  `SubNetwork=Nokia-1,ManagedElement=RNC-3,RncFunction=RNC-1,UtranCell=`WBTS-1/WCEL-1

- The `IubLink` attribute `IubLinkNodeBFunction` is set to the 3GPP DN of the relating `nodeBFunction`. The content of the 3GPP DN must include the Nokia topology.

  `SubNetwork=Nokia-1,ManagedElement=`RNC-3/WBTS-4

- The `UtranCell` attribute `UtranCellIubLink` is set to the 3GPP DN of the relating IubLink.

- The `UtranRelation` attribute `adjacentCell` is set to the 3GPP DN of the target cell. The content of the 3GPP DN must include the Nokia topology.

  `SubNetwork=Nokia-1,ManagedElement=RNC-2,RncFunction=RNC-2,UtranCell=WBTS-3/WCEL-4`

  `SubNetwork=Nokia-1,ExternalUtranCell=EWCE-2`

- The `GsmRelation` attribute `adjacentCell` must be set to the 3GPP DN of the target cell.

5.1.1 Considerations in CM data upload (Nokia model → 3GPP model)

During the upload of a NOM object:

- OMC: mapped to two common R6OM objects: ManagementNode and Subnetwork. The same NOM, `OMCId` (*NmsId*), is used in RDN for both R6OM objects.
- RNC: divided to two different common R6OM objects: Managed element: `Rnc` and `RncFunction`. The same NOM, `rncId`, is used in RDN for both R6OM objects.
- The `IurItem` parameters of the RNC map to the `ExternalRncFunction` with any duplicate parameter information removed.
- WBTS: divided to two different common R6OM objects: Managed element: `nodeB` and `nodeBFunction`. The same NOM, `WBTSId`, is used in RDN for both R6OM objects.
- ANTE: divided to two different common R6OM objects: Managed element: `ANTE` and `Antenna(Function)`. The same NOM, `ANTEId`, is used in RDN for both R6OM objects.

All of R6OM objects can be separately requested to upload: upload is done based on the upload scope. The relevant attributes have to be uploaded for each R6OM object:

- Common attributes specified in the 3GPP R6 specifications for each object.
- The NOM vendor-specific attributes for common objects are provided in `vsDataContainer`.

The following parameters can have wrong values due to range differences between Nokia and 3GPP:

- UtranCell/WCEL:
  - Parameter Name: maximumTransmissionPower/ PtxDLabsMax
    3GPP range: 0..50; Nokia range: -10..50
- Antenna/ANTE:
  - Parameter name: MechanicalOffset/tilt
    3GPP range: 0..360; Nokia range: -90 to 90
  - Parameter name: Height/height
    3GPP range: 0.. 36000000; Nokia range: -1000..1000 m

5.1.2 Considerations in CM data download (3GPP model → Nokia model)

During the download of R6OM objects:

- `ManagementNode` and `Subnetwork` combined in one NOM object: OMC, which holds all `ManagementNode` and `Subnetwork` attributes in the NOM.
- Managed element: `ANTE` and `antenna(Function)` combined in one NOM object: ANTE. That is, all R6OM `ManagedElement:ANTE` and `antenna(Function)` attributes are mapped to be attributes of ANTE in NOM.
- `ManagedElement`: `Rnc` and `RncFunction` combined in one NOM object: RNC. That is, all R6OM `Rnc` and `RncFunction` attributes are mapped to be attributes of RNC in NOM.
- Managed element: `nodeB` and `nodeBFunction` combined in one NOM object: WBTS. That is, all R6OM `NodeB` and `nodeBFunction` attributes are mapped to be attributes of WBTS in NOM.
- Object classes are not requiring creation or modification, but included in a plan are ignored.
- The NOM vendor-specific attributes for common objects are provided in `vsDataContainer`.

Note

All the rules above have to be taken into account when operating over the interface-N. Otherwise the download process fails.

Special requirements for the NOM RNC creation:

- A NOM RNC cannot be created via interface-N: it is always created via the network, that is, uploaded from the network to the regional network element manager.

Special requirements for the NOM WBTS creation:

- `ManagedElement` `NodeB` and `nodeBFunction` creation: the same ID (in NOM: <WBTSId>) must be used for both objects (1:1 relation).
- The content of the R6OM `nodeBFunction` ID in the interface-N DN must include the identity of the `RNCFunction` to which the `NodeBFunction` is connected via its associated `IubLink`. It also has to be of the following format: `RNC-<rncFunctionId>/WBTS-<nodeBFunctionId>`.

Special requirements for the NOM WCEL creation:

- The ID (in NOM: <LcrId>) has to be unique under the associated NodeBFunction (NOM: WBTS) or, rather, under the network.
- The content of the R6OM utranCell ID in the interface-N DN must include the identity of the nodeBFunction to which the utranCell is connected via its associated IubLink. It also has to be of the following format: WBTS-<nodeBFunctionId>/WCEL-<utranCellId>.
- Parameters that are mandatory in an object creation in the regional RAC database, are introduced in Mandatory WCDMA Parameters in Creating BTS Sites.

Note

The object creation via the interface-N has to be done in the NOM order: parent object first - then child objects. That is: WBTS - WCEL - adjacencies.

Special requirements for the object rehosting:

- When a NOM WBTS is rehosted under another RNC, its identity will not remain the same in the interface-N, because its ID contains the parent RNC's ID, for instance, RNC-<RncFunctionId>/WBTS-<nodeBFunctionId>.
- Therefore, the WBTS rehosting requires a delete-create scenario.
- The same applies also to NOM WCEL (UtranCell).
- Since the ExternalGsmCell ID contains ExternalGsmCell LAC parameter, the ID changes in rehosting, if the ExternalGsmCell LAC parameter is changed.
- Therefore, the ExternalGsmCell rehosting requires a delete-create scenario.

Special requirements for the object deletion:

- When the main object deletion is included in a plan, the deletion for all related objects has to be included in the same plan in order to keep the NOM consistent. For instance, if a nodeB is deleted, also the related nodeBFunction has to be deleted from the same plan.
- Related objects cannot be deleted separately (in NOM or over the interface-N). For instance, nodeBFunction cannot be deleted separately.

- If an UtranCell, ExternalUtranCell or ExternalGsmCell is deleted, all incoming UtranRelations and GsmRelations are automatically deleted as well.
- The object deletion via the interface-N has to be done in the NOM order: child objects first - then the parent object. That is, adjacencies - WCEL - WBTS.

Note

R6OM object VsDataContainer as well can include creation, update or deletion commands for vendor-specific objects.

5.2 Vendor-specific objects and parameters

Vendor-specific data is specific for bulk configuration management.

NOM vendor-specific objects and vendor-specific data of the common objects are presented in R6OM (in the interface-N) by using VsDataContainers as shown in the figure below.

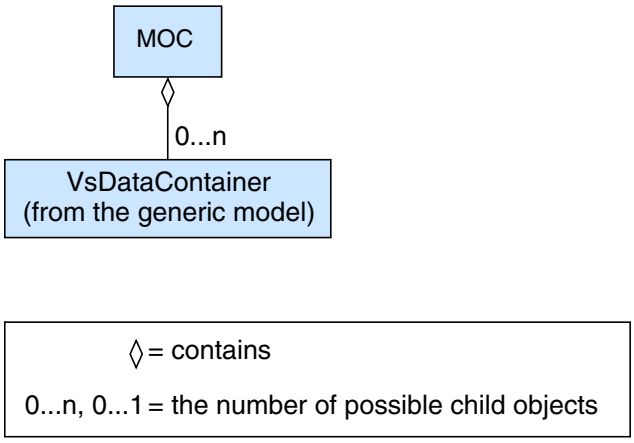

Figure 9. NOM vendor-specific objects and vendor-specific data

Nestings of VsDataContainers are not used in the NOM (in UMTS).

