

Generic CAI3G Interface 1.2

INTERFACE DESCRIPTION

Copyright

© Ericsson AB 2007–2012. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

Ericsson is the trademark or registered trademark of Telefonaktiebolaget LM Ericsson.

All other product or service names mentioned in this document are trademarks of their respective owners.

Abstract

The Customer Administration Interface Third Generation (CAI3G) is an activation interface aiming for providing a simple, up-to-date and unified provisioning interface for the network elements in telecommunication or IT networks or both. It is a web service interface based on Simple Object Access Protocol (SOAP) 1.1. This document introduces the background and usage of CAI3G interface, describes the protocol stack and services in CAI3G interface, and depicts the operations and attributes supported by CAI3G interface. This document also contains a lot of rules and examples in order to provide an easily readable description of the CAI3G facilities.



Contents

1	Introduction	1
1.1	Terminology	1
1.2	Notational Conventions	2
2	CAI3G Overview	5
2.1	CAI3G Web Service	7
2.2	Namespace	9
2.3	Synchronous Interface	9
3	Protocol Stack	11
3.1	SOAP	11
3.1.1	SOAP Message Exchange Model	14
3.1.2	SOAP Document Style	15
3.2	CAI3G Operations	15
3.3	Interface Data Model	16
3.4	MO Attributes	17
4	CAI3G Operation	19
4.1	Session Control Service	19
4.1.1	LOGIN Command	20
4.1.2	LOGOUT Command	21
4.2	Provisioning Service	22
4.2.1	CREATE Operation	22
4.2.2	GET Operation	24
4.2.3	SET Operation	25
4.2.4	DELETE Operation	26
4.2.5	SEARCH Operation	27
4.3	Notification Service	29
4.3.1	Subscribe Operation	30
4.3.2	Notify Operation	33
4.3.3	Unsubscribe Operation	35
4.4	Response and Fault Handling	35
4.4.1	Headerfault	36
4.4.2	CAI3G Fault Details	38
5	CAI3G Parameter	43
5.1	MOType	43
5.2	MOId	43
5.3	MO Attributes	44



5.3.1	Simple Parameter	45
5.3.2	Structured Parameter	48
5.3.3	Sub-object Parameter	50
5.4	Extension	53
5.5	Leaf Attributes	53
5.6	MO Schema File	54
6	CAI3G Integration	57
7	Version Control	59
7.1	CAI3G	59
7.2	CAI3G MO	60
8	Transaction Support	61
9	Security Considerations	63
10	Appendix A (CAI3G Rule Description)	65
10.1	Rules	65
10.2	Recommendations	66
11	Appendix B (CAI3G Services WSDL Files)	67
12	Appendix C (Example of SOAP Message)	81
13	Appendix D (Development Instruction)	83
13.1	Development Process of CAI3G Server	83
13.2	Development Process of CAI3G Client	86
14	Appendix E (Change History)	89
	Glossary	91
	Reference List	93



1 Introduction

CAI3G defines a Web service interface through which one or several Managers (CAI3GManager) can provide user and service data to, and keep track of changes in management information in a System (CAI3GAgent).

CAI3G 1.2 is a synchronous provisioning interface based on SOAP 1.1.

1.1 Terminology

This specification uses a number of terms listed in alphabetical order as follows:

CAI3GAgent	The CAI3GAgent is the system receiving CAI3G request from CAI3GManager and responding to CAI3GManager. It also accepts Notification subscription from CAI3GManager and sends the Notification to CAI3GManager. Both a Mediation Function and a Network Element Function can be a CAI3GAgent.
CAI3GManager	The CAI3GManager is the system sending CAI3G Request, subscribing Notification to CAI3GAgent and receiving responses for the CAI3G operation and CAI3G Notification from CAI3GAgent. An example of CAI3GManager is a business system or a Portal.
Identifier	The structure or data used to identify Managed Object (MO) instances.
Identifier attributes	The elements of a certain identifier are attributes of MO instances. These attributes are named as identifier attributes.
Compound identifier	The identifier is composed by more than one identifier attributes.
MO	A Managed Object (MO) is a definition of a logic object in interface data model, which supports CAI3G provisioning operations. There is difference between an object in logical interface data model and one in CAI3GAgent internal data model. It is possible to have a one-to-many mapping between the objects in those two models though one-to-one mapping is more popular and straightforward in most cases. MO Type and MO instance identifier are two key parameters in CAI3G operations.



MOType	MOType is the name of the MO that has a data type as “xs:string”. MOType has three parts that are an MOName, a constant “@” sign and a URL equal to the namespace of the MO. The MOName MUST always start with an alphabetical character in either upper or lower case followed by zero or more alphabetical character or digit or under score.
MOId	MOId is the instance identifier of the MO defined in argument MOType. The value of MOId is an XML fragment containing identifier parameters. The schema of this XML fragment is not defined in this document. Each CAI3GAgent implementing the interface has to define own schema for MOId.
MOAttributes	MOAttributes are all attribute data for an MO. The value of MOAttributes is an XML fragment whose schema is not defined in this document. Each CAI3GAgent implementing the interface has to define own schema for it.
Session	Session is a series of interactions between two communication end points that occur during the span of a single connection. Typically, one end point requests a connection with another specified end point and if that end point replies agreeing to the connection, the end points take turns exchanging commands and data. The session begins when the connection is established at both ends and terminates when the connection is ended.
Transaction	Transaction is a mechanism that can be used to ensure the data consistency between the Managing and Managed systems.

1.2 Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY” and “OPTIONAL” in this document are to be interpreted as described in RFC-2119, see document Key Words for Use in RFCs to Indicate Requirement Levels.

This specification uses a number of namespace prefixes throughout that are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see XML Information Set).

Table 1 Prefixes and Namespaces used in This Specification

PREFIX	NAMESPACE
cai3g	http://schemas.ericsson.com/cai3g1.2/
http	http://schemas.xmlsoap.org/wsdl/http/



PREFIX	NAMESPACE
jaxws	http://java.sun.com/xml/ns/jaxws
mime	http://schemas.xmlsoap.org/wsdl/mime/
soap	http://schemas.xmlsoap.org/wsdl/soap/
SOAP-ENV	http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENC or soapenc	http://schemas.xmlsoap.org/soap/encoding/
Xjc	http://java.sun.com/xml/ns/jaxb/xjc
Xjb	http://java.sun.com/xml/ns/jaxb
xs or xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance





2 CAI3G Overview

CAI3G is an activation interface aiming for providing a simple, up-to-date and unified provisioning interface for the network elements in telecommunication and other Information Technology (IT) network.

As a descendant of the widely used successful CAI interface, CAI3G inherits the simple operations from its precedent. It gets rid of the drawbacks in CAI by means of the modern Extensible Markup Language (XML) and Web Service technologies.

With the powerful XML document structure, CAI3G is a flexible protocol that can easily adapt to complex data models and integrate aggregation functions. So not only Network Element Function but also Mediation Function can implement this interface for more complicated data model with a lower cost and effort in the service network domain as well as the core network domain.

A typical deployed environment is shown in Figure 1.

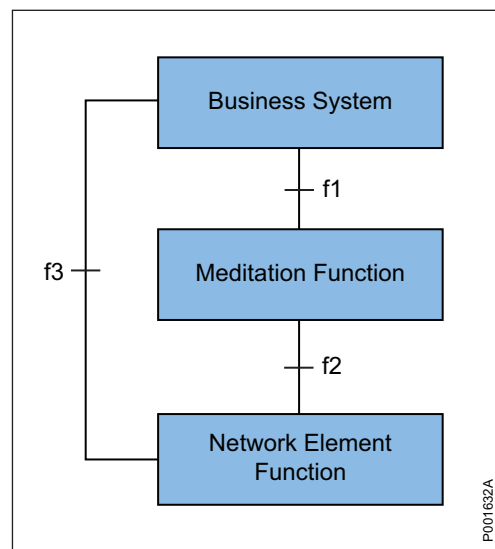


Figure 1 CAI3G Deployment

The Network Element Function provides services to end-users. It implements interface f2 and f3 for provisioning from Business System and Mediation Function respectively.

The Mediation Function is a gateway between Business System and Network Element Functions, which provides a common provisioning interface to Business System probably with some extra services. It consumes interface f2 to provision Network Element Functions when implementing interface f1 for Business Systems.

The Business System handles the billing, provisioning and customer relationships, which consumes either f3 to provision Network Element Functions directly or f1 to provision Network Element Functions via Mediation Function.

CAI3G can be used as a provisioning protocol at f1, f2 and f3 in the graph.

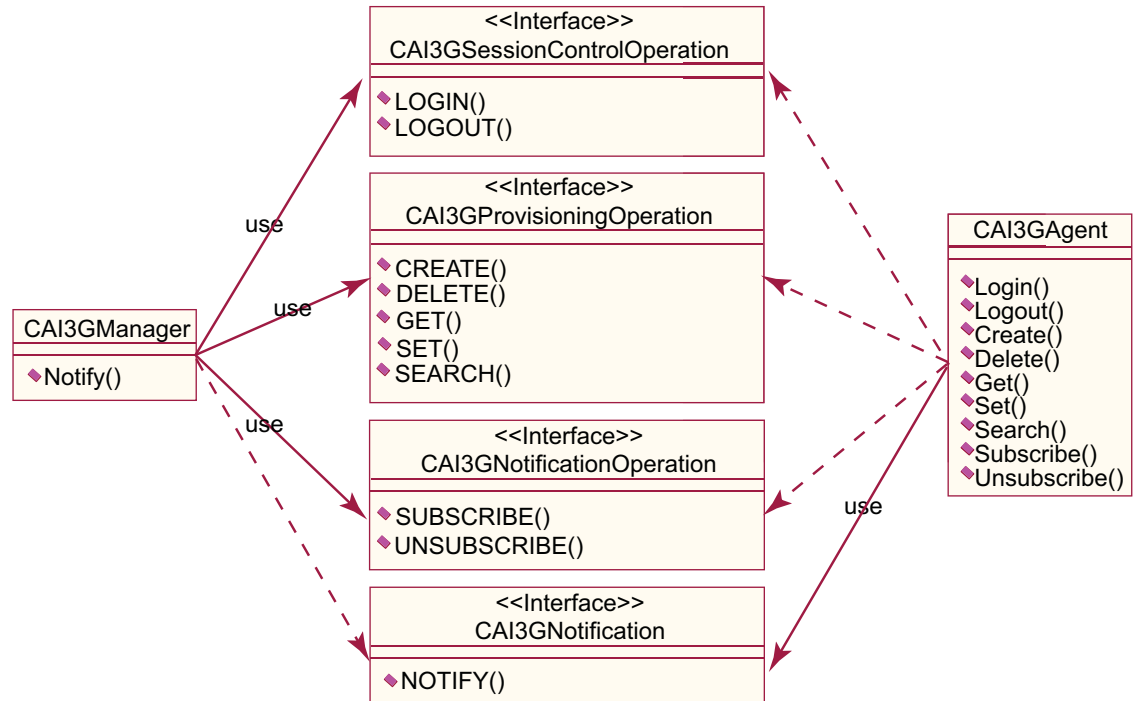


Figure 2 CAI3G Interface Overview

Figure 2 gives an overview of the CAI3G interface. CAI3GManager realizes the CAI3G Notification interface with NOTIFY operation. CAI3GAgent realizes the CAI3GSessionControlOperation interface, CAI3GProvisioningOperation interface, and CAI3GNotificationOperation interface.

A provisioning operation is initiated from the CAI3GManager to CAI3GAgent after CAI3GManager has created a session. CAI3GAgent generates the notification to CAI3GManager after CAI3GManager subscribes the notification service.

An example of CAI3G message flow is shown in Figure 3:

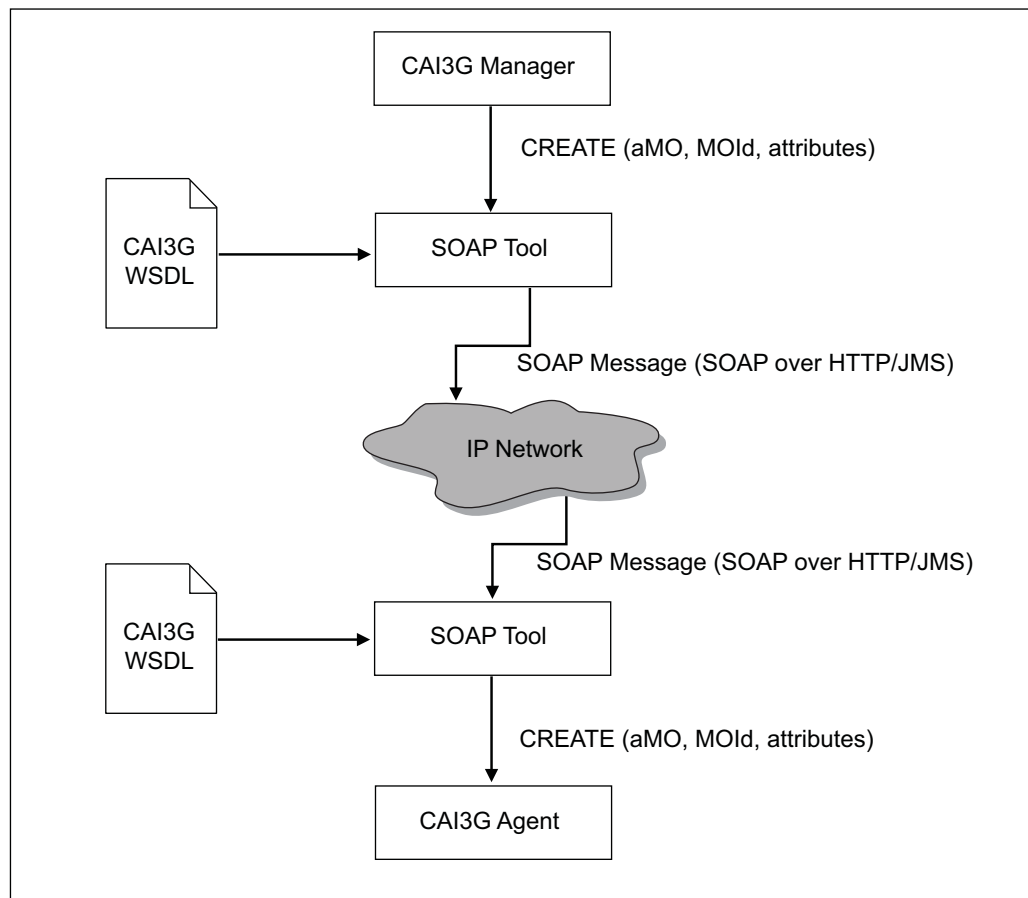


Figure 3 CAI3G Message Flow Example

2.1 CAI3G Web Service

CAI3G web services are published in the form of Web Service Description Language (WSDL), which can use HTTP or JMS as transportation layer.

Figure 4 illustrates the visualized HTTP-based CAI3G web-service structure according to the WSDL files shown in Section 11 on page 67. The SessionControl service contains the Login and Logout operations. The Notification service contains the Subscribe, Unsubscribe, and Notify operations. The SessionControl service and Notification service have separate WSDL files. Regarding the Provisioning service, a specific WSDL file must be provided for each specific MO, and it contains the Create, Get, Set, Delete, and Search operations. The details of the services and the operations are discussed in Section 4 on page 19. The web service document-literal style is used as the standard binding method in CAI3G 1.2.

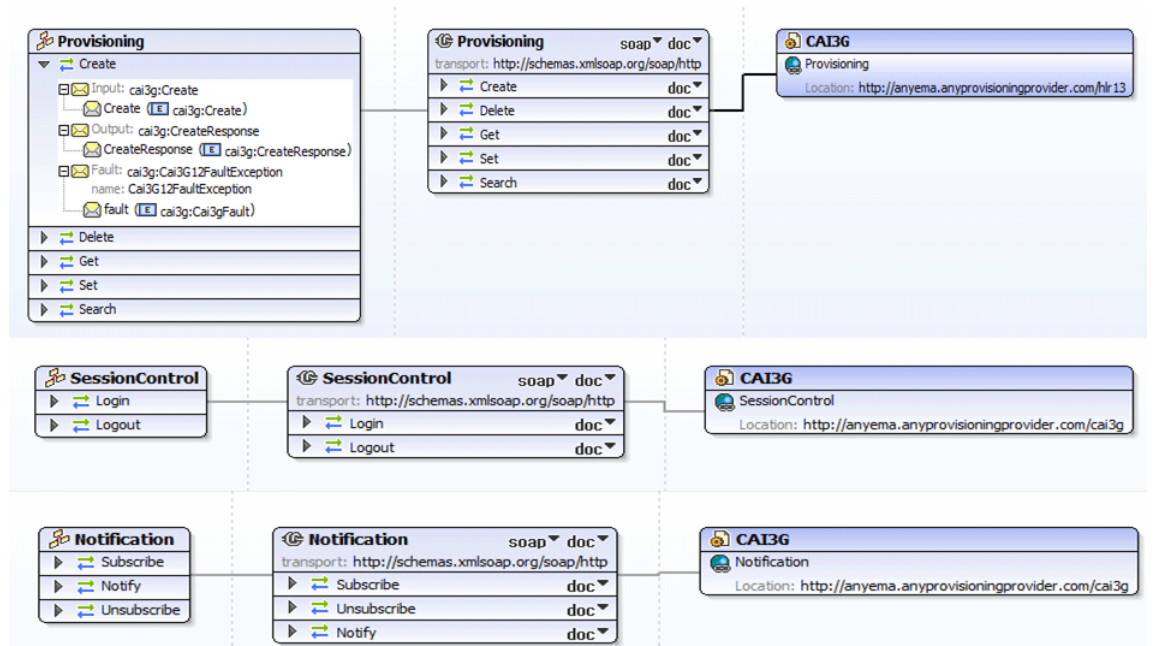


Figure 4 CAI3G Web-Service Structure for HTTP-Based WSDL

Figure 5 illustrates the visualized JMS-based CAI3G web-service structure according to the WSDL files shown in Section 11 on page 67.

