UWL Custom Connector API

Sumit Gogia
Matthias Kruse
Kiran Gangadharappa
Date: 1-October-2005

1. Introduction

The Universal Worklist (UWL) provides unified and centralized way to access user's work and the relevant information in the Enterprise Portal. It collects tasks and notifications from multiple provider systems - Business Workflow, Java Ad Hoc Workflow (a.k.a. Collaboration Tasks), Alert Framework and KM Recent Notifications - in one list for one-stop access.

UWL connects to multiple provider systems through a uniform connector API. UWL interfaces with a connector, which in turn knows how to find and interact with a provider. The data provided by a provider, is mapped/converted into UWL items (task, notifications etc.) by the corresponding connector, before being sent to UWL where it is cached and rendered.

UWL already ships with a number of connectors (objects implementing the connector API). These are:

- WebFlow Connector – fetches work items from SAP Business Workflow.
- AdHocWorkflow Connector – fetches collaboration tasks from SAP AdHoc Workflow.
- Alert Connector- fetches Alerts and follow-up activities from SAP Alert Management System.
- ActionInbox Connector- fetches document related notifications from SAP Knowledge Management.
- Generic ABAP Connector[1] – For ABAP based providers, UWL simplifies the connector API by doing much of the plumbing work in this Generic ABAP connector. The ABAP based provider needs to implement the interface IF_UWL_ITEM_PROVIDER to supply items to UWL.

In most cases, for a provider that wishes to include its items (data) into UWL, one of the above connectors will suffice.

2. Disclaimer

The UWL Connector API is not currently officially supported, and hence subjected to modifications in future. Though, it is available from NW'04 SP14 onwards. For older releases, slight modifications may be needed. For NW'04S release most of the concepts remain same and NW'04S changes are mentioned wherever appropriate.

---

[1] Only in NW04s release onwards

The RSS Connector detailed in this document and the accompanying code is for demo purposes only and SAP does not provide support for it.

3. How this paper is organized

This paper is for the cases where a new connector implementation is deemed necessary. A typical example would be integration of third party workflow generated items. We will walk through the process of writing such a custom connector that fetches data from a provider (not supported by one of the existing connectors), registering and configuring it with UWL.

The rest of the paper is split into the following chapters:

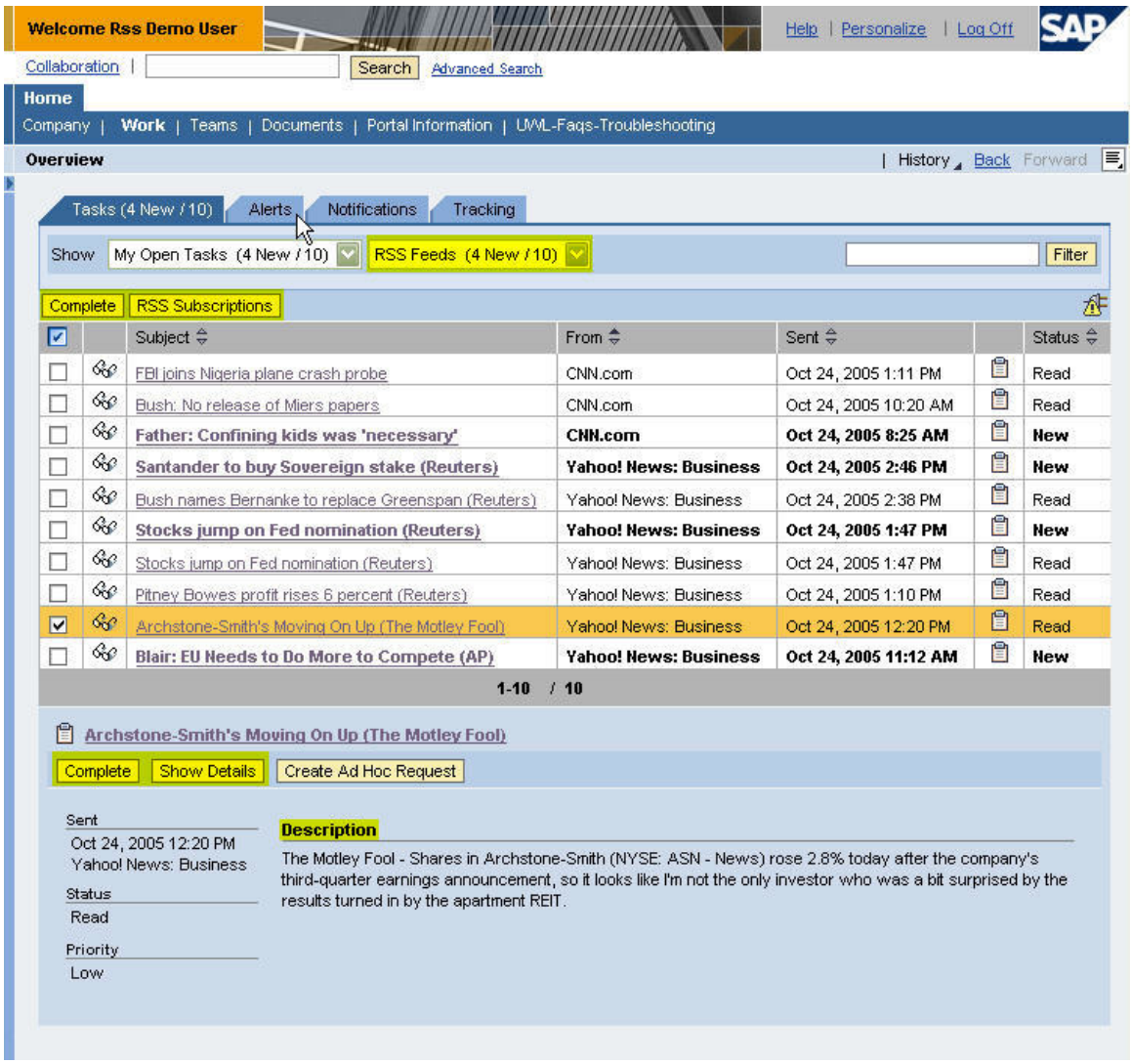
- Overview – An overview of what we want to accomplish.
- UWL Connector API – Information about the connector API
- Implementing a Custom Connector – Details on implementing our custom connector.
- UWL Basics – Minimal and necessary background information regarding UWL.
- RSS Basics – Since our custom connector would fetch RSS feeds (see 4. Overview).

