Luminis III.2 and the uPortal iPerson Object

Providing External User Information to Channels

Document Revision 1.2.1

Creation Date - unknown
Last Modification – 12/10/2004

Jonathan P. Wheat
Programmer / Analyst
Messiah College
1 College Avenue
Grantham, PA 17027

jwheat@messiah.edu

717.766.2511 x3385

Revision history
v1.2 added LDAP information 12/9/2004
v1.2 added new appendix – 12/9/2004
v1.2.1 added cache settings – 12/10/2004

]: My Introduction :[

What the …
To explain quickly, uPortal (the actual free version) has a mechanism to send user information via the WebProxy channel. As with most uPortal specific items I've played with, they don't seem to integrate or simply work with Luminis. This outlines how to get that uPortal object (iPerson object) configured to work and send user information via the Luminis WebProxy channel.

The Quest …
I needed a way to get the ***uid*** (userid number) and ***userName*** from Luminis and pass it to an external web application. After doing research and downloading faqs from various sources, I stumbled upon this document recently re-posted on the Luminis help site (http://campuspipeline.custhelp.com). The document was originally named 'Providing External User Information to Channels' and is answer id : 1058

While this first attempt is not true CPIP, we want eventually to grab the username and encrypted password from LDAP and send them to an external custom application using the WebProxy channel. This document (with my modifications) outlines how I configured my system to fetch the ***uid*** and ***userName*** for a current session.

Below my edited document with my notes about my system. Your mileage may vary, but have a gut feeling that this will work for you as well.

I've placed all of the important information you need to know to configure the system in boxes, as not to be confused with the original text of the document which is useful to read to see how everything interacts. I encourage you to read through the entire document, but I also realize that you may just want to get started like I did.

Not sure if Campus Pipeline / Sungard SCT / Luminis people will care that I edited up this, but hopefully it'll make more sense to you now.

Introduction

uPortal provides a mechanism whereby additional user information can be made accessible to channels. This mechanism presents such information to channels via the IPerson object passed via the ChannelStaticData object. Sources of this additional information are specified in an xml file named PersonDirs.xml located in $CP_ROOT/webapps/luminis/WEB-INF/config. This file also allows a mapping from datasource data element names to alias names used to access those values from the IPerson object's getAttribute() method.

An example of a channel that uses such information is CWebProxy. We will cover its use of such attributes and show how it adds them to into a query string appended to the specified publish-time URL of the channel to enable an external system to customize the returned information for this user. We will also give two working examples showing how to acquire information from an LDAP and JDBC data source.

The PersonDirs.xml file is an XML file and as such must be well-formed XML.

Appendix B
I've posted the default PersonDirs.xml (JDBC) file at the end of this document to serve as an example. To Get started with the configuration, skip down to Appendix A : PersonDirs.xml (JDBC)

Appendix C
I've posted the default PersonDirs.xml (LDAP) file at the end of this document to serve as an example. To Get started with the configuration, skip down to Appendix B : PersonDirs.xml (LDAP)

It consists of an outermost element, <PersonDirs>, that can contain from one to many child <PersonDirInfo> elements. Each PersonDirInfo element defines one data source from which additional attributes will be obtained. Each PersonDirInfo element should define child elements collectively used to access only one data source. The combination of several fields tells the system which data source type will be used. Data can be obtained from either LDAP, JDBC, or container-managed JDBC datasources.

Specifying a Container-Managed JDBC Data Souce

To use a container-managed JDBC data source specify the following elements:

<res-ref-name>
<uidquery>

The existence of the res-ref-name element indicates that a container-managed data source is to be used. The system will obtain a reference to the a "java:comp/env" JNDI context and search within that context for an instance of javax.sql.DataSource bound to "jdbc/<res-ref-name>" where <res-ref-name> refers to the contents of the res-ref-name element.

The uidquery element indicates the SQL query that should be executed to obtain the additional information. Such a query can contain only one placeholder question mark. In place of this question mark the current uPortal user's login ID will be placed prior to executing the query.

For example, if I was going to use a table, EXT_SYS_MAP, having two columns, UPORTAL_ID and EXT_SYS_USER, from which I wished to obtain some external

system's user ID and make it available to channels I would specify a uidquery element like:

<uidquery>SELECT EXT_SYS_USER FROM EXT_SYS_MAP WHERE UPORTAL_ID=?</uidquery>

Specifying a JDBC Data Source

To use a non-container-managed JDBC data source you need to specify the following elements:

<driver>
<url>
<logonid>
<logonpassword>
<uidquery>

The driver element must contain the class that should be instantiated to provide connections to the relational database. Refer to the documentation for the JDBC driver provided to access your database. This class should be available in the classpath for the Luminis server. The jar or zip file containing the driver for your database should therefore be available in the $CP_ROOT/webapps/luminis/WEB-INF/lib directory.

The url element must contain a properly formatted url as dictated by the driver specified in the driver element. This will typically include the database server and port and possibly in instance identifier. Refer to your database's JDBC driver documentation for the proper format for your target database.

The longonid and logonpassword elements must contain id and password respectively of a user for the target database. The system will connect as this user to the database to retrieve the additional information for users.

The uidquery must contain the SQL query used to acquire the additional information. See the container-managed description of this field for the expected format.

Specifying an LDAP Data Source

To use an LDAP data source you must specify the following elements:

<url>
<logonid>
<logonpassword>
<uidquery>
<usercontext>

The url element must contain a URI that conforms to RFC2255, The LDAP URL Format. JNDI is used to access the LDAP repository so the distinguished name specified becomes the initial context of the search to be performed. This means that the context

specified in the usercontext element must be relative to this initial context. See notes for that element below.

