DELI (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introduction

In order for a web server to provide optimized content to different clients it requires a description of the capabilities of the client. Two new compatible standards have been created for describing delivery context based on the Resource Description Framework (RDF): Composite Capabilities / Preferences Profile (CC/PP) created by the W3C and User Agent Profile (UAProf) created by the WAP Forum. These standards allow the efficient transmission of delivery context information to the server even via low bandwidth wireless networks. Instead of sending an entire profile with every request, a client only sends a reference to a profile, stored on a third device known as a profile repository, along with a list of differences specific to this particular client. The process of reassembling the final profile from the profile references and differences is known as profile resolution.

HP Labs has produced an open-source library called DELI that allows Java servlets to resolve HTTP requests containing CC/PP or UAProf information and query the resolved profile. This document describes how DELI may be used within Apache Cocoon. For more information on the DELI library please refer to the DELI web-site. DELI currently uses Jena, an RDF Framework developed at HP Labs. For more details of Jena see Brian McBride's paper and the HP Labs Semantic Web activity homepage.

2. CC/PP

CC/PP is described in CC/PP Structure and Vocabularies, CC/PP Requirements and Architecture and CC/PP Terminology and Abbreviations. A CC/PP profile is broadly constructed as a two level hierarchy: a profile has a number of components and each component has a number of attributes. A protocol for transmitting CC/PP profiles has been proposed but is based on an experimental variant of HTTP known as HTTP-ex so is not compatible with existing servers. Therefore DELI uses the W-HTTP protocol proposed by UAProf. This has identical functionality to the CC/PP protocol based on HTTP-ex, but is compatible with HTTP/1.1.

3. UAProf

The UAProf specification is based on the CC/PP specification. Like CC/PP, a UAProf profile is a two level hierarchy composed of components and attributes. Unlike CC/PP, the UAProf specification also proposes a vocabulary - a specific set of components and attributes - to describe the next generation of WAP phones. The specification also describes two protocols for transmitting the profile >from the client to the server. Currently DELI only supports the W-HTTP protocol.

Profiles using the UAProf vocabulary consist of six components: HardwarePlatform, SoftwarePlatform, NetworkCharacteristics, BrowserUA, WapCharacteristics and PushCharacteristics. These components contain attributes. In DELI each attribute has a distinct name and has an associated collection type, attribute type and resolution rule. In UAProf there are three collection types:

- Simple contains a single value e.g. ColorCapable in HardwarePlatform.
- Bag contains multiple unordered values e.g. BluetoothProfile in the HardwarePlatform component.
- Seq contains multiple ordered values e.g. Ccpp-AcceptLanguage in the SoftwarePlatform component.

In addition attributes can have one of four attribute types:

- String e.g. BrowserName in BrowserUA.
- Boolean e.g. ColorCapable in HardwarePlatform.
- Number is a positive integer e.g. BitsPerPixel in HardwarePlatform.
- Dimension is a pair of positive integers e.g. ScreenSize in HardwarePlatform.

Finally attributes are associated with a resolution rule:

- Locked indicates the final value of an attribute is the first occurrence of the attribute outside the default description block.
- Override indicates the final value of an attribute is the last occurrence of the attribute outside the default description block.
- Append indicates the final value of the attribute is the list of all occurrences of the attribute outside the default description block.

The UAProf vocabulary is described using the file uaprofspec.xml. This describes the attribute name, component, collectionType, attributeType and resolution rule of each component. The vocabulary description file has the following format:

<?xml version="1.0"?> <vocabspec> <attribute> <name>CcppAccept</name> <component>SoftwarePlatform</component> <collectionType>Bag</collectionType> <attributeType>Literal</attributeType> <resolution>Append</resolution> </attribute> </vocabspec>

DELI can also read vocabularies described using RDF schemas. The WAP Forum have published two such schemas to describe the two versions of UAProf currently in use. However RDF Schema does not provide an easy way of describing attributeType or resolution rule. Therefore the UAProf schemas store this information in the comments field for each attribute. Therefore DELI also parses the comments fields to create the vocabulary. This is not an ideal solution so hopefully a more robust way of representing this information will be used in later versions of UAProf.