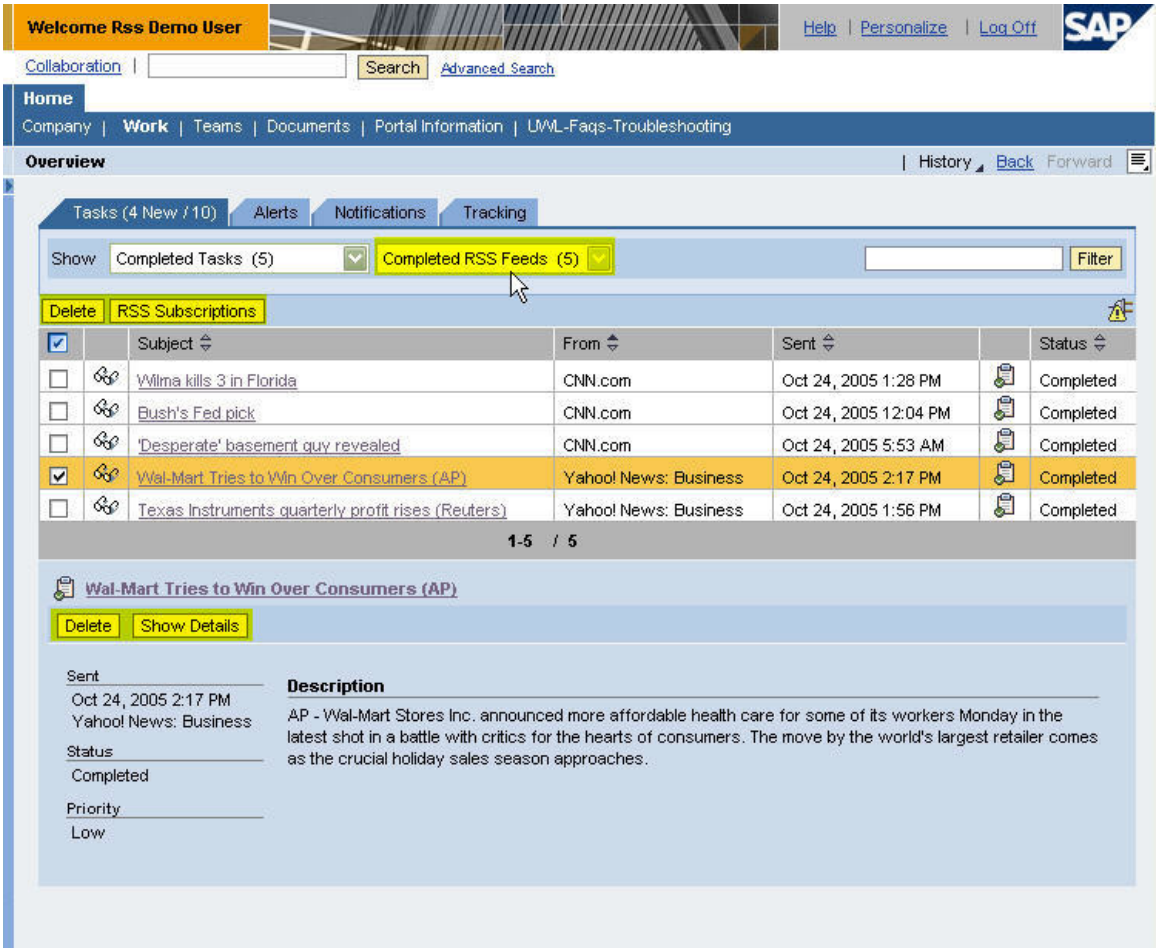
4. Overview

As mentioned in the Introduction, this paper explains the UWL custom connector API, the steps involved in writing a custom connector, and shows what can be accomplished using the connector API.

To make things a little more interesting, we will write a provider which aggregates RSS Feeds (refer to 8. RSS Basics). It will fetch RSS Feeds from various sources from the internet and aggregate them. Our custom connector will take these RSS Feeds (from the provider), convert/map them into UWL items and send them over to UWL. Finally we will configure UWL to display our RSS Items under the tasks tab. We will also configure UWL, so that certain actions can be taken on our RSS items – complete, delete and show details.

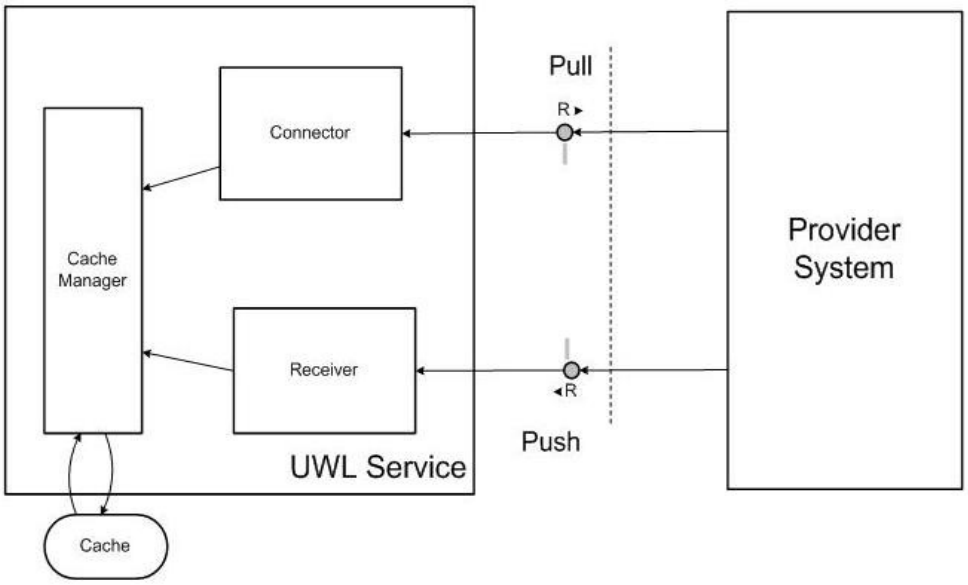
The below screen shots capture the gist of what we will do in the following chapters.


Welcome Rss Demo User
Help | Personalize | Log Off
Collaboration |
Search
Advanced Search
Home
Company | Work | Teams | Documents | Portal Information | UWL-Faqs-Troubleshooting
Overview
| History Back Forward
Tasks (4 New / 10)
Alerts
Notifications
Tracking
Show My Open Tasks (4 New / 10)
RSS Feeds (4 New / 10)
Filter
Complete
RSS Subscriptions
Subject
From
Sent
Status
FBI joins Nigeria plane crash probe | CNN.com | Oct 24, 2005 1:11 PM | Read
Bush: No release of Miers papers | CNN.com | Oct 24, 2005 10:20 AM | Read
Father: Confining kids was 'necessary' | CNN.com | Oct 24, 2005 8:25 AM | New
Santander to buy Sovereign stake (Reuters) | Yahoo! News: Business | Oct 24, 2005 2:46 PM | New
Bush names Bernanke to replace Greenspan (Reuters) | Yahoo! News: Business | Oct 24, 2005 2:38 PM | Read
Stocks jump on Fed nomination (Reuters) | Yahoo! News: Business | Oct 24, 2005 1:47 PM | New
Stocks jump on Fed nomination (Reuters) | Yahoo! News: Business | Oct 24, 2005 1:47 PM | Read
Pitney Bowes profit rises 6 percent (Reuters) | Yahoo! News: Business | Oct 24, 2005 1:10 PM | Read
Archstone-Smith's Moving On Up (The Motley Fool) | Yahoo! News: Business | Oct 24, 2005 12:20 PM | Read
Blair: EU Needs to Do More to Compete (AP) | Yahoo! News: Business | Oct 24, 2005 11:12 AM | New
1-10 / 10
Archstone-Smith's Moving On Up (The Motley Fool)
Complete
Show Details
Create Ad Hoc Request
Sent
Oct 24, 2005 12:20 PM
Yahoo! News: Business
Status
Read
Priority
Low
Description
The Motley Fool - Shares in Archstone-Smith (NYSE: ASN - News) rose 2.8% today after the company's third-quarter earnings announcement, so it looks like I'm not the only investor who was a bit surprised by the results turned in by the apartment REIT.



5. UWL Connector API

The following picture illustrates the basic idea.



The important and relevant things for us are the following:

- A connector implements the IProviderConnector interface and acts as the mediator between the UWL Service and the actual Item Provider.
- An instance of the connector needs to be registered with the UWL Service and should be co-located with the UWL Service in the same web container. There is no need for the connector to be created remotely. Once registered with UWL, the connector is called at regular intervals to retrieve items that the UWL renders.
- The actual Item Provider could be located remotely. The connector knows how to talk to the provider (in a timely fashion). For our implementation, we will keep things simple and co-locate the provider with the connector.
- The connector maps/converts the data (RSS feeds in our case) from the provider into the UWL Items.

