CompatibleOne

Accords Parser v1.02

Iain James Marshall & Jean Pierre Laisné

11/05/2012

This document describes the operation of the ACCORDS Platform CORDS Parser.

Table des matières

Introduction

This document provides a detailed description of the operation of the ACCORDS Parser. It is intended for those wishing to understand the operational behavior of the tool in order to be able to correctly design CORDS documents and their associated schemas for the description of extensions to the ACCORDS platform be they tools or even secondary provisioning systems.

Overview

The ACCORDS Parser performs an important role in the operation of the ACCORDS Platform. It is responsible for the validation and processing of XML resource description documents submitted to

the platform. These documents are composed using the CompatibleOne Resource Description Schema (CORDS).

The processing of CORDS documents, as performed by the ACCORDS Parser, is the means by which infrastructure and other resource descriptions are introduced into the distributed knowledge base of the ACCORDS Platform. This knowledge base is composed of a collection of server components each responsible for the management of their particular collection of information categories, and instances thereof. Access to these information management components is through standardized OCCI/REST/HTTP interfaces. For further information concerning OCCI, its core model, it infrastructure categories and its renderings, please consult the corresponding specifications [5], [6] and [7].

Document Type

As is customary with XML documents, a namespace is expected to be provided by the value of an "xmlns" attribute of the outer document element. This attribute value MUST provide an URL from which the namespace may be retrieved and MUST take the form of an XSD Schema document. The schema will be used during document processing not only to validate the content of the document but also to control the way in which processing will be performed. The XSD document "manifest.xsd" will be retrieved from [1],[2],[3] and must be specified as the value of the namespace attribute for manifest documents describing infrastructure level provisioning (IAAS) using CORDS Versions 1 and 2 as described in the Cords Reference Manual [4].

```
<?xml version='1.0' encoding='UTF-8'?>
<manifest xmlns='http://www.compatibleone.fr/schemes/manifest.xsd'>
<…/>
</manifest>
```

Further manifests will be made available for the validation and control of processing of other document types amongst which platform as a service (PAAS) provisioning and platform configuration are amongst the targeted applications.

Operation

The ACCORDS Parser tool will be launched for the processing of a resource description document. The name of the document will be provided at the command line and may indicate a local file or a remote URL. The processing performed on this resource occurs in phases as is shown in the following figure:

XML

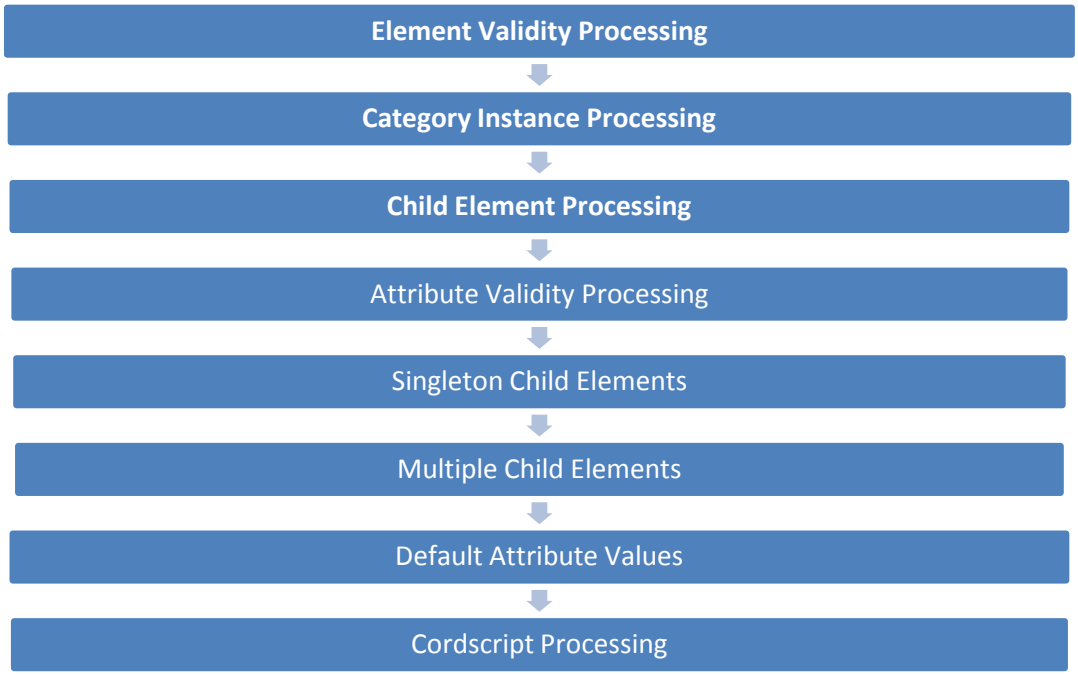
The XML file will be retrieved, as required, and then processed by the standard XML parser to build a document object model (DOM) tree structure representing the elements and attributes described in the document. Failure to retrieve the XML file will return an error as will any syntax errors encountered during the XML parsing required for the construction of the DOM. This document object model will then be submitted for processing by the ACCORDS Parser but first the name space definition must be retrieved.

