



## CHAPTER 2

# Extensible Markup Language Processing

---

**Revised: June 24, 2009, OL-15335-04**

This chapter describes the Extensible Markup Language (XML) process in the Common Object Request Broker Architecture (CORBA) adapter.

## XML and Components

Along with XML, the primary component of the CORBA adapter is the CORBA interface servant (CIS) Java program. In addition, there are dependencies on related components in the managed object (MO) Java package. The required packages are as follows:

- Apache XML
- Xerces (parser)
- ECS (XML Document Building Tool Kit)

XML and the CIS Java packages are central, functional components in the Cisco BTS 10200 Softswitch. There is no larger object model applied to the Softswitch. A larger object model is deferred until a Cisco Systems standard model is created. This model covers applications for packet telephony for a variety of applications.

## XML in the CORBA Interface Servant

This section describes how to use XML in the CIS. Terms used in this section follow those used in XML specifications. This is to avoid confusion in the use of terms such as element, subelement, and attribute.

## CIS Functions

Schemas are provided for client-side verification of the XML document structure. These schemas cover the following items:

- ManagedObject
- Request
- Reply

The schema for a ManagedObject follows the format listed below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="ManagedObject">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="MOAttribute" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Verb" type="xs:string" use="required"/>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="MOAttribute">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Required"/>
        <xs:element ref="Type"/>
        <xs:element ref="Default"/>
        <xs:element ref="Width"/>
        <xs:element ref="HelpText"/>
        <xs:element ref="Label"/>
        <xs:element ref="Parser" minOccurs="0"/>
        <xs:element ref="Permitted" minOccurs="0"/>
        <xs:element ref="Fk" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Required" type="xs:boolean"/>

  <xs:element name="Type">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="single"/>
        <xs:enumeration value="text"/>
        <xs:enumeration value="multi"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

  <xs:element name="Default" type="xs:string"/>

  <xs:element name="HelpText" type="xs:string"/>

  <xs:element name="Label" type="xs:string"/>

  <xs:element name="Noun" type="xs:string"/>

  <xs:element name="Param" type="xs:string"/>

  <xs:element name="Parser">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="JavaScript"/>
        <xs:element ref="RegExp"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="JavaScript" type="xs:string"/>

```

```

<xs:element name="RegExp" type="xs:string"/>

<xs:element name="Permitted" type="xs:string"/>

<xs:element name="Width" type="xs:int" />

<xs:element name="Fk">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Noun"/>
      <xs:element ref="Param"/>
      <xs:element ref="Fk" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>

```

The schema for a *Request* follows the format listed below.

```

<?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="Request">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Entry" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Verb" type="xs:string" use="required"/>
        <xs:attribute name="Noun" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Entry">
      <xs:complexType>
        <xs:attribute name="Key" type="xs:string" use="required"/>
        <xs:attribute name="Value" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>

```

The schema for a *Reply* follows the format listed below.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="Reply">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Status"/>
        <xs:element ref="Reason"/>
        <xs:element ref="Size"/>
        <xs:element ref="AbsoluteSize"/>
        <xs:element ref="StartRow"/>
        <xs:element ref="DataTable"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Status" type="xs:boolean"/>

  <xs:element name="Reason" type="xs:string"/>

  <xs:element name="Size" type="xs:integer"/>

```

```

<xs:element name="AbsoluteSize" type="xs:integer" />

<xs:element name="StartRow" type="xs:integer" />

<xs:element name="DataTable">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Row" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Row">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Column" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:integer" />
  </xs:complexType>
</xs:element>

<xs:element name="Column" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="id" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

</xs:schema>

```

The interface definition language (IDL) allows access to XML description documents for each noun and verb combination. For example, the **add subscriber** command generates a matching XML document that defines the element and attributes of this command. The IDL allows command processing based on a well-formed but unverified XML document.

The IDL allows command access to supported media gateway (MGW) devices. The command strings do not follow the XML access format defined in the schema. The CIS supports MGW command strings that are native to the MGW internal command structure.

All XML documents that originate in the BTS 10200 are dynamically generated. This includes all command description documents.

## ManagedObject

A ManagedObject has one element, the MOAttribute. A ManagedObject also has two attributes: the Id of the ManagedObject and the Verb. The Id represents the object on which some action is to be taken. The Verb indicates the action to be taken. For example, subscriber, or termination is a valid id. This is a required attribute.

The following list describes the various parts of the ManagedObject and their values:

- **Id**—This attribute represents the object on which to take some action.
- **Verb**—This attribute defines the action to take on a given ManagedObject. This is a required attribute and is composed of character data.
- **MOAttribute**—The ManagedObject can contain none, one, or more MOAttributes. It has one attribute named Id. This character data acts as a label for the element. The order of these elements does not imply any specific behavior. They are arbitrarily listed.
- **Required**—This subelement has two values, true and false.
- **Type**—This subelement defines whether the MOAttribute has a single value, multiple values, or is a text. The multiple or single options imply that a list of choices is offered in the permitted element.
- **Default**—This subelement is informational. It indicates the default value for the MOAttribute.
- **Width**—This subelement indicates the total field width of the data. For example, if the MOAttribute is a description, this indicates the length of the description.
- **HelpText**—This subelement offers a brief text to indicate the nature of the MOAttribute.
- **Permitted**—This subelement specifies the possible values or ranges for the MOAttribute.
- **Parser**—This subelement indicates what type of validation is required. There is a single attribute for this subelement. The attribute is an Id field constructed of character data. The subelements are listed below:
  - **JavaScript**—This subelement indicates a possible JavaScript to perform validation or regular expression matching.
  - **RegExp**—This subelement defines the regular expression in character data format.