A connector, as described above, "pulls" data from a provider at regular intervals. UWL also allows a provider to push data into UWL (the receiver box in the above picture). Refer to the java docs for details on this and for the connector API in general.

6. Implementing the Custom Connector

We need to do the following to implement a custom connector and have UWL use it:

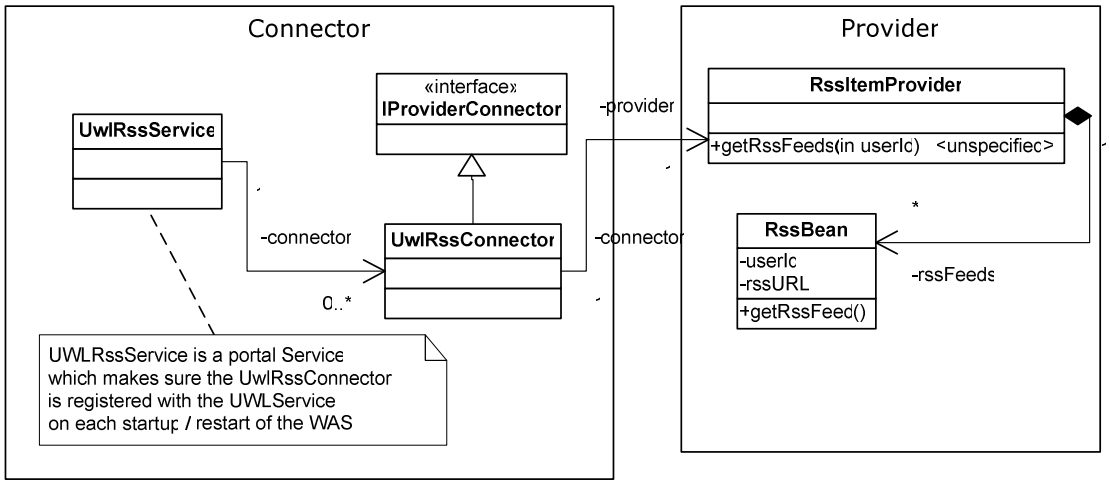
1. Write a connector which implements the IProviderConnector interface. This connector talks to a provider to fetch the actual data. In our case, we will also write a provider (an RSS aggregator) as mentioned in 4. Overview
2. Register this connector with UWL. This has to be done programmatically.
3. Indicate to the connector, which physical system the provider is located. This is done through the UWL System Configurations UI.
4. Upload a custom xml configuration to UWL to indicate our Item Types, actions that can be taken on them, and our custom second level view.

Refer to the UWL Basics chapter, to get a quick overview on UWL basic concepts and terms like Item Types, Views, and Actions etc.

Let's delve into the details of each step.

6.1 RSS Connector and RSS Item Provider

The picture below summarizes our overall design of the RSS Connector and RSS Item Provider.



The class UwlRssConnector is our connector and hence implements the IProviderConnector interface. It mediates between the UWL Service and our Provider, which we call the RssItemProvider.

The RssItemProvider acts as the aggregator of the RSS Feeds (modeled as RssBean) from all the RSS sources that the logged in user has subscribed to (each source is identified by the string rssURL in RssBean).

The RssBean models one RSS feed for one user. It contains information about the URL to the syndicated content and uses the ROME libraries to build the RSS Feed objects. Refer to 8. RSS Basics below for a quick introduction to RSS and the ROME library.

The UwlRssService is a portal service that registers our connector with UWL on every startup or restart of the WAS. This is required because the connector registrations with UWL are not persistent! If the UWL service / portal application is manually stopped & restarted, the same would have to be done for our UwlRssService. This is ensured by making our UwlRssService a dependent of the UWLService.

A brief description of the methods of the IProviderConnector interface, and their specific implementation for our RSS Connector:

- getId

  To begin with, each connector needs a unique identifier. This is used by UWL to identify each connector. We use the constant string "RssConnector" to identify our RSS connector to UWL.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getId()
 */
public String getId() {
        return UwlRssConnector.RSS_CONNECTOR;
}
```

- supportsItemType

UWL invokes this method to find which Item Types are supported by this connector. It then calls the getItems method on the connector *only* when it is displaying these Item Types. The supported item types by RssConnector are defined as string constants "uwl.task.rss" and "uwl.completedtask.rss". This means that the RSS items will be displayed under the "Tasks" tab. (Check UWL Basics section or the official UWL documentation for more information about Item Types).

We use the static helper function "matchType(childType, parentType)" from ItemType to compare the Item Types. Essentially, if the item type to be displayed by UWL is parent of RssConnector's supported item types (e.g. uwl.task or uwl.completedtask), then we return true.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#supportsItemType(java.lang.String)
 */
public boolean supportsItemType(String itemType) {
        if(itemType != null) {
            if(ItemType.matchType(RSS_ITEM_TYPE, itemType) ||
             (ItemType.matchType(RSS_COMPLETED_ITEM_TYPE,itemType)) )
                return true;
        }
        return false;
}
```

- getItems

  This method is called whenever UWL is going to display item types that are supported by the connector (see supportsItemType above). A connector makes a call to the Item Provider here, to fetch the items. Since we define both the RSS Connector, and the RSS Item Provider, we are free to decide the communication API between them. In general, it is a good idea to let the connector do any sort of caching of the items if required at all. This is because the connector is co-located with the UWL Service.

  In our case, we will keep the things simple. The RSS Item Provider does not have its own persistence. It just fetches the ***new*** RSS Feeds from the content sources on the web, and returns them to the connector. Since we know that UWL caches (persists) items in a generic manner, we let the connector make use of this. The RSS connector converts the feeds from the provider [Refer to 6.1.1 below], and packs them as a ***delta result*** before sending them over to UWL. A delta result means that only changed or new items are being returned. UWL then retains the older items (in its persistent cache) too.

  A few caveats though:

  - A fetched RSS Item could get changed both at the content source, and locally by the UWL user. E.g. the status of the RSS UWL item can change from new to read or completed. In this case we need to retain information from both sides. So every time we fetch items from the content sources, we look for the same items in the UWL cache too, and merge the information before returning the delta result. We use the UWL pushChannel api to retrieve the cached item from UWL (e.g. to read the current status of the item)

- When user wishes to delete an RSS (UWL) item, if we simply delete items, they might re-appear on a subsequent re-fetch from the content source. So we only mark them as deleted and also set an expiry date. The "expiry date" makes sure that it gets deleted from the UWL cache after the expiry date, and hence does not remain in the UWL cache forever.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getItems
 * (com.sap.netweaver.bc.uwl.UWLContext, java.lang.String,
 * com.sap.netweaver.bc.uwl.connect.ConnectorFilter, java.lang.String)
 */
public ConnectorResult getItems(UWLContext context,
                                String itemType,
                                ConnectorFilter connectorFilter,
                                String system)
throws ConnectorException {

        /*
         * get all the rss feeds for the user
         * convert them to UWL Items
         * add them to the connectorresult and give it back to UWL.
         */

        ConnectorResult result = null;
        List items = null;
        if(ItemType.matchType(RSS_ITEM_TYPE,itemType)){
                Hashtable feeds = rssProvider.getRssFeeds(context.getUserId());
                items = mapFeedsToUwlItems(context,
                                           itemType,
                                           connectorFilter,
                                           system,
                                           feeds);
        }
        else if(ItemType.matchType(RSS_COMPLETED_ITEM_TYPE,itemType)){
                /*
                 * in this case we don't return any new or modified items!
                 * let uwl use items from its cache.
                */
                items = new ArrayList();
        }