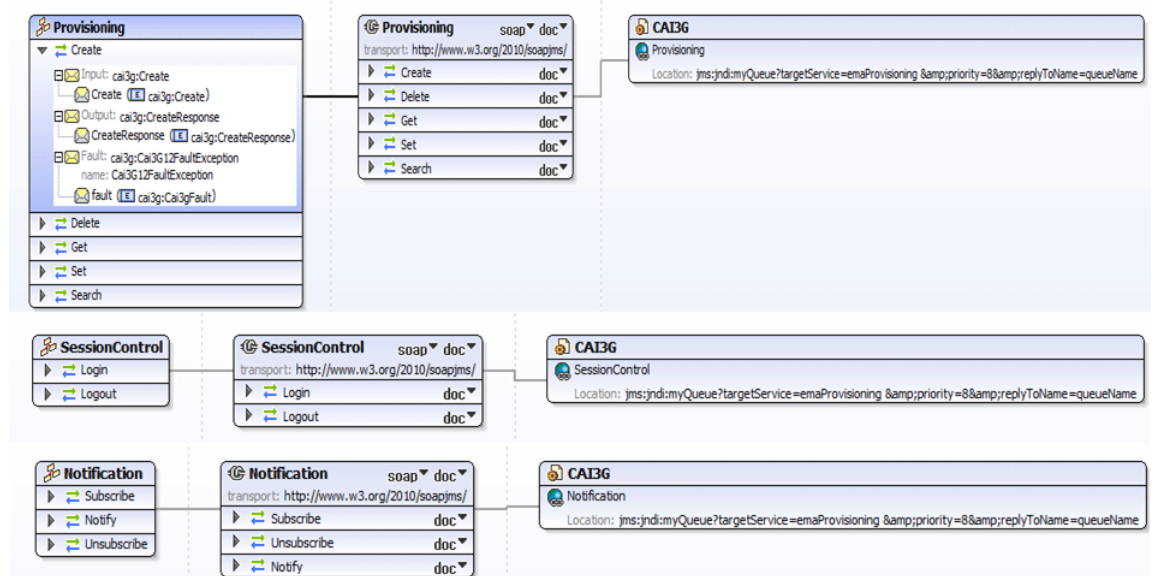


Figure 5 CAI3G Web-Service Structure for JMS-Based WSDL

Note: The CAI3G web-service structure for JMS-based WSDL is similar to the structure for HTTP-based WSDL except the definition of transportation layer in the binding element and service element of the WSDL file.



Any application that uses the CAI3G interface can send SOAP requests directly. Because SOAP creates some overhead information, it is recommended to use a SOAP Third Party Product (3PP) for this purpose. By using a SOAP client together with the CAI3G WSDL files, the application obtains a simple Application Programming Interface (API) to use and the actual SOAP messages are hidden by the SOAP client.

Below are the main advantages of CAI3G Web Service Interface:

- Using open web service specification (SOAP, XML, WSDL, and JMS), supported by a wide range of 3PPs
- Easy integration by use of API
- Hide network complexity
- Capable of managing complex data model
- Support for notification towards business systems

2.2 Namespace

The namespace of CAI3G 1.2 is as follows:

`http://schemas.ericsson.com/cai3g1.2/`

2.3 Synchronous Interface

CAI3G 1.2 is a pure synchronous interface that means a CAI3GManager MUST NOT send a request on the same logical session before the response of the previous request comes back from CAI3GAgent.





3 Protocol Stack

CAI3G 1.2 is based on SOAP 1.1 and has five levels in its protocol stack.

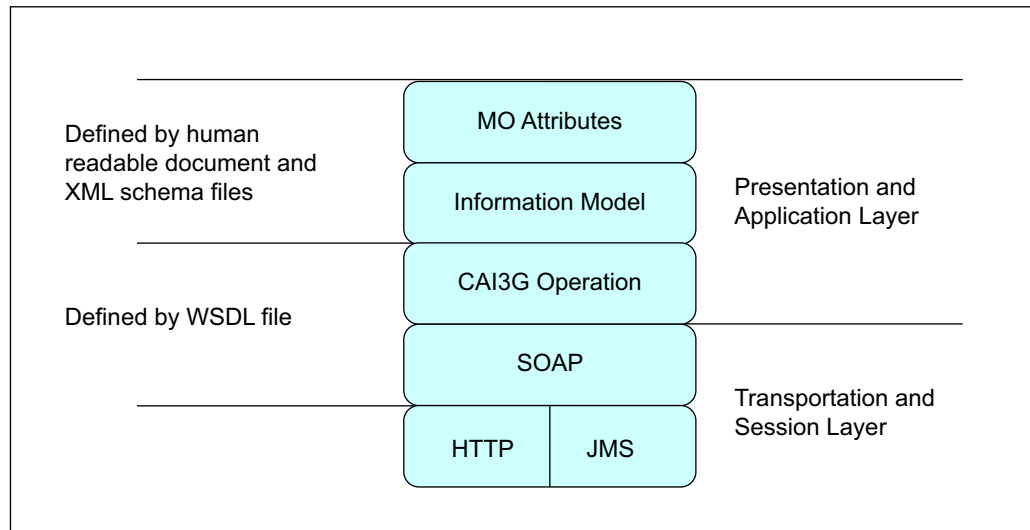


Figure 6 CAI3G Communication Protocol Stack

Basic features of the CAI3G are:

- SOAP-message based presentation and session layer
- Multiple concurrent sessions
- Programming language and platform independent
- Working in a security environment with firewall
- Using open web service specification (SOAP, XML, WSDL, and JMS), supported by wide variety of 3PP

The following standards are used as transport and network layers:

- SOAP Specification version 1.1 (see Simple Object Access Protocol (SOAP) 1.1)
- HTTP/1.1 (see Hypertext Transfer Protocol HTTP/1.1)
- JMS 1.1 (see Java Message Service)

3.1 SOAP

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an Extensible Markup Language (XML) based protocol (see document Extensible Markup Language (XML) 1.1 (Second Edition)).



SOAP can potentially be used in combination with a variety of other protocols, such as HTTP, HTTPS, HTTP Extension Framework, or JMS.

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to Remote Procedure Call (RPC).

A SOAP message contains the following:

- 1 **SOAP envelope construct** defines an overall framework for expressing what is in a message, which should deal with it, and whether it is optional or mandatory.
- 2 **SOAP Header** is a generic mechanism for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties.
- 3 **SOAP Body** is a container for mandatory information intended for the ultimate recipient of the message. SOAP defines one element for the body, which is the Fault element used for reporting errors (see Section 4.4 on page 35).

Below are given two examples of SOAP messages embedded in HTTP request and response respectively.



```

POST/CAI3GAgent HTTP/1.1
Host:www.mycompany.com
Content-Type:text/xml;charset="utf-8"
Content-Length:326
SOAPAction:"CAI3G#Create"

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  xmlns:abc="http://schemas.mycompany.com/myservice/">
  <SOAP-ENV:Header>
    <cai3g:SessionId>1234567</cai3g:SessionId>
    <cai3g:TransactionId>4294967</cai3g:TransactionId>
    <cai3g:SequenceId>42949672</cai3g:SequenceId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <cai3g:Create>
      <cai3g:M0Type>User@http://www.mycompany.com/myservice/<cai3g:M0Type>
      <cai3g:M0Id>
        <abc:socialId>123-456-789</abc:socialId>
      </cai3g:M0Id>
      <cai3g:M0Attributes>
        <abc:CreateUser socialId="310102197204084417">
          <abc:name>Alice</abc:name>
          <abc:gender>Female</abc:gender>
          <abc:age>46</abc:age>
        </abc:CreateUser>
      </cai3g:M0Attributes>
    </cai3g:Create>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example 1 SOAP Message Embedded in HTTP Request

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=6DA3BCFC2FAD827686D58D4295C830D0; Path=/
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Tue, 24 Apr 2012 06:02:12 GMT

```

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <SessionId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 1234567</SessionId>
    <TransactionId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 4294967</TransactionId>
    <SequenceId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 42949672</SequenceId>
  </S:Header>
  <S:Body>
    <CreateResponse xmlns="http://schemas.ericsson.com/cai3g1.2/">
      <M0Id>
        <socialId xmlns="http://schemas.mycompany.com/myservice/"
  xmlns:cai3="http://schemas.ericsson.com/cai3g1.2/" xmlns:soapenv="http://schemas.xmlsoap.org/s
        </M0Id>
      </CreateResponse>
    </S:Body>
  </S:Envelope>

```

Example 2 SOAP Message Embedded in HTTP Response

The SOAP message embedded JMS request/response has the same payload as the one of the HTTP request/response. A customized property with the specified name SOAPAction in JMS request is mandatory, and other properties are optional. The SOAPAction property in JMS request has the same value as the one in HTTP request. All the headers in JMS request are optional.



```
JMSXDeliveryCount      1
JMSExpiration           0
JMSRedelivered         false
JMSDeliveryMode        1
SOAPAction              CAI3G#Create
JMSPriority             4
JMSXGroupSeq           0
JMSTimestamp            1316758545614
```

Example 3 JMS Properties in JMS Request

```
Correlation ID
Destination          queue://receive
Expiration           0
Group
Message ID           ID:ema6751.eapac.ericsson.com-36849-1316758545438-0:0:1:1:1
Persistence          Non Persistent
Priority             4
Redelivered          false
Reply To
Sequence            0
Timestamp            2011-09-23 14:15:45:614 CST
Type
```

Example 4 JMS Headers in JMS Request

In Example 1, a SOAP request is sent to a CAI3G provisioning service. The request tries to create a “User” MO instance for that service. In Example 2, the response message contains the HTTP message with the SOAP message as the payload.

The SOAP Envelope element is the top element of the XML document representing the SOAP message. XML namespaces are used to disambiguate SOAP identifiers from application specific identifiers. It is worth noting that the rules governing XML payload format in SOAP are entirely independent of the fact that the payload is carried in HTTP.

3.1.1 SOAP Message Exchange Model

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but as illustrated in Example 1 and Example 2, SOAP messages are often combined to implement patterns such as request/response.

SOAP implementations can be optimized to exploit the unique characteristics of particular network systems. For example, the HTTP binding provides for SOAP response messages to be delivered as HTTP responses, using the same connection as the inbound request.

Regardless of the protocol to which SOAP is bound, messages are routed along a so-called “message path”, which allows for processing at one or more intermediate nodes in addition to the ultimate destination.

A SOAP application receiving a SOAP message **MUST** process that message by performing the following actions in the order listed below:

- 1 Identify all parts of the SOAP message intended for that application.



- 2 Verify that all mandatory parts identified in step 1 are supported by the application for this message and process them accordingly. If this is not the case then discard the message. The processor MAY ignore optional parts identified in step 1 without affecting the outcome of the processing.
- 3 If the SOAP application is not the ultimate destination of the message then remove all parts identified in step 1 before forwarding the message.

3.1.2 SOAP Document Style

WSDL 1.1 (see Web Service Description Language (WSDL) 1.1) supports two message styles, RPC and document.

When the document style is used, the client understands that it should make use of XML schemas rather than remote procedure calling conventions. The latter requires the use of RPC style.

There are advantages and disadvantages around both methods. CAI3G 1.2 will use SOAP Document style in order to get the following benefits:

- Document style makes full use of XML
- Document style does not require a rigid contract
- Document style is better suited for asynchronous processing
- Document style makes object exchange more flexible

Note: The choice of document style does not necessarily impact the programming model against the SOAP toolkit used to build a CAI3GManager and/or CAI3GAgent. It is in many cases possible to get an “RPC like” API with methods and parameters even though the style “on the wire” is document.

CAI3G 1.2 uses SOAP Document binding method.

3.2 CAI3G Operations

CAI3G Interface comprises of three categories of operations. They are Provisioning Operations, Session Control Operations and Notification Operations.

The Provisioning Operations contain five basic operations that support the provisioning to different MOs defined in the interface data model, see Section 3.3 on page 16. They are CREATE, DELETE, SET, GET and SEARCH operations. A CREATE operation adds a new MO instance in CAI3GAgent's “database” (for implication of database, see the Note at the end of this section) via CAI3G interface, while a DELETE operation removes an MO instance from the database. A SET operation modifies the data for an existing MO instance in the database, while a GET operation retrieves the data for a dedicated MO instance if the identifiers are specified in the request or, optionally, all MO instance identifiers

if no identifier is provided in the request. A SEARCH operation retrieves the MO instance ids according to the inputted filter constraints.

The Session Control Operations contain LOGIN and LOGOUT. A LOGIN operation establishes a session between CAI3GManager and CAI3GAgent. A LOGOUT operation just terminates a session created by a LOGIN operation.

The Notification Operations contain Subscribe, Unsubscribe and Notify operations. CAI3GManager uses Subscribe and Unsubscribe operations to register and unregister the notification service provided by CAI3GAgent. CAI3GAgent uses Notify operation to inform CAI3GManager a notification.

Note: The database here is a logic database. In the case that the target CAI3GAgent is a Network Element Function, it could be a physical database. To a Mediation Function, there is no physical database. Instead it should be stored in a real database in the Network Element Function the Mediation Function connected to.

3.3 Interface Data Model

The information model contains all MOs that defined in the interface. A Managed Object (MO) is a definition of an object in data model, which can be operated and maintained by the application.

There is difference between an object in the logical interface data model and one in CAI3GAgent internal data model. In most of cases, one MO in CAI3GAgent internal data model can be one-to-one mapped into a logic MO in the interface data model. But CAI3G 1.2 standard also supports a many-to-one mapping between MOs in CAI3GAgent data model and MO in interface data model in some special cases.

The following example shows the two different mapping scenarios. In this simple example, only two types of object exist in the physical data model. One is the Subscriber type, and the other is the User type. There is a one-to-many has/belong-to relationship between those two object types so that one Subscriber object instance has several User object instances and one User object instance only belongs to one Subscriber object instance.

There are two ways to design a provisioning interface data model in this tiny physical data model. One way is to define two MOs in the interface data model, which has a one-to-one mapping from the physical data model. So CAI3G has a Subscriber object and a User object in our interface data model. Each of them has own operations. The other way is to define one MO in the interface data model that is the Subscriber object. And inside this MO, a sub-object parameter, User, is defined for it. In the second case, only the Subscriber object has the operations.

For more detailed information about sub-object parameter, see Section 5.3.3 on page 50.

Although both solutions are valid in CAI3G 1.2. The one-to-one mapping solution is more flexible and stable in the case there are many object instances related to



one instance of other object, while the sub-object parameter solution is simpler if the number of sub-object instances is not big.

Designer SHOULD use one-to-one mapping in the case there are lots of sub-object instances and use sub-object parameters in other cases.

The MOs defined in interface data model are distinguished by the local-name of the MO Type and the namespace. Each namespace and local MO Type pair MUST be unique against others. This gives the designer the opportunity to name MO type with the same name in different CAI3GAgent. All MOs that are defined in the CAI3G interface have to contain at least one MO instance identifier distinguishing MO instance from MO instance.

Each MO in CAI3G interface MUST contain at least one MO instance identifier.

3.4 MO Attributes

The “MO Attributes” is at the top level of the protocol stack. In CAI3G 1.2, each MO contains multiple attributes. Each attribute follows the structure listed in this section.

Attribute	An Attribute can be either a SimpleAttribute, or a StructureAttribute, or a SubobjectAttribute.
SimpleAttribute	A SimpleAttribute can be either a SingleValueAttribute or a MultipleValueAttribute.
SingleValueAttribute	A SingleValueAttribute comprises of an AttributeName and an AttributeValue.
AttributeName	An AttributeName is a plain text string following the regular expression, [A-Za-z][_A-Za-z0-9]*.
AttributeValue	An AttributeValue is any value defined in MO schema file.
MultipleValueAttribute	A MultipleValueAttribute comprises of an AttributeName and several AttributeValues.
StructureAttribute	A StructureAttribute comprises of an AttributeName and several Attributes.
SubobjectAttribute	A SubobjectAttribute comprises of an AttributeName, a Key and several SubAttributes.
Key	A Key can be either a SingleKey or a CompoundKey.



SingleKey	A SingleKey is a SingleValueAttribute.
CompoundKey	A CompoundKey comprises of several SingleValueAttributes.

Above example shows the structure definition of the attributes. More detailed information of attributes is described in Section 5.3 on page 44.



4 CAI3G Operation

The CAI3G operation suite contains three services. They are the Session Control service, Provisioning service, and Notification service. This section describes each of them in details.

4.1 Session Control Service

CAI3G provisioning operations and notification operations can only be sent in a CAI3G session or WS-Security profile.

Note: When the WS-Security profile is used, the Session Control service can be optional, which is decided by the implementation of the CAI3GAgent. There is no specific requirement for the WS-Security profile from the generic CAI3G perspective. The CAI3GAgent decides how to implement the WS-Security profile.

A CAI3G session established by a valid user of the CAI3GAgent has the same authority context as that of the user.

A CAI3G session starts when a LOGIN command is sent from CAI3GManager. It stays as active until either a LOGOUT command received by CAI3GAgent or a session time-out occurs.

A CAI3GAgent MAY implement session time-out mechanism that a session MAY be closed and/or considered as invalid if the timer of this session maintained in CAI3GAgent expires. The timer MUST be reset when a request coming from CAI3GManager arrives on this logical session. This document does not cover the way to define the length of the timer. It depends on each implementation.

After a LOGIN command is received by CAI3GAgent, it checks the username and password in the request. If that information is correct, a system unique (for implication of “system unique” see next paragraph) sessionId among all current active sessions is generated and sent back to CAI3GManager in the response. After that all requests and responses containing that sessionId are considered as the requests and responses of that session.

“System Unique” in this document means that the uniqueness MUST be guaranteed within a CAI3G provisioning service node. If a CAI3G provisioning service node contains several load-sharing CAI3GAgents, the uniqueness of the sessionId MUST be provided to all CAI3GAgents involved.

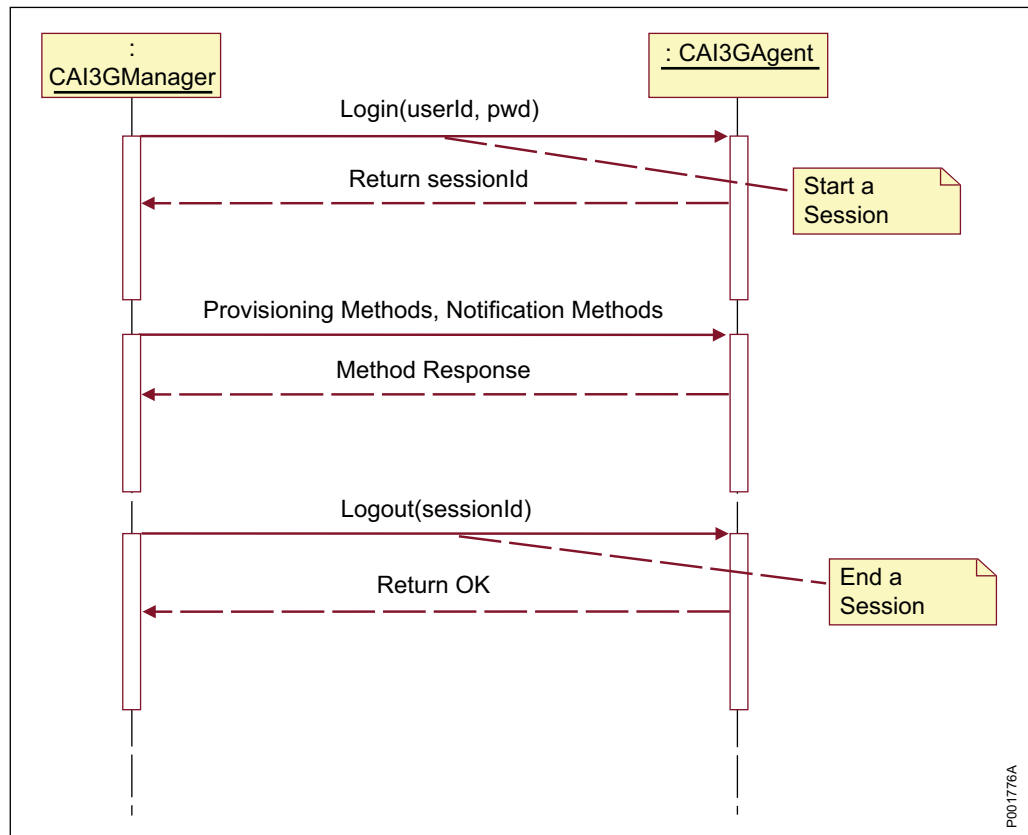


Figure 7 Session Lifecircle

4.1.1 LOGIN Command

User sends a LOGIN command in order to establish a provisioning session between CAI3GManager and CAI3GAgent. The pseudo-method of LOGIN command looks like:

LOGIN (IN userId, IN pwd, OUT sessionId, OUT baseSequenceId)

There are two IN attributes and two OUT attributes in this command.

userId

The userId contains the name of the user who would like to start a session for provisioning. It SHOULD always start with an alphabetical character in either upper or lower case followed by zero or more alphabetical characters or digits or underscores. This is an input string expected from CAI3GManager.

pwd

The pwd contains the password of that user. It is a plain text string that is not encrypted. This is an input string expected from CAI3GManager.

**sessionId**

The sessionId is the system unique session identifier automatically generated by CAI3GAgent. CAI3GAgent returns this attribute to CAI3GManager.

CAI3GAgent SHOULD guarantee the sessionId is system unique.

baseSequenceId

The baseSequenceId is the random number in the range of 0 to $2^{32}-1$ automatically generated by CAI3GAgent, which is the start number of the sequenceId in each CAI3G request from CAI3GManager. The usage of the sequenceId is described in Section 8 on page 61.

Here is the XML schema definition of the LOGIN request and response.

