

Upfront Customization – Content Sorting

Combining UTML, XSLT and JavaScript

Every once in a while you are being asked to provide an apparently simple feature of automatically or manually specifying order of Upfront NewsItems and NewsBoxes in each folder. In the standard Upfront templates, there is no property that would allow you to do that. However, the Velocis database that is hosting the Upfront database, is capable of adding new properties to any of Upfront objects.

Design phase

To accomplish desired result of allowing Upfront Administrators rearrange the order of items in any Upfront folder, we will automatically add a new property called SortOrder. We will use UTML to obtain data from Upfront, while XSLT, CSS and JavaScript will be used to process and render that information into HTML format. This will be consumer's only custom theme, so we will not touch any of the administrative tasks, that will be still handled by the standard70 Upfront theme.

Technology Overview

One of the many ways to query Upfront database, is to use the UTML language. It stands for Upfront Template Markup Language and is a language that is used in Upfront user interface templates. Upfront 6.0 templates used UTML version 1.0. Upfront 7.0 templates use UTML version 2.0.

XSLT, which stands for Extensible Style Sheet Templates, is a language for transforming XML documents into other XML documents. HTML, and especially XHTML, is one of many XML document types. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary. XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.

CSS, which stands for Cascading Style Sheets, is a style sheet language that allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of documents from the content of documents, CSS2 simplifies Web authoring and site maintenance. CSS2 builds on CSS1 (see [CSS1]) and, with very few exceptions, all valid CSS1 style sheets are valid CSS2 style sheets. CSS2 supports media-specific style sheets so that authors may tailor the presentation of their documents to visual browsers, aural devices, printers, braille devices, handheld devices, etc. It also supports content positioning, downloadable fonts, table layout, features for internationalization, automatic counters and numbering, and some properties related to user interface.

JavaScript is Netscape's cross-platform, object-oriented scripting language. Core JavaScript contains a core set of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects. JavaScript lets you create applications that run over the Internet. Client applications run in a browser, such as Netscape Navigator, and server applications run on a server, such as Netscape Enterprise Server. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases.

Implementation

Our new Upfront custom theme will be called gazebo. First we need to create two new folders, with the exact same name as the theme: one in the cer3/webcontent/upfront/en/, which will be hosting our static HTML files, JavaScript files and images; second in the cer3/templates/upfront/en/, which will be hosting our dynamic templates. We will also need a set of subfolders in both of them, too. Use your favorite tool, Windows Explorer or DOS prompt, to create the following folder structure:

```
cer3/webcontent/upfront/en/gazebo/common
cer3/webcontent/upfront/en/gazebo/ie4
```

Darek Danielewski
Partner

October 2004
BrightStar Partners, Inc

Let's start with fairly simple tasks:

- In cer3/webcontent/upfront/en/gazebo/common, create an empty file called blank.html. We will use it later as default page to display for our content element.
- From cer3/webcontent/upfront/en/standard70/common, copy iwr_wizard.js and utilities.js to the same folder.
- Copy styles.css from cer3/webcontent/upfront/en/standard70/ie4 to cer3/webcontent/upfront/en/gazebo/ie4
- From cer3/webcontent/upfront/en/standard70/images, copy anim_busy.gif, dialog_header.gif, dialog_line1.gif, icon_return.gif, icon_save.gif, icon_saveas.gif, space.gif, toolbar_email.gif, toolbar_email_d.gif, toolbar_email_r.gif, toolbar_runoptions.gif, toolbar_runoptions_d.gif, toolbar_runoptions_r.gif, toolbar_separator.gif, toolbar_view_excel.gif, toolbar_view_excel_d.gif, toolbar_view_excel_r.gif, toolbar_view_html.gif, toolbar_view_html_d.gif, toolbar_view_html_r.gif, toolbar_view_pdf.gif, toolbar_view_pdf_d.gif, toolbar_view_pdf_r.gif, toolbar_view_separator.gif, toolbar_view_text.gif, toolbar_view_text_d.gif, toolbar_view_text_r.gif, to cer3/webcontent/upfront/en/gazebo/images (or copy all images altogether)
- From cer3/templates/upfront/en/standard70/common copy common.uinc, footer_left.uinc, footer_right.uinc, guide_to_docs.utml, header_bottom.uinc, header_top.uinc, iwr_modifyprompts.utml, iwr_outputview_frame.utml, iwr_outputview_header.utml, iwr_outputview_ribbon.utml, iwr_running.utml, meta_tags.uinc to cer3/templates/upfront/en/gazebo/common to be able to test some of the functionality
- Start Upfront administration console and register the new gazebo theme.