        ProviderStatus  status = new ProviderStatus(true,
                                           system,
                                           UwlRssConnector.RSS_CONNECTOR);
        result = ConnectorResult.createDeltaResult(new ItemCollection(items),status);
        return result;
}
```

- getAllActionsForItem

This method is a chance for the connector to add actions dynamically. The RSS connector doesn't do that, and hence returns the currently defined actions (which are passed as a parameter to this call).

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getAllActionsForItem
 * (com.sap.netweaver.bc.uwl.UWLContext, java.util.Map, com.sap.netweaver.bc.uwl.Item)
 */
public Map getAllActionsForItem(UWLContext ctx,
                                Map currentActions,
                                Item item)
throws ConnectorException {
        return currentActions;
}
```

- getActionHandler

  In 6.4 (Defining custom configuration and uploading it to UWL), we define certain actions that a user can take on the RSS items. E.g. complete, delete, show details etc. For some actions (e.g. show details) we use a UWL predefined action handler. For actions, for which we don't use a pre-defined action handler (e.g. complete, delete), we write our own action handler object. This action handler implements the IActionHandler interface.

  UWL calls this method to get a reference to the Action Handler. Only when the action handler id equals Action.PROVIDER_ACTION_HANDLER , we return our own action handler. For other cases, we let UWL use the pre-defined action handler that was mentioned for the action in the configuration xml.

  The RSS Action Handler primarily implements the performAction method, where it checks for the current action and takes an appropriate action (Refer to the connector API Java Docs for details on the rest of the methods) –

  - Action: Complete – It sets the item's status to complete and also changes it's item type to "uwl.completedtask.rss". This ensures that the completed item shows up in the separate "completetasks" view (Refer 6.4).
  - Action: Delete – It sets the item's status to deleted and also sets an expiry date.
  - Action: Mark as Read – It changes the item's status to read.

  In all the actions, the action handler uses the push channel to update the item in UWL's cache. This makes the changes to item persistent.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getActionHandler(java.lang.String)
 */
public IActionHandler getActionHandler(String actionHandlerId) {
        if(Action.PROVIDER_ACTION_HANDLER.equals(actionHandlerId))
                return actionHandler;
        else
                return null;
}
```

- getItem

  This method is called to ensure that the item still exists and is valid, before executing an action on it. The connector is supposed to fetch the current state of the item from the provider, and return it to UWL.

  In our case, our defined actions (read, complete, delete) that can be taken on the RSS Items don't really depend on the latest updated version of the item. Even if the RSS Item has been removed from the web source, it is ok to let the user complete/delete it. In essence, fetching all items from the source at this point won't be worth it. So we retrieve the RSS item from UWL's cache (our persistent store) using the push channel API and return it to UWL.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getItem
 * (com.sap.netweaver.bc.uwl.UWLContext, java.lang.String, java.lang.String)
```

```
 */
public Item getItem(UWLContext ctx,String systemId,String externalId)
throws ConnectorException {
        try {
                ItemKey key = new ItemKey(ctx.getUserId(),
                                        UwlRssConnector.RSS_CONNECTOR,
                                        systemId,externalId);
                return uwlService.getPushChannel().getItemByKey(key);
        }
        catch(UWLException ue) {
                throw new ConnectorException(ue);
        }
}
```

- getAttachmentHeaders

  RSS Items don't have attachments. So we return Item.EMPTY_ATTACHMENTS. It is important to return a non null value here. A null value is not expected.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#getAttachmentHeaders
 * (com.sap.netweaver.bc.uwl.UWLContext, com.sap.netweaver.bc.uwl.Item)
 */
public Attachment[] getAttachmentHeaders(UWLContext context,Item item)
throws ConnectorException {
        return Item.EMPTY_ATTACHMENTS;
}
```

- getDescription

  This is called only if the Item's description is not filled in the getItems call. Since we fill the item's description in the getItems method, in this method we can either throw a not implemented exception or just return item's description again. We do the latter for simplicity sake.

```
/* (non-Javadoc)
 * @see com.sap.netweaver.bc.uwl.connect.IProviderConnector#
 * getDescription(com.sap.netweaver.bc.uwl.Item, com.sap.netweaver.bc.uwl.UWLContext)
 */
public String getDescription(Item item, UWLContext context)
throws ConnectorException{
        /*
         * just returning the item's description once again.
         */
        return item.getDescription();
}
```

The remaining methods of the IProviderConnector interface are simple and are default implemented. Check java docs for details on these methods.

6.1.1 Mapping RSS Feed Items to UWL Items

As mentioned in the getItems method description above, the connector needs to map / convert the RSS Items to the UWL Item structure, which UWL understands. A Typical RSS feed looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
```

```
<channel>
      <title>CNN.com</title>
      <link>http://www.cnn.com/rssclick/?section=cnn_topstories</link>
      <description>CNN.com delivers up-to-the-minute news and information on the latest top
      stories,
                  weather, entertainment, politics and more.</description>
      <language>en-us</language>
      <copyright>© 2005 Cable News Network LP, LLLP.</copyright>
      <pubDate>Wed, 31 Aug 2005 18:31:47 EDT</pubDate>
      <ttl>5</ttl>
      <item>
         <title>Health fears as bodies lie in water</title>
         <link>

         http://www.cnn.com/rssclick/2005/WEATHER/08/31/katrina.impact/index.html?section=cnn_topst
      ories
         </link>
         <description>
         Public health emergency declared; cholera, typhoid fearsBush:
         "One of the worst national disasters"New Orleans mayor says toll in his city will be
      hundredsExtra
         10,000 National Guard troops called up
         </description>
         <pubDate>Wed, 31 Aug 2005 17:29:17 EDT</pubDate>
      </item>
 </channel>
</rss>
```

The root element is <rss> which has one <channel> element, which contains information about the channel. Each channel has one or more items, which represent the content from the corresponding web source. For E.g. top stories from CNN or Google News. These are the items that we want to show in the tasks tab in UWL.

To read these items/entries from the RSS feed, we use the ROME library which handles variations in different RSS versions (Refer to the RSS Basics chapter for a quick introduction to RSS and the ROME library).