```
<!-- Login Request -->
<xs:element name="Login">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="userId" type="xs:string"/>
      <xs:element name="pwd" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Login Response -->
<xs:element name="LoginResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sessionId" type="SessionIdType"/>
      <xs:element name="baseSequenceId" type="xs:unsignedLong"/>
    </xs:sequence>
  </xs:complexType>

<!-- Session Id Data Type -->
<xs:simpleType name="SessionIdType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[d\w]{1,}" />
  </xs:restriction>
</xs:simpleType>
```

Example 5 LOGIN request and response

In the case, a CAI3GManager sends another LOGIN after it has already built up a session and that session is still active. The CAI3GAgent SHOULD still answer the request with a new sessionId. It will totally depend on CAI3GManager's implementation to handle the response.

One important step in login procedure is the check of version support. A session MUST be set up only if CAI3GAgent can support the version mentioned in the request from CAI3GManager. A special error will be returned in the case when the version is not matched between CAI3GManager and CAI3GAgent.

4.1.2**LOGOUT Command**

User uses LOGOUT command to terminate an established session. The pseudo-method of LOGOUT looks like:

LOGOUT(IN SessionId)



There is only one parameter in this method, which is the session identifier of the session that needs to be terminated. The CAI3GAgent stops the session according to the SessionId in the LOGOUT request only if the SessionId in the message part equals to the one in the header. Otherwise, the error SHOULD be reported.

In the same session, any received request after a successful LOGOUT will receive an error response indicating of invalid session id.

Here is the XML schema definition of the LOGOUT request and response.

```
<xs:element name="Logout">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sessionId" type="SessionIdType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LogoutResponse">
  <xs:complexType/>
</xs:element>

<!-- Session Id Data Type -->
<xs:simpleType name="SessionIdType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d\w]{1,}"/>
  </xs:restriction>
</xs:simpleType>
```

Example 6 LOGOUT request and response

4.2 Provisioning Service

The Provisioning service is the main part of the CAI3G interface. The prerequisite to execute a provisioning command or method is that a session has been built up or WS-Security is used. See Section 4.1 on page 19 for detailed information about session control in CAI3G interface.

The generic Provisioning service specification is only a template reference for a CAI3G Agent to define its specific provisioning services (schema and WSDL file).

4.2.1 CREATE Operation

The CREATE operation is used to create an MO instance in the CAI3G interface data model. The method used in CREATE operation is method CREATE. The structure of this pseudo-method looks like:

CREATE(IN MOType, [IN MOId], [IN MOAttributes], [IN extension], OUT MOId, [OUT MOAttributes], [OUT extension])

MOType The MOType in a CREATE request indicates the type of the target MO.



MOId

The MOId in a CREATE request indicates that the MO instance needs to be generated. It is optional in the request. The MOId in a CREATE response indicates that the MO instance has been created.

If the MOId is absent from a CREATE request, the CAI3GAgent SHOULD generate or allocate a new identifier that is returned in the CREATE response. Otherwise, CAI3GAgent just simply replies the MOId in the request.

MOAttributes

The MOAttributes in a CREATE request contains all data prepared for an MO creation. It is optional in the request. All mandatory attributes for an MO have to be filled in a CREATE request. If an MO has no mandatory attributes except MOId, the MOAttributes SHOULD be omitted in the CREATE request. The MOAttributes MAY be included also in a CREATE response containing all data in the created MO.

extension

The extension in a CREATE request is defined with the data type "AnySequenceType". This parameter can contain extra data except MOType, MOId and MOAttributes.

Here is the XML schema fragment of the CREATE request and response.

```
<!-- Create Request -->
<xs:element name="Create">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOType" type="MoType"/>
      <xs:element name="MOId" type="AnyMOIdType" minOccurs="0"/>
      <xs:element name="MOAttributes" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Create Response -->
<xs:element name="CreateResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOId" type="AnyMOIdType"/>
      <xs:element name="MOAttributes" type=
        "GetResponseMOAttributesType" minOccurs="0"/>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 7 CREATE request and response



4.2.2 GET Operation

GET operation is used to fetch all data of an MO instance or the identifiers of all instances for a certain MO type. The method used in READ operation is GET method. The structure of this pseudo-method looks like:

```
GET (IN MOType, [IN MOId], [IN MOAttributes], [IN extension], [OUT MOAttributes], [OUT extension]); //Fetch data of an MO.
```

```
GET (IN MOType, [IN MOAttributes], [IN extension], [OUT MOIds], [OUT extension]); //Fetch all MO instance identifiers
```

The **MOType** in a GET request indicates the type of the target MO.

The **extension** in a GET request is defined with the data type “AnySequenceType”. This parameter can contain extra data except MOType, MOId and MOAttributes.

MOId The MOId is optional in a GET request. When contained in a GET request, it indicates the specific MO instance needs to be fetched. When being absent it indicates to fetch all MO instance identifiers.

MOAttributes The MOAttributes is optional in a GET request containing filtering data for fetching MO instance(s). The MOAttributes MAY be included also in a GET response containing data of the fetched MO instance(s).

4.2.2.1 Fetch Data of an MO

An MO instance's data is fetched from the GET method, when the input parameter MOId is present and has a value.

The **MOId** existing in a GET request indicates the MO instance that will be retrieved. If the MOId exists in a GET request, the MOAttributes is returned from CAI3G Agent with attribute data in the MO instance specified by MOId.

The input parameter MOAttributes can also be present together with MOId in a GET method. This input parameter MOAttributes acts as filtering data for fetching data of an MO instance. The purpose of input parameter MOAttributes is implementation dependent.

4.2.2.2 Fetch all MO Instance Identifiers

GET method fetches the identifiers of all instances for a certain MO type when:

- The input parameter “MOId” has an empty value or there is no “MOId” in the request. And in this case “MOAttributes” also must be empty
- The input parameter “MOAttributes” has an empty value or there is no “MOAttributes” in the request

It depends on each CAI3G realization to support this function in all MOs, or partly, or none.



There is no output “MOAttributes” argument in the response. Instead, zero or more “MOId” are sent back from CAI3GAgent to CAI3GManager.

4.2.2.3 XML Schema for GET Request and Response

This section gives the XML schema fragment of a GET request and response.

```
<!-- Get Request -->
<xs:element name="Get">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOType" type="MoType"/>
      <xs:element name="MOId" type="AnyMOIdType" minOccurs="0"/>
      <xs:element name="MOAttributes" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="extension" type="AnySequenceType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Get Response -->
<xs:element name="GetResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOId"
        type="AnyMOIdType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="MOAttributes"
        type="GetResponseMOAttributesType" minOccurs="0"/>
      <xs:element name="extension" type="AnySequenceType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 8 GET request and response

4.2.3 SET Operation

The SET operation is used to change the MO instance data. The method of this operation is SET method. The pseudo-method structure of SET command looks like:

SET(IN MOType, IN MOId, IN MOAttributes, [IN extension], [OUT MOAttributes], [OUT extension])

MOType	The MOType in a SET request indicates the type of the target MO.
MOId	The MOId in a SET request indicates the MO instance need to be updated.
MOAttributes	The MOAttributes is mandatory in a SET request containing all update data for an MO instance. The MOAttributes MAY be included also in a SET response containing all changed data in the updated MO.



extension The extension in a SET request is defined with the data type “AnySequenceType”. This parameter can contain extra data except MOType, MOId and MOAttributes.

Here is the XML schema fragment of a SET request and response.

```
<!-- Set Request -->
<xs:element name="Set">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOType" type="MoType"/>
      <xs:element name="MOId" type="AnyMOIdType"/>
      <xs:element name="MOAttributes">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Set Response -->
<xs:element name="SetResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOAttributes" type=
        "GetResponseMOAttributesType" minOccurs="0"/>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 9 SET request and response

4.2.4 DELETE Operation

The DELETE operation is used to remove an MO instance from the interface data model. The method used in DELETE operation is DELETE. This pseudo-method has a structure that looks like:

DELETE (IN MOType, IN MOId, [IN MOAttributes], [IN extension], [OUT MOId], [OUT MOAttributes], [OUT extension]).

There are four parameters in DELETE method.

MOType The MOType in a DELETE request indicates the type of the target MO.

MOId The MOId in a DELETE request indicates that the MO instance needs to be removed.



MOAttributes	The MOAttributes in both DELETE request and response is an optional parameter. To the input parameter MOAttributes, it acts as filtering data for deletion of an MO instance and the purpose of it is implementation dependent. To the output parameter MOAttributes, it is returned from CAI3GAgent with all attribute data in the MO instance specified by MOId.
extension	The extension in a DELETE request is defined with the data type "AnySequenceType". This parameter can contain extra data except MOType, MOId and MOAttributes.

Here is the XML schema fragment of a DELETE request and response.

```

<!-- Delete Request -->
<xs:element name="Delete">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOType" type="MoType"/>
      <xs:element name="MOId" type="AnyMOIdType"/>
      <xs:element name="MOAttributes" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="extension" type="AnySequenceType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Delete Response -->
<xs:element name="DeleteResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOId" type="AnyMOIdType" minOccurs="0"/>
      <xs:element name="MOAttributes"
        type="GetResponseMOAttributesType" minOccurs="0"/>
      <xs:element name="extension" type="AnySequenceType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Example 10 DELETE request and response

4.2.5 SEARCH Operation

The SEARCH operation is used to retrieve the MO instances for a certain MO type that meet the filter condition from the interface data model. The method used in SEARCH operation is SEARCH. This pseudo-method has a structure that looks like:

SEARCH(IN MOType, IN filters, [IN extension], [OUT MOId], [OUT extension])

There are three parameters in SEARCH method:

MOType	The MOType in a SEARCH request indicates the type of the target MO.
---------------	---

**filters**

The filters in a SEARCH is a mandatory input argument from CAI3GManager. It specifies a filter constraint that CAI3GAgent will use to filter MO instances. CAI3GAgent will add the MO into response if the MO instance satisfies the filter constraint. The filter string should use the XPath expression.

The parameters that are filterable in CAI3G is MOAttributes. The filters is an XML structure of type SearchFiltersType which contains a number of elements of type SearchFilterType.

The filterable parameters MUST be supplied in accordance to the following rules: The relation between different filters is "OR". The relation between different "MOAttributes" is "AND".

MOAttributes is the only filterable parameter contained in filter and it must support XPATH expression string, and the different implementation MAY decide the subset to support.eg.,

```
<filters>
  <filter>
    <MOAttributes>/SPAdministrator[spId =
      "200"]
    </MOAttributes>
  </filters>
</filter>
```

This filter is to match the attribute spId under MO "SPAdministrator" with value 200.

extension

The extension in a SEARCH request is defined with the data type "AnySequenceType". This parameter can contain extra data except MOType, MOId and MOAttributes.

Here is the XML schema fragment of a SEARCH request and response.



```
<!--Search Request-->
<xs:element name="Search">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="M0Type" type="MoType"/>
      <xs:element name="filters" type="SearchFiltersType"/>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Search Response -->
<xs:element name="SearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="M0Id" type="AnyM0IdType" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="extension" type="AnySequenceType"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 11 SEARCH request and response

4.3 Notification Service

Notification service contains the notification related operations. In order to get notification from CAI3GAgent, a CAI3GManager has to register itself first. The procedure of the registration is called Subscribe. The notification is terminated when CAI3GManager unsubscribes a pre-registration.

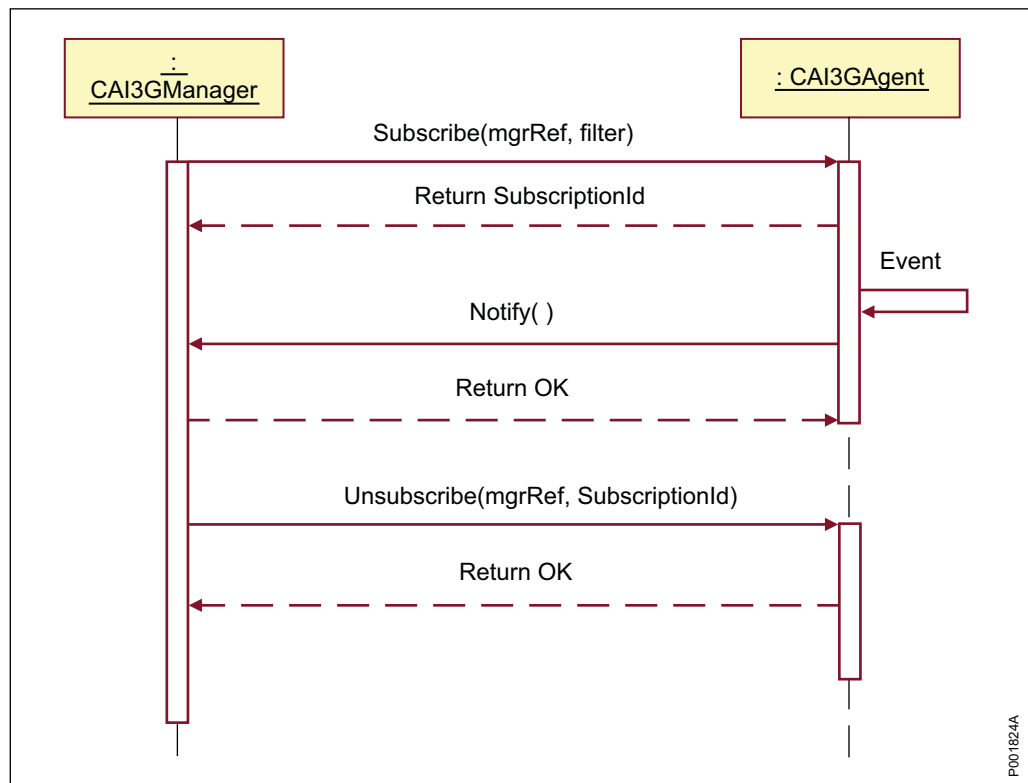


Figure 8 CAI3G Notification

Before sending Subscribe or Unsubscribe methods to CAI3GAgent, CAI3GManager has to set up a session using the commands described in Section 4.1 on page 19.