XSD

The outer element of the DOM MUST provide an "xmlns" attribute specifying the local filename or remote URL of the XSD Schema that defines the structure of the current document object model. The XSD file will be retrieved and processed by the standard XML parser to build a document object model of this schema. Failure to retrieve the XSD file will return an error as will any syntax errors encountered during the XSD parsing required for the construction of the document object model of the schema.

DOM

The initial document object model will then be processed recursively by the ACCORDS Parser in a depth first manner using the document object model of the schema to control each level. This processing involves several distinct stages, at each level, and will be described in detail in the following sections of this document. The aim of the DOM processing is the production of a persistent graph composed of OCCI instances representing each of the elements described in the original XML document. This processing occurs in the phases as indicated in the following figure:



OCCI

Element Validity Processing

Each element of the XML document must be a valid child element of the parent element as declared by the current schema type description retrieved from the schema. For the outermost element this

type definition will be the schema document itself. An error condition will be raised for all elements which are not permitted to be used as child elements of a particular type definition. The names of elements should be prefixed by the name space identification tag as provided by the names space description schema declaration attribute of the form “xmlns:tag=’url’”. An element accepted by this stage will be submitted, along with its schema type definition, for processing by the next stage.

Category Instance Processing

This stage involves the resolution of the category instance identifier of the OCCI category instance that the element represents and describes. The name of the current element, under processing, is taken and used as the OCCI category name for which a request will be issued through the ACCORDS Publisher, the ACCORDS service publication and discovery service, in order to localize the OCCI category manager for this particular category of service. If an OCCI category manager cannot be localized for a particular element under consideration then an error condition will be raised.

Once the OCCI category manager has been localized then the following alternative processing conditions are possible depending on the attributes provided by the element description and the current state of not only the category manager but also the ACCORDS platform distributed knowledge base.

The ID attribute

When an ID attribute has been specified then the one of the following conditions will occur:

a. **the corresponding instance can be retrieved**
   In this case the explicit reference to the category instance is accepted and retained for the resolution of the element.

b. **the corresponding instance cannot be retrieved**
   Here the explicit reference to a category instance is incorrect and an error condition will be raised.

When a value for the ID attribute has not been specified then the NAME attribute will be considered as is described in the following section.

The NAME attribute

When a value for the NAME attribute has been provided then one of the following three conditions will result:

a. **a single named instance can be located and retrieved**
   The instance identifier of the named instance will be taken and used to resolve the element.

b. **multiple named instances can be located**
   In this case a choice of instances must be offered and the corresponding multiple choice condition will be raised.

c. **a single named instance cannot be retrieved**
   A new category instance will be created, of the required name and description, and the resulting instance identifier will be retained for resolution of the element.

When a value has not been specified for the NAME attribute then a new, un-named, category instance will be created, of the required description, and the resulting instance identifier will be retained as the resolution of the element.