So converting RSS feed entries into UWL items is as easy as reading the corresponding properties from Feed entries and creating a new UWL item with those properties. This is done in the mapFeedsToUwlItems method.

```
private List mapFeedsToUwlItems (UWLContext context,
                                 String itemType,
                                 ConnectorFilter qp,
                                 String system,
                                 Hashtable feeds) {
        ArrayList retItems = new ArrayList();
        if( (feeds == null) || (feeds.isEmpty()) ) return retItems;

        Enumeration keys = feeds.keys();
        while(keys.hasMoreElements()) {
            String key = (String)keys.nextElement();
            SyndFeed feed = (SyndFeed)feeds.get(key);
            if(feed == null) continue;
            List entries = feed.getEntries();
            for(int j=0; j<entries.size();j++) {
                SyndEntry entry = (SyndEntry)entries.get(j);
                /*
                 * If the item was previously fetched, then we need to keep
                 * some attributes from the cached item - e.g. status.
                 */
                StatusEnum itemStatus = null;
```

```
                try {
                        ItemKey itemKey = new ItemKey(context.getUserId(),
                                                        UwlRssConnector.RSS_CONNECTOR,
                                                        system,entry.getUri());
                        Item cachedItem = uwlService.getPushChannel().getItemByKey(itemKey);
                        if(cachedItem != null) {
                                itemStatus = cachedItem.getStatus();
                                //no need to update completed items.
                                if(StatusEnum.COMPLETED.equals(itemStatus)) continue;
                        }
                }
                catch (UWLException ue) {
                        Category.APPLICATIONS.logThrowable(Severity.FATAL,
                                                loc,
                                                (Object)"mapFeedsToUwlItems()",
                                                new Object[]{this},
                                                "Error fetching cached item from PushChannel",
                                                ue);
                }
                Item uwlItem = new Item(UwlRssConnector.RSS_CONNECTOR, //connectorId
                                        system,                         //systemId
                                        entry.getUri(),                 //externalId
                                        context.getUserId(),            //userId
                                        -1,                             //attachment count
                                        entry.getPublishedDate(),       //date created
                                        entry.getAuthor(),              //creator id
                                        null,                           //due date
                                        null,                           //external object id
                                        RSS_ITEM_TYPE,                  //external type
                                        RSS_ITEM_TYPE,                  //item type
                                        PriorityEnum.LOW,               //priority
                                        (itemStatus==null)?
                                        StatusEnum.NEW:itemStatus,      //status
                                        entry.getTitle() );             //subject

                if((uwlItem.getCreatorId()==null) || (uwlItem.getCreatorId().equals("")) ) {
                        String title = feed.getTitle();
                        if( (title == null) || (title.equals("")) ) {
                                uwlItem.setCreatorId(key);
                        }
                        else uwlItem.setCreatorId(title);
                }
                uwlItem.setDescription(entry.getDescription().getValue());
                uwlItem.setExecutionUrl(entry.getLink());
                retItems.add(uwlItem);
            }
        }
        return retItems;
}
```

6.2 Registering Connector with UWL

Each Connector has to be registered with the UWL Service for it to be able to use the connector. This can only be done programmatically. More over the registration with the UWL Service is not persistent. Every time the UWL Service gets restarted (either manually by the administrator or because the portal itself was restarted), the connector has to be registered again with the UWL Service.

The easiest way to do this is to write a portal service such that:

- Its "startup" attribute in portalapp.xml is set to "true". This ensures that this portal service gets started/initialized at every (re)start of the Web Application Server.
- It looks up the UWL service and registers the connector in its afterInit method. It is not a good idea to do this in the init method, since it might get called before the UWL Service has been started.
- It has a reference to the UWL Service (technical name: com.sap.netweaver.bc.uwl) in its portalapp.xml. This makes it a dependent of the UWL service, and ensures that it gets restarted whenever the UWL service is restarted.

In our case, the registration code in the afterInit method looks something like this (exception handling omitted for brevity)

```
//look up the UWLService.
IUWLService uwlService =
        (IUWLService)PortalRuntime.getRuntimeResources().getService(IUWLService.ALIAS_KEY);
//create the connector instance
connector = new UwlRssConnector(uwlService);

//register with UWL
uwlService.registerProviderConnector(connector);
```

6.3 Connecting the connector to a provider system

The connector is registered and co-located on the same system as the UWL. But the provider need not be running on the same system as UWL. The provider could be a remote Java Application, or an ABAP application. It can be anything as long as there is a connector that can communicate with it and get items for UWL.

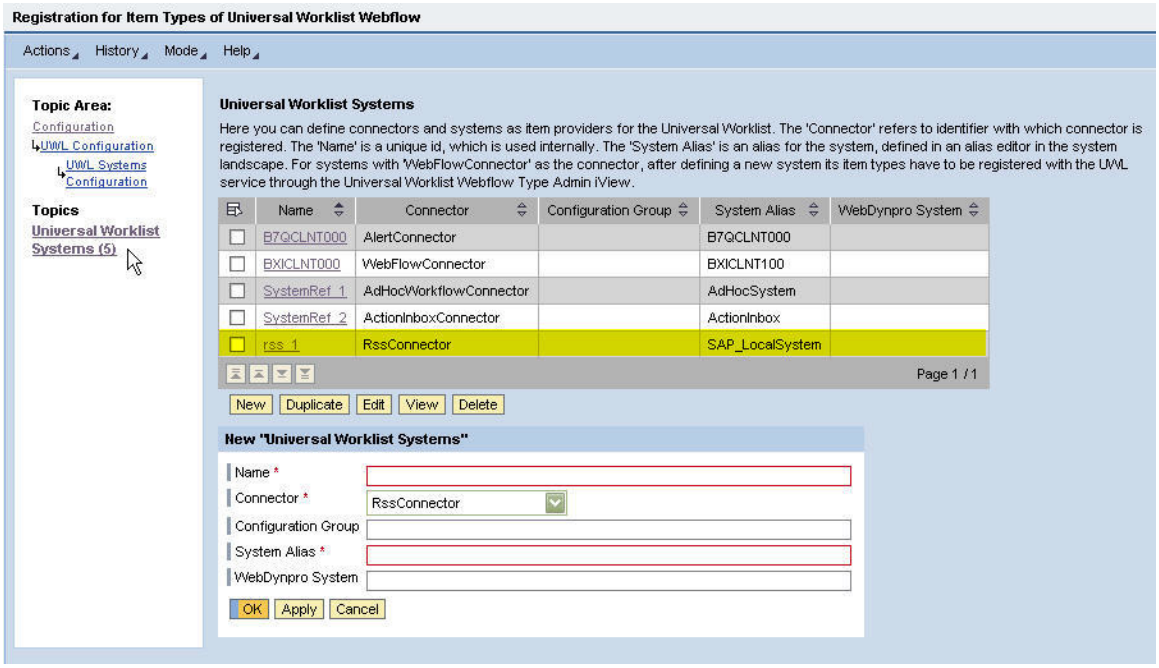
UWL provides a mechanism to configure this through its Configuration user interface. UWL uses the portal system landscape directory for this purpose. So, one can define a System (in the portal SLD) with all the relevant information and then map the connector to the System Alias through the UWL configuration user interface. The UWL Service then passes the System Alias name as a parameter in all the calls to the connector. (This step is done once the Connector has been deployed to the WAS/Portal – check 10. Deploying and Running the RSS Connector Service).

Since we decided to keep things simple for our RSS connector and RSS Item Provider by co-locating both of them together with the UWL Service, we don't really need to configure the system information. The RSS connector simply maintains a Java reference to a local instance of the RSS Item Provider. But to satisfy UWL, we do need to configure the connector with a system name. Otherwise the UWL service won't make calls to the RSS connector. A dummy system alias name will suffice for our purpose.

Note:

For NW04 only - As shown in the screenshot below, the custom connector name ("RssConnector" in the screenshot) will not show up in the Connector drop down until the UWLSystems.cc.xml file is modified and uploaded/deployed along with the custom connector's par/sda file. Refer to 10.1 UWLSystems.cc.xml

For NW04s onwards – The custom connector name ("RssConnector" in the screenshot) will show up automatically in the drop down, once the custom connector is registered with UWL (which normally is done during deployment of the Custom Connector portal service – check 10. Deploying and Running the RSS Connector Service)



6.4 Defining custom configuration and uploading it to UWL

Up until this point we have defined an Item Provider, a connector that fetches items (feeds) from the item provider, and we have registered the connector with UWL. The end result would be that the items show up under the default view for tasks. So the logged in user can look at top stories from CNN (for example) under his/her tasks view. To make this really useful to the user we need/should do the following:

- In order to avoid cluttering the user's tasks view, we define a sub view that only displays the items (RSS feeds) from the RSS connector.
- Let the user carry out some basic actions on the items. E.g. complete or delete them or navigate to the content (web) source to read the full details about the feed.
- The complete action is equivalent to archiving which will move the items to a completed tasks view. From there the user can delete the items permanently.