Nokia Object Model (NOM) has the following vendor-specific objects in UMTS:

- FMCG
- FMCI

- FMCS
- HOPG
- HOPI
- HOPS
- WSMLC
- WLCSE
- WANE
- WSG
- WCAL
- GCAL

In the NOM, there are also some transmission objects, such as AXC, IMAG and PORT that are not related to bulk CM in the interface-N and are not supported. The object NEMU is not supported due to restrictions in 3GPP specifications.

The following graphic presents the containment for Frequency Measurement Control (FMCx) objects in the Interface-N, as well as the maximum instance numbers.

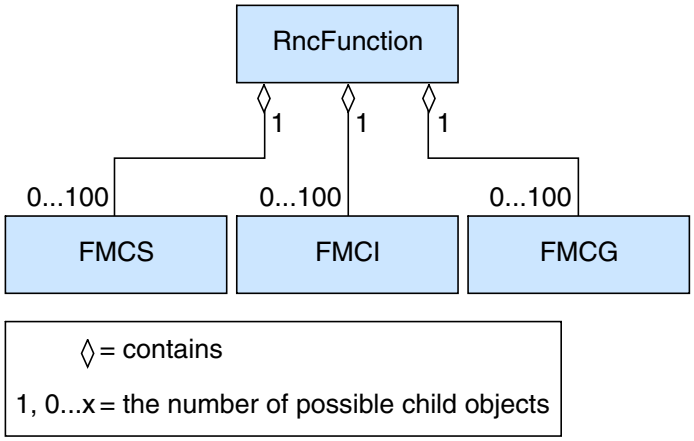

Figure 10. Containment for Frequency Measurement Control Objects

The following graphic presents the containment for Handover path (HOPx) objects in the Interface-N, as well as the maximum instance numbers.

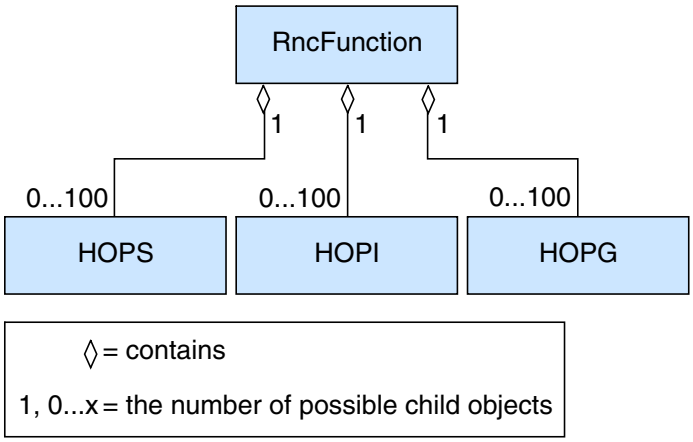

Figure 11. Containment for Handover path objects

The following graphic presents the containment for Location Services -related objects in the Interface-N, as well as the maximum instance numbers.

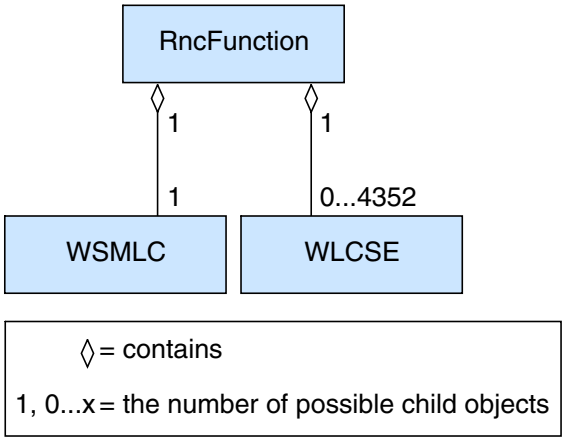

Figure 12. Containment for Location Services -related objects

The following graphic presents the containment for IMSI based HO -related objects in the Interface-N, as well as the maximum instance numbers.

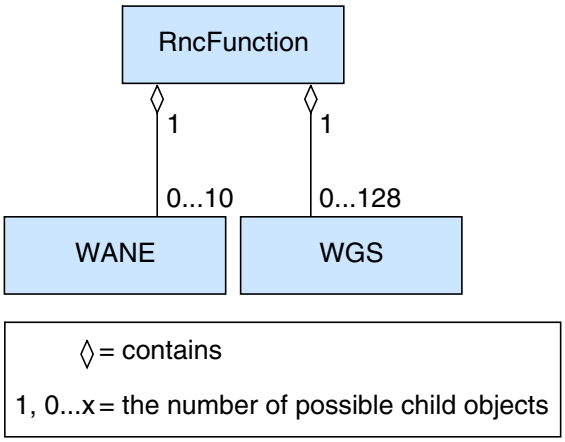

Figure 13. Containment for IMSI based HO -related objects

The following common NOM objects, which have a corresponding object in R6OM, have a vendor-specific extension in R6OM:

- ADJG
- ADJI
- ADJS
- COCO
- RNC
- WBTS
- WCEL
- ANTE

In addition, the following NOM objects have a Nokia-specific vsDataContainer extension in R6OM in order to support mandatory adjacency parameters (the extension is not specified in the 3GPP specifications):

- BTS (FBTS)
- EWCE

All of the NOM objects mentioned above always have one `vsDataContainer` instance per corresponding R6OM object in the interface-N.

Special case: `ExternalRncFunction` - which maps to`NOM RNC` parameter `IurItem` - always has one `VsDataContainer` instance.

The identification of the vendor-specific objects and vendor-specific data of the common objects is performed by using `vsDataContainerId` in the R6OM (in the interface-N XML files). The `vsDataContainerId` is formulated by using a combination of the NOM object name and the NOM identifier, for instance:

- `NOM HOPSId` = "1" (vendor-specific object) corresponds to `vsDataContainerId` = "`HOPS-1`" (in R6OM / in the interface-N)
- `NOM WCELId` = "12345" (vendor-specific part of a common object UtranCell) corresponds to `vsDataContainerId` = "`WCEL-12345`" (in R6OM / in the interface-N).

Nokia UTRAN vendor-specific object/data mappings

Table 19. Nokia UTRAN vendor specific object/data mappings

Object class	Mapping	Nokia DN	3GPP DN	Notes
ADJG [1]	`vsDataContainerId` = *adjgRDN*, `vsDataType` = `ADJG`	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJG-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, GsmRelation=ADJG-1, VsDataContainer=ADJG-1	The Nokia-specific parameters cannot be carried in 3GPP's `GsmRelation` object, but a separate vendor-specific object is needed.
ADJI [1]	`vsDataContainerId` = *adjiRDN*, `vsDataType` = `ADJI`	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJI-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, UtranRelation=ADJI-1, VsDataContainer=ADJI-1	The Nokia-specific parameters cannot be carried in 3GPP's `UtranRelation` object, but a separate vendor-specific object is needed.
ADJS [1]	`vsDataContainerId` = *adjsRDN*, `vsDataType` = `ADJS`	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1/ADJS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, UtranRelation=ADJS-1, VsDataContainer=ADJS-1	The Nokia-specific parameters cannot be carried in 3GPP's `UtranRelation` object, but a separate vendor-specific object is needed.
COCO	`vsDataContainerId` = *cocoRDN*, `vsDataType` = `COCO`	PLMN-PLMN/RNC-1/COCO-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1,IubLink=COCO-1	The Nokia-specific parameters cannot be carried in 3GPP's `IubLink` object but a separate vendor specific object is needed under `IubLink`.