Next, in the cer3/templates/upfront/en/gazebo/common, create a new file, called main.utml. This will be our starting point for each user. It is time to do some coding, but don't worry, I'll explain each step.

Listing 1 contains the source code of our main template. First we need to obtain the content for the folder that the user requested:

```
<utml:content>
  <DescribeNewsBox RequestId="1" numlevels="1">
    <utml:switch>
      <utml:case test="<%id%>"><Id><%id%></Id></utml:case>
      <utml:default><Id><%SYSTEM.BaseNewsBoxId#safe%></Id></utml:default>
    </utml:switch>
    <Module Name="ProviderTypeDescriptor"/>
    <Module Name="Base"/>
  </DescribeNewsBox>
</utml:content>
```

When a request is sent to Upfront, it might be accompanied by a NewsBox ID or by default, it will be intended for the root of all NewsBoxes. In the first case, the ID element provided in the GET or POST call, can be obtained within an Upfront template, by enclosing the element name in <%%> brackets. In this case we need to use <%id%>. When we are looking for the NewsIndex object, which is the default name of the root of all NewsBoxes, we can simply call for a global Upfront variable called SYSTEM.BaseNewsBoxId. We still have to enclose it in the special brackets.

Because there are many different modules available to fully describe any Upfront object, we are specifically requesting only the Base and ProviderTypeDescriptor modules. We also specify that we need only the immediate children of the NewsBox to be returned, by using the numlevels attribute with a value of 1. If you'd like to return more levels of folder's descendants, you need to provide a higher value or specify All, to return all children and descendants of a NewsBox.

After you receive data from Upfront you can start presenting it within the <utml:presentation> tags. As there can be multiple <DescribeContent> requests, you have to be specific which content results you are about to process. In the title section we will have a standard heading, that displays the Upfront heading with the folder's name. Next, we define the basic styling of our web page. The way CSS styling works, is that every style element, which name starts with a dot, like .niElement and .nbElement, will apply to all HTML object, regardless of their type, that will specify the style's name as their class attribute. However, you can extend the style for certain element types, by prefixing it with the HTML object name, in our case TD.nbElement and TD.niElement. The overflow:hidden attribute of the BODY style, tells any browser to keep

everything in one window without any scrollbars. This is the desired effect in our case, however, in other instances it would clip object's content, if the object was bigger than the display space.

All folder' content in our web page will be rendered with the help of two HTML tables. The first table will contain simple links to our sorting routines as well as the folder' dynamic access path. Second table will present all NewsBox and NewsItem children of the folder we are processing.

Our first encounter with XSLT starts with rendering the folder' access path. Every folder contains this element to specify it's location within the Upfront tree:

```
<FullPath>
  <NewsBoxPath>
    <Id>107cb0c090b611d7a44e92ffed0956d7</Id>
    <Name>NewsIndex</Name>
    <NewsBoxPath>
      <Id>109cc7b090b611d7a44e92ffed0956d7</Id>
      <Name>Administration</Name>
      <NewsBoxPath>
        <Id>10992fcc7bb611d7a44ef090ed0d7956</Id>
        <Name>IWR Drill Through Reports</Name>
      </NewsBoxPath>
    </NewsBoxPath>
  </NewsBoxPath>
</FullPath>>
```