As it turns out, configuring the UWL user interface to suit our requirements above is a straight forward task. We need to define a custom configuration XML and then upload it to UWL.

The custom configuration XML is really straight forward and self explanatory. Check the UWL Basics chapter below for a quick overview of basic UWL terms and concepts. Also check the configuration DTD documentation in the java docs for details about what can validly go in the configuration xml.

Few things to note though:

- UWL administration and configuration UI allows you to download existing configuration files. So it is easier to take the relevant sections (e.g. view definition) from the existing configuration xml and then modify it as per needs.
- We define two views, rssView and rssCompletedView to display our item types "uwl.task.rss" and "uwl.completedtask.rss". This creates a second level sub view for the supported item type(s).
- Actions complete and delete refer to the actual action definition by the same name, in the UWL standard configuration.
- Action showDetails uses the pre-defined UrlHandler action-handler which opens a url in a new window. The url is given as a property with value coming from the item's executionUrl property. We set this property to the RSS Feed's link while creating UWL items (Refer back to Mapping RSS Feed Items to UWL Items above – method mapFeedsToUwlItems).
- Actions inside the view apply to multiple items. E.g. multiple items can be selected and completed or deleted in one shot. The corresponding action needs to be there in the Item Type as well.

Here is the full version of our custom conf xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE UWLConfiguration (View Source for full doctype...)>
 <UWLConfiguration version="1.0">
 <Actions>
 <Action name="showDetails" handler="UrlLauncher" groupAction="no"
      returnToDetailViewAllowed="yes" launchInNewWindow="no">
 <Properties>
  <Property name="url" value="${item.executionUrl}" />
     </Properties>
  <Descriptions default="Show Details" />
     </Action>
 <Action name="RSS Subscriptions" groupAction="no" handler="SAPWebDynproLauncher"
      returnToDetailViewAllowed="yes" launchInNewWindow="no">
 <Properties>
  <Property name="WebDynproApplication" value="RssSubscriptions" />
  <Property name="WebDynproDeployableObject" value="sap.com/uwl.rss.subscriptions.ui" />
  <Property name="System" value="SAP_LocalSystem" />
  <Property name="type" value="button" />
     </Properties>
     </Action>
     </Actions>
 <ItemTypes>
 <ItemType name="uwl.task.rss" connector="RssConnector" defaultView="rssView"
      defaultAction="showDetails" executionMode="default">
 <Actions>
  <Action name="showDetails" reference="showDetails" groupAction="no"
      returnToDetailViewAllowed="yes" launchInNewWindow="no" />
  <Action name="complete" reference="complete" groupAction="no" returnToDetailViewAllowed="yes"
      launchInNewWindow="no" />
     </Actions>
     </ItemType>
 <ItemType name="uwl.completedtask.rss" connector="RssConnector"
      defaultView="rssCompletedView" defaultAction="showDetails" executionMode="default">
 <Actions>
  <Action name="showDetails" reference="showDetails" groupAction="no"
      returnToDetailViewAllowed="yes" launchInNewWindow="no" />
  <Action name="delete" reference="delete" groupAction="no" returnToDetailViewAllowed="yes"
      launchInNewWindow="no" />
```

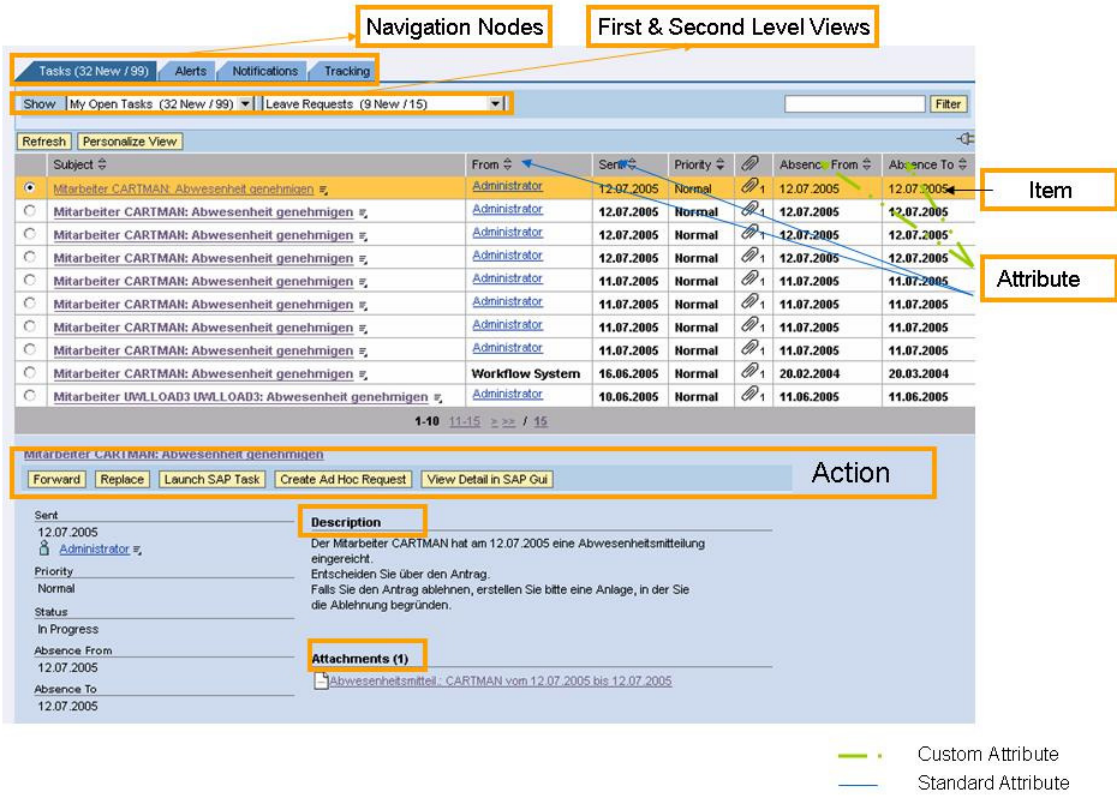
```
    </Actions>
    </ItemType>
    </ItemTypes>
<Views>
<View name="rssView" width="98%" supportedItemTypes="uwl.task.rss"
     columnOrder="detailIcon,subject,creatorId,createdDate,statusIcon,status"
     sortby="creatorId:ascend, createdDate:descend" emphasizedItems="new"
     selectionMode="MULTISELECT" tableDesign="STANDARD" visibleRowCount="10"
     headerVisible="yes" tableNavigationFooterVisible="yes" tableNavigationType="CUSTOMNAV"
     actionRef="" tableNavigationHeaderVisible="no" displayOnlyDefinedAttributes="yes"
     actionPosition="top" dynamicCreationAllowed="yes" queryRange="undefined">
 <Descriptions default="RSS Feeds" />
<DisplayAttributes>
 <DisplayAttribute name="createdDate" type="datetime" width="" sortable="yes" format="medium"
     referenceBundle="sent_date" hAlign="LEFT" vAlign="BASELINE" maxTextWidth="0"
     headerVisible="yes" />
    </DisplayAttributes>
<Actions>
 <Action reference="complete" groupAction="no" returnToDetailViewAllowed="yes"
     launchInNewWindow="no" />
 <Action reference="RSS Subscriptions" groupAction="no" returnToDetailViewAllowed="yes"
     launchInNewWindow="no" />
    </Actions>
    </View>
<View name="rssCompletedView" width="98%" supportedItemTypes="uwl.completedtask.rss"
     columnOrder="detailIcon,subject,creatorId,createdDate,statusIcon,status"
     sortby="creatorId:ascend, createdDate:descend" emphasizedItems="new"
     selectionMode="MULTISELECT" tableDesign="STANDARD" visibleRowCount="10"
     headerVisible="yes" tableNavigationFooterVisible="yes" tableNavigationType="CUSTOMNAV"
     actionRef="" tableNavigationHeaderVisible="no" displayOnlyDefinedAttributes="yes"
     actionPosition="top" dynamicCreationAllowed="yes" queryRange="undefined">
 <Descriptions default="Completed RSS Feeds" />
<DisplayAttributes>
 <DisplayAttribute name="createdDate" type="datetime" width="" sortable="yes" format="medium"
     referenceBundle="sent_date" hAlign="LEFT" vAlign="BASELINE" maxTextWidth="0"
     headerVisible="yes" />
    </DisplayAttributes>
<Actions>
 <Action reference="delete" groupAction="no" returnToDetailViewAllowed="yes"
     launchInNewWindow="no" />
 <Action reference="RSS Subscriptions" groupAction="no" returnToDetailViewAllowed="yes"
     launchInNewWindow="no" />
    </Actions>
    </View>
    </Views>
    </UWLConfiguration>
```

6.4.1 Uploading custom configuration xml to UWL

The different ways of uploading a custom configuration xml to UWL are outlined in 7.2, which has references to the UWL documentation. In our case, we simply put the custom configuration in the PORTAL-INF/classes directory, and package it as part of our par (portal archive) file. During deployment, the custom configuration is automatically picked up by "UWL (SDM) Deployment Hook" and is uploaded to UWL.

Note that the custom configuration file name needs to be in the following format uwl_<configfilename>.xml

7. UWL Basics



The picture above marks up the basic elements of the UWL user interface. The Navigation nodes, second level views, item (custom) attributes and much more is configurable through the UWL configuration xml. Refer to the UWL Configuration Guide for details.

The relevant (empty) parts of the configuration xml are listed in the xml below.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE UWLConfiguration PUBLIC '-//SAP//UWL1.0//EN'
'http://www.sap.com/dtds/nw/uwl/uwl_configuration.dtd' []>
<UWLConfiguration version="1.0">
    <DescriptionBundles>
    </DescriptionBundles>
    <Actions (global for the configuration file)>
    </Actions>
    <ItemTypes>
    </ItemTypes>
    <Views>
        <Actions (view-local visibility)>
        </Actions>
    </Views>
    <NavigationNodes>
    </NavigationNodes>
</UWLConfiguration>
```