Table 19. Nokia UTRAN vendor specific object/data mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
BTS (FBTS)	vsDataContainerId = *egceRDN*, vsDataType = EGCE	PLMN-PLMN/BSC-0/BCF-0/BTS-123 321	SubNetwork=Nokia-1, ExternalGsmCell=EGCE-123 321, VsDataContainer=EGCE-123 321	It is not necessary to carry ExccId in the 3GPP DN. Note: currently FBTS (foreign BTS) is used but it will be replaced with EGCE in the future NetAct/RAC releases. For more information, see examples of instance mappings on Table 18.
EWCE	vsDataContainerId = *ewceRDN*, vsDataType = EWCE	PLMN-PLMN/EXCC-1/EWCE-1234	SubNetwork=Nokia-1, ExternalUtranCell=EWCE-1234, VsDataContainer=EWCE-1234	It is not necessary to carry ExccId in the 3GPP DN. Operators can define the EWCE instance number but the requirement is that the identification has to be unique in the subNetwork. The maximum character amount is 10.
FMCG	vsDataContainerId = *fmcgRDN*, vsDataType = FMCG	PLMN-PLMN/RNC-1/FMCG-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=FMCG-1	WCEL MOs have references to FMCx MOs: FMCx MO instance has to exist (at least in the same plan) before it can be referred to.
FMCI	vsDataContainerId = *fmciRDN*, vsDataType = FMCI	PLMN-PLMN/RNC-1/FMCI-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=FMCI-1	WCEL MOs have references to FMCx MOs: FMCx MO instance has to exist (at least in the same plan) before it can be referred to.
FMCS	vsDataContainerId = *fmcsRDN*, vsDataType = FMCS	PLMN-PLMN/RNC-1/FMCS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=FMCS-1	WCEL MOs have references to FMCx MOs: FMCx MO instance has to exist (at least in the same plan) before it can be referred to.

Table 19. Nokia UTRAN vendor specific object/data mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
HOPS	`vsDataContainerId = hopsRDN, vsDataType = HOPS`	PLMN-PLMN/RNC-1/HOPS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=HOPS-1	ADJx MOs have references to HOPx MOs: HOPx MO instance has to exist (at least in the same plan) before it can be referred to.
HOPI	`vsDataContainerId = hopiRDN, vsDataType = HOPI`	PLMN-PLMN/RNC-1/HOPI-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=HOPI-1	ADJx MOs have references to HOPx MOs: HOPx MO instance has to exist (at least in the same plan) before it can be referred to.
HOPG	`vsDataContainerId = hopgRDN, vsDataType = HOPG`	PLMN-PLMN/RNC-1/HOPG-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=HOPG-1	ADJx MOs have references to HOPx MOs: HOPx MO instance has to exist (at least in the same plan) before it can be referred to.
WSMLC	`vsDataContainerId = wsmlcRDN, vsDataType = WSMLC`	PLMN-PLMN/RNC-1/WSMLC-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=WSMLC-1	There is one WSMLC per each RNC.
WLCSE	`vsDataContainerId = wlcseRDN, vsDataType = WLCSE`	PLMN-PLMN/RNC-1/WLCSE-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=WLCSE-1	WLCSE MOs have references to WCEL MOs: WCEL instance has to exist (at least in the same plan) before it can be referred to.
WANE	`vsDataContainerId = waneRDN, vsDataType = WANE`	PLMN-PLMN/RNC-1/WANE-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=WANE-1	
WGS	`vsDataContainerId = wgsRDN, vsDataType = WGS`	PLMN-PLMN/RNC-1/WGS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=WGS-1	WGS MOs have references to WANE MOs: WANE instance has to exist (at least in the same plan) before it can be referred to.

Table 19. Nokia UTRAN vendor specific object/data mappings (Continued)

Object class	Mapping	Nokia DN	3GPP DN	Notes
RNC [1]	`vsDataContainerId` = *rncRDN*, `vsDataType` = `RNC`	PLMN-PLMN/RNC-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RcnFunction=RNC-1, VsDataContainer=RNC-1	The Nokia-specific parameters cannot be carried in 3GPP's `RncFunction` object, but a separate vendor specific object is needed.
RNC/IurItem	`VsDataContainerId` = *NRncId (ID of the neighbouring RNC)*, `vsDataType` = `RNC`	no corresponding Nokia DN	SubNetwork=Nokia-1, ExternalRncFunction=RNC-1, `vsDataContainer=RNC-1`	This is a special mapping. Check Figure 4.
WBTS [1]	`vsDataContainerId` = *wbtsRDN*, `vsDataType` = `WBTS`	PLMN-PLMN/RNC-1/WBTS-1	SubNetwork=Nokia-1, ManagedElement=RNC-1/WBTS-1, NodeBFunction=RNC-1/WBTS-1, VsDataContainer=WBTS-1	The Nokia-specific parameters cannot be carried in 3GPP's `NodeBFunction` object, but a separate vendor specific object is needed.
WCEL [1]	`vsDataContainerId` = *wcelRDN*, `vsDataType` = WCEL	PLMN-PLMN/RNC-1/WBTS-1/WCEL-1	SubNetwork=Nokia-1, ManagedElement=RNC-1, RncFunction=RNC-1, UtranCell=WBTS-1/WCEL-1, VsDataContainer=WCEL-1	The Nokia-specific parameters cannot be carried in 3GPP's `UtranCell` object, but a separate vendor specific object is needed.
ANTE	`vsDataContainerId` = *rRDN*, `vsDataType` = `ANTE`	PLMN-PLMN/ANTC-1/ANTE-1/	SubNetwork=Nokia-1, ManagedElement=ANTC-1/ANTE-1, Antenna=ANTE-1, vsDataContainer=ANTE-1	The Nokia-specific parameters cannot be carried in 3GPP's `AnteFunction` object, but a separate vendor specific object is needed. VSD shall not appear as a child of RNC
WCAL	`vsDataContainerId` = *rRDN*, `vsDataType` = `WCAL`	PLMN-PLMN/ANTC-1/ANTE-1/WCAL-1	SubNetwork=Nokia-1, ManagedElement=ANTC-1, ANTE-1, Antenna=ANTE-1, VsDataContainer=WCAL-1	
GCAL	`vsDataContainerId` = *rRDN*, vsDataType = GCAL	PLMN-PLMN/ANTC-1/ANTE-1/GCAL-1	SubNetwork=Nokia-1, ManagedElement=ANTC-1, ANTE-1, Antenna=ANTE-1, VsDataContainer=GCAL-1	

[1] Note that in this context, this denotes the Nokia-specific part of the object.

Note

The set of vendor-specific parameters is different per adjacency/relation. It depends on the adjacency type in the Nokia Object Model.