The logonid must contain the DN of a user on behalf of which the system will log into LDAP and perform the query specified in the uidquery element against the subtree of the context specified in the usercontext element.

The logonpassword must contain the password of the user specified with logonid.

The uidquery element must contain a search filter that conforms to RFC 2254, The String Representation of LDAP Search Filters. It can also contain the variable identifier string “{0}” which will be replaced with the uPortal login ID of the current user. This variable can be placed wherever an *attr*, *matchingrule*, or *value* production appear in the search filter grammar in section 4 of RFC 2254. The search is performed with sub-tree scope in the named context specified in the usercontext element and retrieves all available attributes for the object that results. If an attribute is multi-valued only one value is currently returned.

The usercontext element must contain relative distinguished name that is the starting point of the search. This must be relative to the initial context specified in the url element. The initial context is appended to the usercontext value to form the distinguished name used in the LDAP search sent to the server.

Mapping Attributes

Once a data source has been activated you then need to indicate which pieces of data being brought back from that data source should be placed in the IPerson object and what their key should be to retrieve them out of the IPerson object. This is accomplished through an attributes element in each PersonDirInfo element. Each must have its own set of attributes defined.

In the case of a JDBC data source’s SQL query the content of the name element for each attribute declaration should be the name of a column returned in the query’s result set. The value used must match the case of the column name returned by the RDBM in the result set. This is typically in all upper case. Check with your data base documentation and information on the JDBC driver being used.

For an LDAP data source the content of the name element for each attribute declaration should be the name of an attribute returned for the object. For LDAP attribute names are case insensitive so case does not matter. Note that with multi-valued attributes only one value will be returned but it is currently undefined which of the multiple values it will be. This will be changed in the future to pass all values through via an array of java.lang.String objects.

The key by which the attribute should be extracted from the IPerson object is specified as the value of the alias element for each attribute declaration. So if I queried an LDAP

object that had a cn attribute and I wanted to acquire that attribute in a channel via a key of “common name” I would specify an attribute declaration like:

```
<attribute>
  <name>cn</name>
  <alias>common name</alias>
</attribute>
```

Similarly, if I had a JDBC data source that executed a query like:

SELECT EXT_SYS_ID FROM SOME_TABLE WHERE UP_ID=?

and I wished to acquire that value from the IPerson object via a key of “externalId” I would specify an attribute declaration like:

```
<attribute>
  <name>EXT_SYS_ID</name>
  <alias>externalId</alias>
</attribute>
```

If attribute declarations are included that have an empty name element those declarations are ignored.

Examples

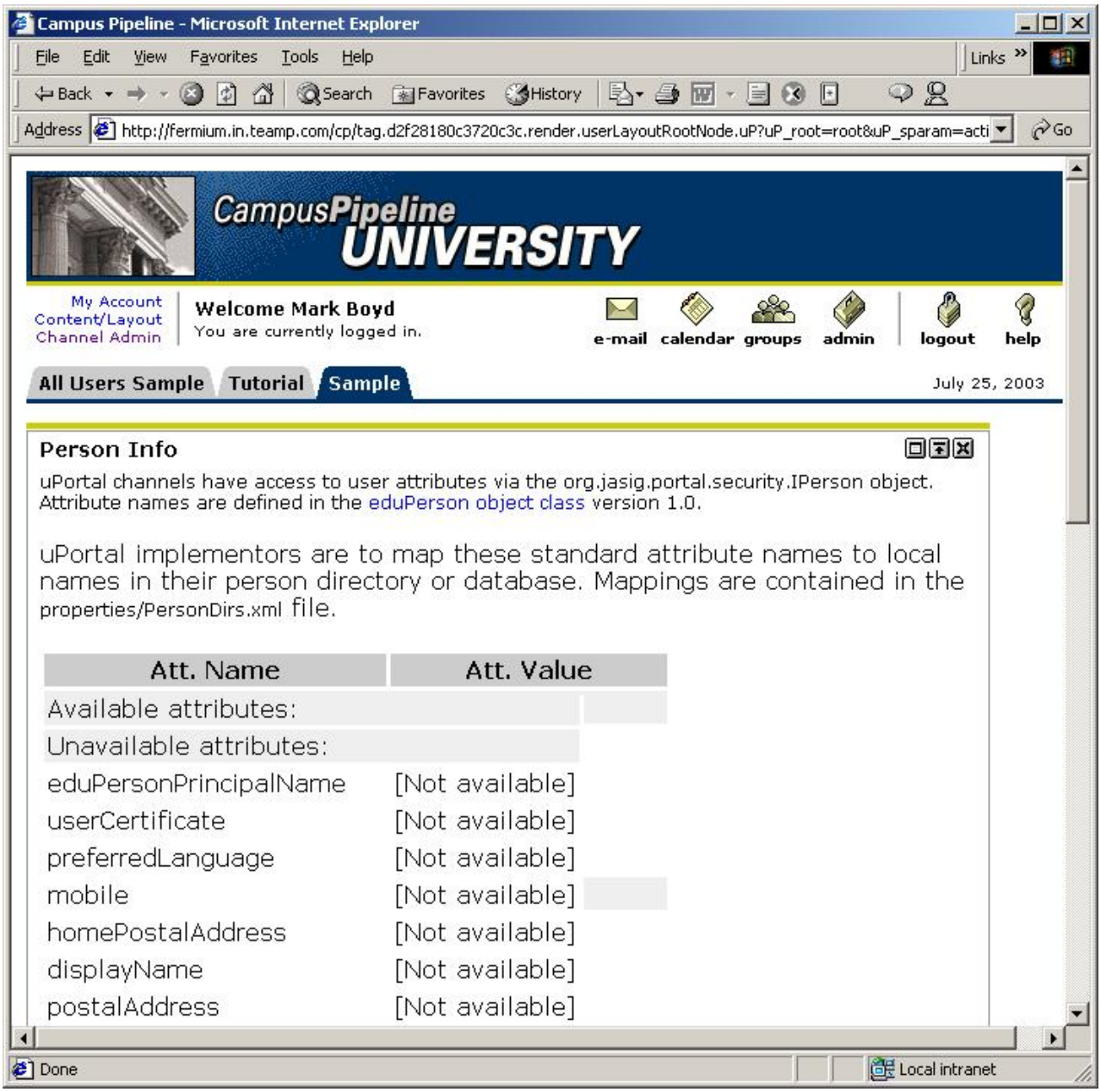
To illustrate the use of PersonDirs.xml we will present two examples. One will show how to set up a JDBC source and the other will show how to set up an LDAP source. To view the affects of each one we will make use of a channel that presents all information residing in the IPerson object passed to it and hence to all channels in a user’s layout.