Refer to the dtd definition for details about all the elements that can be there in the UWL configuration xml file (java docs). The basic and necessary information that we need for now, is detailed below.

7.1 UWL Basic Types

7.1.1 ItemType

ItemType is a UWL construct that captures all the necessary meta-information about the Item (and hence defines the Item). This includes:

- Definition of any custom attributes for an item – what custom attributes, how they are retrieved and the source from where they are retrieved.
- Valid Actions for the item that this ItemType defines.
- Reference to the default view definition, which will render this view in the best way.

7.1.2 View

A view need not be defined from scratch. It can refer to an existing view (definition) and can redefine values for some of its properties.

7.1.3 Action

Actions can be defined for views and items. Item actions are displayed in the preview and detail view as action buttons. View actions are also displayed as buttons and can have an effect on none, one or multiple selected items. Group actions can be used in views to render decision options directly as check boxes in the item rows, e.g. for mass approval scenarios.

Each item type contains a defaultAction, which is "the" action for the item, executed when the users clicks on the item subject. Each item can however support any number of additional actions.

7.2 Uploading configuration to UWL

There are three different ways of uploading / registering the custom configuration xml with UWL (Refer to configuration guide for details).

- We can manually upload it through the UWL Administration iView.
- We can include it as part of our business package and UWL picks up our custom xml automatically during deployment.
- Programmatically register the configuration file using the UWL Service API (Appendix Section 10.3 in the configuration guide).

8. RSS Basics

RSS (Really Simple Syndication) is a web content syndication (distribution) format. This format conforms to the XML 1.0 specification, and in essence makes the web content machine understandable. Websites use RSS feeds to distribute their content (usually the new content) and increase traffic. For example, major news sites use RSS to distribute top stories, breaking news etc. Once the content is in the RSS format, any RSS-aware program can check the feed for changes, and react to changes appropriately. For example, an RSS-aware program can help you keep up

with your favorite news sites, or weblogs by checking their RSS feeds and displaying any new or changed content from each of the feed sources.
Check References for further reading on RSS.

8.1 Reading RSS Feeds

Since RSS conforms to the xml specification, one can use a java parser to parse the xml and read out the content one wants. But as it turns out, there are close to nine different RSS formats. All of them are xml, but differ in their schemas.
RSS Versions & incompatibilities
This means that while writing an RSS feed parser, one has to either take care of the different versions, or just settle with one version. The latter would mean that the feed parser would not work with RSS feeds from a lot of content providers who use a different RSS version.

Luckily for us, a few sun engineers have taken on the task of mapping these different schemas into one model. The result is the ROME library for Java (ROME). ROME includes a set of parsers and generators for the various flavors of syndication feeds, as well as converters to convert from one format to another.
Even though there are a number of other libraries that do the same, I found ROME to be the most comprehensive and easy to use and hence decided to use it. Check this overview tutorial and other tutorials for a quick start on ROME.

9. Application Persistence

Up until this point, if you have wondered where the RSS Item Provider finds a user's subscription URLs, then this is the chapter that gives you the answers.

In general it is up to the author of the data provider to choose a persistence mechanism for its data. It is separate from the connector implementation. It is discussed here for two reasons. First to make our RSS sample complete and useful, and secondly to highlight UWL feature of pre-defined action handlers which help in seamlessly navigating from UWL user interface to a completely different application.

To keep things simple for our sample, the RSS Item provider reads the logged in user's subscription URLs from an xml file. This xml file is packaged as part of our portal service. So we manually add entries into this xml for each user, and redeploy our portal service. Not so user friendly but really simple as a first step. The initialize method in RSS Item Provider does the work of reading the user's subscription URLs from the xml file. It uses the JDOM API once again.

As a second step, we define the "RSS Subscriptions" action for our RSS views in our custom configuration xml (refer back to 6.4). The relevant action definition section is this:

```
<Action name="RSS Subscriptions" groupAction="no" handler="SAPWebDynproLauncher"
returnToDetailViewAllowed="yes" launchInNewWindow="no">
        <Properties>
                <Property name="WebDynproApplication" value="RssSubscriptions" />
                <Property name="WebDynproDeployableObject"
                value="sap.com/uwl.rss.subscriptions.ui" />
                <Property name="System" value="SAP_LocalSystem" />
                <Property name="type" value="button" />
```