5.3 Interface-N XML file example

The namespaces of the parameters in vsDataContainers refer to the RAN release of the object in question. For example, the namespace vsFMCG_RN2.0 indicates that the object belongs to RAN release RN2.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<bulkCmConfigDataFile
xmlns="http://www.3gpp.org/ftp/specs/archive/32_series/32.615#configData"
xmlns:xn="http://www.3gpp.org/ftp/specs/archive/32_series/32.625#genericNrm"
xmlns:un=" http://www.3gpp.org/ftp/specs/archive/32_series/32.645#utranNrm"
xmlns:gn=" http://www.3gpp.org/ftp/specs/archive/32_series/32.655#geranNrm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:vsRNC_152.2="nokia_rnc_152.2_cd0279.xsd"
xmlns:vsWBTS_152.2="nokia_wbts_152.2_cd0279.xsd"
xmlns:vsWCEL_152.2="nokia_wcel_152.2_cd0279.xsd"
[...]
xmlns:vsEWCE_1.0="nokia_ewce_1.0_cd0279.xsd"
xmlns:vsEGCE_1.0="nokia_egce_1.0_cd0279.xsd"
xmlns:vsRNC_RN2.0="nokia_rnc_rn2.0_cd0279.xsd"
xmlns:vsWBTS_RN2.0="nokia_wbts_rn2.0_cd0279.xsd"
xmlns:vsWCEL_RN2.0="nokia_wcel_rn2.0_cd0279.xsd"
[...]
xsi:schemaLocation =" bulkCmConfigDataFile.xsd configData.xsd
nokia_rnc_152.2_cd0279.xsd nokia_rnc_152.2_cd0279.xsd
nokia_wbts_152.2_cd0279.xsd nokia_wbts_152.2_cd0279.xsd
nokia_wcel_152.2_cd0279.xsd nokia_wcel_152.2_cd0279.xsd
[...]
nokia_rnc_rn2.0_cd0279.xsd nokia_rnc_rn2.0_cd0279.xsd
nokia_wbts_rn2.0_cd0279.xsd nokia_wbts_rn2.0_cd0279.xsd
nokia_wcel_rn2.0_cd0279.xsd nokia_wcel_rn2.0_cd0279.xsd
[...] ">
  <fileHeader fileFormatVersion="32.615 V4.4" vendorName="Nokia"/>
    <configData>
      <xn:SubNetwork id="Nokia-25">
        <xn:attributes/>
          <xn:ManagedElement id="RNC-195">
            <un:RncFunction id="RNC-195">
              <xn:VsDataContainer id="FMCG-2">
                <xn:attributes>
                  <xn:vsDataType>FMCG</xn:vsDataType>
                   <xn:vsDataFormatVersion>nokia_fmcg_rn2.0_cd0279
                   </xn:vsDataFormatVersion>
                     <vsFMCG_RN2.0:vsDataFMCG_RN2.0>
                     <vsFMCG_RN2.0:FMCGChangeOrigin>2
                     </vsFMCG_RN2.0:FMCGChangeOrigin>
                     <vsFMCG_RN2.0:GSMcauseCPICHEcNo>0
                     </vsFMCG_RN2.0:GSMcauseCPICHEcNo>
                     [...]
                     </vsFMCG_RN2.0:vsDataFMCG_RN2.0>
               </xn:attributes>
```

```
              </xn:VsDataContainer>
              <un:UtranCell id="WBTS-1/WCEL-3">
                <un:attributes>
                  <un:userLabel></un:userLabel>
                  <un:cId>1200</un:cId>
                  <un:localCellId>3</un:localCellId>
                  <un:uarfcnDl>2000</un:uarfcnDl>
                  <un:primaryScramblingCode>62</un:primaryScramblingCode>
                  <un:primaryCpichTxPower>330</un:primaryCpichTxPower>
                  <un:primarySchPower>-30</un:primarySchPower>
                  <un:secondarySchPower>-30</un:secondarySchPower>
                  <un:bchPower>-50</un:bchPower>
                  <un:lac>4001</un:lac>
                  <un:rac>100</un:rac>
                  <un:sac>101</un:sac>
                  <un:uraList>1 1 1 4 1 1 1 4</un:uraList>
                 <un:utranCellIubLink>SubNetwork=Nokia-25,ManagedElement=RNC-
195,RncFunction=RNC-195,IubLink=COCO-21
                  </un:utranCellIubLink>
                </un:attributes>
             <xn:VsDataContainer id="WCEL-3">
               <xn:attributes>
                 <xn:vsDataType>WCEL</xn:vsDataType>
                  <xn:vsDataFormatVersion>nokia_wcel_rn2.0_cd0279
                  </xn:vsDataFormatVersion>
                    <vsWCEL_RN2.0:vsDataWCEL_RN2.0>
                      <vsWCEL:ACBarredList>0</vsWCEL:ACBarredList>
                      <vsWCEL:AICHTraTime>0</vsWCEL:AICHTraTime>
                      <vsWCEL:AMRModeDL>1</vsWCEL:AMRModeDL>
                      <vsWCEL:AMRModeUL>1</vsWCEL:AMRModeUL>
                        [...]
                    </vsWCEL_RN2.0:vsDataWCEL_RN2.0>
                </xn:attributes>
              </xn:VsDataContainer>
            </un:UtranCell>
          </un:RncFunction>
        </xn:ManagedElement>
      </xn:SubNetwork>
    </configData>
<fileFooter dateTime="2004-05-14T13:24:31+02:00"/>
</bulkCmConfigDataFile>
```

6 Troubleshooting

The following steps describe how to troubleshoot the 3GPP CORBA Bulk CM Northbound Interface.

To restart the 3GPP CORBA Bulk CM Northbound Interface

1. Get the PID from reference file:

   `ls -l /var/opt/nokianms/ref/*rac3g*`

   If the reference file exists, the result should be as the following example:

   `/var/opt/nokianms/ref/rc25rac.rac3gpmx.<XXXX>`

   where `<XXXX>` is the process ID (PID) of the process.

2. Restart CM NBIF with

   `kill -USR2 <PID>`

3. The CM NBIF restarts automatically in couple of minutes.

To stop the 3GPP CORBA Bulk CM Northbound Inferface process

1. Stop the process with

   `kill -TERM <PID>`

   The process will be shutdown and removed from WPMANA control.

2. Start the process manually with

   `/opt/nokiaoss/configurator/rac/rac3gp/bin/rac3gpmx.sh&`

7 Where to find more information

For more information on Radio Access Configurator, see Radio Access Configurator Principles, DN03317888.

For more information on technical aspects of Radio Access Configurator, see *Radio Access Configurator Technical Reference Guide*, DN03317864.

For the allowed object identifiers, see Managed Object Reference, DN00126336.

For detailed information about RNC parameters supported by Radio Access Configurator, see WCDMA RAN RNW Parameters, DN00270387.

For information about non-network parameters supported by Radio Access Configurator, see *Non-network RNW Parameters*, DN04191918.

For information about changes in RNC parameters supported by Radio Access Configurator, see *Changes in WCDMA RAN RNW Parameters*, DN05227811.

For WCDMA parameter mapping, see *Interface-N: WCDMA (3G) Parameter Mapping*, DN03373371.

For more information about the 3GPP CORBA Northbound Interface, see the following documents in NED and in the NOLS documentation set:

- Configuring NetAct for 3GPP CORBA Northbound Interface, DN05171594
- 3GPP CORBA Northbound Interface Technical Reference Guide, DN05171613
- 3GPP CORBA Northbound Interface Principles, DN05171637

See the 3rd Generation Partnership Project (3GPP) web pages at http://www.3gpp.org/ftp/Specs/ for the following specifications:

For 3GPP R6 object models (termed Network Resource Models):

- TS 32.622-6.4.0 Generic Network Resources IRP: Network Resource Model (2005-03)
- 3GPP TS 32.642-6.4.0 UTRAN Network Resources IRP: Network Resource Model (2005-03)

- 3GPP TS 32.652-6.4.0 GERAN Network Resources IRP: Network Resource Model (2005-03)
- 3GPP TS 32.300-6.4.0 Name Convention for Managed Objects (2005-03)

Appendix A. BulkCmIRPConstDefs.idl