Child Element Processing

For the element currently under processing, each of the child elements will then be submitted, in sequential order, for recursive depth wise processing starting at the stage named "*Element Validity Processing*" as described above.

Attribute Validity Processing

Once the processing of the child elements has completed then the resulting list of attributes, either explicitly specified or accumulated, for the element under consideration, will be tested for validity against the schema type definition provided for the level of processing. Error conditions will be raised for all attributes that are present for the element but are not defined in the schema type definition.

Singleton Child Element Processing

The collection of direct nested child elements will now be processed to detect the presence of valid attributes, of the same name as a child element, defined for the element currently being processed. When an attribute with a name matching the name of a nested child element is encountered then the value of the ID attribute of the child element will be extracted and used as the value of the attribute of the same name, of its parent element. For example, if a child element named "example" exists, for which an attribute named EXAMPLE has been defined for its parent element, then the value of the ID attribute of the child element "example" will be taken and used as the value of the EXAMPLE attribute of the parent element. This is an important feature of the processing performed by the ACCORDS Parser whereby category instances provide the basic building blocks contributing their properties to the resulting model.

Multiple Occurrence Child Element Processing

The collection of nested child elements will then be processed to detect the first element for which multiple occurrences are permitted, or required, by the schema type description of the parent element. These elements will be processed to produce OCCI LINK instances between the parent category instance and each of the child instance categories. This is another important feature of the processing performed by the ACCORDS Parser for the construction of the OCCI linkage graph.

Default Attribute Values

The collection of attributes, defined by the schema type description of the current element under processing, will be inspected to localize attributes for which a default value has been specified and for which an attribute has not already been provided. For each of these attributes, found to be missing, the specified default value will be used for the creation of an attribute that will be appended to the attribute list of the element under processing. The following example demonstrates the way in which default values may be specified in the XSD Schema document attribute definition.

```
<xsd:attribute name="example" default="DefaultValueString"/>
```

Cordscript Processing

The collection of attributes, of the current element under processing, will be inspected to localize attribute values indicating a need for further processing. This may be signaled in two ways. Firstly, the presence of "Cordscript:" prefixed to the value of an attribute indicates that the value is to be

submitted to the Cordscript parser for processing. Secondly, the value of any attribute named "Cordscript" will also be submitted for processing. In the second case the attribute will be removed from the attribute list of its parent element once processing has completed.

Cordscript statements take the following generic form:

```
{Attribute}.{method}( {parameter list} );
```

Here the terms `{Attribute}`, `{method}` and `{parameter list}` must be replaced by the required attribute name, the Cordscript action or method and an eventual list of parameter values. Attribute values of this kind may be composed of multiple Cordscript statements as required to perform the desired operation.

```
"Example.new(id.value); Example.start();"
```

Here we see an example comprising two Cordscript statements.

The first statement will build a new instance of the "example" category and store the resulting category instance identifier as the current value of the EXAMPLE attribute. The value of the ID attribute will be passed as the single parameter of the resulting OCCI POST request and will be identified in the rendering with the following format `{domain}.{element}.id` The values for the terms {domain} and {element} will be taken from the name of the application domain and the name of the current element being processed.

The second statement will invoke an OCCI "start" action request for the category instance identified by the current value of the indicated named attribute.

The attribute name resolution mechanism is very flexible and is based on a principle of "*cascading attribute values*". This is similar to the technique used in "*cascading style sheets*" where attributes or properties defined by outer parental elements are inherited, and may consequently be accessed directly by inner child element Cordscript expressions, unless of course they have been masked by identical named attributes of closer scope.

```
<outer name="example" xmlns="test.xsd">
<inner name="other" value="Cordscript: value.new( name.value, xmlns.value );"/>
</outer>
```

In this example the value of the first parameter, NAME, will be "other", being that of the "inner" element. The value of the second parameter, XMLNS, will be "test.xsd" which not having been masked, will cascade through to be inherited for use by the inner element's Cordscript expressions.