4. W-HTTP Protocol

An example W-HTTP request using this protocol is shown below:

GET /ccpp/html/ HTTP/1.1 Host: localhost x-wap-profile:"http://127.0.0.1:8080/ccpp/profiles/test09defaults.rdf", "1-Rb0sq/nuUFQU75vAjKyiHw==" x-wap-profile-diff:1;<?xml version="1.0"?> <rdf:RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#"> <rdf:Description rdf:ID="MyDeviceProfile"> <prf:component> <rdf:Description rdf:ID="HardwarePlatform"> <rdf:type rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#HardwarePlatform"/> <prf:BitsPerPixel>16</prf:BitsPerPixel> </rdf:Description> </prf:component> </rdf:Description> </rdf:RDF>

The first two lines describe the resource that is being requested by the client, http://localhost/ccpp/html, and the method being used to make the request, GET, and the protocol being used HTTP/1.1. The remaining lines of the request describe the device delivery context. This is specified using a profile reference and a profile-diff. The profile is referenced via the x-wap-profile line and has the URI

http://127.0.0.1:8080/ccpp/profiles/test09defaults.rdf.

After the profile reference, there is a value

1-Rb0sq/nuUFQU75vAjKyiHw==

known as a profile-diff digest. The first part of the profile-diff-digest, 1-, is the profile-diff sequence number. This is used to indicate the order of the profile-diffs and to indicate which profile-diff the profile-diff digest refers to. The remainder of the profile-diff digest is generated by applying the MD5 message digest algorithm and Base64 algorithm to the corresponding profile-diff. The MD5 algorithm takes as input a message of arbitary length and produces as output a 128-bit "fingerprint" or "message-digest" of the input. The Base64 algorithm takes as input arbitary binary data and produces as output printable encoding data.

After the profile-diff digest, the next line contains the x-wap-profile-diff. This request header field also has a profile-diff sequence number which indicates that this profile-diff corresponds to the previous profile-diff-digest. The profile-diff itself consists of the XML fragment which spans the remainder of the request. Multi-line request header fields are permitted by the HTTP/1.1 specification as long as each subsequent line starts with either a tab character or a whitespace.

When the server receives a HTTP request with UAProf request headers, it has to perform profile resolution i.e. retrieve the referenced profile(s) and any further profiles referenced via default blocks. It then has to merge these profiles and the profile-diffs while applying the UAProf resolution rules.

5. Configuring DELI

In order to use DELI, you need to enable the DELI component. In addition, you may want to configure DELI using deliConfig.xml, the DELI configuration file or legacyDevices.xml, the DELI legacy device support file.

5.1. Cocoon.xconf

In order to use DELI you need to configure Deli and specify the configuration file. You can either do this in deli.xconf in which case you will need to rebuild Cocoon or change the deployed cocoon.xconf in the web-server WEB-INF directory:

```
<deli class="org.apache.cocoon.components.deli.DeliImpl"> <parameter
name="deli-config-file" value="deli/config/deliConfig.xml"/> </deli>
```

5.2. Sitemap.xmap

In order to make profile information available to your stylesheet then you need to add <map:parameter name="use-deli" value="true"/> to the match that specifies your stylesheet in sitemap.xmap. Here is the match used for the deli test stylesheet:

```
<map:match pattern="deli.html"> <map:generate src="docs/samples/hello-page.xml"/>
<map:transform src="stylesheets/deli_test.xsl" type="deli"> <map:parameter name="use-deli"
value="true"/> </map:transform> <map:serialize type="html"/> </map:match>
```

5.3. Main Configuration File

DELI also has its own configuration files that are found in the resources\deli\config directory. The most important one, deliConfig.xml is used to configure the main DELI options:

```
<?xml version="1.0"?> <deli>
<localProfilesFile>WEB-INF/deli/config/localProfiles.xml</localProfilesFile>
<localProfilesPath>WEB-INF/deli/legacyProfiles</localProfilesPath>
<useLocalProfilesIfNoCCPP>true</useLocalProfilesIfNoCCPP>
<useLocalProfilesInAdditionToCCPP>false</useLocalProfilesInAdditionToCCPP>
```

<debug>false</debug> <printDefaults>true</printDefaults>
<printProfileBeforeMerge>false</printProfileBeforeMerge>
<processUndefinedAttributes>true</processUndefinedAttributes>
<useCapabilityClasses>false</useCapabilityClasses>
<capabilityClassFile>WEB-INF/deli/config/capClass.xml</capabilityClassFile>
<namespaceConfigFile>WEB-INF/deli/config/namespaceConfig.xml</namespaceConfigFile>
<!-- note that the default uri is also hardcoded -->
<rdfsUri>http://www.w3.org/1999/PR-rdf-schema-19990303#</rdfsUri>
<rdfsUri>http://www.w3.org/TR/PR-rdf-schema#</rdfsUri> </deli>

This file can contain a number of configuration directives described in in the following sections:

5.3.1. Caching options

The caching options control how the server caches referenced profiles. DELI can either cache profiles indefinitely or update stale profiles after a set interval. It is also possible to configure the maximum size of the profile cache.