```
// File: BulkCmIRPConstDefs.idl

#ifndef _BULKCMIRPCONSTDEFS_IDL_
#define _BULKCMIRPCONSTDEFS_IDL_

// This statement must appear after all include statements
#pragma prefix "3gppsa5.org"

/* ## Module: BulkCmIRPConstDefs
This module contains type definitions for the Bulk CM IRP
================================================================
*/
module BulkCmIRPConstDefs
{

   /*
   This block identifies the notification types defined by
   this Bulk CM IRP version.
   This string is used in the second field of the Structured
   Event.
   */
   interface NotificationType
   {
      const string NOTIFY_SESSION_STATE_CHANGED = "x1";
      const string NOTIFY_GET_SESSION_LOG_ENDED = "x2";
   };

   /*
   This block assigns value for the name of the NV of the Structured Event.
   */
   interface AttributeNameValue
   {
      const string SESSION_ID = "k";
      const string SOURCE_INDICATOR = "m";
      const string ERROR_INFORMATION = "n";
   };

   /*
   This block defines all possible values for sessionState.
   One of these strings appear in the event_name of the
   Structured Event of notifySessionStateChanged notification.
   */
   interface SessionStateChangeNotification
   {
      const string UPLOAD_FAILED = "x1";
      const string UPLOAD_COMPLETED = "x2";
      const string DOWNLOAD_FAILED = "x3";
      const string DOWNLOAD_COMPLETED = "x4";
      const string ACTIVATION_FAILED = "x5";
      const string ACTIVATION_PARTLY_REALISED = "x6";
      const string ACTIVATION_COMPLETED = "x7";
      const string FALLBACK_FAILED = "x8";
      const string FALLBACK_PARTLY_REALISED = "x9";
      const string FALLBACK_COMPLETED = "x10";
      const string VALIDATION_FAILED = "x11";
      const string VALIDATION_COMPLETED = "x12";
      const string PREACTIVATION_FAILED = "x13";
      const string PREACTIVATION_PARTLY_REALISED = "x14";
      const string PREACTIVATION_COMPLETED = "x15";
   };

   /*
   This block defines all possible values for sessionLogStatus
   One of these strings appear in the event_name of the Structured
   Event of notifyGetSessionLogEnded notification.
   */
```

```
interface LogStateNotification
{
   const string GET_SESSION_LOG_COMPLETED_SUCCESSFULLY = "x1";
   const string GET_SESSION_LOG_COMPLETED_UNSUCESSFULLY = "x2";
};

/*
For each started configuration session a unique identifier is generated
by the IRPManager. An sessionId can not be used for an upload if it is
already in use of a download configuration and vice versa.
*/
typedef string SessionId;

/*
This string field is used in order to provide additional error information
if an operation has failed.
*/
typedef string ErrorInformation;

/*
Defines the different subphases of a configuration session
e.g. thus it is easy to implement a detection of an upload
or a download/activate session.
*/
enum SubPhase {IdlePhase, DownloadPhase, UploadPhase, ActivationPhase,
               FallbackPhase, PreactivationPhase, ValidationPhase};

/*
Defines the different substates of a configuration session. This includes
the transition state as well.
*/
enum SubState {Completed, Failed, PartlyRealised, InProgress};

/*
Defines state of a configuration session with the phase and the substate
of the configuration.
*/
struct SessionState
{
   SubPhase sub_phase;
   SubState sub_state;
};

/*
Contains the list of all current sessionIds
*/
typedef sequence <SessionId> SessionIdList;

/*
Specifies a complete destination path (including filename).
*/
typedef string FileDestination;

/*
The format of Distinguished Name is specified in
the Naming Conventions for Managed Objects; TS 32.300.
e.g. "SubNetwork=10001,ManagedElement=400001" identifies a
ManagedElement instance of the object model.
*/
typedef string DistinguishedName;

/*
Used within the upload method to give filter critera
*/
typedef string FilterType;

/*
Defines the kind of scope to use in a search together with
```

```
   SearchControl.level, in a SearchControl value.
   SearchControl.level is always >= 0. If a level is bigger than the
   depth of the tree there will be no exceptions thrown.
   */
   enum ScopeType {BaseOnly, BaseNthLevel, BaseSubtree, BaseAll};

   /*
   Controls the searching for MOs during upload, and contains:
   the type of scope ("type" field),
   the level of scope ("level" field),
   the filter ("filter" field),
   The type and level fields are mandatory.
   The filter field is mandatory (The filter will have to be
   set to an empty string if it has no other value).
   */
   struct SearchControl
   {
      ScopeType type;
      unsigned long level;
      FilterType filter;
   };

   /*
   This indicates how the activation is executed, either with least service
   impact or least elapsed time.
   */   enum ActivationMode {LeastServiceImpact, LeastElapsedTime};

   /*
   This indicates the level of verification of bulk configuration data done,
   either full or limited checking.
   */   enum VerificationMode {FullChecking, LimitedChecking};

     /* ActivationModeTypeOpt is a type carrying an optional parameter.
      If the boolean is TRUE, the value is present.
      Otherwise, the value is absent.
   */
 union ActivationModeTypeOpt switch(boolean)
   {
      case TRUE: ActivationMode activation_mode;
   };

     /* VerificationModeTypeOpt is a type carrying an optional parameter.
      If the boolean is TRUE, the value is present.
      Otherwise, the value is absent.
   */
   union VerificationModeTypeOpt switch(boolean)
   {
      case TRUE: VerificationMode verification_mode;
   };

};

#endif // _BULKCMIRPCONSTDEFS_IDL_
```

Appendix B. BulkCmIRPSystem.idl

```
// File: BulkCMIRPSystem.idl

#ifndef _BULKCMIRPSYSTEM_IDL_
#define _BULKCMIRPSYSTEM_IDL_

#include "BulkCmIRPConstDefs.idl"
#include "ManagedGenericIRPConstDefs.idl"
#include "ManagedGenericIRPSystem.idl"

// This statement must appear after all include statements
#pragma prefix "3gppsa5.org"

/* ## Module: BulkCmIRPSystem
This module implements capabilities of Bulk CM IRP.
================================================================
*/
module BulkCmIRPSystem
{
   /*
   The request cannot be processed due to a situation of concurrency.
   E.g. two concurrent activation requests involving the same ManagedElement
   instance. The semantics carried in reason is outside the scope of this IRP.
   */
   exception ConcurrencyException { string reason; };

   /*
   The provided filter is malformed or invalid. The semantics carried in reason
   is outside the scope of this IRP.
   */
   exception IllegalFilterFormatException { string reason; };

   /*
   The provided Distinguished Name is malformed or invalid. The semantics
   carried in reason is outside the scope of this IRP.
   */
   exception IllegalDNFormatException { string reason; };

   /*
   The provided scope type is illegal. The semantics carried in reason is
   outside the scope of this IRP.
   */
   exception IllegalScopeTypeException { string reason; };