4.3.1 Subscribe Operation

CAI3GManager cannot get any notification information from CAI3GAgent until it registers itself into the notification service provided by CAI3GAgent. The pseudo-method of Subscribe operation looks like:

Subscribe(IN managerRef, IN filters, OUT subscriptionId)

There are two input arguments and one output argument in the method.

The **managerRef** is a mandatory input argument from CAI3GManager. It specifies the reference of CAI3GManager to which CAI3GAgent will send notification. The data type of this attribute is a URI.

The **filters** is a mandatory input argument from CAI3GManager. It specifies a filter constraint that CAI3GAgent will use to filter notification. CAI3GAgent will notify CAI3GManager if the event satisfies the filter constraint.

The parameters that are filterable in CAI3G are cai3gUser, MOType, operation, MOId and MOAttributes (see Section 4.3.2 on page 33). The filters is an XML



structure of type NotificationFiltersType which contains a number of elements of type NotificationFilterType.

The filterable parameters MUST be supplied in accordance to the following rules:

cai3gUser	Exact matches are supported. It means "user1" can be matched to "user1".
MOType	MUST support both match as whole word and wildcard format: "A@*", "*@B" and "*@*", the "*" can be matched to any characters.
operation	MUST match as whole word, and only support enum values {Create, Set, Delete}
MOId	MUST support XPATH expression string, and the different implementation MAY decide the subset to support, e.g., <MOId>/mo1[usrid > 1000]</MOId>.
MOAttributes	MUST support XPATH expression string, and the different implementation MAY decide the subset to support, e.g., <MOAttributes>/moattr[corpId = "200"]</MOAttributes>.



```

<xs:element name="Subscribe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="managerRef" type="xs:anyURI"/>
      <xs:element name="filters" type="NotificationFiltersType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="OperationUnique">
    <xs:selector xpath="filter/operation"/>
    <xs:field xpath="."/>
  </xs:unique>
</xs:element>

<xs:element name="SubscribeResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="subscriptionId" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="NotificationFiltersType">
  <xs:sequence>
    <xs:element name="filter" type="NotificationFilterType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="NotificationFilterType">
  <xs:sequence>
    <xs:element name="cai3gUser" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="MOType" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="operation"
      type="NotificationOperationType" minOccurs="0" maxOccurs="3"/>
    <xs:element name="MOId" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="MOAttributes"
      type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="NotificationOperationType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Create"/>
    <xs:enumeration value="Delete"/>
    <xs:enumeration value="Set"/>
  </xs:restriction>
</xs:simpleType>

```

Example 12

The interpretation of the contents within one NotificationFilter or multiple NotificationFilters are as follows:

If the contents is within one NotificationFilter, all elements with the same name make up the vector of that parameter. The whole condition space is the cross multiplication of all vectors, i.e., the values of the elements with the same name have "OR" relation, while the values of different elements have "AND" relation.

Particularly, for those elements which contains XML structured XPATH string (MOId and MOAttributes), the interpretation applies "AND" relation. For example, if XPath is used as the value in the implementation for MOId, although "<MOId>/mo1[userid>100]</MOId>" and "<MOId>/mo2[registerDate>2004/01/23]</MOId>" has the same element name, they are regarded as different elements.

The relation between different NotificationFilters is "OR".



The following example sets a filter on operation of John's Create operation in which, the userid should be larger than 1000, the registerDate should be later than 2004/05/06 and the corpId should be equal to 200 (MOType: "mo1", MOAttributes: "corpId=200", MOId: "userid > 1000" and "registerDate > 2004/05/06") or Elle's operation in which the MOType is mo2 (MOType: "mo2"):

```
<filters>
  <filter>
    <cai3gUser>John</cai3gUser>
    <MOType>mo1@http://schemas.abc.com/service</MOType>
    <operation>ObjectCreation</operation>
    <MOId>/mo1[userid>1000]</MOId>
    <MOId>/mo1[registerDate>2004/05/06]</MOId>
    <MOAttributes>/moattr[corpId="200"]</MOAttributes>
  </filter>
  <filter>
    <cai3gUser>Elle</cai3gUser>
    <MOType>mo2@http://schemas.abc.com/service</MOType>
  </filter>
</filters>
```

Example 13 CAI3G Notification Filter

4.3.2

Notify Operation

After a successful registration, when an event fulfills the trigger condition, a notification will be sent from CAI3GManager that subscribe that event. The pseudo-method for notify looks like:

NOTIFY(IN notificationHeader, [IN correlatedNotifications], [IN additionalText], [IN sourceIndicator], IN notificationData)

The **notificationHeader** is a complex argument including a structure of following member attributes:

cai3gUser: This parameter specifies the creator of the session in which the operation is executed.

MOType: This parameter specifies the type of the MO in which the event occurred.

MOId: This parameter specifies the instance of the MO in which the event occurred.

notificationId: This parameter provides an identifier for the notification, which may be carried in the **correlatedNotifications** parameter of future notifications. This parameter SHOULD be chosen to be unique across all notifications of a particular MO throughout the time that correlation is significant. This is an optional parameter.

eventTime: It indicates the event occurrence time. This is a mandatory parameter.

notificationActor: It carries the Universal Resource Identifier (URI) of CAI3GAgent that detects the network event and generates the notification. This is an optional parameter.

operation: It carries identification of the type of event reported by the notification. Allowed event types in CAI3G 1.2 are Create, Delete and Set.



subscriptionId: The **subscriptionId** is sent back to CAI3GManager to show that the Notify operation is from the CAI3GAgent that the CAI3GManager has subscribed to.

The **correlatedNotifications** is an optional multi-value parameter that specifies the related notification items. The value of each correlated notification is a notification identity.

The **additionalText** is an optional parameter with additional information of the notification.

The **sourceIndicator** is an optional parameter indicates the source of the notification. This argument MAY be used when there are one or more Mediation Functions in the notification path between an end Network Element Function and CAI3GManager. It MUST be a value of notificationActor.

The **notificationData** is an argument containing the information of the content of the event, i.e. the whole CAI3G request.

Here is the XML schema definition of the Notify request.

```
<!-- Notification Request -->
<xs:element name="Notify">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="notificationHeader" type="NotificationHeaderType"/>
      <xs:element name="correlatedNotifications"
        type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="additionalText" type="xs:string" minOccurs="0"/>
      <xs:element name="sourceIndicator" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="notificationData">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="NotifyResponse">
  <xs:complexType/>
</xs:element>

<!-- Notification Header Type -->
<xs:complexType name="NotificationHeaderType">
  <xs:sequence>
    <xs:element name="cai3gUser" type="xs:string"/>
    <xs:element name="MOType" type="MoType"/>
    <xs:element name="MOId" type="AnyMOIdType"/>
    <xs:element name="notificationId" type="xs:string" minOccurs="0"/>
    <xs:element name="eventTime" type="xs:dateTime"/>
    <xs:element name="notificationActor" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="operation" type="NotificationOperationType"/>
    <xs:element name="subscriptionId" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Example 14 Notify request

When a Network Element Function sends out a notification, the **notificationActor** in the notification header argument and the **sourceIndicator** in the request have the same value. If there is a Mediation Function in the notification path, after it receives the request, it SHOULD change the **notificationActor** as its own



distinguished name without touching sourceIndicator and forward the notification to the next application in the path.

The response of a “Notify” method is the generic SOAP response that is described in details in Section 4.4 on page 35.

4.3.3 Unsubscribe Operation

CAI3GManager can send an Unsubscribe request to abort a registered notification subscription. The structure of the pseudo-method looks like:

Unsubscribe (IN managerRef, [IN subscriptionId])

The **managerRef** is a mandatory input argument from CAI3GManager. It specifies the reference of CAI3GManager. CAI3GManager SHOULD supply its valid “managerRef”. This is the necessary requirement for the operation to be successful. The data type of this attribute is a URI.

The **subscriptionId** is an optional input argument from CAI3GManager. It is the “subscriptionId” carried as the OUT parameter in the Subscribe operation. CAI3GManager will supply a specific subscriptionId if CAI3GManager wants to unsubscribe that particular subscription. If this argument is absent or has an empty value, CAI3GAgent will stop all subscriptions established between CAI3GAgent and this “managerRef” specified CAI3GManager.

Here is the XML schema definition of Unsubscribe request.

```
<xs:element name="Unsubscribe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="managerRef" type="xs:anyURI"/>
      <xs:element name="subscriptionId" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="UnsubscribeResponse">
  <xs:complexType/>
</xs:element>
```

Example 15 Unsubscribe request

4.4 Response and Fault Handling

CAI3G 1.2 is an interface that each CAI3G request will have a response message. CAI3GManager SHOULD not send a request till the response of the previous request returns. The SOAP Fault element will be used to carry error and/or status information within a CAI3G response.

If a CAI3G method does not have any output parameter, the receiver of an empty SOAP body in CAI3G response assumes the request was handled successfully. For a CAI3G method that has output parameters, CAI3GManager receives them inside a valid SOAP message after an OK response of HTTP.

CAI3G 1.2 uses the fault handling mechanism of SOAP protocol for its fault handling. The SOAP fault handling defines following four sub-elements: faultcode, faultstring, faultactor and detail (see Simple Object Access Protocol (SOAP) 1.1). Other fault sub-elements MAY be present, if they are namespace qualified.

The detail element is intended for carrying application specific error information related to the Body element. CAI3G error information is present in the detail element in the SOAP Fault element if the content of the Body element could not be successfully processed. The CAI3G request header faults are defined separately in SOAP Header using the SOAP headerfault element.

4.4.1 Headerfault

The **headerfault** elements which appear inside soap:header and have the same syntax as soap:header allows specification of the header type(s) that are used to transmit error information pertaining to the header defined by the soap:header. The SOAP specification states that errors pertaining to headers must be returned in headers, and this mechanism allows specification of the format of such headers.

There are three header parameters defined in CAI3G interface. They are sessionId, transactionId and sequenceId, also see Section 8 on page 61.

The sessionId has the following fault codes:

- Invalid SessionId: A sessionId does not equal to any sessionId currently maintained in the system.
- Session Timeout: A sessionId is not valid any longer because of the session timeout.
- SessionId Syntax Error: The syntax of sessionId cannot be recognized in CAI3GAgent.

The transactionId has the following fault code:

- Invalid TransactionId: reserved for future.

The sequenceId has the following fault code:

- Invalid SequenceId: A sequenceId is lower than or equals to the latest session request sequence identifier in CAI3GAgent.

Similar to CAI3G Fault element, each headerfault element also contains two other sub-elements. Element faultactor is a URI string specifies where the error is generated. Element description provides the developer the opportunity to add more detailed information for the error. The XML schema of SessionIdFault type is shown below. The TransactionIdFault and SequenceIdFault are similar to this one.



```

<xs:complexType name="SessionIdFault" final="restriction">
  <xs:complexContent>
    <xs:extension base="HeaderFaultType">
      <xs:sequence>
        <xs:element name="faultcode">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Invalid SessionId"/>
              <xs:enumeration value="Session Timeout"/>
              <xs:enumeration value="SessionId Syntax Error"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SequenceIdFault" final="restriction">
  <xs:complexContent>
    <xs:extension base="HeaderFaultType">
      <xs:sequence>
        <xs:element name="faultcode">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Invalid SequenceId"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TransactionIdFault"
  final="restriction">
  <xs:complexContent>
    <xs:extension base="HeaderFaultType">
      <xs:sequence>
        <xs:element name="faultcode">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Invalid TransactionId"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="HeaderFaultType">
  <xs:sequence>
    <xs:element name="faultactor" type="xs:string"/>
    <xs:element name="description" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

Example 16

There are two values reserved for parameter “faultactor” in this document. “MF” indicates the node reporting the fault is a Mediation Function, while “NEF” indicates the node reporting the fault is a Network Element Function. The CAI3G interface designer MAY add own actor definition except those two reserved values.

In order to solve the problem that some 3PPs do not deal with any information in SOAP header, when the header fault occurs, the body part should also be set to fault. The CAI3G fault code between 1000~1999 are defined and reserved for the corresponding fault in SOAP header.

4.4.2 CAI3G Fault Details

A CAI3G Fault component is a sub-element of the SOAP Fault element. It contains the error information in SOAP body of a CAI3G request.

The XML representation for a CAI3G Fault component is an element information item with the following infoset properties:

- A [local name] of `cai3gfault`
- A [namespace name] of “`http://schemas.ericsson.com/cai3g1.2/`”
- Three or Four element information item amongst its [children] as follows:
 - A `faultcode` element information item intended for use by CAI3GManager to provide an algorithmic mechanism for identifying the fault.
 - A `faultreason` element information item intended to provide a human readable explanation of the fault and is not intended for algorithmic processing.
 - A [local name] of `faultreason`
 - One or more fault text element information items amongst its [children]
 - A `faultrole` element information item identifying the role being played by the node which generated the fault
 - An optional `details` element information item intended for carrying CAI3GAgent specific error information related to the Body element. It is a supplement to CAI3G fault where each CAI3GAgent can put their specific error codes and information. The absence of the detail element in the CAI3G Fault element indicates that the CAI3G fault is good enough to understand the fault.

CAI3GAgent's specific error information SHOULD only be put in CAI3G Fault Details element.

The XML Schema fragment of the CAI3G Fault element looks like:



```
<xs:element name="Cai3gFault">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="faultcode" type="xs:integer"/>
      <xs:element name="faultreason">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="reasonText" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="faultrole" type="xs:string"/>
      <xs:element name="details" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 17 CAI3G Fault element

There are two values reserved for parameter “faultrole” in this document. “MF” indicates the node reporting the fault is a Mediation Function, while “NEF” indicates the node reporting the fault is a Network Element Function. The CAI3G interface designer MAY add own role definition except those two reserved values.

An example of a CAI3G fault response is given as the following:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <SequenceId xmlns="http://schemas.ericsson.com/cai3g1.2/">2147483647</SequenceId>
    <SessionId xmlns="http://schemas.ericsson.com/cai3g1.2/">1234567</SessionId>
    <TransactionId xmlns="http://schemas.ericsson.com/cai3g1.2/">2147483647</TransactionId>
  </S:Header>
  <S:Body>
    <S:Fault>
      <faultcode>S:Client</faultcode>
      <faultstring>This is a client error.</faultstring>
      <faultactor></faultactor>
      <detail>
        <Cai3gFault xmlns="http://schemas.ericsson.com/cai3g1.2/">
          <faultcode>3002</faultcode>
          <faultreason>
            <reasonText>Target Object Instance Does Not Exist.</reasonText>
          </faultreason>
          <faultrole>MF</faultrole>
          <details>...</details>
        </Cai3gFault>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

Example 18 A CAI3G Fault Response

The following table gives out all current allowed error codes in CAI3G 1.2.



Table 2 CAI3G Error Code List

CAI3G ERROR VALUE	SOAP ERROR CODE	DESCRIPTION	COMMENTS
Reserved (1-999)			
Header Errors (1001-1999)			
1001	Client	Invalid SessionId	
1002	Client	Session Timeout	The session indicated by the SessionId is timeout.
1003	Client	SessionId Syntax Error	The SessionId is not a numeric string.
1101	Client	Invalid SequenceId	
1201	Client	Invalid TransactionId	
Request Errors (2001-2999)			
2001	Client	Invalid Managed Object Type	Unknown MO to CAI3GAgent
2002	Client	Invalid Managed Object Id	The format of MOId is not correct.
2003	Client	Unsupported Data Type	The data type is not supported.
2999	Client	Other Request Error	Request error that cannot be put into above categories.
Client Errors (3001-3999)			
3001	Client	Operation Not Allowed	Operation is not allowed due to the prerequisites are not fulfilled.
3002	Client	Object Does Not Exist	Try to access an inexistent MO instance.
3003	Client	Object Already Exists	Try to re-create an existing MO instance.
3004	Client	Invalid User ID	The userId in LOGIN operation does not exist.
3005	Client	Invalid Password	The pwd in LOGIN operation is invalid.
3006	Client	Invalid SessionId	The sessionId in LOGOUT operation is not correct.
3007	Client	Invalid Filter	The format of filter in Subscribe operation is not correct.
3008	Client	Invalid Subscription ID	The subscriptionId in Unsubscribe operation does not exist.
3009	Client	Invalid Managed Object ID	The MOId elements are not correct.
3010	Client	Invalid MO Attribute	The value of input attribute violates XML schema definition.
3011	Client	Insufficient MO Attributes	The value of input attribute violates XML schema definition.
3012	Client	Insufficient Parameter	There are some mandatory parameters missed in the request.
3013	Client	Invalid Parameter	There are invalid parameters in the request.
3014	Client	Login Failure	Login Failure
3015	Client	SessionId Not Consistent	The sessionId in header and body in a LOGOUT request are not the same.



CAI3G ERROR VALUE	SOAP ERROR CODE	DESCRIPTION	COMMENTS
3999	Client	Other Client Error	Client error that cannot be put into above categories.
Server Errors (4001-4999)			
4001	Server	Operation Not Supported.	CAI3GAgent cannot support the operation in current request.
4002	Server	Object Not Supported.	CAI3GAgent cannot support the MOType in current request.
4003	Server	Filter Not Supported	The content of the filter in Subscribe operation is not supported in CAI3GAgent.
4004	Server	Function Busy	A temporary error caused by short of computing resource at CAI3GAgent.
4005	Server	Internal Fatal Error	A fatal error in CAI3GAgent when processing the request.
4006	Server	External Error	An error caused outside the CAI3GAgent causes the abort of the processing request.
4007	Server	CAI3G Version Not Supported	The version of CAI3G request from CAI3GManager is not supported by CAI3GAgent.
4008	Server	MO Version Not Supported	The MO version is not supported in CAI3GAgent.
4009	Server	Reached the limitation	Over limitation of maximum numbers of soap connections.
4010	Server	Reached the limitation	Over limitation of maximum numbers of CAI3G sessions.
4999	Server	Other Server Error	Server error that cannot be put into any above categories.





5 CAI3G Parameter

5.1 MOType

MOType is a plain text string based on the type “xs:string”. An MO type contains two parts. One is the namespace of the MO, and the other is the MO name string always starting with an alphabetical character in either upper or lower case followed by zero or more alphabetical characters or digits or underscores. Those two parts are connected with symbol “@”. The whole MOType string looks like “MO_Name@MO_Namespace”.

The name string of an MO type MUST follow the regular expression: [A-Za-z][_A-Za-z0-9]*.

The MO name plus MO namespace MUST be global unique.

5.2 MOId

MOId is an XML fragment containing the MOId parameter-value pairs that are used to identify an MO instance in the interface data model. CAI3G 1.2 standard supports compound MO identifiers or multiple MO identifiers. An example of MOId looks like the following:

```
<MOId>
  <MSISDN>46455395000</MSISDN>
  <IMSI>46234563545000</IMSI>
</MOId>
```

Example 19 Compound MO Identifier Example

Multi-identifier refers to cases where one MO can be identified in more than one way:

```
<MOId>
  <name>Ericsson Shanghai</name>
</MOId>
and:
<MOId>
  <id>RDC</id>
</MOId>
and maybe also:
<MOId>
  <name>Ericsson Shanghai</name>
  <id>RDC</id>
</MOId>
```

Example 20 Multiple MO Identifiers Examples

Note: In one specific request only one of the above can be used.

Generic CAI3G schema is only a template for reference, so it does not describe specific definition of the MO identifier.



Each implementation of CAI3G interface SHOULD define own logic relationship of MOId. It is also the CAI3GAgent's responsibility to interpret this parameter correctly by means of either the hard-coded logic or the dynamic parsing of the schema.

5.3 MO Attributes

As it is mentioned in Section 3.4 on page 17, CAI3G supports three categories of attributes. CAI3G requires each implementation instance to use XML schema to define the structure of the attributes in each methods. This section provides the XML representation, XML schemas and examples of those attributes, which the CAI3G interface designer must follow.

MO attributes are identified by one or more key attributes which can also be represented by MOId, see Section 5.2 on page 43. The key attributes can be XML attributes or XML elements in MO attributes at the top level of different methods. The following table describes the MO key attributes definition rules for each CAI3G method.

Table 3 MO Key Attributes

CAI3G Method	Key Attributes ⁽¹⁾
Create	Key attributes should be defined as XML attributes and XML elements at the top level if needed.
Set	Key attributes should be defined as XML attributes. An XML element can also be defined for MO identifier changeover if needed.
Get	Key attributes should be defined as XML elements at the top level. XML elements can also be defined for filter purposes if needed.
Delete	Key attributes should be defined as XML attributes. An XML element can be also defined at the top level for filter purposes if needed.

(1) If key attributes are defined in MO attributes at the top level, the MOId should be same as those attributes.

Note: When defining the responses for above methods, the key attributes should be defined as XML attributes.

The following table gives out the XML schema binding for different parameters:

Table 4 MO Attributes XML Schema Mapping Table

MO Attributes Type	Type	minOccurs	maxOccurs
Single Value Parameter	Any simple type defined in standard XML schema and customer schema	0 or 1	1
Multiple Value Parameter	Any simple type defined in standard XML schema and customer schema	<=maxOccurs	>1
Structured Parameter	Any complex type without “key” in customer schema	0 or 1	1
Sub-object Parameter	Any complex type having “key” in customer schema	<=maxOccurs	>=1



5.3.1 Simple Parameter

5.3.1.1 Single Value Parameter

The XML representation for a single value parameter definition component is an element information item with the following Infoset properties:

- A [local name] of the single value parameter name
- A [namespace name] of the single value parameter namespace name
- An annotation element information model amongst its [children], in order as
 - A documentation element information model
 - An optional appinfo element information model (see Section 5.5 on page 53)
- The value of the parameter

The XML schema structure of a single value parameter looks like:

```
<xs:element name="paramName" type="paramDataType"
  default="paramDefaultValue" minOccurs="0" maxOccurs="1" nillable="[true|false]">
  <xs:annotation>
    <xs:documentation>
      A single value parameter example.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

Example 21 Structure of Single Value Parameter

In the above schema string, XML attribute “maxOccurs” MUST always be set to “1” for a single value parameter.

XML attribute “default” is an optional attribute only needed when the single value parameter has a default value.

When XML attribute “minOccurs” equals to 0, this single value parameter is an optional one.

An example of a single value parameter in XML instance document looks like:

```
<Zip>200126</Zip>
```

Example 22 A CAI3G Single Value Parameter

CREATE Scenario

In CREATE operation, the following XML fragment tells CAI3GAgent to set the single value parameter “Zip” with value “200126”. This MUST be supported in a CAI3G interface realization.

```
<Zip>200126</Zip>
```



The following XML fragment tells CAI3GAgent to set the single value parameter “Zip” with default value if it exists. If there is no default value for “Zip”, CAI3GAgent SHOULD interpret the XML fragment as the CAI3GManager wants to set the value of “Zip” to empty.

```
<Zip/>
```

SET Scenario

In SET operation, the following XML fragment tells CAI3GAgent to set the single value parameter “Zip” with value “200126”. This MUST be supported in a CAI3G interface realization.

```
<Zip>200126</Zip>
```

The following XML fragment tells CAI3GAgent to set the single value parameter “Zip” with default value if it exists. If there is no default value for “Zip”, CAI3GAgent SHOULD interpret the XML fragment as the CAI3GManager wants to set the value of “Zip” to empty. This MUST be supported in a CAI3G interface realization.

```
<Zip/>
```

The following request tells CAI3GAgent to remove this single value parameter from the MO in the data model. This MAY be supported in a CAI3G interface realization.

```
<Zip xsi:nil="true"/>
```

5.3.1.2

Multiple Value Parameter

Multiple value parameter is the second type of simple parameter. It is a parameter contains more than one value.

The XML presentation of a multiple value parameter is very similar to that of a single value parameter. The only difference is that for a multiple value parameter the XML attribute “maxOccurs” in schema is greater than “1”. So that a multiple value parameter’s name-value pair MAY occurs more than once in an instance document.

The XML schema structure of a multiple value parameter looks like:

```
<xs:element name="paramName" type="paramDataType" default=
"paramDefaultValue" minOccurs="minimumOccursTimes" maxOccurs=
"maximumOccursTimeGreaterThanOne" nillable="[true|false]">
  <xs:annotation base="documentation">
    A multiple value parameter example.
  </xs:annotation>
</xs:element>
```

Example 23 Structure of a Multiple Value Parameter

In the above schema string, XML attribute “maxOccurs” MUST always be greater than “1” for a multiple value parameter.



XML attribute “default” is an optional attribute only needed when a multiple value parameter has a default value.

When XML attribute “minOccurs” equals to 0, this multiple value parameter is an optional one.

An example of a multiple value parameter in XML instance document looks like:

```
<children>Bob</children>
<children>Claus</children>
<children>John</children>
```

Example 24 A CAI3G Multiple Value Parameter

CREATE Scenario

In CREATE operation, the following request tells CAI3GAgent to set the multiple value parameter “children” with value “Bob”, “Claus” and “John”. This MUST be supported in a CAI3G interface realization.

```
<children>Bob</children>
<children>Claus</children>
<children>John</children>
```

The following request tells CAI3GAgent to set a value to the multiple value parameter “children” with default value if it exists. If there is no default value for “children”, CAI3GAgent SHOULD set the value of “children” to empty. So in this case, the CAI3GAgent MIGHT return error since normally the name of “children” does not have a default value. This MUST be supported in a CAI3G interface realization.

```
<children/>
```

SET Scenario

In SET operation, the following request tells CAI3GAgent to replace the multiple value parameter “children” with value “Bob”, “Claus”, “John” and “Lena”. This MUST be supported in a CAI3G interface realization.

```
<children>Bob</children>
<children>Claus</children>
<children>John</children>
<children>Lena</children>
```

The following request tells CAI3GAgent to replace all values of the multiple value parameter “children” with the default value if it exists. If there is no default value for “children”, which is true in this example, CAI3GAgent SHOULD set the value of “children” to empty. This MUST be supported in a CAI3G interface realization.

```
<children/>
```



The following request tells CAI3GAgent to remove this multiple value parameter from the MO in the data model. This MAY be supported in a CAI3G interface realization.

```
<children xsi:nil="true"/>
```

5.3.2 Structured Parameter

A structured parameter is a single value parameter that contains other type of parameter as its attribute. It is a recursive structure that can further contain another structured parameter as its attribute.

The XML representation for a structured parameter definition component is an element information item with the following Infoset properties:

- A [local name] of the structured parameter name
- A [namespace name] of the structured parameter namespace name
- One or more element information items amongst its [children] as follows:
 - An annotation element information model amongst its [children], in order as
 - A documentation element information model
 - An optional appinfo element information model (see Section 5.5 on page 53)
 - One or more simple parameter information items (see Section 5.3.1 on page 45)
 - Zero or more structured parameter information items
 - Zero or more sub-object parameter information items (see Section 5.3.3 on page 50)

The XML schema structure of a structured parameter looks like:

```
<xs:element name="paramName" type="paramDataType"
  minOccurs="0" maxOccurs="1" nillable="true|false">
  <xs:annotation base="documentation">
    A structured parameter example.
  </xs:annotation>
</xs:element>

<xs:complexType name="paramDataType">
  <xs:sequence>
    <xs:element name="someSParam" type="aSingleValueParamType"/>
    <xs:element name="someMParam" type="aMultiValueParamType"/>
    <xs:element name="someStructParam" type="aStructParamType"/>
    <xs:element name="someSubMoParam" type="aSubMoParamType"/>
  </xs:sequence>
</xs:complexType>
```

Example 25 Structure of a Structured Parameter



An example of a structured parameter in XML instance document looks like:

```
<user>
  <name>Alice</name>
  <gender>female</gender>
  <status>married</status>
  <children>Bob</children>
  <children>John</children>
  <homeAddr>
    <street>No.128, Liverpool Street</street>
    <city>London</city>
    <postcode>N1 1LX</postcode>
    <country>UK</country>
  </homeAddr>
</user>
```

Example 26 A CAI3G Structured Parameter

The behavior of the structured parameter in CREATE and SET methods depends on the behavior of its sub-parameters in those methods. In Example 26, the sub-parameter “name” SHOULD follow the rules described in Section 5.3.1.1 on page 45, and the sub-parameter “children” SHOULD follow the rules described in Section 5.3.1.2 on page 46 and so on.

CREATE Scenario

In CREATE operation, the following request tells CAI3GAgent to set the value of a structured parameter. This MUST be supported in a CAI3G interface realization.

```
<user>
  <name>Alice</name>
  <gender>female</gender>
  <status>married</status>
  <children>Bob</children>
  <children>John</children>
  <homeAddr>
    <street>No.128, Liverpool Street</street>
    <city>London</city>
    <postcode>N1 1LX</postcode>
    <country>UK</country>
  </homeAddr>
</user>
```

Example 27 CREATE operation

SET Scenario

In SET operation, the following request tells CAI3GAgent to update “user” parameter with the value specified in the XML fragment structure. In this example, the “status” of “user” is changed and the values of “children” are replaced by all new values provided in this structure. The “street” value in “homeAddr” is also updated. This MUST be supported in a CAI3G interface realization.

```
<user>
  <status>divorce</status>
  <children>Bob</children>
  <children>John</children>
  <children>Li</children>
  <homeAddr>
    <street>No.101, Zunyi Road</street>
  </homeAddr>
</user>
```

Example 28 SET operation

The following request tells CAI3GAgent to remove the structured parameter from the MO in the data model. This MAY be supported in a CAI3G interface realization.

```
<user xsi:nil="true"/>
```

5.3.3 Sub-object Parameter

In Section 3.3 on page 16, two ways of interface data model design are described, one of which is to use sub-object parameter.

In some sense, a structured parameter can be considered as a special sub-object parameter that can only have one sub-object instance. A sub-object parameter can represent a sub-object having more than one instance. Similar to that an MO requires MO identifiers to specify a single instance, the sub-object needs keys to distinguish an instance from others.

A key attribute is a unique attribute that occurs as both an XML element and an XML attribute in an instance document, which distinguishes the sub-object instance one from another.

The XML representation for a sub-object parameter definition component is an element information item with the following Infoset properties:

- A [local name] of the structured parameter name
- A [namespace name] of the structured parameter namespace name
- One or more key attribute information items amongst its [attributes]
- One or more element information items amongst its [children] as follows:
 - An annotation element information model amongst its [children], in order as
 - A documentation element information model
 - An optional appinfo element information model (see Section 5.5 on page 53)
 - Zero or more key element information items
 - One or more simple parameter information items (see Section 5.3.1 on page 45)
 - Zero or more structured parameter information items (see Section 5.3.2 on page 48)
 - Zero or more sub-object parameter information items

The XML schema structure of a sub-object parameter looks like:



```
<xs:element name="paramName" type="paramDataType" minOccurs="minimumOccursTimes"
maxOccurs="maximumOccursTimeGreaterThanOne" nillable="[true|false]">
  <xs:annotation>
    <xs:documentation>
      A sub-object parameter example.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="paramDataType">
        <xs:attribute name="key1" type="xs:string" use="required"/>
        <xs:attribute name="key2" type="xs:string" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <key name="paramName" id="CAI3GKey1">
    <selector xpath="paramName" />
    <field xpath="@key1" />
    <field xpath="@key2" />
  </key>
</xs:element>

<xs:complexType name="paramDataType">
  <xs:sequence>
    <xs:element name="key1" type="aSingleValueParamType" />
    <xs:element name="key2" type="aSingleValueParamType" />
    <xs:element name="someSVParam" type="aSingleValueParamType" />
    <xs:element name="someMVParam" type="aMultiValueParamType" />
    <xs:element name="someStructParam" type="aStructParamType" />
    <xs:element name="someSubMoParam" type="aSubMoParamType" />
  </xs:sequence>
</complexType>
```

Example 29 Structure of a Sub-object Parameter

Any key attribute MUST be a single value parameter without default value.

An example of a sub-object parameter in XML instance document looks like:

```
<user socialId="123-456-789">
  <socialId>123-456-789</socialId>
  <name>Alice</name>
  <gender>female</gender>
  <status>married</status>
  <children>Bob</children>
  <children>John</children>
  <homeAddr>
    <street>No.128, Liverpool Street</street>
    <city>London</city>
    <postcode>N1 1LX</postcode>
    <country>UK</country>
  </homeAddr>
</user>
```

Example 30 A CAI3G Sub-object Parameter

CREATE Scenario

In CREATE operation, the following XML fragment tells CAI3GAgent to create an instance for sub-object user with key “socialId” as “123-456-789” and other attribute values. This MUST be supported in a CAI3G interface realization.

```
<user socialId="123-456-789">
  <socialId>123-456-789</socialId>
  <name>Alice</name>
  <gender>female</gender>
  <status>married</status>
  <children>Bob</children>
  <children>John</children>
  <homeAddr>
    <street>No.128, Liverpool Street</street>
    <city>London</city>
    <postcode>N1 1LX</postcode>
    <country>UK</country>
  </homeAddr>
</user>
```

Example 31 CREATE operation

SET Scenario

In SET operation, the following XML fragment tells CAI3GAgent to set an instance for sub-object user with key “socialId” as “123-456-789” and other attribute values. This MUST be supported in a CAI3G interface realization.

```
<user socialId="123-456-789">
  <name>Alice</name>
  <gender>female</gender>
  <status>married</status>
  <children>Bob</children>
  <children>John</children>
  <homeAddr>
    <street>No.128, Liverpool Street</street>
    <city>London</city>
    <postcode>N1 1LX</postcode>
    <country>UK</country>
  </homeAddr>
</user>
```

Example 32 SET operation

The following XML fragment tells CAI3GAgent to update the data of the sub-object “user” instance whose key “socialId” equals to “123-456-789” with the attribute specified in the request. In this case, a new MO attribute “husband” is added, the “children” list is replaced by the value of all new “children” and the “street” MO sub-attribute is also changed in “homeAddr”. This MUST be supported in a CAI3G interface realization.

```
<user socialId="123-456-789">
  <husband>Del</husband>
  <children>Bob</children>
  <children>Claus</children>
  <children>John</children>
  <homeAddr>
    <street>No.101, Liverpool Street</street>
  </homeAddr>
</user>
```

The following XML fragment tells CAI3GAgent to change the key attribute of a sub-object. In this case the key attribute “socialId” of Alice is changed from “123-456-789” to “123-456-788”, and the other attributes of the sub-object are kept same as before. This MAY be supported in a CAI3G interface realization.



```
<user socialId="123-456-789">
  <socialId>123-456-788</socialId>
</user>
```

The following XML fragment tells CAI3GAgent to remove a “user” sub-object instance whose key “socialId” is “123-456-788” from the data model. This **MUST** be supported in a CAI3G interface realization.

```
<user socialId="123-456-788" xsi:nil="true"/>
```

The following XML fragment tells CAI3GAgent to remove all “user” sub-object instances from the data model. This **MAY** be supported in a CAI3G interface realization.

```
<user xsi:nil="true"/>
```

5.4 Extension

This optional parameter is added to operation CREATE, DELETE, GET, SET and SEARCH. It is defined with the data type “AnySequenceType”. This parameter can contain data that is not covered by MOType, MOId and MOAttributes and is semantically agreed between the CAI3G Manager and CAI3G Agent. The exact structure of extension is not defined in the schema. The CAI3G Agent or Manager can interpret the data based on the agreements between them.