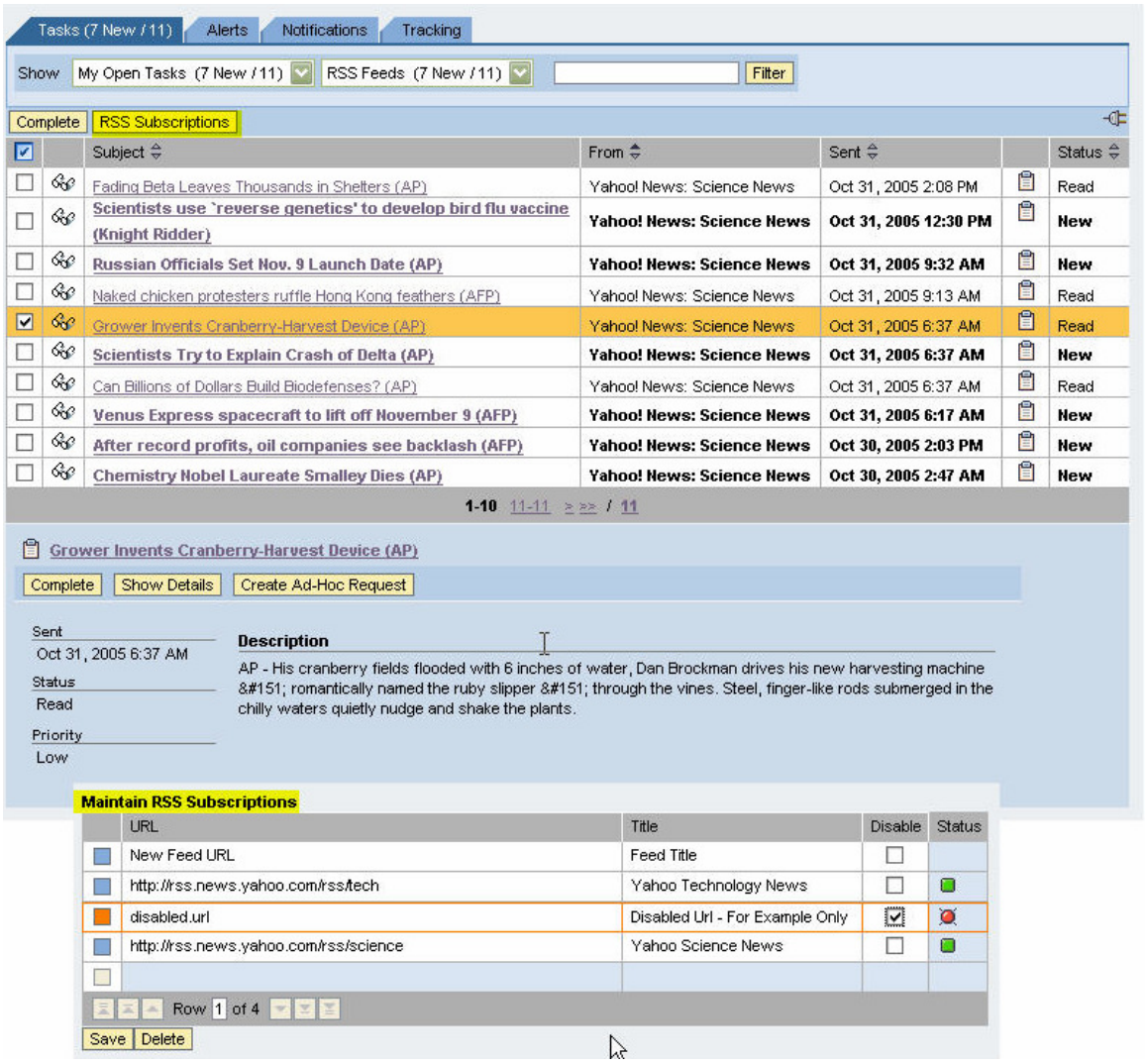
```
        </Properties>
</Action>
```

This action essentially tells UWL to use its SAPWebDynproLauncher action handler to launch the webdynpro application specified using the property element. Refer to the Action Handler Java Documentation to find out what other action handlers are provided by UWL.

We refer to this action from our view like this (once again refer back to 6.4):

```
<View name="rssView" … >
        …
        <Actions>
                …
                <Action reference="RSS Subscriptions" groupAction="no"
                returnToDetailViewAllowed="yes" launchInNewWindow="no" />
        </Actions>
</View>
```

Once we have done this, we see a button labeled “RSS Subscriptions” as part of the UWL user interface. This button will launch the webdynpro (or any other) application we specified in the action definition.



This application can be used to maintain the user's RSS subscriptions. Of course one has to write this application. Look for a simple one in the sources directory. It reads/writes to the same xml file that the RSS Item Provider uses.

10. Deploying and Running the RSS Connector Service

The sources for the RSS connector sample were written using the Netweaver Developer Studio (640 & 700). A Local Development Component of type "Enterprise Portal Service" was created. The Build result of this development component is a "sda" archive (which contains the usual "par" archive). This sda can be deployed to the Web Application Server through SDM either directly from the Dev studio or using the Remote GUI for SDM.

Another development component of type "External Library" was created to keep the external jars (JDOM & ROME libraries) and then reference them from our first DC. Check the NWDI documentation for details on using Development components.

Also, the jar file containing the the UWL Connector API classes (bc.uwl.service.api_api.jar) is kept in the "External Library" DC and referenced in the main Portal Service DC. You will find this jar file under your portal installation at <WAS Install directory>\<SID>\JC<Instance Number>\j2ee\cluster\server0\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal\portalapps\com.sap.netweaver.bc.uwl\lib

To use the sample development components as is, change the root location of your development components in the Netweaver Dev Studio to point to the directory where the sample's .dcdef file is located (This is done under Windows->Preferences->Java Development Infrastructure->Development Configurations). The DCs would then be visible in the Netweaver Dev Studio. (*Remember to replace the bc.uwl.service.api_api.jar with the one from your portal installation*)

10.1 UWLSystems.cc.xml[2]

As mentioned in 6.3 Connecting the connector to a provider system, the custom connector name does not show up in the drop down box in the UWL Systems Configuration UI until the following is done.

1. Locate the following file under your portal installation directory: com.sap.netweaver.bc.uwl.configarchive (it is a zip file). It is located at <WAS Install directory>\<SID>\JC<InstanceNumber>\j2ee\cluster\server0\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal\portalapps\com.sap.netweaver.bc.uwl\config.
2. Copy this config archive to a temp directory.
3. Extract the install\meta\lib\com.sap.netweaver.bc.uwl.configmeta file (it is a zip file) from the configarchive file, and then extract the com.sap.netweaver.bc.uwl\systems\UWLSystems.cc.xml from the configmeta file.
4. Edit the UWLSystems.cc.xml file and add your connector name in the list of connectors.
5. Now update the configmeta file with the modified UWLSystems.cc.xml file and then update the configarchive file with this modified configmeta file. Make sure to maintain the relative paths of the files in the zip archives.
6. Copy this modified configarchive to your portal project location under the PORTAL-INF\config directory. This ensures that when you build your portal project, the resulting par file contains the configarchive. (When you create a portal project in the Netweaver Developer Studio, the PORTAL-INF directory is created under the "dist" directory.

A batch file to automatically do the above steps is included with the sources. Look for updateConfigArchive.bat. It automatically modifies the configarchive file and places it under your PORTAL-INF/config directory of your project, so that anytime you build your connector par/sda, it gets packaged as well. It only needs to be run once.

10.2 Proxy Settings in your portal

[2] This is applicable only for NW'04 release. For NW'04S this step is not needed.

The RSS connector connects to web sources on the internet for RSS Feeds. If you (or your organization) use a proxy server to access the web, the necessary proxy settings would be required in the portal.

This is done at the following location (once you log as administrator):

System Administration->System Configuration->Service Configuration->Applications->com.sap.portal.ivs.httpservice->Services->proxy

Check the portal documentation for details.

11. References

UWL End User Guide

UWL Administration/Configuration Guide

RSS Technical Overview

RSS Overview & History

UWL Central note 676253