   /*
   The provided scope level is illegal. The semantics carried in reason is
   outside the scope of this IRP.
   */
   exception IllegalScopeLevelException { string reason; };

   /*
   The request cannot be processed because no fallback data is available, i.e.
   fallback capability was previously not asked for.
   */
   exception NoFallbackException {};

   /*
   The provided sessionId value is already used for another configuration
   session. The semantics carried in reason is outside the scope of this IRP.
   */
   exception SessionIdInUseException { string reason; };

   /*
   The provided URL is malformed or invalid. The semantics carried in reason is
   outside the scope of this IRP.
   */
   exception IllegalURLFormatException{ string reason; };
```

```
/*
The provided sessionId value does not identify any existing configuration
session.
*/
exception UnknownSessionIdException {};

/*
The request cannot be processed because it is not valid in the current state
of the configuration session.
*/
exception NotValidInCurrentStateException
{
   BulkCmIRPConstDefs::SessionState current_state;
};

/*
The request cannot be processed because the maximum number of simultaneously
running configuration sessions has been reached. The semantics carried in
reason is outside the scope of this IRP.
*/
exception MaxSessionReachedException { string reason; };

/*
The provided ActivationMode type is illegal. The semantics carried in reason
is outside the scope of this IRP.
*/
exception IllegalActivationModeException { string reason; };

/*
The provided VerificationMode type is illegal. The semantics carried in
reason is outside the scope of this IRP.
*/
exception IllegalVerificationModeException { string reason; };

/*
System otherwise fails to complete the operation.  System can provide reason
to qualify the exception.  The semantics carried in reason
is outside the scope of this IRP.
*/
exception GetBulkCmIRPVersionsException { string reason; };
exception UploadException { string reason; };
exception DownloadException { string reason; };
exception ActivateException { string reason; };
exception ValidateException { string reason; };
exception PreactivateException { string reason; };
exception GetBulkCMIRPOperationProfileException { string reason; };
exception GetBulkCMIRPNotificationProfileException { string reason; };
exception GetSessionLogException { string reason; };
exception StartSessionException { string reason; };
exception GetSessionStatusException { string reason; };
exception FallbackException { string reason; };
exception EndSessionException { string reason; };
exception AbortSessionOperationException { string reason; };
exception GetSessionIdsException { string reason; };

/*
Defines the System interface of a EM. It defines all methods which are
necessary to control a configuration session from a IRPManager.
*/
interface BulkCmIRP
{
   /*
   Return the list of all supported Bulk CM IRP versions.
   */
   ManagedGenericIRPConstDefs::VersionNumberSet get_bulk_CM_IRP_versions (
   )
   raises (GetBulkCmIRPVersionsException);
```

```
/*
Return the list of all supported operations and their supported
parameters for a specific BulkCM IRP version.
*/
ManagedGenericIRPConstDefs::MethodList get_bulk_CM_IRP_operation_profile (
   in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
)
raises (GetBulkCMIRPOperationProfileException,
        ManagedGenericIRPSystem::OperationNotSupported,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Return the list of all supported notifications and their supported
parameters for a specific BulkCM IRP version.
*/
ManagedGenericIRPConstDefs::MethodList
   get_bulk_CM_IRP_notification_profile
(
   in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
)
raises (GetBulkCMIRPNotificationProfileException,
        ManagedGenericIRPSystem::OperationNotSupported,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Uploads a configuration from the subnetwork. The result is put in a
configuration data file in an area specified by the IRPManager.
The MIB of the subnetwork is iterated by means of containment search,
using a SearchControl to control the search and the returned results.
All MOs in the scope constitutes a set that the filter works on.
In case of a concurrent running session the function will
return an exception. If the value of the given baseObject or FilterType
does not exist then this asynchronous error condition will be notified.
*/
void upload (
   in BulkCmIRPConstDefs::SessionId session_id,
   in BulkCmIRPConstDefs::FileDestination sink,
   in BulkCmIRPConstDefs::DistinguishedName base_object,
   in BulkCmIRPConstDefs::SearchControl search_control
)
raises (UploadException, UnknownSessionIdException,
        MaxSessionReachedException, NotValidInCurrentStateException,
        ConcurrencyException,
        IllegalDNFormatException, IllegalFilterFormatException,
        IllegalScopeTypeException, IllegalScopeLevelException,
        IllegalURLFormatException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Indicates the EM that it can download a configuration data file from
a given configuration data file storage area. The EM will check the
consistence of the configuration data and the software compatibilty.
*/
void download (
   in BulkCmIRPConstDefs::SessionId session_id,
   in BulkCmIRPConstDefs::FileDestination source
)
raises (DownloadException, UnknownSessionIdException,
        MaxSessionReachedException, NotValidInCurrentStateException,
        IllegalURLFormatException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Activates a previously downloaded and sucessfully parsed configuration
inside a session.  This means that the configuration will be introduced
in the live sub-network. In case of a concurrent running session
the function will return an exception.
```

```
*/
void activate (
   in BulkCmIRPConstDefs::SessionId session_id,
   in BulkCmIRPConstDefs::ActivationModeTypeOpt activation_mode,
   in boolean fallback
)
raises (ActivateException, UnknownSessionIdException,
        NotValidInCurrentStateException, ConcurrencyException,
        IllegalActivationModeException,
        ManagedGenericIRPSystem::ParameterNotSupported,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Uploads a log from the subnetwork which is usally used for error
analysis. The log is put in a logfile in the filesystem which can
be accessed by the EM. If there are no log entries an empty log file
is uploaded.
*/
void get_session_log (
   in BulkCmIRPConstDefs::FileDestination sink,
   in BulkCmIRPConstDefs::SessionId session_id,
   in boolean only_error_info
)
raises (GetSessionLogException, UnknownSessionIdException,
        IllegalURLFormatException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Creates an instance of the configuration session state machine. The
IDLE_PHASE & COMPLETED is notified
*/
void start_session (
   in BulkCmIRPConstDefs::SessionId session_id
)
raises (StartSessionException, SessionIdInUseException,
        MaxSessionReachedException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Returns the state of a configuration session.
*/
BulkCmIRPConstDefs::SessionState get_session_status (
   in BulkCmIRPConstDefs::SessionId session_id,
   out BulkCmIRPConstDefs::ErrorInformation error_information
)
raises (GetSessionStatusException, UnknownSessionIdException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Actives a fallback area. Each time a configuration is activated a
fallback area can be created, s. activate parameter.
This area is backup of the complete configuration which can be
restored by this method. The process is as follows:
1. When the method activate(...,..., TRUE) is used,
   a copy of the valid area is taken before the activation
   of the new planned data has started. Only one fallback area can
   exists at a time for a specific scope of the subnetwork.
2. When a fallback area is avilable and triggered by this method, the
   previous valid area is replaced with the data stored in
   the fall back area.
If the EM detects that the former configuration has never been
changed it returns an exception because it does not trigger an
activation of the former data.
*/
void fallback (
   in BulkCmIRPConstDefs::SessionId session_id
)
raises (FallbackException, UnknownSessionIdException, NoFallbackException,
```