The example below shows the usage of extension to identify the delete type is “forcedelete”.

```
<extension>
  <DeleteType>forcedelete</DeleteType>
</extension>
```

Example 33 Extension Example

5.5 Leaf Attributes

The leaf attribute is a special type of attribute the value of which itself is an XML fragment that should be considered as a single entry so that a CAI3GAgent **MUST** NOT update part of it.

CAI3GAgent **MAY** or **MAY NOT** verify the content of the leaf attribute.

Any of the following attribute schema definitions **SHOULD** be considered as a leaf attribute.



```
<xs:element name="leafattr1">
  <xs:complexType>
    <xs:any namespace="valid_xml_namespace_or_macro"
      processContents="lax_or_ignore_or_strict"/>
  </xs:complexType>
</xs:element>

<xs:element name="leafattr2" type="xs:anyType"/>

<xs:element name="leafattr3">
  <xs:annotation>
    <xs:documentation>...</xs:documentation>
    <xs:appinfo>
      <leaf>true</leaf>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Example 34

Which solution SHOULD be used depends on the MO schema designers own preference. CAI3GAgent has to parse the content of leaf elements presented in the 2nd and 3rd ways according to XML schema standard. The content of the element presented in the 1st way MAY or MAY NOT be parsed depending on the value of processContents.

CAI3GAgent MUST only parse and verify the content of leaf attributes but ignore the rule interpretation in it.

5.6 MO Schema File

The description of an MO in CAI3G instance is defined in an XML schema file. The XML schema file consists of MO identifier and MO attribute definition.

Note: All MO identifiers MUST be defined as the key attributes of the userdefined MO attributes elements.

This section gives a simple example on how to write the schema file (see Example 35 for the generic schema file) during MO development and how to override the generic CAI3G definitions. Example 36 defines the User MO in provisioning interface for myservice at mycompany. The MOId of this MO is socialId. The actual MO attribute element is called CreateUserAttributes that corresponds to CreateMODefinition in the Create element in the generic CAI3G schema.



```

<xs:schema targetNamespace="http://schemas.mycompany.com/myservice/"
xmlns="http://schemas.mycompany.com/myservice/"
xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="CreateUser">
    <xs:complexType>
      <xs:complexContent>
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="gender">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Male"/>
                <xs:enumeration value="Female"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="Age">
            <xs:simpleType>
              <xs:restriction base="xs:int">
                <xs:minInclusive value="1"/>
                <xs:maxInclusive value="200"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="status">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="widow"/>
                <xs:enumeration value="divorced"/>
                <xs:enumeration value="married"/>
                <xs:enumeration value="single"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="children" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="homeAddr">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Street">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:maxLength value="255"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="postcode" type="xs:string"/>
                <xs:element name="country">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string"/>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="socialId" type="xs:unsignedInt"/>
        </xs:sequence>
        <xs:attribute name="socialId" type="xs:unsignedInt" use="required"/>
      </xs:complexContent>
    </xs:complexType>
    <xs:key name="CreateUserMessage" id="CAI3GKey1">
      <xs:selector xpath="."/>
      <xs:field xpath="@socialId"/>
    </xs:key>
  </xs:element>
  <xs:element name="socialId" type="xs:unsignedInt"/>
</xs:schema>

```

Example 35 MO Definition





6 CAI3G Integration

After defining the specific MO schema file (see Section 5.6 on page 54), the CAI3G interface designer must follow the procedures below to publish the northbound CAI3G interface:

1. Modify the generic CAI3G schema template based on the specific MO CAI3G schema.
2. Build and publish the specific WSDL file.

The following is an example that shows how to override the generic CAI3G schema template (tags in bold style in the following example denotes the overridden part).

```
<xs:schema targetNamespace="http://schemas.ericsson.com/cai3g1.2/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.ericsson.com/cai3g1.2/"
xmlns:up="http://schemas.mycompany.com/myservice/"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://schemas.mycompany.com/myservice/"
    schemaLocation="myservice.xsd"/>
  <xs:element name="Create">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="M0Type" type="xs:string"/>
        <xs:element name="M0Id" type="tns:AnyM0IdType" minOccurs="0"/>
        <xs:element name="M0Attributes" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="ns1:CreateUserAttributes"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ....
  <xs:complexType name="AnyM0IdType">
    <xs:sequence>
      <xs:element ref="up:socialId"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Example 36 Overridden Generic CAI3G Schema Template

For information about how to build a specific WSDL file, see Section 11 on page 67.





7 Version Control

7.1 CAI3G

CAI3G version comprises of major version and minor version delimited by dot. This document describes CAI3G 1.2 that has a major version “1” and minor version “2”. The version of the interface is defined in the target namespace in CAI3G WSDL file. The following XML string gives the current version definition in CAI3G 1.2:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xopenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
targetNamespace="http://schemas.ericsson.com/cai3g1.2/">
```

Example 37 CAI3G 1.2 Namespace

For instance cai3g namespace with version information is included in the CAI3G SOAP requests as the following example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <m:SessionId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">1234567</m:SessionId >
    <m:TransactionId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">4294967295
  </m:TransactionId>
    <m:SequenceId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">4294967295
  </m:SequenceId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:Create xmlns:m="http://schemas.ericsson.com/cai3g1.2/">
      <m:MOType>User</m:MOType>
      </m:MOType>
      <m:MOId>
        <abc:socialId xmlns:abc="http://www.mycompany.com/myservice/">310102197204084417
      </abc:socialId>
      </m:MOId>
      <m:MOAttributes>
        <abc:CreateUser socialId="310102197204084417"
xmlns:abc="http://www.mycompany.com/myservice/">
          <abc:name>Alice</abc:name>
          <abc:gender>Female</abc:gender>
          <abc:age>46</abc:age>
        </abc:CreateUser>
      </m:MOAttributes>
    </m:Create>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 38 CAI3G Request Header and Body

CAI3G interface is a version data aware interface. One important step in login procedure is the check of version support. A session MUST be set up only if CAI3GAgent can support the version mentioned in the request from CAI3GManager. A special error will be returned in the case when the version is not matched between CAI3GManager and CAI3GAgent.



7.2 CAI3G MO

Not only the generic CAI3G interface but also each MO defined in the logical interface data model MAY have its version.

The MO version information is included in the MO namespace that MAY always be checked by CAI3GAgent. CAI3GAgent MAY reject a request due to a wrong version of MO.



8 Transaction Support

In the SOAP header, there are two optional elements designed for transaction support. One is the "transactionId", which is an `xs:string` data type, and the other is the "sequenceId", which is an `xs:unsignedLong` data type. The "transactionId" is generated according to its own rule when a CAI3GManager starts a transaction. The "sequenceId" is generated based on the "baseSequenceId" retrieved during the LOGIN operation. The second request sent by CAI3GManager after LOGIN MUST have a sequenceId that is larger than the baseSequenceId.

The CAI3GManager MUST guarantee the later the request the larger the sequenceId.

CAI3GAgent SHOULD keep the latest sequenceId for each session and reject any income request that has a smaller sequenceId than the latest sequenceId. A response MUST contain the same "transactionId" and "sequenceId" as the value in the corresponding request.

The sequenceId has an "xs:unsignedLong" type defined in XML Schema Part 2: Datatypes, which has an upper limit to $2^{64}-1$. The base of sequenceId in a session is randomly generated between 0 and $2^{63}-1$. At least 2^{63} requests can be handled before the sequenceId is exhausted in the worst case, which means the life of that session will be nearly 300 thousand years if the capacity of the session is 1 million requests per second, when the base sequenceId equals to $2^{63}-1$. So in this release of the specification, no action is considered when the sequenceId is exhausted. The CAI3GManager SHOULD LOGOUT and re-LOGIN if such a scenario happens.

Together with the aforementioned "sessionId", those two parameters provide a practical way to distinguish requests according to the session, transaction and request identifier.

Here is the XML schema for "transactionId" and "sequenceId":

```
<xs:element name="TransactionId" type="xs:string"/>
<xs:element name="SequenceId" type="xs:unsignedLong"/>
```

By providing those two attributes, CAI3G 1.2 provides the ability to support transaction handling. A CAI3G session can have many transactions each of which can have many requests. Below is an example of the SOAP header containing the sessionId, transactionId and sequenceId.

```
<SOAP-ENV:Header>
  <m:SessionId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">127</m:SessionId>
  <m:TransactionId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">42</m:TransactionId>
  <m:SequenceId xmlns:m="http://schemas.ericsson.com/cai3g1.2/">42295</m:SequenceId>
</SOAP-ENV:Header>
```

Example 39 CAI3G Message Header

How to use those parameters in CAI3GAgent internal logic is not covered in this document. It depends on the designers of CAI3GAgent in each implementation to explore the data carried in these header entries.





9 Security Considerations

The CAI3GAgent can secure the transportation layer through SOAP over HTTPS or the WS-Security profile. When the WS-Security profile is used, the SessionControl service can be skipped according to the implementation of the CAI3GAgent. Example 40 shows a SOAP request carrying the UsernameToken security profile to interact with the CAI3GAgent:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cai3="http://schemas.ericsson.com/cai3gl.2/">
  <soapenv:Header>
    <wsse:Security xmlns:wsse=
      "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu=
        "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken">
        <wsse:Username>sogadm</wsse:Username>
        <wsse:Password Type=
          "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
          sogadm</wsse:Password>
        </wsse:UsernameToken>
      </wsse:Security>
    <cai3:SequenceId>1</cai3:SequenceId>
    <cai3:TransactionId>123</cai3:TransactionId>
  </soapenv:Header>
  <soapenv:Body>
    <cai3:Create>
      <cai3:MOType>Subscription@http://schemas.ericsson.com/ema/UserProvisioning/GsmAuc/</cai3:MOType>
      <cai3:MOId><imsi>2341003000</imsi></cai3:MOId>
      <cai3:MOAttributes>
        <createSubscription imsi="2341003000"
          xmlns="http://schemas.ericsson.com/ema/UserProvisioning/GsmAuc/"
          xmlns:cai3g="http://schemas.ericsson.com/cai3gl.2/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <imsi>2341003000</imsi>
          <ki>12345678901234567890123456789012</ki>
          <a38ind>1</a38ind>
          <ad>1</ad>
          <adkey>300</adkey>
          <addinfo>10258ade</addinfo>
        </createSubscription>
      </cai3:MOAttributes>
    </cai3:Create>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 40 SOAP Request





10 Appendix A (CAI3G Rule Description)

This section gathers all rules and recommendations described in previous sections. The rules are the MUST requirements that the CAI3G interface implements have to obey. On the other hand, the recommendations are the best practical requirements suggested by the author of this document.

10.1 Rules

[Rule-Sync]: A CAI3GManager MUST NOT send a request on the same logical session before the response of the previous request comes back from CAI3GAgent.

[Rule-MO-Unique]: Each MO namespace and local MO Type (MO Name) pair MUST be global unique.

[Rule-MO-ID-1]: Each MO in CAI3G interface MUST contain at least one MO instance identifier.

[Rule-MO-ID-2]: The MOId is also the key attributes that MUST be defined in the top-level element.

[Rule-MO-Type]: The name string of an MO type MUST follow the regular expression: `[A-Za-z][_A-Za-z0-9]*`

[Rule-Session-Version]: A session MUST be set up only if CAI3GAgent can support the version mentioned in the request from CAI3GManager.

[Rule-Leaf-Attr]: CAI3GAgent MUST only parse and verify the content of leaf attributes but ignore the rule interpretation in it.

[Rule-MO-Schema-1]: The CAI3G interface designer MUST write XML schema file to describe the MO definition.

[Rule-MO-Schema-4]: All MO Identifiers MUST be defined as the key attributes of the user defined MO Attributes elements in MO definition Schema.

[Rule-MO-Name-Convention-1]: The name of the top element of MOAttributes in the Request MUST follow the rule, `[OPERATION] + [MOName]`, for example "createUser".

[Rule-MO-Name-Convention-2]: The name of the top element of MOAttributes in the Response MUST follow the rule, `[OPERATION] + "Response" + [MOName]`, for example "getResponseUser".

[Rule-Sequence-ID]: The CAI3GManager MUST guarantee the later the request the larger the sequenceId.

[Rule-Get-All-Instance]: Without MOId and MOAttributes that means the system will get all the instances according to the MOType.



[Rule-Response-Header]: A response MUST contain the same “transactionId” and “sequenceId” as the value in the corresponding request.

[Rule-Notify-SourceIndicator]: The sourceIndicator MUST be a value of notificationActor.

10.2 Recommendations

[Rec-MO-Design]: Designer SHOULD use one-to-one mapping in the case there are lots of sub-object instance and use sub-object parameter in the rest cases.

[Rec-Session-ID]: CAI3GAgent SHOULD guarantee the system uniqueness of the session identifier in the header of a SOAP message.

[Rec-Relogin]: In the case, a user sends another LOGIN after he has already built up a session and that session is still active. The CAI3GAgent SHOULD still answer the request with a new sessionId.

[Rec-Generate-MO-ID]: If the MOId is provided with an empty value or absent in a CREATE request, the CAI3GAgent SHOULD generate or allocate a new identifier that is returned in the CREATE response.

[Rec-Create]: If an MO has no mandatory attributes except MOId, the MOAttributes SHOULD be omitted in the CREATE request.

[Rec-Notify-NotificationId]: The notification identifier SHOULD be chosen to be unique across all notifications of a particular MO throughout the time that correlation is significant.

[Rec-Notify-MediationFunction]: If there is a Mediation Function in the notification path, after it receives the request, it SHOULD change the notificationActor as its own distinguished name without touching sourceIndicator and forward the notification to the next application in the path.



11 Appendix B (CAI3G Services WSDL Files)

This section describes the WSDL files for the interface and the schema file for type definition. The WSDL file in this section is compliant to WSDL 1.1 standard defined in Web Service Description Language (WSDL) 1.1, and the schema file in this section is compliant to XML schema standard defined in XML Schema Part 0: Primer, XML Schema Part 1: Structures, and XML Schema Part 2: Datatypes.

The WSDL file for CAI3G SessionControl service is as follows. It describes the SessionControl service that a CAI3GAgent can provide toward a CAI3GManager.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/">
  <types>
    <xs:schema
      <xs:import namespace="http://schemas.ericsson.com/cai3g1.2/"
        schemaLocation="cai3g1.2_.xsd"/>
    </xs:schema>
  </types>
  <message name="Login">
    <part name="Login" element="cai3g:Login"/>
  </message>
  <message name="LoginResponse">
    <part name="LoginResponse" element="cai3g:LoginResponse"/>
  </message>
  <message name="Cai3G12FaultException">
    <part name="fault" element="cai3g:Cai3gFault"/>
  </message>
  <message name="Logout">
    <part name="Logout" element="cai3g:Logout"/>
    <part name="SessionId" element="cai3g:SessionId"/>
  </message>
  <message name="LogoutResponse">
    <part name="LogoutResponse" element="cai3g:LogoutResponse"/>
    <part name="SessionId" element="cai3g:SessionId"/>
  </message>
  <portType name="SessionControl">
    <operation name="Login">
      <input message="cai3g:Login"/>
      <output message="cai3g:LoginResponse"/>
      <fault name="Cai3G12FaultException" message="cai3g:Cai3G12FaultException"/>
    </operation>
    <operation name="Logout" parameterOrder="Logout SessionId">
      <input message="cai3g:Logout"/>
      <output message="cai3g:LogoutResponse"/>
      <fault name="Cai3G12FaultException" message="cai3g:Cai3G12FaultException"/>
    </operation>
  </portType>
  <binding name="SessionControlImplPortBinding" type="cai3g:SessionControl">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Login">
      <soap:operation soapAction="CAI3G#Login"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="Cai3G12FaultException">
        <soap:fault name="Cai3G12FaultException" use="literal"/>
      </fault>
    </operation>
```



```
<operation name="Logout">
  <soap:operation soapAction="CAI3G#Logout"/>
  <input>
    <soap:body parts="Logout" use="literal"/>
    <soap:header message="cai3g:Logout" part="SessionId" use="literal"/>
  </input>
  <output>
    <soap:body parts="LogoutResponse" use="literal"/>
    <soap:header message="cai3g:LogoutResponse" part="SessionId" use="literal"/>
  </output>
  <fault name="Cai3G12FaultException">
    <soap:fault name="Cai3G12FaultException" use="literal"/>
  </fault>
</operation>
</binding>
<service name="SessionControlService">
  <port name="SessionControlImplPort" binding="cai3g:SessionControlImplPortBinding">
    <soap:address location="http://127.0.0.1:8998/cai3g12/SessionControl"/>
  </port>
</service>
</definitions>
```