Element Name	Default Value	Description
maxCachedProfileLifetime	24 hours	The maximum lifetime of a cached profile in hours.
maxCacheSize	100	The maximum number of profiles in the profile cache.
refreshStaleProfiles	false	Do we refresh cached profiles after the maximum lifetime has expired?

5.3.2. Debugging options

The debugging options are used to control the information that DELI prints to the Servlet engine console.

Element Name	Default Value	Description
debug	true	Is the automatic debug log information turned on?
printDefaults	true	Print both default and override values of attributes for debugging purposes?
printProfileBeforeMerge	false	Print the profile before merging for debugging purposes?

5.3.3. Legacy device options

As already mentioned DELI can support legacy devices by recognising the user-agent string supplied by a client and mapping it on to a profile. In order to use this facility it is necessary to supply an XML file that contains information about legacy device user-agent strings and the corresponding profile URLs. The format for the legacy device file is described in a subsequent section.

Element Name	Default Value	Description
useLocalProfilesIfNoCCPP	true	Use the legacy device database if devices send no

		CC/PP information?
localProfilesFile	legacyDevice.xml	The file containing the legacy device database.

5.3.4. Protocol options

It is possible to switch on whitespace normalisation in profile-diffs prior to calculating the profile-diff-digest so that additional whitespaces added by the proxy are ignored. To use this option clients must also support whitespace normalisation.

Element Name	Default Value	Description
normaliseWhitespaceInProfileDif	true	Is whitespace normalisation of the profile-diff prior to calculating the profile-diff-digest turned on?

5.3.5. Vocabulary options

DELI has a number of vocabulary options. Firstly it is possible to configure the vocabulary using an XML file, or if you are using UAProf using a UAProf RDF Schema. You can find examples of both approaches in this distribution. Secondly it is possible to specify the URI to be used for the RDF namespace. Thirdly it is possible to set the string used to represent components and defaults in the vocabulary. This is important because the two standards currently use different cases for the first letter of default elements (CC/PP uses "default" whereas UAProf uses "Default").

Element Name	Default Value	Description
vocabularyFile	uaprofspec.xml	The file containing the vocabulary specification.
schemaVocabularyFile	ccppschema-20000405.rdfs	The file containing the vocabulary specification as an UAProf RDF Schema. Use the attribute namespace to configure which namespace the schema corresponds to.
rdfUri	http://www.w3.org/1999/02/22-rd	The namespace used for RDF constructs.
componentProperty	component	The name for components.
defaultProperty	Default	The name for defaults

5.4. Configuring Legacy Devices

It is easy to configure DELI to recognise legacy devices via user-agent strings. A user-agent string is a string sent by the client to the server as part of the HTTP request. It allows different browsers to be identified. The legacy device configuration file maps user-agent strings on to profile either on the local filestore or on a profile repository. By default this is done in the legacyDevice.xml file which has the following format:

<?xml version="1.0" encoding="UTF-8"?> <devices> <!-- Alcatel --> <device> <ua value="Alcatel-BF4/2.0" profile="Alcatel_OT512.rdf"/> </device> <device> <ua value="Alcatel-BE4/1.0" profile="Alcatel_OT301.rdf"/> </device> </devices>

Where vale is a device unique string found in the user-agent string of the device and profile is either a local file or a URL for the appropriate legacy profile.