```
        NotValidInCurrentStateException, ConcurrencyException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
The IRPManager invokes this operation to delete all its temporary
entities and the related sessionId which belong to the scope of
a configuration session. This includes the related error and log
informationen too.
*/
void end_session (
   in BulkCmIRPConstDefs::SessionId session_id
)
raises (EndSessionException, UnknownSessionIdException,
        NotValidInCurrentStateException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
The IRPManager invokes this operation to abort an active operation
during a configuration session. It is only effecting
a configuration session in state IN_PROGRESS. In this case the
current session task is interrupted, e.g. the activating in progress,
using best effort strategy, and a state change is notified
*/
void abort_session_operation (
   in BulkCmIRPConstDefs::SessionId session_id
)
raises (AbortSessionOperationException, UnknownSessionIdException,
        NotValidInCurrentStateException,
        ManagedGenericIRPSystem::InvalidParameter);

/*
Returns a list all sessionIds of current running configuration sessions.
*/
BulkCmIRPConstDefs::SessionIdList get_session_ids (
)
raises (GetSessionIdsException);

/*
Validates previously downloaded bulk configuration data inside a session.
Detects errors in the data prior to requesting preactivation or
activation.
*/
void validate (
   in BulkCmIRPConstDefs::SessionId session_id,
   in BulkCmIRPConstDefs::ActivationModeTypeOpt activation_mode
)
raises (ValidateException, UnknownSessionIdException,
           NotValidInCurrentStateException, ConcurrencyException,
           IllegalActivationModeException,
           ManagedGenericIRPSystem::ParameterNotSupported,
           ManagedGenericIRPSystem::InvalidParameter,
           ManagedGenericIRPSystem::OperationNotSupported);

/*
Preactivates previously downloaded bulk configuration data inside a
session. This operation validates configuration data changes in the
context of the current data and pre-processes the configuration data
changes.
*/
void preactivate (
   in BulkCmIRPConstDefs::SessionId session_id,
   in BulkCmIRPConstDefs::VerificationModeTypeOpt verification_mode,
   in BulkCmIRPConstDefs::ActivationModeTypeOpt activation_mode,
   in boolean fallback
)
raises (PreactivateException, UnknownSessionIdException,
        NotValidInCurrentStateException, ConcurrencyException,
        IllegalActivationModeException, IllegalVerificationModeException,
```

```
                ManagedGenericIRPSystem::ParameterNotSupported,
                ManagedGenericIRPSystem::InvalidParameter,
                ManagedGenericIRPSystem::OperationNotSupported);

   };
};


module SimpleUploadBulkCMIRPSystem
{
   exception GetSimpleUploadBulkCmIRPVersionsException { string reason; };
   exception GetSimpleUploadBulkCMIRPOperationProfileException
      { string reason; };
   exception GetSimpleUploadBulkCMIRPNotificationProfileException
      { string reason; };

   interface SimpleUploadBulkCMIRP
   {
      /*
      Return the list of all supported Bulk CM IRP versions.
      */
      ManagedGenericIRPConstDefs::VersionNumberSet
          get_simple_upload_bulk_CM_IRP_versions (
      )
      raises (GetSimpleUploadBulkCmIRPVersionsException);

      /*
      Return the list of all supported operations and their supported
      parameters for a specific BulkCM IRP version.
      */
      ManagedGenericIRPConstDefs::MethodList
         get_simple_upload_bulk_CM_IRP_operation_profile (
         in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
      )
      raises (GetSimpleUploadBulkCMIRPOperationProfileException,
              ManagedGenericIRPSystem::OperationNotSupported,
              ManagedGenericIRPSystem::InvalidParameter);

      /*
      Return the list of all supported notifications and their supported
      parameters for a specific BulkCM IRP version.
      */
      ManagedGenericIRPConstDefs::MethodList
         get_simple_upload_bulk_CM_IRP_notification_profile
      (
         in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
      )
      raises (GetSimpleUploadBulkCMIRPNotificationProfileException,
              ManagedGenericIRPSystem::OperationNotSupported,
              ManagedGenericIRPSystem::InvalidParameter);


      /*
      Uploads a configuration from the subnetwork. The result is put in a
      configuration data file in an area specified by the IRPManager.
      The MIB of the subnetwork is iterated by means of containment search,
      using a SearchControl to control the search and the returned results.
      All MOs in the scope constitutes a set that the filter works on.
      In case of a concurrent running session the function will
      return an exception. If the value of the given baseObject or FilterType
      does not exist then this asynchronous error condition will be notified.
      */
      void upload (
         in BulkCmIRPConstDefs::SessionId session_id,
         in BulkCmIRPConstDefs::FileDestination sink,
         in BulkCmIRPConstDefs::DistinguishedName base_object,
         in BulkCmIRPConstDefs::SearchControl search_control
```

```
      )
      raises (
          BulkCmIRPSystem::UploadException,
          BulkCmIRPSystem::UnknownSessionIdException,
          BulkCmIRPSystem::MaxSessionReachedException,
          BulkCmIRPSystem::NotValidInCurrentStateException,
          BulkCmIRPSystem::ConcurrencyException,
          BulkCmIRPSystem::IllegalDNFormatException,
          BulkCmIRPSystem::IllegalFilterFormatException,
          BulkCmIRPSystem::IllegalScopeTypeException,
          BulkCmIRPSystem::IllegalScopeLevelException,
          BulkCmIRPSystem::IllegalURLFormatException,
          ManagedGenericIRPSystem::InvalidParameter);
   };

};  // end of module SimpleUploadBulkCMIRPSystem


module ControlledUploadBulkCMIRPSystem
{
    exception GetControlledUploadBulkCmIRPVersionsException { string reason; };
    exception GetControlledUploadBulkCMIRPOperationProfileException
        { string reason; };
    exception GetControlledUploadBulkCMIRPNotificationProfileException
        { string reason; };

    interface ControlledUploadBulkCMIRP
    {
        /*
        Return the list of all supported Bulk CM IRP versions.
        */
        ManagedGenericIRPConstDefs::VersionNumberSet
            get_controlled_upload_bulk_CM_IRP_versions (
        )
        raises (GetControlledUploadBulkCmIRPVersionsException);

        /*
        Return the list of all supported operations and their supported
        parameters for a specific BulkCM IRP version.
        */
        ManagedGenericIRPConstDefs::MethodList
            get_controlled_upload_bulk_CM_IRP_operation_profile (
                in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
        )
        raises (GetControlledUploadBulkCMIRPOperationProfileException,
            ManagedGenericIRPSystem::OperationNotSupported,
            ManagedGenericIRPSystem::InvalidParameter);

        /*
        Return the list of all supported notifications and their supported
        parameters for a specific BulkCM IRP version.
        */
        ManagedGenericIRPConstDefs::MethodList
            get_controlled_upload_bulk_CM_IRP_notification_profile (
                in ManagedGenericIRPConstDefs::VersionNumber bulk_CM_IRP_version
        )
        raises (GetControlledUploadBulkCMIRPNotificationProfileException,
            ManagedGenericIRPSystem::OperationNotSupported,
            ManagedGenericIRPSystem::InvalidParameter);


        /*
        Uploads a configuration from the subnetwork. The result is put in a
        configuration data file in an area specified by the IRPManager.
        The MIB of the subnetwork is iterated by means of containment search,
        using a SearchControl to control the search and the returned results.
        All MOs in the scope constitutes a set that the filter works on.
```