Do This
Create this channel as specified below. It will help you see when the system is configured properly.

So first publish and subscribe to the channel usnig the following parameters:

Publish-Time Parameter	**Value**
Channel Type	Custom Java
Channel Title, Name, Functional Name, and Description	Person Attributes
Channel Timeout	3000
Channel Secure	No
Channel Class	org.jasig.portal.channels.CPersonAttributes
Channel Controls	None
Selected Categories	Applications
Selected Groups	Public

When you log in as a user before activating any PersonDirs.xml data source the channel will look as shown below. Note that no attributes are shown as available. Also note that the channel indicates that the PersonDirs.xml file is located in a properties directory instead of the location in the $CP_ROOT/webapps/luminis/WEB-INF/config directory. This is because this channel is provided as a sample in the uPortal codes base and is referring to deployment in uPortal not in Luminis.



Example using JDBC

To pull in some piece of data unique to each person and based on the login ID of that person we will use a table used by Luminis that is part of the uPortal data store. This table contains among other things an integer ID for each login ID. The data base used to store relation information in Luminis is specified in the $CP_ROOT/webapps/luminis/WEB-INF/config/rdbm.properties file. We can use the properties defined in this file to activate our example JDBC data source.

Example rdbm.properties file

```
# rdbm.properties

jdbcDriver=com.campuspipeline.rdb.conn.ConnectionPoolDriver
jdbcUser=uportal
jdbcPassword= wh4t3V3rPa$$w0rD

jdbcUrl=jdbc:cp:uPortalPooled;driver=oracle.jdbc.driver.OracleDriver;url=jdbc:oracle:t
hin:@db.messiah.edu:1521:PRTL
```

To get this simple example working first uncomment the following lines in PersonDirs.xml.:

Uncomment lines

To do that, remove the <!- - and - -> from the preceding and trailing lines

<driver>the fully qualified class of the driver</driver>
<url>the jdbc url for the driver</url>
<logonid>the db logon id</logonid>
<logonpassword>the db password</logonpassword>
<uidquery>SELECT USER_ID FROM UP_USER WHERE USER_NAME=?</uidquery>

Then paste in the values found in rdbm.properties. If my rdbm.properties file contained the following property definitions:

then the PersonDirs.xml content for the above elements would be:

Modified PersonDirs.xml

```
<driver>com.campuspipeline.rdb.conn.ConnectionPoolDriver</driver>

<url>jdbc:cp:uPortalPooled;driver=oracle.jdbc.driver.OracleDriver;url=jdbc:oracle:thin
:@db.messiah.edu:1521:PRTL</url>

<logonid>uportal</logonid>
<logonpassword> wh4t3V3rPa$$w0rD </logonpassword>
<uidquery>SELECT USER_ID FROM UP_USER WHERE USER_NAME=?</uidquery>
```

Once you have configured the data source you then need to map the returned attribute, USER_ID in this case, to the key by which it will be made available in the IPerson object.

Getting values via the SQL statement

Basically you're using the SQL string in the <uidquery> section above. The default is *SELECT USER_ID FROM …*

So all you're going to get is the user_id (iud) which is the user number, not the login name. So modify the line above in the PersonDirs.xml file to this –

```
<uidquery>SELECT USER_ID,USER_NAME FROM UP_USER WHERE USER_NAME=?</uidquery>
```

Now you can use the user_name as well as the user_id.

Set up the following attribute declaration:

Errata in original document

In the original document posted to the custhelp site, there is a mistake that renders the example useless. The original document tells you to this –

> Set up the following attribute declaration:
>
> ```
> <attribute>
>   <name>uid</name>
>   <alias>uid</alias>
> </attribute>
> ```

Below is the code that actually works, which oddly enough is the original XML attribute code in the default PersonDirs.xml code, so leave it (unless for some reason it does not look like the code below)

What works (original xml code)

```
<attribute>
  <name>USER_ID</name>
  <alias>uid</alias>
</attribute>
```

Since we also added USER_NAME to the SQL query, we need to set up an attribute for that as well.