Example 41 HTTP-Based WSDL File for CAI3G SessionControl Service

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapjms="http://www.w3.org/2010/soapjms/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/"
  <import namespace="http://schemas.ericsson.com/cai3g1.2/"
    location="cai3g1.2_.xsd" />
  <message name="LoginRequest">
    <part name="parameters" element="cai3g:Login" />
  </message>
  <message name="LoginResponse">
    <part name="parameters" element="cai3g:LoginResponse" />
  </message>
  <message name="LogoutRequest">
    <part name="parameters" element="cai3g:Logout" />
  </message>
  <message name="LogoutResponse">
    <part name="parameters" element="cai3g:LogoutResponse" />
  </message>
  <message name="HeadInfo">
    <part name="sessionId" element="cai3g:SessionId" />
    <part name="transactionId" element="cai3g:TransactionId" />
    <part name="sequenceId" element="cai3g:SequenceId" />
  </message>
  <message name="Cai3gFault">
    <part name="parameters" element="cai3g:Cai3gFault" />
  </message>
  <message name="Cai3gHeaderFault">
    <part name="sessionIdFault" type="cai3g:SessionIdFault" />
    <part name="transactionIdFault" type="cai3g:TransactionIdFault" />
    <part name="sequenceIdFault" type="cai3g:SequenceIdFault" />
  </message>
  <portType name="SessionControl">
    <operation name="Login">
      <input message="cai3g:LoginRequest" />
      <output message="cai3g:LoginResponse" />
      <fault name="Cai3gFault" message="cai3g:Cai3gFault" />
    </operation>
    <operation name="Logout">
      <input message="cai3g:LogoutRequest" />
      <output message="cai3g:LogoutResponse" />
      <fault name="Cai3gFault" message="cai3g:Cai3gFault" />
    </operation>
  </portType>
  <binding name="SessionControl" type="cai3g:SessionControl">
    <soap:binding style="document" transport="http://www.w3.org/2010/soapjms/" />
    <operation name="Login">
```




```

<soap:operation soapjms:soapAction="CAI3G#Login"
  style="document" />
<input>
  <soap:body use="literal" />
</input>
<output>
  <soap:body use="literal" />
</output>
<fault name="Cai3gFault">
  <soap:fault name="Cai3gFault" use="literal" />
</fault>
</operation>
<operation name="Logout">
  <soap:operation soapjms:soapAction="CAI3G#Logout"
    style="document" />
  <input>
    <soap:body use="literal" />
    <soap:header message="cai3g:HeadInfo" part="sessionId"
      use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
    <soap:header message="cai3g:HeadInfo" part="sessionId"
      use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault"
        part="sessionIdFault" use="literal" />
    </soap:header>
  </output>
  <fault name="Cai3gFault">
    <soap:fault name="Cai3gFault" use="literal" />
  </fault>
</operation>
</binding>

<service name="CAI3G">
  <port name="SessionControl" binding="cai3g:SessionControl">
    <soap:address
      location="jms:jndi:myQueue?targetService=emaProvisioning&
        priority=8&replyToName=queueName" />
  </port>
</service>
</definitions>

```

Example 42 JMS-Based WSDL File for CAI3G SessionControl Service

The WSDL file for CAI3G Provisioning service must be provided by CAI3GAgent for each specific MO.

The CAI3GAgent must prepare a specific schema file according to specific MO requirements (see Section 5.6 on page 54) and CAI3G Provisioning operation definitions (see Section 6 on page 57).

The following is an example for reference:

```

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/">
  <import namespace="http://schemas.ericsson.com/cai3g1.2/" location="cai3g1.2_.xsd"/>
  <message name="CreateRequest">
    <part name="parameters" element="cai3g:Create"/>
  </message>
  <message name="CreateResponse">
    <part name="parameters" element="cai3g:CreateResponse"/>
  </message>
  <message name="GetRequest">
    <part name="parameters" element="cai3g:Get"/>
  </message>

```



```
</message>
<message name="GetResponse">
  <part name="parameters" element="cai3g:GetResponse"/>
</message>
<message name="SetRequest">
  <part name="parameters" element="cai3g:Set"/>
</message>
<message name="SetResponse">
  <part name="parameters" element="cai3g:SetResponse"/>
</message>
<message name="DeleteRequest">
  <part name="parameters" element="cai3g:Delete"/>
</message>
<message name="DeleteResponse">
  <part name="parameters" element="cai3g:DeleteResponse"/>
</message>
<message name="SearchRequest">
  <part name="parameters" element="cai3g:Search"/>
</message>
<message name="SearchResponse">
  <part name="parameters" element="cai3g:SearchResponse"/>
</message>
<message name="HeadInfo">
  <part name="sessionId" element="cai3g:SessionId"/>
  <part name="transactionId" element="cai3g:TransactionId"/>
  <part name="sequenceId" element="cai3g:SequenceId"/>
</message>
<message name="Cai3gFault">
  <part name="parameters" element="cai3g:Cai3gFault"/>
</message>
<message name="Cai3gHeaderFault">
  <part name="sessionIdFault" type="cai3g:SessionIdFault"/>
  <part name="transactionIdFault" type="cai3g:TransactionIdFault"/>
  <part name="sequenceIdFault" type="cai3g:SequenceIdFault"/>
</message>
<portType name="Provisioning">
  <operation name="Create">
    <input message="cai3g:CreateRequest"/>
    <output message="cai3g:CreateResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Delete">
    <input message="cai3g:DeleteRequest"/>
    <output message="cai3g:DeleteResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Get">
    <input message="cai3g:GetRequest"/>
    <output message="cai3g:GetResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Set">
    <input message="cai3g:SetRequest"/>
    <output message="cai3g:SetResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Search">
    <input message="cai3g:SearchRequest"/>
    <output message="cai3g:SearchResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
</portType>
<binding name="Provisioning" type="cai3g:Provisioning">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Create">
    <soap:operation soapAction="CAI3G#Create" style="document"/>
    <input>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault">
```



```

        use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
          use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
          use="literal"/>
      </soap:header>
    </output>
    <fault name="Cai3gFault">
      <soap:fault name="Cai3gFault" use="literal"/>
    </fault>
  </operation>
  <operation name="Delete">
    <soap:operation soapAction="CAI3G#Delete" style="document"/>
    <input>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
          use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
          use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
          use="literal"/>
      </soap:header>
    </output>
    <fault name="Cai3gFault">
      <soap:fault name="Cai3gFault" use="literal"/>
    </fault>
  </operation>
  <operation name="Get">
    <soap:operation soapAction="CAI3G#Get" style="document"/>
    <input>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
          use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
          use="literal"/>
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
          use="literal"/>
      </soap:header>
    </output>
    <fault name="Cai3gFault">
      <soap:fault name="Cai3gFault" use="literal"/>
    </fault>
  </operation>
  <operation name="Set">
    <soap:operation soapAction="CAI3G#Set" style="document"/>
    <input>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>

```



```

</input>
<output>
  <soap:body use="literal"/>
  <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
      use="literal"/>
  </soap:header>
  <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
      use="literal"/>
  </soap:header>
  <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
      use="literal"/>
  </soap:header>
</output>
<fault name="Cai3gFault">
  <soap:fault name="Cai3gFault" use="literal"/>
</fault>
</operation>
<operation name="Search">
  <soap:operation soapAction="CAI3G#Search" style="document"/>
  <input>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
        use="literal"/>
    </soap:header>
  </output>
  <fault name="Cai3gFault">
    <soap:fault name="Cai3gFault" use="literal"/>
  </fault>
</operation>
</binding>
<service name="CAI3G">
  <port name="Provisioning" binding="cai3g:Provisioning">
    <soap:address location="http://anyema.anyprovisioningprovider.com/cai3g"/>
  </port>
</service>
</definitions>

```

Example 43 HTTP-Based WSDL File for CAI3G Provisioning Service

```

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapjms="http://www.w3.org/2010/soapjms/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/">
  <import namespace="http://schemas.ericsson.com/cai3g1.2/" location="cai3g1.2_.xsd"/>
  <message name="CreateRequest">
    <part name="parameters" element="cai3g:Create"/>
  </message>
  <message name="CreateResponse">
    <part name="parameters" element="cai3g:CreateResponse"/>
  </message>
  <message name="GetRequest">
    <part name="parameters" element="cai3g:Get"/>
  </message>

```



```

<message name="GetResponse">
  <part name="parameters" element="cai3g:GetResponse"/>
</message>
<message name="SetRequest">
  <part name="parameters" element="cai3g:Set"/>
</message>
<message name="SetResponse">
  <part name="parameters" element="cai3g:SetResponse"/>
</message>
<message name="DeleteRequest">
  <part name="parameters" element="cai3g:Delete"/>
</message>
<message name="DeleteResponse">
  <part name="parameters" element="cai3g:DeleteResponse"/>
</message>
<message name="SearchRequest">
  <part name="parameters" element="cai3g:Search"/>
</message>
<message name="SearchResponse">
  <part name="parameters" element="cai3g:SearchResponse"/>
</message>
<message name="HeadInfo">
  <part name="sessionId" element="cai3g:SessionId"/>
  <part name="transactionId" element="cai3g:TransactionId"/>
  <part name="sequenceId" element="cai3g:SequenceId"/>
</message>
<message name="Cai3gFault">
  <part name="parameters" element="cai3g:Cai3gFault"/>
</message>
<message name="Cai3gHeaderFault">
  <part name="sessionIdFault" type="cai3g:SessionIdFault"/>
  <part name="transactionIdFault" type="cai3g:TransactionIdFault"/>
  <part name="sequenceIdFault" type="cai3g:SequenceIdFault"/>
</message>
<portType name="Provisioning">
  <operation name="Create">
    <input message="cai3g:CreateRequest"/>
    <output message="cai3g:CreateResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Delete">
    <input message="cai3g:DeleteRequest"/>
    <output message="cai3g:DeleteResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Get">
    <input message="cai3g:GetRequest"/>
    <output message="cai3g:GetResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Set">
    <input message="cai3g:SetRequest"/>
    <output message="cai3g:SetResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
  <operation name="Search">
    <input message="cai3g:SearchRequest"/>
    <output message="cai3g:SearchResponse"/>
    <fault name="Cai3gFault" message="cai3g:Cai3gFault"/>
  </operation>
</portType>
<binding name="Provisioning" type="cai3g:Provisioning">
  <soap:binding style="document" transport="http://www.w3.org/2010/soapjms"/>
  <operation name="Create">
    <soap:operation style="document" soapjms:soapAction="CAI3G#Create"/>
    <input>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
      <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
        use="literal"/>
    </output>
  </operation>

```



```
</soap:header>
<soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
  <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
    use="literal"/>
</soap:header>
<soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
  <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
    use="literal"/>
</soap:header>
</output>
<fault name="Cai3gFault">
  <soap:fault name="Cai3gFault" use="literal"/>
</fault>
</operation>
<operation name="Delete">
  <soap:operation style="document" soapjms:soapAction="CAI3G#Delete"/>
  <input>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
        use="literal"/>
    </soap:header>
  </output>
  <fault name="Cai3gFault">
    <soap:fault name="Cai3gFault" use="literal"/>
  </fault>
</operation>
<operation name="Get">
  <soap:operation style="document" soapjms:soapAction="CAI3G#Get"/>
  <input>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
        use="literal"/>
    </soap:header>
  </output>
  <fault name="Cai3gFault">
    <soap:fault name="Cai3gFault" use="literal"/>
  </fault>
</operation>
<operation name="Set">
  <soap:operation style="document" soapjms:soapAction="CAI3G#Set"/>
  <input>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
  </input>
```



```

<output>
  <soap:body use="literal"/>
  <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
      use="literal"/>
  </soap:header>
  <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
      use="literal"/>
  </soap:header>
  <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
    <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
      use="literal"/>
  </soap:header>
</output>
<fault name="Cai3gFault">
  <soap:fault name="Cai3gFault" use="literal"/>
</fault>
</operation>
<operation name="Search">
  <soap:operation style="document" soapjms:soapAction="CAI3G#Search"/>
  <input>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
    <soap:header message="cai3g:HeadInfo" part="sessionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sessionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="transactionId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="transactionIdFault"
        use="literal"/>
    </soap:header>
    <soap:header message="cai3g:HeadInfo" part="sequenceId" use="literal">
      <soap:headerfault message="cai3g:Cai3gHeaderFault" part="sequenceIdFault"
        use="literal"/>
    </soap:header>
  </output>
  <fault name="Cai3gFault">
    <soap:fault name="Cai3gFault" use="literal"/>
  </fault>
</operation>
</binding>
<service name="CAI3G">
  <port name="Provisioning" binding="cai3g:Provisioning">
    <soap:address location=
"jms:jndi:myQueue?targetService=emaProvisioning &amp;priority=8&amp;replyToName=queueName"/>
  </port>
</service>
</definitions>

```

Example 44 JMS-Based WSDL File for CAI3G Provisioning Service

The WSDL file for CAI3G Notification service is as follows:

```

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/">
  <types>
    <xs:schema>
      <xs:import namespace="http://schemas.ericsson.com/cai3g1.2/"
        schemaLocation="cai3g1.2_.xsd"/>
    </xs:schema>
  </types>
  <message name="Notify">

```



```
<part name="Notify" element="cai3g:Notify"/>
</message>
<message name="NotifyResponse">
  <part name="NotifyResponse" element="cai3g:NotifyResponse"/>
</message>
<message name="Cai3G12FaultException">
  <part name="fault" element="cai3g:Cai3gFault"/>
</message>
<message name="Unsubscribe">
  <part name="Unsubscribe" element="cai3g:Unsubscribe"/>
  <part name="SessionId" element="cai3g:SessionId"/>
  <part name="TransactionId" element="cai3g:TransactionId"/>
  <part name="SequenceId" element="cai3g:SequenceId"/>
</message>
<message name="UnsubscribeResponse">
  <part name="UnsubscribeResponse" element="cai3g:UnsubscribeResponse"/>
  <part name="SessionId" element="cai3g:SessionId"/>
  <part name="TransactionId" element="cai3g:TransactionId"/>
  <part name="SequenceId" element="cai3g:SequenceId"/>
</message>
<message name="Subscribe">
  <part name="Subscribe" element="cai3g:Subscribe"/>
  <part name="SessionId" element="cai3g:SessionId"/>
  <part name="TransactionId" element="cai3g:TransactionId"/>
  <part name="SequenceId" element="cai3g:SequenceId"/>
</message>
<message name="SubscribeResponse">
  <part name="SubscribeResponse" element="cai3g:SubscribeResponse"/>
  <part name="SessionId" element="cai3g:SessionId"/>
  <part name="TransactionId" element="cai3g:TransactionId"/>
  <part name="SequenceId" element="cai3g:SequenceId"/>
</message>
<portType name="Notification">
  <operation name="Notify">
    <input message="cai3g:Notify"/>
    <output message="cai3g:NotifyResponse"/>
    <fault name="Cai3G12FaultException" message="cai3g:Cai3G12FaultException"/>
  </operation>
  <operation name="Unsubscribe"
    parameterOrder="Unsubscribe SessionId TransactionId SequenceId">
    <input message="cai3g:Unsubscribe"/>
    <output message="cai3g:UnsubscribeResponse"/>
    <fault name="Cai3G12FaultException" message="cai3g:Cai3G12FaultException"/>
  </operation>
  <operation name="Subscribe" parameterOrder="Subscribe SessionId TransactionId SequenceId">
    <input message="cai3g:Subscribe"/>
    <output message="cai3g:SubscribeResponse"/>
    <fault name="Cai3G12FaultException" message="cai3g:Cai3G12FaultException"/>
  </operation>
</portType>
<binding name="NotificationImplPortBinding" type="cai3g:Notification">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Notify">
    <soap:operation soapAction="CAI3G#Notify"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="Cai3G12FaultException">
      <soap:fault name="Cai3G12FaultException" use="literal"/>
    </fault>
  </operation>
  <operation name="Unsubscribe">
    <soap:operation soapAction="CAI3G#Unsubscribe"/>
    <input>
      <soap:body parts="Unsubscribe" use="literal"/>
      <soap:header message="cai3g:Unsubscribe" part="SessionId" use="literal"/>
      <soap:header message="cai3g:Unsubscribe" part="TransactionId" use="literal"/>
      <soap:header message="cai3g:Unsubscribe" part="SequenceId" use="literal"/>
    </input>
    <output>
      <soap:body parts="UnsubscribeResponse" use="literal"/>
      <soap:header message="cai3g:UnsubscribeResponse" part="SessionId" use="literal"/>
      <soap:header message="cai3g:UnsubscribeResponse" part="TransactionId" use="literal"/>
    </output>
  </operation>
</binding>
```