```
  In case of a concurrent running session the function will
  return an exception. If the value of the given baseObject or FilterType
  does not exist then this asynchronous error condition will be notified.
  */
  void upload (
      in BulkCmIRPConstDefs::SessionId session_id,
      in BulkCmIRPConstDefs::FileDestination sink,
      in BulkCmIRPConstDefs::DistinguishedName base_object,
      in BulkCmIRPConstDefs::SearchControl search_control
  )
  raises (
      BulkCmIRPSystem::UploadException,
      BulkCmIRPSystem::UnknownSessionIdException,
      BulkCmIRPSystem::MaxSessionReachedException,
      BulkCmIRPSystem::NotValidInCurrentStateException,
      BulkCmIRPSystem::ConcurrencyException,
      BulkCmIRPSystem::IllegalDNFormatException,
      BulkCmIRPSystem::IllegalFilterFormatException,
      BulkCmIRPSystem::IllegalScopeTypeException,
      BulkCmIRPSystem::IllegalScopeLevelException,
      BulkCmIRPSystem::IllegalURLFormatException,
      ManagedGenericIRPSystem::InvalidParameter);


/*
Uploads a log from the subnetwork which is usally used for error
analysis. The log is put in a logfile in the filesystem which can
be accessed by the EM. If there are no log entries an empty log file
is uploaded.
*/
void get_session_log (
   in BulkCmIRPConstDefs::FileDestination sink,
   in BulkCmIRPConstDefs::SessionId session_id,
   in boolean only_error_info
)
raises (
    BulkCmIRPSystem::GetSessionLogException,
    BulkCmIRPSystem::UnknownSessionIdException,
    BulkCmIRPSystem::IllegalURLFormatException,
    ManagedGenericIRPSystem::InvalidParameter);

/*
Creates an instance of the configuration session state machine. The
IDLE_PHASE & COMPLETED is notified
*/
void start_session (
   in BulkCmIRPConstDefs::SessionId session_id
)
raises (
    BulkCmIRPSystem::StartSessionException,
    BulkCmIRPSystem::SessionIdInUseException,
    BulkCmIRPSystem::MaxSessionReachedException,
    ManagedGenericIRPSystem::InvalidParameter);

/*
Returns the state of a configuration session.
*/
BulkCmIRPConstDefs::SessionState get_session_status (
   in BulkCmIRPConstDefs::SessionId session_id,
   out BulkCmIRPConstDefs::ErrorInformation error_information
)
raises (
    BulkCmIRPSystem::GetSessionStatusException,
    BulkCmIRPSystem::UnknownSessionIdException,
    ManagedGenericIRPSystem::InvalidParameter);

/*
The IRPManager invokes this operation to delete all its temporary
```

```
      entities and the related sessionId which belong to the scope of
      a configuration session. This includes the related error and log
      informationen too.
      */
      void end_session (
         in BulkCmIRPConstDefs::SessionId session_id
      )
      raises (
          BulkCmIRPSystem::EndSessionException,
          BulkCmIRPSystem::UnknownSessionIdException,
          BulkCmIRPSystem::NotValidInCurrentStateException,
          ManagedGenericIRPSystem::InvalidParameter);


      /*
      The IRPManager invokes this operation to abort an active operation
      during a configuration session. It is only effecting
      a configuration session in state IN_PROGRESS. In this case the
      current session task is interrupted, e.g. the activating in progress,
      using best effort strategy, and a state change is notified
      */
      void abort_session_operation (
         in BulkCmIRPConstDefs::SessionId session_id
      )
      raises (
          BulkCmIRPSystem::AbortSessionOperationException,
          BulkCmIRPSystem::UnknownSessionIdException,
          BulkCmIRPSystem::NotValidInCurrentStateException,
          ManagedGenericIRPSystem::InvalidParameter);


      /*
      Returns a list all sessionIds of current running configuration sessions.
      */
      BulkCmIRPConstDefs::SessionIdList get_session_ids (
      )
      raises (
          BulkCmIRPSystem::GetSessionIdsException);

     };

};  // end of module ControlledUploadBulkCMIRPSystem

#endif  // _BULKCMIRPSYSTEM_IDL_
```

Appendix C. BulkCMIRPNotifications.idl

```
// File: BulkCMNotifications.idl

#ifndef _BULKCMIRPNOTIFICATIONS_IDL_
#define _BULKCMIRPNOTIFICATIONS_IDL_

#include <NotificationIRPNotifications.idl>
#include <BulkCmIRPConstDefs.idl>

// This statement must appear after all include statements
#pragma prefix "3gppsa5.org"

module BulkCMIRPNotifications
{

    interface NotifySessionStateChange: NotificationIRPNotifications::Notify
    {
        // This is the type_name (2nd field) of the fixed header.
        const string EVENT_TYPE =
            BulkCmIRPConstDefs::NotificationType::NOTIFY_SESSION_STATE_CHANGED;

        // ------------------------------
        // One of the strings here is the event_name (3rd field) of the
        // fixed header.
        // The first 2 are relevant for IS-defined packages Simple
        // Upload and Controlled Upload.
        // All are relevant for IS-defined package
        // Controlled Upload & Provisioning.

        const string UPLOAD_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::UPLOAD_FAILED;
        const string UPLOAD_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::UPLOAD_COMPLETED;
        const string DOWNLOAD_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::DOWNLOAD_FAILED;
        const string DOWNLOAD_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::DOWNLOAD_COMPLETED;
        const string ACTIVATION_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::ACTIVATION_FAILED;
        const string ACTIVATION_PARTLY_REALISED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::ACTIVATION_PARTLY_REALISED;
        const string ACTIVATION_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::ACTIVATION_COMPLETED;
        const string FALLBACK_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::FALLBACK_FAILED;
        const string FALLBACK_PARTLY_REALISED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::FALLBACK_PARTLY_REALISED;
        const string FALLBACK_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::FALLBACK_COMPLETED;
        const string VALIDATION_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::VALIDATION_FAILED;
        const string VALIDATION_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::VALIDATION_COMPLETED;
        const string PREACTIVATION_FAILED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::PREACTIVATION_FAILED;
        const string PREACTIVATION_PARTLY_REALISED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::PREACTIVATION_PARTLY_REALISED;
        const string PREACTIVATION_COMPLETED = BulkCmIRPConstDefs::
            SessionStateChangeNotification::PREACTIVATION_COMPLETED;
        // ------------------------------

        const string SESSION_ID =
            BulkCmIRPConstDefs::AttributeNameValue::SESSION_ID;

        const string SOURCE_INDICATOR =
            BulkCmIRPConstDefs::AttributeNameValue::SOURCE_INDICATOR;
      };
```

```
    interface NotifyGetSessionLogEnded: NotificationIRPNotifications::Notify
    {
        // This is the type_name (2nd field) of the fixed header.
        const string EVENT_TYPE =
            BulkCmIRPConstDefs::NotificationType::NOTIFY_GET_SESSION_LOG_ENDED;

        // -----------------
        // One of the 2 strings here is the event_name (3rd field) of the
        // fixed header.
        const string GET_SESSION_LOG_COMPLETED_SUCCESSFULLY =
            BulkCmIRPConstDefs::LogStateNotification::
                GET_SESSION_LOG_COMPLETED_SUCCESSFULLY;
        const string GET_SESSION_LOG_COMPLETED_UNSUCESSFULLY =
            BulkCmIRPConstDefs::LogStateNotification::
                GET_SESSION_LOG_COMPLETED_UNSUCESSFULLY;
        // ----------------------

        const string SESSION_ID =
            BulkCmIRPConstDefs::AttributeNameValue::SESSION_ID;

        const string SOURCE_INDICATOR =
            BulkCmIRPConstDefs::AttributeNameValue::SOURCE_INDICATOR;
    };

};

#endif // _BULKCMIRPNOTIFICATIONS_IDL_
```

Index

W

X