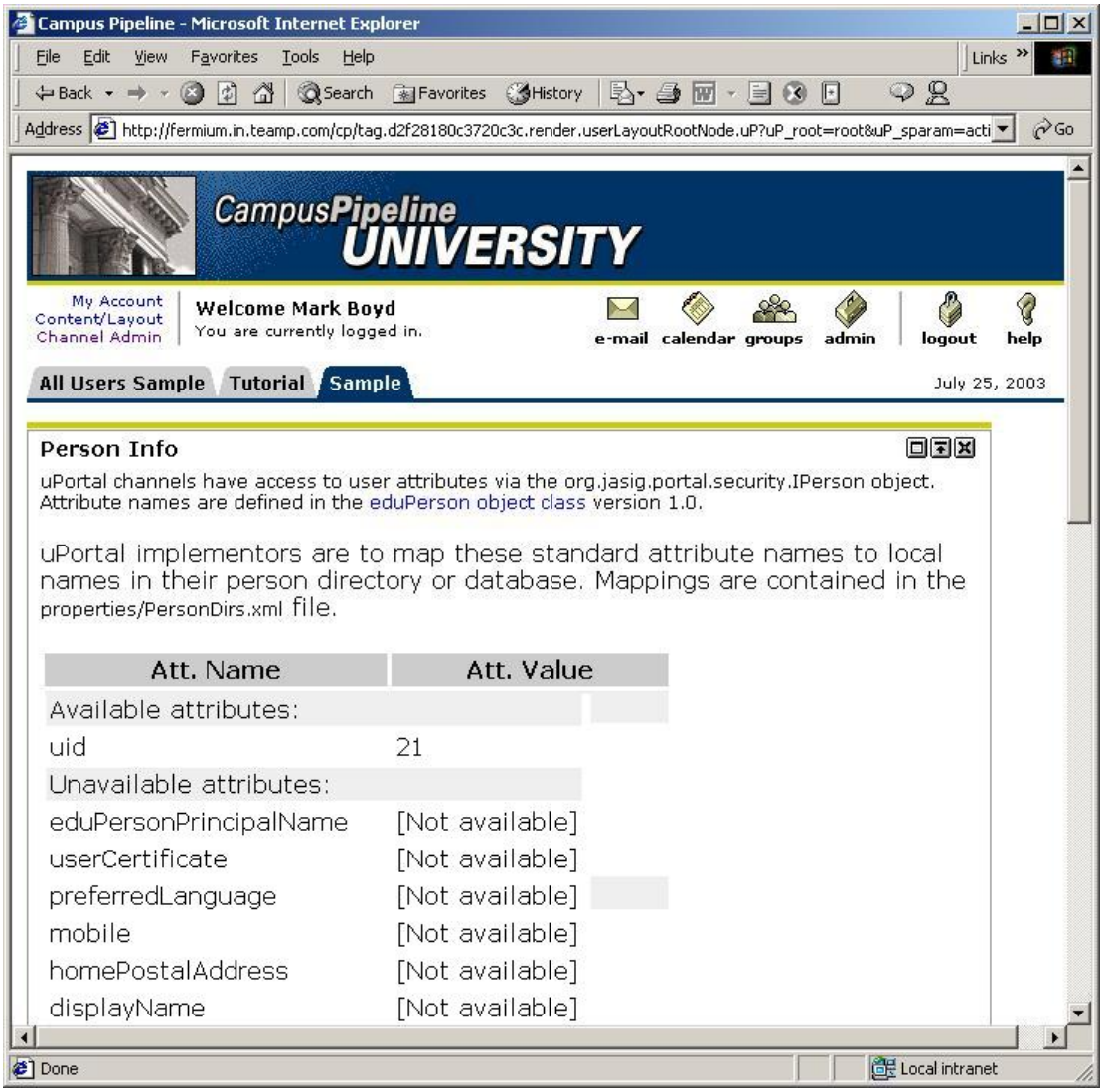
Also Added to configure the username

```
<attribute>
  <name>USER_NAME</name>
  <alias>userName</alias>
</attribute>
```

And Modified the SQL string to be

```
<uidquery>SELECT USER_ID,USER_NAME FROM UP_USER WHERE USER_NAME=?</uidquery>
```

Save these files.
Then stop and start the Luminis server. When you log back in as a user the Person Attributes channel should now look as shown below. Note that if you log in as a user who has not logged in to Luminis before there will be no USER_ID found until you log in a second time. This is because the PersonDirs.xml processing takes place before the system identifies this as a new user and sets up their uPortal information. This is only an artifact of the UP_USER table being used for an example.



LDAP

We have recently implemented the LDAP portion of this and commented out the jdbc calls. LDAP uses the Luminis LDAP to fetch user information. Because of this approach, there is much more information that can be gathered.

Example Using LDAP

To show an example of additional information being brought into the IPerson object from an LDAP repository we will use the Luminis LDAP server. The PersonDirs.xml processing uses simple authentication and connect to the LDAP server. Any user defined in Luminis can be used but will be restricted in which attributes they can view for other users. Anonymouse searches of the Luminis LDAP server are not allowed.

If my user ID were “mpeterson” and my passwcd ~vim Pord to log into Luminis were “uguess1t” and the Luminis LDAP server was running on port 389 of host luminis.teamp.com then I would activate an LDAP data source with the following declaration.

```
<url>ldap://luminis.teamp.com:389/o=in.teamp.com,o=cp</url>
<logonid>uid=mpeterson,ou=People,o=in.teamp.com, o=cp</logonid>
<logonpassword>uguess1t</logonpassword>
<uidquery>(uid={0})</uidquery>
<usercontext>ou=People</usercontext>
```

User Submission

Greg Marshall submitted his section of the PersonDirs.xml file. He also send this link to an LDAP browser to help get the correct parameters (yours will vary a bit) :

http://www-unix.mcs.anl.gov/~gawor/ldap/

```
<!-- LDAP Properties -->
<url>ldap://luminis1.truman.edu:389/o=truman.edu,o=truman</url>
<logonid>uid=cpadmin,ou=People,o=truman.edu,o=truman</logonid>
<logonpassword>NotTheReal1</logonpassword>
<uidquery>(uid={0})</uidquery>
<usercontext>ou=People</usercontext>
```

NOTE : To get this to work you need to make sure you comment out the lines for the jdbc connection above.