```

    <soap:header message="cai3g:UnsubscribeResponse" part="SequenceId" use="literal"/>
  </output>
  <fault name="Cai3G12FaultException">
    <soap:fault name="Cai3G12FaultException" use="literal"/>
  </fault>
</operation>
<operation name="Subscribe">
  <soap:operation soapAction="CAI3G#Subscribe"/>
  <input>
    <soap:body parts="Subscribe" use="literal"/>
    <soap:header message="cai3g:Subscribe" part="SessionId" use="literal"/>
    <soap:header message="cai3g:Subscribe" part="TransactionId" use="literal"/>
    <soap:header message="cai3g:Subscribe" part="SequenceId" use="literal"/>
  </input>
  <output>
    <soap:body parts="SubscribeResponse" use="literal"/>
    <soap:header message="cai3g:SubscribeResponse" part="SessionId" use="literal"/>
    <soap:header message="cai3g:SubscribeResponse" part="TransactionId" use="literal"/>
    <soap:header message="cai3g:SubscribeResponse" part="SequenceId" use="literal"/>
  </output>
  <fault name="Cai3G12FaultException">
    <soap:fault name="Cai3G12FaultException" use="literal"/>
  </fault>
</operation>
</binding>
<service name="NotificationService">
  <port name="NotificationImplPort" binding="cai3g:NotificationImplPortBinding">
    <soap:address location="http://127.0.0.1:8998/cai3g12/Notification"/>
  </port>
</service>
</definitions>

```

Example 45 HTTP-Based WSDL File for CAI3G Notification Service

```

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapjms="http://www.w3.org/2010/soapjms/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/"
  targetNamespace="http://schemas.ericsson.com/cai3g1.2/"
  <import namespace="http://schemas.ericsson.com/cai3g1.2/"
    location="cai3g1.2_.xsd" />
  <message name="NotifyResponse">
    <part name="parameters" element="cai3g:NotifyResponse" />
  </message>
  <message name="UnsubscribeResponse">
    <part name="parameters" element="cai3g:UnsubscribeResponse" />
  </message>
  <message name="HeadInfo">
    <part name="sessionId" element="cai3g:SessionId" />
    <part name="transactionId" element="cai3g:TransactionId" />
    <part name="sequenceId" element="cai3g:SequenceId" />
  </message>
  <message name="SubscribeRequest">
    <part name="parameters" element="cai3g:Subscribe" />
  </message>
  <message name="SubscribeResponse">
    <part name="parameters" element="cai3g:SubscribeResponse" />
  </message>
  <message name="UnsubscribeRequest">
    <part name="parameters" element="cai3g:Unsubscribe" />
  </message>
  <message name="NotifyRequest">
    <part name="parameters" element="cai3g:Notify" />
  </message>
  <message name="Cai3gFault">
    <part name="parameters" element="cai3g:Cai3gFault" />
  </message>
  <message name="Cai3gHeaderFault">
    <part name="sessionIdFault" type="cai3g:SessionIdFault" />
    <part name="transactionIdFault" type="cai3g:TransactionIdFault" />
    <part name="sequenceIdFault" type="cai3g:SequenceIdFault" />
  </message>
  <portType name="Notification">

```



```
<operation name="Subscribe">
  <input message="cai3g:SubscribeRequest" />
  <output message="cai3g:SubscribeResponse" />
  <fault name="Cai3gFault" message="cai3g:Cai3gFault" />
</operation>
<operation name="Notify">
  <input message="cai3g:NotifyRequest" />
  <output message="cai3g:NotifyResponse" />
</operation>
<operation name="Unsubscribe">
  <input message="cai3g:UnsubscribeRequest" />
  <output message="cai3g:UnsubscribeResponse" />
  <fault name="Cai3gFault" message="cai3g:Cai3gFault" />
</operation>
</portType>

<binding name="Notification" type="cai3g:Notification">
  <soap:binding style="document" transport="http://www.w3.org/2010/soapjms/" />
  <operation name="Subscribe">
    <soap:operation soapjms:soapAction="CAI3G#Subscribe"
      style="document" />
    <input>
      <soap:body use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sessionId"
        use="literal" />
      <soap:header message="cai3g:HeadInfo" part="transactionId"
        use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sequenceId"
        use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sessionId"
        use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault"
          part="sessionIdFault" use="literal" />
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="transactionId"
        use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault"
          part="transactionIdFault" use="literal" />
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="sequenceId"
        use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault"
          part="sequenceIdFault" use="literal" />
      </soap:header>
    </output>
    <fault name="Cai3gFault">
      <soap:fault name="Cai3gFault" use="literal" />
    </fault>
  </operation>
  <operation name="Unsubscribe">
    <soap:operation soapjms:soapAction="CAI3G#Unsubscribe"
      style="document" />
    <input>
      <soap:body use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sessionId"
        use="literal" />
      <soap:header message="cai3g:HeadInfo" part="transactionId"
        use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sequenceId"
        use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
      <soap:header message="cai3g:HeadInfo" part="sessionId"
        use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault"
          part="sessionIdFault" use="literal" />
      </soap:header>
      <soap:header message="cai3g:HeadInfo" part="transactionId"
        use="literal">
        <soap:headerfault message="cai3g:Cai3gHeaderFault"
          part="transactionIdFault" use="literal" />
      </soap:header>
    </output>
  </operation>
</binding>
```



```

</soap:header>
<soap:header message="cai3g:HeadInfo" part="sequenceId"
  use="literal">
  <soap:headerfault message="cai3g:Cai3gHeaderFault"
    part="sequenceIdFault" use="literal" />
</soap:header>
</output>
<fault name="Cai3gFault">
  <soap:fault name="Cai3gFault" use="literal" />
</fault>
</operation>
<operation name="Notify">
  <soap:operation soapjms:soapAction="CAI3G#Notify"
    style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<service name="CAI3G">
  <port name="Notification" binding="cai3g:Notification">
    <soap:address
      location="jms:jndi:myQueue?targetService=
        emaProvisioning &priority=8&replyToName=queueName" />
    </port>
  </service>
</definitions>

```

Example 46 JMS-Based WSDL File for CAI3G Notification Service





12

Appendix C (Example of SOAP Message)

An example of a CAI3G SOAP request message is as follows:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <cai3g:SessionId
      xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/">
      1234567
    </cai3g:SessionId>
    <cai3g:TransactionId
      xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/">
      4294967295
    </cai3g:TransactionId>
    <cai3g:SequenceId
      xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/">
      4294967295
    </cai3g:SequenceId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <cai3g:Create
      xmlns:cai3g="http://schemas.ericsson.com/cai3g1.2/">
      <cai3g:MOType>
        User@http://www.mycompany.com/myservice/
      </cai3g:MOType>
      <cai3g:M0Id>
        <abc:socialId
          xmlns:abc="http://schemas.mycompany.com/myservice/">
          310102197204084417
        </abc:socialId>
      </cai3g:M0Id>
      <cai3g:M0Attributes>
        <CreateUser socialId="310102197204084417"
          xmlns="http://schemas.mycompany.com/myservice/">
          <name>Alice</name>
          <gender>Female</gender>
          <age>46</age>
          <socialId>310102197204084417</socialId>
        </CreateUser>
      </cai3g:M0Attributes>
      <cai3g:extension />
    </cai3g:Create>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 47 CAI3G SOAP Request

The above example creates a user instance whose social ID is “310102197204084417” and name is “Alice” in “myservice” provided by “mycompany”. The namespace of that service is “http://schemas.mycompany.com/service/”.

The response message to the above CAI3G SOAP request is as follows:



```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <SessionId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 1234567</SessionId>
    <TransactionId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 4294967</TransactionId>
    <SequenceId xmlns="http://schemas.ericsson.com/cai3g1.2/"> 42949672</SequenceId>
  </S:Header>
  <S:Body>
    <CreateResponse xmlns="http://schemas.ericsson.com/cai3g1.2/">
      <MOId>
        <socialId xmlns="http://schemas.mycompany.com/myservice/"
          xmlns:cai3="http://schemas.ericsson.com/cai3g1.2/" xmlns:soapenv="http://schemas.xmlsoap.org/soap
        </socialId>
        </MOId>
      </CreateResponse>
    </S:Body>
  </S:Envelope>
```

Example 48 Response Message



13 Appendix D (Development Instruction)

The developer starts from WSDL and XML schema definition to develop a Web Service System. Generally speaking, the developer uses a Web Service tool as a development environment. The tool can generate client (stub) and service (skeleton) code. With the generated code, the developer only needs to develop their own business logic and avoids programming of network and protocol.

The CAI3G system development also follows this development process. This section provides an example on how to develop a simple CAI3G system, including CAI3GAgent and CAI3GManager, with JAX-WS.

The example in this section provides the tips through the whole process of developing the CAI3G System.

1. Developing CAI3G server and deploying it as web service on the Web container Tomcat, Version 6.
2. Developing CAI3G client.

Note: The example does not cover the development for JMS protocol. The implementation methods of JMS-based web service are various. Different JMS implementation providers may define their own wire protocol and deployment instruction. For information about how to implement the CAI3GAgent and CAI3GManager over JMS, refer to the JMS implementation provider's document.

The example does not provide general guide on how to develop with JAX-WS for Java. For further information, refer to Java API for XML Web Services (JAX-WS) Users Guide.

Before developing the CAI3G system, download the following tool kits:

- JAVA JDK1.6
- Eclipse, an IDE Java development tool

In addition, the following tool kit is recommended for development of the CAI3G system:

- Apache Tomcat, a web container for web services

13.1 Development Process of CAI3G Server

The development process is described as follows:

Note: Users can develop the SessionControl, Notification, and Provisioning services by following the steps below.



1. Use the `wsimport` tool to generate CAI3G skeleton and CAI3G data type code. `Wsimport` is distributed with JDK1.6, and the user manual for this tool can be found in Java standard documents.

The unix command is as follows:

```
$JAVA_HOME/bin/wsimport -keep -s generated \  
myservice.wsdl
```

The `myservice.wsdl` file in the above command is provided in Example 43. The XSD file in the `myservice.wsdl` file is provided in Example 36.

Make sure that the XSD files referred to in the preceding WSDL file are available. The `/generated` in the above command is an existing directory to store generated skeleton files.

The directory structure for the generated codes is as follows:

```
generated/com/ericsson/schemas/cai3g1/AnyMOIdType.java  
...  
generated/com/ericsson/schemas/cai3g1/Create.java  
generated/com/ericsson/schemas/cai3g1/CreateMODefinition.java  
generated/com/ericsson/schemas/cai3g1/CreateResponse.java  
...  
generated/com/ericsson/schemas/cai3g1/Provisioning.java  
generated/com/ericsson/schemas/cai3g1/ProvisioningService.java  
...
```

2. Implement the generated interface of the server side.

In this case, implement the interface “Provisioning” that is defined as `<portType>` in the WSDL file.

Part of the `ProvisionImpl.java` file that implements the interface “Provisioning” is as follows:

```
package com.ericsson.ema.webservice.impl;  
  
...  
  
@javax.jws.WebService(targetNamespace = "http://schemas.ericsson.com/cai3g1.2/",  
    endpointInterface = "com.ericsson.schemas.cai3g1.Provisioning")  
public class ProvisioningImpl implements Provisioning {  
  
    ...  
  
    public com.ericsson.schemas.cai3g1.CreateResponse create  
        (com.ericsson.schemas.cai3g1.Create create) throws Cai3G12FaultException {  
  
        com.ericsson.schemas.cai3g12.CreateResponse _return = null;  
        //implement your business logical here  
        return _return;  
    }  
  
    ...  
  
}
```

3. Create a WAR package for deployment.



Add the following classes and description files to the `myservice.war` package.

```
WebContent/
WebContent/META-INF
WebContent/META-INF/context.xml
WebContent/WEB-INF
WebContent/WEB-INF/sun-jaxws.xml
WebContent/WEB-INF/web.xml
WebContent/WEB-INF/classes/com/*
```

The `context.xml` file is as follows:

```
<Context reloadable="true" crossContext="true">
    <!-- servlets that I want to access from subdomain are contained in this path -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

The `sun-jaxws.xml` file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
  version="2.0">
  <endpoint
    name="myservice"
    implementation="com.ericsson.ema.webservice.impl.ProvisioningImpl"
    url-pattern="/" />
</endpoints>
```

Note: Replace the `implementation` content with actual class name.

The `web.xml` file is as follows:



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
  <listener>
    <listener-class>
      com.sun.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>myservlet</servlet-name>
    <servlet-class>
      com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>h1r13sub</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>120</session-timeout>
  </session-config>
</web-app>
```

Note: You can modify the `servlet-name`, `url-pattern`, and `session-timeout` according to the configuration.

4. Deploy and test the previously created WAR package.

Copy the `myservice.war` package to your web-server destination directory.

Check the log file `/<Your Destination Directory>/catalina.log`.

If the `myservice.war` package is successfully loaded, open a web browser and go to `http://youraddress:<Your Port>/myservice/`.

The result page is shown in Figure 9.

Web Services

Endpoint	Information
Service Name: {http://schemas.ericsson.com/cai3g1.2/}ProvisioningImplService	Address: http://127.0.0.1/myservice
Port Name: {http://schemas.ericsson.com/cai3g1.2/}ProvisioningImplPort	WSDL: http://127.0.0.1/myservice/?wsdl
	Implementation class: com.ericsson.ema.webservice.impl.ProvisioningImpl

Figure 9 Web Services Result Page

13.2 Development Process of CAI3G Client

The development process is described as follows:

Note: Users can develop the Notification and Provisioning services by following the steps below.



1. Use the `wsimport` tool to generate CAI3G stubs and CAI3G data-type code. For detailed commands, see Step 1.
2. Create class `Client` that depends on the stubs.

```
public class Client
{
    public static void main(String[] args) throws Exception
    {
        //Log in to the CAI3G server here if needed

        URL url= new URL("http://localhost/myservice/?wsdl");

        ProvisioningService service = new ProvisioningService
            (url, new QName("http://schemas.ericsson.com/cai3g1.2/", "ProvisioningService"));

        Provisioning provisioning = service.getCAI3GImplPort();
        Create create = new Create();
        //fulfill the request here
        ...

        CreateResponse response = provisioning.create(create);
        //handle the response here
        ...
        //Log out from the CAI3G server if needed
    }
}
```

3. Compile and run the client.





14 Appendix E (Change History)

Table 5 Change History

Date	Modification	Version
2004/2/19	Change the namespace	1.0
	Change NotificationFiltersType	
	Change NotificationHeaderType	
	Change NotificationOperationType, e.g: ObjectCreation -> Create	
	Naming convention, e.g.: moType -> MOType , NotificationHeader -> notificationHeader	
	Remove "KeyString" definition and replace it's usage with "UserIdType"	
2004/3/24	Change the cai3gUser in NotificationFilterType from UserIdType to xs:string	1.0
	Change the userid in Login from UserIdType to xs:string	
	Remove simple type "UserIdType"	
2004/3/30	Change the eventTime in NotificationHeaderType from xs:time to xs:dateTime	
2004/4/7	Remove some unnecessary type definition	1.0
	Adjust the order of some elements definition	
2004/11/23	Add definition for "Search" request/response	1.1
	Add a optional input parameter to Create/Delete/Get/Set/Search operation to contain any extension data except MOType, MOId and MOAttributes	
2004/12/14	Add "filters" to the "selector" xpath in the "OperationUnique" definition	1.1
2004/12/31	Add "MOId" and " MOAttributes" as optional response for Delete operation	1.1
2006/10/17	Add "extension" as optional response for Create/Set/Delete/Get/Search operation	1.1
2007/6/11	Add input "MOAttributes" as optional request for Delete/Get operation	1.2
	A rule added for the naming convention of the top level elements in MOAttributes	
2011/9/27	Make CAI3G 1.2 compatible with web-service standards	1.2
	Introduce security profile and allow the SessionControl service to be optional	
	Introduce JMS transportation	





Glossary

3PP

Third Party Product

API

Application Programming Interface

CAI3G

Customer Administration Interface Third Generation

GUI

Graphical User Interface

HTTP

HyperText Transfer Protocol

IT

Information Technology

MIME

Multipurpose Internet Mail Extensions

MO

Managed Object

RFC

Request For Comments

RPC

Remote Procedure Call

SOAP

Simple Object Access Protocol

TLS

Transport Layer Security

URI

Universal Resource Identifier

WSDL

Web Service Description Language

XML

eXtensible Markup Language





Reference List

- [1] Key Words for Use in RFCs to Indicate Requirement Levels, <http://www.ietf.org/rfc/rfc2119.txt>
- [2] XML Information Set, <http://www.w3.org/TR/xml-infoset>
- [3] Simple Object Access Protocol (SOAP) 1.1, <http://www.w3.org/TR/soap>
- [4] Hypertext Transfer Protocol HTTP/1.1, <http://www.ietf.org/rfc/rfc2616.txt>
- [5] Extensible Markup Language (XML) 1.1 (Second Edition), <http://www.w3.org/TR/REC-xml>
- [6] Web Service Description Language (WSDL) 1.1, <http://www.w3.org/TR/wsdl>
- [7] XML Schema Part 0: Primer, <http://www.w3.org/TR/xmlschema-0/>
- [8] XML Schema Part 1: Structures, <http://www.w3.org/TR/xmlschema-1/>
- [9] XML Schema Part 2: Datatypes, <http://www.w3.org/TR/xmlschema-2/>
- [10] Java API for XML Web Services (JAX-WS) Users Guide, <http://jax-ws.java.net/nonav/2.1.7/docs/UsersGuide.html>
- [11] Java Message Service, <http://www.oracle.com/technetwork/java/index-js-p-142945.html>