## Request

The *Request* schema consists of one element, containing none, one, or more *Entry* elements, and two attributes and their values.

The following list describes the various parts of the schema and its values:

- **Noun**—This attribute defines the item on which some operation is requested. This is expressed as character data.
- **Verb**—This attribute defines the action to perform on the "Noun" attribute. This is expressed as character data.

The *Entry* element is allowed to be empty. It can also contain two attributes. These attributes are defined as follows:

- **Key**—This is the *id* value derived from the *MOAttribute* in the *ManagedObject*. It is expressed as character data.

- **Value**—This is the client-derived value to assign to the key specified above. The *Value* is expressed as character data. It should also conform to the subelements in *MOAttribute* from which this key/value pair was derived.

## Reply

The *Reply* schema defines the structure of returned data generated in response to a *Request*. The *Reply* contains three elements and no attributes. These elements are defined as follows:

- **Status**—The *Reply* contains one *Status* element. It has two possible values. Either true or false is applied to this element.
- **Reason**—The *Reason* element contains character data. This element explains the cause for an error in processing a command or returns a success indication.
- **DataTable**—This element has one attribute and one subelement, which are defined below. This element is used as the container for data that results from the execution of a request. Each *Reply* can contain a *DataTable* element.
  - **Row**—This subelement defines a single complete item of data. A *DataTable* can contain one or more *Row* subelements. A *Row* has one attribute. This character data defines the row ID. The ID is always a sequential value based on the number of returned rows. The *id* attribute is required.
  - **Col**—Each *Row* contains a subelement known as a *Col*. This subelement has one attribute. The value of the element is expressed as character data. The attribute for *Col* is *id*. It is expressed as a character value. This is the same *id* value used in the *MOAttribute*. This is a required attribute.

## CORBA Interface Servant Adapter Implementation

The CIS is an adapter implementation that specifies an external interface. This section provides more detail about the structure of the document interchange between the CIS program and a client-side program. One global issue for the external interface is that all documents covered here are defined as *well-formed* but not *verified*. This means that the schema is not an embedded part of the XML document. When the schema are embedded, parser packages can be used to validate the structure of the document. However, this impedes the transition to XML schemas, should schemas be desired by other customers. The client side can still use the schema, included in this document, to perform validation.

## Cisco BTS 10200 Softswitch IDL Code

This section presents an example of the system IDL file for the CORBA adapter (CAD) interface in the BTS 10200. This IDL applies to BTS 10200, Release 4.x.

```
// Copyright (c) 2002, 2006 by Cisco Systems, Inc.
//=====
//
// Name:          bts10200.idl
// Author:       A. J. Blanchard
// Description:
// This is the IDL for the entire provisioning infrastructure of the
// BTS 10200. The text strings are all XML well-formed documents. The
// current procedure is to maintain separate schema(s). This allows later
// migration to schemas and away from schema for document validation.
//
```

```

// All commands are expressed as XML documents. The template document for
// each NOUN/VERB pair is accessible from the a separate method. This XML
// interface is table oriented and follows the same nomenclature and syntax
// as the other BTS 10200 adapter interfaces.
//
//
//=====
#ifndef bts10200_idl
#define bts10200_idl

//-----
// Set up modules to match java package tree for the OAM&P
//-----
module cad {

    typePrefix cad "oam.sswitch.com";

    //-----
    // Exceptions
    //-----
    exception CadExceptions {
        long    error_code;
        string  error_string;
    };

    interface Bts10200_Security {

        //-----
        // Create a session key.
        //-----
        void    login(in string          name,
                     in string          password,
                     out string         key)
                raises(CadExceptions);

        //-----
        // Destroy a session key.
        //-----
        void    logout(in string         key)
                raises(CadExceptions);

    }; // end Bts10200_Security

    interface Bts10200 {

        //-----
        // Fetch a command (noun/verb) XML document
        //-----
        void    getCommandDoc(in string          noun,
                             in string          verb,
                             in string          key,
                             out string         xml_doc)
                raises(CadExceptions);

        //-----
        // Fetch an Extended command (noun/verb) XML document - with foreign keys
        //-----
        void    getExtCommandDoc(in string      noun,
                                in string      verb,
                                in string      key,
                                out string     xml_doc)
                raises(CadExceptions);
    };
};

```

```

//-----
// Issue a command XML document (add, change, delete, show)
//-----
void      request(in string          xml_request,
                  in string          key,
                  out string         xml_reply)
          raises(CadExceptions);

}; // end Bts10200

interface Macro {

//-----
// Process a Macro Command XML document (add, change, delete, show)
//-----
void      execute(in string          request,
                  in string          key,
                  out string         reply)
          raises(CadExceptions);

}; // end Macro

}; // end cad
#endif // end bts10200_idl
```