Note that the Luminis LDAP schema holds user information under DNs like:

uid=mpeterson,ou=People,o=in.teamp.com,o=cp

Also remember that the usercontext value is relative to the initial context specified in the url. This means that you could just as easily have specified the following declaration and achieved the same result.

```
<url>ldap://luminis.teamp.com:389/o=cp</url>
<logonid>uid=vgoenka,ou=People,o=in.teamp.com, o=cp</logonid>
<logonpassword>vgoenka</logonpassword>
<uidquery>(uid={0})</uidquery>
<usercontext>ou=People,o=in.teamp.com</usercontext>
```

Attributes

You set these to whatever you want to fetch out of LDAP. The name is the name within LDAP (this is where the browser is useful). The alias is what it is called once it is pulled out, and what you enter in the WebProxy channel to get it to send. So for example this :

```
<attribute>
  <name>uid</name>
  <alias>uid</alias>
</attribute>
```

Obviously pulls the **uid** from the database and calls it **uid**

This however

```
<attribute>
  <name>uid</name>
  <alias>TheUserId</alias>
</attribute>
```

Pulls the **uid** from the database and calls the variable (if you will) **TheUserId**

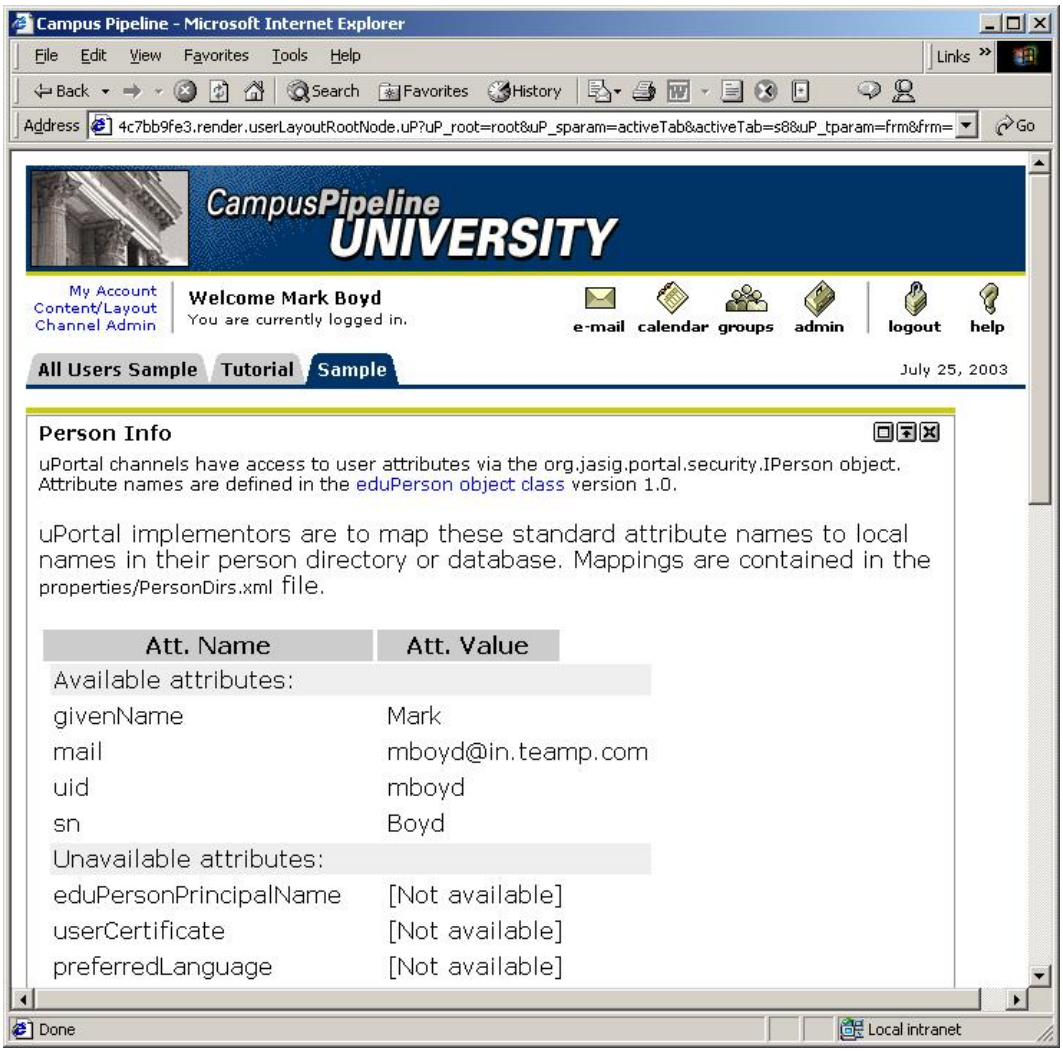
Once we have defined the data source we then need to map the attributes being returned to alias names by which they will be inserted into the IPerson object and by which channels can extract them back out. Define the following attributes:

```
<attribute>
  <name>uid</name>
  <alias>uid</alias>
</attribute>

<attribute>
  <name>cn</name>
  <alias>commonName</alias>
</attribute>

<attribute>
  <name>givenName</name>
  <alias>givenName</alias>
</attribute>

<attribute>
  <name>mail</name>
  <alias>mail</alias>
</attribute>
```

```
<attribute>
  <name>sn</name>
  <alias>sn</alias>
</attribute>
```

Now stop and start the Luminis server.

When you log back in as a user the ***Person Attributes*** channel should now look as shown below.



SAME FROM HERE OUT
All the rest of the document is the same regardless of which one you configure jdbc / LDAP. Make sure you do the examples as explained, and Please feel free to build the web proxy channel with the url on my server, it'll display the entire QueryString for you as well as the userid (number) and username (if you're passing these)

Do This

Do the following example with the static html page first to make sure you have the settings correct. Then continue to my .asp example to prove it is working.