6. Writing CC/PP and UAProf aware stylesheets

Once you have got DELI running on Cocoon, the next step in creating a CC/PP aware site is to create some stylesheets that use profile information. DELI makes CC/PP or UAProf attributes available to XSLT stylesheets as parameters. In the process of doing this, DELI 'flattens' the profiles by omitting the component information. Hence to retrieve the CcppAccept attribute you use the XPath deli-capabilities/browser/CcppAccept whereas to retrieve the ScreenSize attribute you use the XPath deli-capabilities/browser/ScreenSize. In addition where attributes contain multiple values e.g. Bags or Sequences those values are separated using <li> elements. Hence to retrieve the individual elements you use the XPath deli-capabilities/browser/CcppAccept/li. The following code demonstrates how to retrieve both a simple and a complex attribute. For more complex examples see the deli_test.xsl stylesheet with Cocoon.

```
<?xml version="1.0"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> <xsl:param name="deli-capabilities"/> <xsl:template match="/"> <xsl:if test="contains($deli-capabilities/browser/CcppAccept/li,'wml')"> <xsl:call-template name="wmldevice"/> </xsl:if> <xsl:if test="not(contains($deli-capabilities/browser/CcppAccept/li,'wml'))"> <xsl:call-template name="htmldevice"/> </xsl:if> </xsl:template> <xsl:template name="wmldevice"> <wml> <card id="init" newcontext="true"> <p> <xsl:call-template name="capabilities"/> </p> </card> </wml> </xsl:template> <xsl:template name="htmldevice"> <html> <head> <title>Test Page for DELI in Cocoon</title> </head> <body> <xsl:call-template name="capabilities"/> </body> </html> </xsl:template> <xsl:template name="capabilities"> <xsl:if test="$deli-capabilities/browser/ImageCapable"> ImageCapable: <xsl:value-of select="$deli-capabilities/browser/ImageCapable"/> <br/> </xsl:if> <xsl:if test="$deli-capabilities/browser/CcppAccept"> CcppAccept: <xsl:for-each select="$deli-capabilities/browser/CcppAccept/li"> <xsl:value-of select="."/>, </xsl:for-each> <br/> </xsl:if> </xsl:template> </xsl:stylesheet>
```

7. Incorporating DELI into other Cocoon components

If you want to use DELI in other Cocoon components, DELI needs to go through an initialization phase in order to read in configuration files. This occurs in initialization() in DeliImpl.java in the Cocoon DELI component in this line:

```
Workspace.getInstance().configure(this.servletContext, this.deliConfig);
```

i.e. it constructs a DELI workspace object. In addition, DeliImpl.java has a number of methods to get profile information, but the one which is of most use is getProfile(). For examples of how to call DELI, see DeliTransformer.java - here are the relevant pieces.

First import DELI:

```
import org.apache.cocoon.components.deli.Deli;
```

Then in service(), initialize DELI:

```
if (this.manager.hasComponent(Deli.ROLE)) { if (this.getLogger().isDebugEnabled()) { getLogger().debug("Looking up " + Deli.ROLE); } this.deli = (Deli) this.manager.lookup(Deli.ROLE); } else { if (this.getLogger().isDebugEnabled()) { getLogger().debug("Deli is not available"); } }
```

Then in getLogicSheetParameters, call DELI to get a profile and convert it into a DOM tree to give to

XSLT as a parameter:

```
if (this.deli != null && this._useDeli) { try { Request request = ObjectModelHelper.getRequest(objectModel); if (map == null) { map = new HashMap(); } org.w3c.dom.Document deliCapabilities = this.deli.getUACapabilities(request); map.put("deli-capabilities", deliCapabilities); } catch (Exception e) { getLogger().error("Error setting DELI info", e); } }
```

However, you don't want to pipe DELI to XSLT so here is some more example code showing how to use DELI in Cocoon if you want to use the DELI API directly:

Import the deli component and the DELI API:

```
import org.apache.cocoon.components.deli.Deli; import com.hp.hpl.deli.Profile; import com.hp.hpl.deli.ProfileAttribute;
```

Add code to initialize the service method as before

```
public void service(ServiceManager manager) throws ServiceException { try { deli = (Deli)manager.lookup(Deli.ROLE); } catch(ServiceException ce) { logger.log(Level.ERROR, "Cannot get ref to Deli", ce); ce.printStackTrace(); } }
```

Some example code that shows how to extract WmlDeckSize from a DELI profile

```
try { Profile theProfile = deli.getProfile(request); if(theProfile != null) { ProfileAttribute attrib = theProfile.getAttribute("WmlDeckSize"); if(attrib != null) { Vector temp = (Vector)attrib.get(); if(temp != null) { deckSizeString = temp.get(0).toString(); System.out.println("Determined WmlDeckSize from DELI: " + deckSizeString); } } } if(deckSizeString == null) { deckSizeString = "1100"; System.out.println("Using fallback WmlDeckSize of " + deckSizeString); } } catch(Exception ex) { System.out.println(ex.toString()); ex.printStackTrace(); } }
```

8. More information ?

For more information on the DELI library please refer to the DELI web-site.

1. Comments

add your comments