For precise information pertaining to the Cordscript instructions and methods that may be used in the construction of these attribute action expressions, please refer to the corresponding section of the Cords Reference Manual [4].

Element Completion

On completion of the preceding stages the OCCI category instance, identified by the resolved category instance identifier, will be updated to reflect any changes that may have occurred during the different stages of processing. Control will be transferred to the processing of the parent element in the case of nested processing.

Example

A complete example of a fictive CORDS document schema will now be described in order to show the way in which the elements and attributes will be used by the Parser.

Example Schema

Here we have a simple yet exhaustive example of a trivial CORDS document schema.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:annotation>
		<xsd:documentation xml:lang="en">
			An example of a CORDS Schema
		</xsd:documentation>
	</xsd:annotation>
	<xsd:element name="example" type="cordsExample"/>
	<xsd:complexType name="cordsExample">
		<xsd:attribute name="id" type="xsd:string"/>
		<xsd:attribute name="name" type="xsd:string"/>
		<xsd:attribute name="status" type="xsd:integer"/>
		<xsd:attribute name=”xmlns” type=”xsd:string”/>
		<xsd:attribute
			name="configuration"
			type="xsd:string"
			default=”Cordscript: configuration.new(name.value);”
			/>
		<xsd:attribute
			name="cordscript"
			type="xsd:string"
			default=”plan.new(id.value,name.value);/>
	</xsd:complexType>
</xsd:schema>
```

In the above example of a schema document for the element “*example*” the final two attributes with their default values are of particular interest as they demonstrate the two ways in which Cordscript processing can be defined for default invocation by the ACCORDS parser during the processing of a document.

In the first case of the attribute named “configuration” the default value of the attribute will be added if a configuration attribute is not present. The associated Cordscript expression will be invoked at the end of processing for the element and will result in the creation of a configuration category instance of the same name as the example element.

The second case may be seen in the above definition of the “Cordscript” attribute. This will be appended to the element attribute list, the expression provided by its associated value will be invoked and then the attribute will be removed.

Example Document

An example document, as described by the preceding schema, now follows.

```
<?xml version=”1.0”?>
<example name=”test” xmlns=”file:///example.xsd”/>
```

This simple document demonstrates the way in which a named element will be completed by the default valued attributes of its associated schema definition.

Resulting OCCI Graph

The OCCI instance graph resulting from the processing of the preceding CORDS document is shown in the figure below.

Example

Name=test
Xmlns="file:///example.xsd"
Configuration="efa778e4-4456-edfa-56fe-456edf778ad4"

Conclusion

This version of the ACCORDS Parser is now ready for use for the parsing and processing of CORDS documents described by XSD schemas, other than that of the standard "manifest.xsd". New schemas can be written for the description and representation of third party distributed provisioning systems. This technique opens the door for the description of services offered not only as platform as a service (PAAS) but also as software as a service (SAAS) and business process as a services (BPAAS).

These new document schemas will define new categories of service as required to meet their particular needs. The provision of the appropriate category instance management services is all that is necessary for the integration of these new services for use from within the base ACCORDS platform framework.

Finally it can be envisaged that, within the scope of the object oriented nature of the second version of CORDS, where complex nodes may be described by external types, the service provisioning system may be extended such that it allows for the federation, aggregation and integration of heterogeneous service types.

References

1. Cords Manifest Schema:
   http://www.compatibleone.fr/schemes/manifest.xsd

2. Cords Type Definition Schema:
   http://www.compatibleone.fr/schemes/cordstypes.xsd

3. Cords Category Import Schema:
   http://www.compatibleone.fr/schemes/cords.xsd

4. Cords Reference Manual:
   http://www.compatibleone.fr/occi/publisher/CordsReferenceManualV2.08.pdf

5. OCCI Core Model Specification: http://www.ogf.org/documents/GFD.183.pdf

6. OCCI Cloud Infrastructure Specification : http://www.ogf.org/documents/GFD.184.pdf

7. OCCI HTTP Binding Specification : http://www.ogf.org/documents/GFD.185.pdf