Example Using Web Proxy

The Web Proxy channel is used to incorporate existing web content from other web based applications. It is published with a target URL from which it is to extract content to be inserted into the web page. The Web Proxy code runs on the Luminis server and pulls the information from the external server via that URL and inserts it into the page being served back to the Luminis user. To facilitate passing user identification to that external server Web Proxy can append query parameters including the PersonDirs.xml added attributes and their values to the URL being called to acquire the external content.

This example will not actually show an external service being used. The purpose here is to show how the additional attributes can be sent to the external server as part of the Web Proxy request. To mimic a service I created a simple html page in $CP_ROOT/webapps/luminis named test.html. This file will then be accessible via a URL of http://luminisHost/cp/test.html. I then configured Luminis to use an http proxy when opening any http resource. The contents of the test.html file were:

```
<html>
<body>
<H1>Hi there class.</H1>
</body>
</html>
```

My Notes on this section

First open notepad and paste in the text above. Save it to your \luminis\webapps\luminis\cps directory (not the CP directory referred to above).

Then test it with a browser before you do anything else to make sure you have the url right –

http://portal.messiah.edu/cps/hello.html

Once you get the page to view, continue on.

New WebProxy Channel

I then published a Web Proxy channel with the following parameters. NOTE: IPerson attributes will only be appended to the URL passed to the server if the ***Pass-through Type is set to all, or application, or if the URL contains any query parameters with name*** cw_inChannelLink. (See the Web Proxy documentation for the definition of the pass through feature and html links used in the channel.) Note that only those values that were changed from their default are shown.

My Changes
Since all we grabbed in our JDBC example is userid and username, make the channel with the following settings (these are modified from the example in the original document) If you've used the LDAP example (good for you !) you need to set the value for these to whatever parameters (specified by the alias attribute in the xml file). Here's the LDAP settings provided you want to send all of this to the page : `uid,commonName,givenName,mail,sn`

Here are the WebProxy Channel Settings

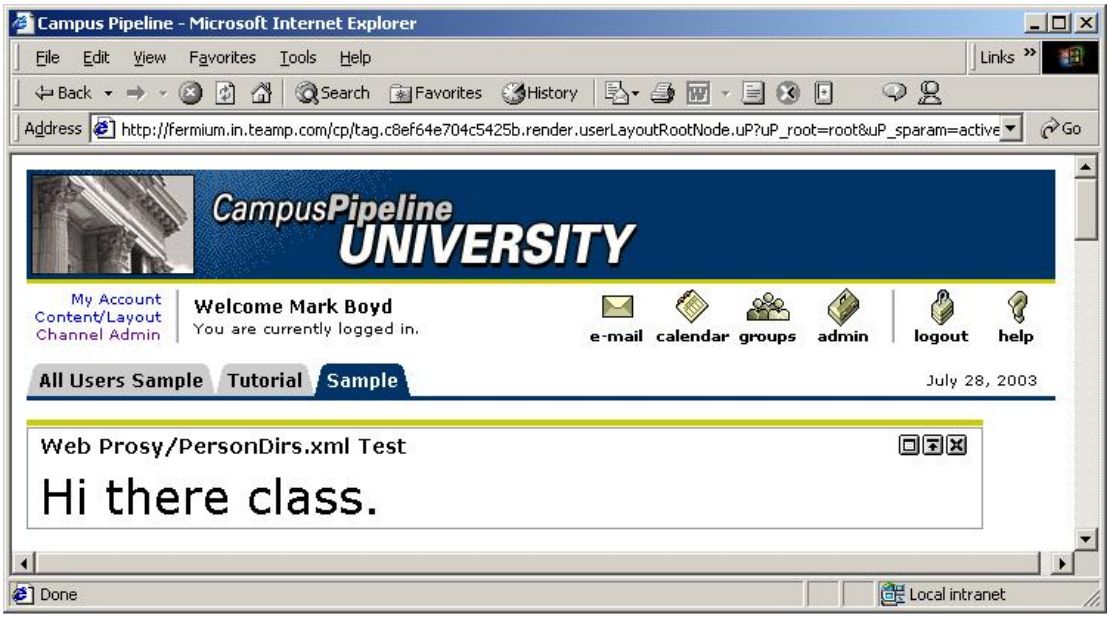
Publish-Time Parameter	**Value**
Channel Type	Web Proxy
Channel Title, Name, Functional Name, and Description	iPerson – send Attribs
Channel Timeout	3000
Channel Secure	No
Application URI	http://portal.messiah.edu/cps/hello.html
Pass-through Type	All
Default IPerson Attributes to Pass	**uid,userName**
Restrict IPerson Attributes Passing to These	**uid,userName**
Channel Controls	None
Selected Categories	Applications
Selected Groups	Public

Two other settings
There is a cache issue here. The channel will render fine the first time, but it'll forget the settings once you change tabs or refresh. So change these settings also – **Default Cache Timeout (in seconds): 9000** **Default Cache Mode: all**

Almost Done
Subscribe to the channel and see what happens.

After subscribing to the channel I then see its content as shown below clearly showing that the contents of our test.html file were indeed accessed by the channel. However, of more importance is the URL used to acquire that file. The URL as recorded in the http proxy was:

Connecting to fermium.in.teamp.com:80
GET
/cp/test.html?uid=mboyd&commonName=&givenName=Mark&mail=mboyd%40in.teamp.com&sn=Boyd HTTP/1.1



APPENDIX A : ASP Example

To get the paramers passed to an .asp page (externally hosted), do this.

Create an .asp page with this following code :

```
<html>
<head>
<title>send test</title>
</head>

<body>
<%
	username = Request.Querystring("userName")
	uid = Request.QueryString("uid")

	Response.write("username :" & username & " / uid :" & uid)

%>
</body>
</html>
```

Keep in mind this MUST be properly formed html / XHTML or else you'll get errors. Save that on an .asp server somewhere and figure out what the url is to get to the page. I saved my page as persondir.asp and my url is : http://apps.messiah.edu/LuminisXML/persondir.asp (yes that is an actual url too)

Then I edited my channel that I created above calling the 'local' Hello.html file.
All you have to do is change :

Application URI : http://apps.messiah.edu/LuminisXML/persondir.asp

Save the channel.

I had to delete the channel and re-add it to my layout to get it to work. And Bam… there it is :

APPENDIX B – PersonDirs (JDBC)

PERSONDIRS.XML

```
<?xml version="1.0"?>
<!-- $Header: /src/luminis/webapps/luminis/WEB-INF/config/PersonDirs.xml,v 1.2
2003/10/01 23:13:48 mboyd Exp $ -->

<!--
     PersonDirs.xml makes possible a standard way for the uPortal
     framework and its installed channels to refer to particular person
     attributes.  For example, if a channel wants to display a user's
     email address, it will look for an attribute called "mail", which
     appears as an alias in this file.  This alias maps to the actual
     name of the field containing an email address in your data repository.
     If no data repository at your organization contains a user's email
     address then simply don't fill in a <name> corresponding to the "mail"
     alias.  The requesting channel will simply get a null when requesting
     the attribute value of "mail" indicating that this attribute is not
     available.  Providing this level of indirection for attribute references
     makes it possible for organizations to share channels that refer to
     user's attributes.

     The <PersonDirs> element should contain one or more <PersonDirInfo>
     elements.  Each <PersonDirInfo> element represents a source for
     obtaining person attributes and should contain a set of attribute
     alias/name pairs in addition to settings for obtaining the data
     from either

       1) LDAP
       2) JDBC
       3) Container-managed JDBC DataSource

     This means that it is possible to obtain person attributes from more
     than one source.

     If a particular attribute alias is specified more than
     once, the name value obtained from the last occurrence of that attribute
     will be used.

     Author: Howard Gilbert, howard.gilbert@yale.edu
     Version $Revision: 1.2 $
-->

<PersonDirs>

  <PersonDirInfo>

    <!-- Within this <PersonDirInfo> element, set either the LDAP properties,
         the JDBC properties, or a container-defined JDBC resource.
         If you want to obtain person attributes from
         more relational databases AND/OR LDAP directories, add additional
         <PersonDirInfo> elements under <PersonDirs>.
    -->


    <!-- LDAP Properties -->
    <!--
    <url>ldap://yu.yale.edu:389/dc=itstp, dc=yale, dc=edu</url>
    <logonid>cn=bogus,cn=Users,dc=itstp,dc=yale,dc=edu</logonid>
    <logonpassword>foobar</logonpassword>
```

```
    <uidquery>(cn={0})</uidquery>
    <usercontext>cn=Users</usercontext>
    -->


    <!-- JDBC Properties -->
    <!-- This example source following these comments returns the uPortal
         user_id from the up_user table used by the portal and maps it to an
        alias 'uid' attribute in the iPerson object. Note the mapping
        specified in the <attributes> section below.

        To enable this example of additional information being pushed into the
        IPerson object uncomment lines immediately following these comments.
        Modify the values to reflect those found in

        WEB-INF/config/rdbm.properties.

        In other words copy the following rdbm.properties values into the
        specified tags as follows:

        jdbcDriver    property should be placed in the <driver> tag,
        jdbcUrl       property should be placed in the <url> tag,
        jdbcUser      property should be placed in the <logonid> tag,
        jdbcPassword  property should be placed in the <logonpassword> tag

        The <uidquery> below will pull the user_id from the up_user table
        given the user's login ID. For this sample where the user_id is pulled
        from the up_user table there is an anomaly. When a user logs in for
        the first time there will be no value in that table for that user.
        This processing takes place during authentication of that user and a
        value is not written to that table until just after authentication.
        But the next time the user logs in a value will exist and will be
        pushed into the IPerson object's attributes.

        This is only an artifact of using that table for an example.
    -->
    <driver>the fully qualified class of the driver</driver>
    <url>the jdbc url for the driver</url>
    <logonid>the db logon id</logonid>
    <logonpassword>the db password</logonpassword>
    <uidquery>SELECT USER_ID FROM UP_USER WHERE USER_NAME=?</uidquery>



    <!-- DataSource from container -->
    <!--
    <res-ref-name>PersonDb</res-ref-name>
    <uidquery>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME, LAST_NAME,
EMAIL FROM UP_PERSON_DIR WHERE USER_NAME=?</uidquery>
    -->


    <attributes>

      <!-- uPortal channels refer to user attributes using standard
           attribute names found in the eduPerson object class. If
           a name is found for an eduPerson attribute name or "alias", it
           will be stuffed into the org.jasig.portal.security.IPerson object.
           See http://www.educause.edu/eduperson/ for an explanation of each
           of these aliases.
      -->

      <attribute>
```

```
        <name>USER_ID</name>
        <alias>uid</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonAffiliation</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonNickname</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonOrgDN</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonOrgUnitDN</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonPrimaryAffiliation</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>eduPersonPrincipalName</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>c</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>cn</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>description</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>displayName</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>facsimileTelephoneNumber</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>givenName</alias>
      </attribute>
```

```
      <attribute>
        <name></name>
        <alias>homePhone</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>homePostalAddress</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>initials</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>jpegPhoto</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>l</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>labeledURI</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>mail</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>mobile</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>o</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>ou</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>pager</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>postalAddress</alias>
      </attribute>

      <attribute>
        <name></name>
```

```
        <alias>postalCode</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>postOfficeBox</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>preferredLanguage</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>seeAlso</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>sn</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>st</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>street</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>telephoneNumber</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>userCertificate</alias>
      </attribute>

      <attribute>
        <name></name>
        <alias>userSMIMECertificate</alias>
      </attribute>

    </attributes>

  </PersonDirInfo>

</PersonDirs>
```

APPENDIX C : PersonDirs.xml (LDAP)

```
<?xml version="1.0"?>
<!-- $Header: /src/luminis/webapps/luminis/WEB-INF/config/PersonDirs.xml,v 1.2
2003/10/01 23:13:48 mboyd Exp $ -->

<!--
     PersonDirs.xml makes possible a standard way for the uPortal
     framework and its installed channels to refer to particular person
     attributes.  For example, if a channel wants to display a user's
     email address, it will look for an attribute called "mail", which
     appears as an alias in this file.  This alias maps to the actual
     name of the field containing an email address in your data repository.
     If no data repository at your organization contains a user's email
     address then simply don't fill in a <name> corresponding to the "mail"
     alias.  The requesting channel will simply get a null when requesting
     the attribute value of "mail" indicating that this attribute is not
     available.  Providing this level of indirection for attribute references
     makes it possible for organizations to share channels that refer to
     user's attributes.

     The <PersonDirs> element should contain one or more <PersonDirInfo>
     elements.  Each <PersonDirInfo> element represents a source for
     obtaining person attributes and should contain a set of attribute
     alias/name pairs in addition to settings for obtaining the data
     from either

       1) LDAP
       2) JDBC
       3) Container-managed JDBC DataSource

     This means that it is possible to obtain person attributes from more
     than one source.

     If a particular attribute alias is specified more than
     once, the name value obtained from the last occurrence of that attribute
     will be used.

     Author: Howard Gilbert, howard.gilbert@yale.edu
     Version $Revision: 1.2 $
-->

<PersonDirs>

  <PersonDirInfo>

    <!-- Within this <PersonDirInfo> element, set either the LDAP properties,
         the JDBC properties, or a container-defined JDBC resource.
         If you want to obtain person attributes from
         more relational databases AND/OR LDAP directories, add additional
         <PersonDirInfo> elements under <PersonDirs>.
    -->


    <!-- LDAP Properties -->
    <url>ldap://YOURLDAP.HOST.edu:389/o=messiah.edu,o=messiah.edu</url>
    <logonid>uid=cpadmin,ou=People,o=messiah.edu,o=messiah.edu</logonid>
    <logonpassword>xxxxxx</logonpassword>
    <uidquery>(uid={0})</uidquery>
    <usercontext>ou=People</usercontext>


    <!-- JDBC Properties -->
    <!-- This example source following these comments returns the uPortal
```

```
         user_id from the up_user table used by the portal and maps it to an
       alias 'uid' attribute in the iPerson object. Note the mapping
       specified in the <attributes> section below.

       To enable this example of additional information being pushed into the
       IPerson object uncomment lines immediately following these comments.
       Modify the values to reflect those found in

       WEB-INF/config/rdbm.properties.

       In other words copy the following rdbm.properties values into the
       specified tags as follows:

       jdbcDriver    property should be placed in the <driver> tag,
       jdbcUrl       property should be placed in the <url> tag,
       jdbcUser      property should be placed in the <logonid> tag,
       jdbcPassword  property should be placed in the <logonpassword> tag

       The <uidquery> below will pull the user_id from the up_user table
       given the user's login ID. For this sample where the user_id is pulled
       from the up_user table there is an anomaly. When a user logs in for
       the first time there will be no value in that table for that user.
       This processing takes place during authentication of that user and a
       value is not written to that table until just after authentication.
       But the next time the user logs in a value will exist and will be
       pushed into the IPerson object's attributes.

       This is only an artifact of using that table for an example.
    -->
    <!--
    <driver>com.campuspipeline.rdb.conn.ConnectionPoolDriver</driver>

<url>jdbc:cp:uPortalPooled;driver=oracle.jdbc.driver.OracleDriver;url=jdbc:oracle:thin
:@bantestdb.messiah.edu:1521:PRTL</url>
    <logonid>uportal</logonid>
    <logonpassword>fortune63</logonpassword>
    <uidquery>SELECT USER_ID,USER_NAME FROM UP_USER WHERE USER_NAME=?</uidquery>
   -->


    <!-- DataSource from container -->
    <!--
    <res-ref-name>PersonDb</res-ref-name>
    <uidquery>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME, LAST_NAME,
EMAIL FROM UP_PERSON_DIR WHERE USER_NAME=?</uidquery>
    -->


    <attributes>

      <!-- uPortal channels refer to user attributes using standard
           attribute names found in the eduPerson object class. If
           a name is found for an eduPerson attribute name or "alias", it
           will be stuffed into the org.jasig.portal.security.IPerson object.
           See http://www.educause.edu/eduperson/ for an explanation of each
           of these aliases.
      -->

      <attribute>
        <name>givenName</name>
        <alias>givenName</alias>
      </attribute>
```

```
      <attribute>
        <name>sn</name>
        <alias>sn</alias>
      </attribute>

      <attribute>
        <name>userPassword</name>
        <alias>userPassword</alias>
      </attribute>

      <attribute>
        <name>displayName</name>
        <alias>displayName</alias>
      </attribute>

      <attribute>
        <name>uid</name>
        <alias>uid</alias>
      </attribute>

      <attribute>
        <name>cn</name>
        <alias>commonName</alias>
      </attribute>

      <attribute>
        <name>pdsRole</name>
        <alias>pdsRole</alias>
      </attribute>

    </attributes>

  </PersonDirInfo>

</PersonDirs>
```