As you can see, it is a recursive type of tree, where a NewsBoxPath element, contains another NewsBoxPath element, that just might contain another NewsBoxPath element, and so on and so forth. XSLT is a perfect tool to process this type of data, as it is based on templates and a template is allowed to call itself with same or different set of parameters. Here is the code for our recursive template:

```
<xsl:template name="build-path">
  <xsl:param name="current-node"/>
  <xsl:param name="template"/>
  <xsl:if test="$current-node">
    <I><a class="nbHeader">
      <xsl:attribute name="href">
        %SYSTEM.UpfrontPath#html%
        <xsl:value-of select="'?xmlcmd=<GetPage><Template>'" />
        <utml:escape escape-style="encode-canhtml"><xsl:value-of
select="$template"/></utml:escape>
        <xsl:value-of select="'</Template></GetPage>&id='"/>
        <utml:escape escape-style="encode-canhtml"><xsl:value-of
select="$current-node/Id"/></utml:escape>
      </xsl:attribute>
      [ <utml:escape escape-style="encode-canhtml"><xsl:value-of select="$current-
node/Name"/></utml:escape> ]
    </a></I>
    <xsl:if test="$current-node/NewsBoxPath">
      <xsl:value-of select="' - '"/>
      <xsl:call-template name="build-path">
        <xsl:with-param select="$current-node/NewsBoxPath" name="current-
node"/>
        <xsl:with-param select="$template" name="template"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:if>
</xsl:template>
```

This template is using two parameters: current-node and Upfront template name to be used for rendering. In our case the, the template name will be always main.utml. First, we test if we have a non-empty node to process. If we do, we create an A HTML element, with all the right attributes. Within XSLT templates it is required to specify text using <xsl:value-of select="string to display"/> element, if the text contains characters like < and >, used by XSL itself. If you want to obtain a variable's value that is available to a template, you do it with the same xsl:value-of element, however, your select attribute should read \$variable_name, in our case \$current-node and \$template. As you can see, you may refer to a particular node of an XML variable by using the construct \$variable_name/node_name. For example, we need the node's ID value to correctly identify the folder to the Upfront template, so we obtain it by specifying \$current-node/Id.

After we process current node, we check if there is another NewsBoxPath element available and call the template itself, with a new value of current-node, the \$current-node/NewsBoxPath element. Looking at the

example above, we would display NewsIndex – Administration – IWR Drill Through Reports. We need to call this template from within our code by using:

```
<TD><xsl:call-template name="build-path">
  <xsl:with-param name="current-node" select="<%UP.FullPath.NewsBoxPath%>" />
  <xsl:with-param name="template" select="'main.utml'" />
</xsl:call-template></TD>
```

Now it is time to process the folder's content. First we need to check if the SortOrder property is present by executing test="Property/Value[../Name='SortOrder']". You should read it as "find the Value element of the Property element, having (the Property element) another child with the Name element with its text value equal to string SortOrder".

```
<Property>
  <Name>SortOrder</Name>
  <Value>a7265ff0a37611d7b2b88e6dfd308b8a|a76b2360a37611d7b2b88e6dfd308b8a</Value>
</Property>
```

When we find such element, we process the getListItem template, otherwise we will process standard template for NewsItem and NewsBox elements. Later, both templates will end up using the same rendering template, however, some initial steps need to be done differently.

If SortOrder property had a non-empty value, it would contain a pipe delimited set of Upfront objects' IDs. We obtain that string and pass it to the getListItem template along with the value of the separator, in our case '|' character. This template is another example of a recursive template where we obtain the value of the first ID, process it, remove it from the list, and call the same template again with short list of IDs to process. It is using the formatItem template as a helper to get to the formatUpfrontItem template. As I mentioned, regardless if we have the SortOrder property or not, we will use the same final template, the formatUpfrontItem, to format current element. However, there is a difference in processing XML element when using xsl:call-template and xsl:apply-templates.

When we use xsl:call-template, your context, the place where you are inside the XML tree, does not change. So that is why in the formatItem template, we obtain the value of currentElement by using select="*[./Id=\$itemValue]", which reads "from all the children of the current node, find all that have an ID element with its text value equal to the value of itemValue variable".

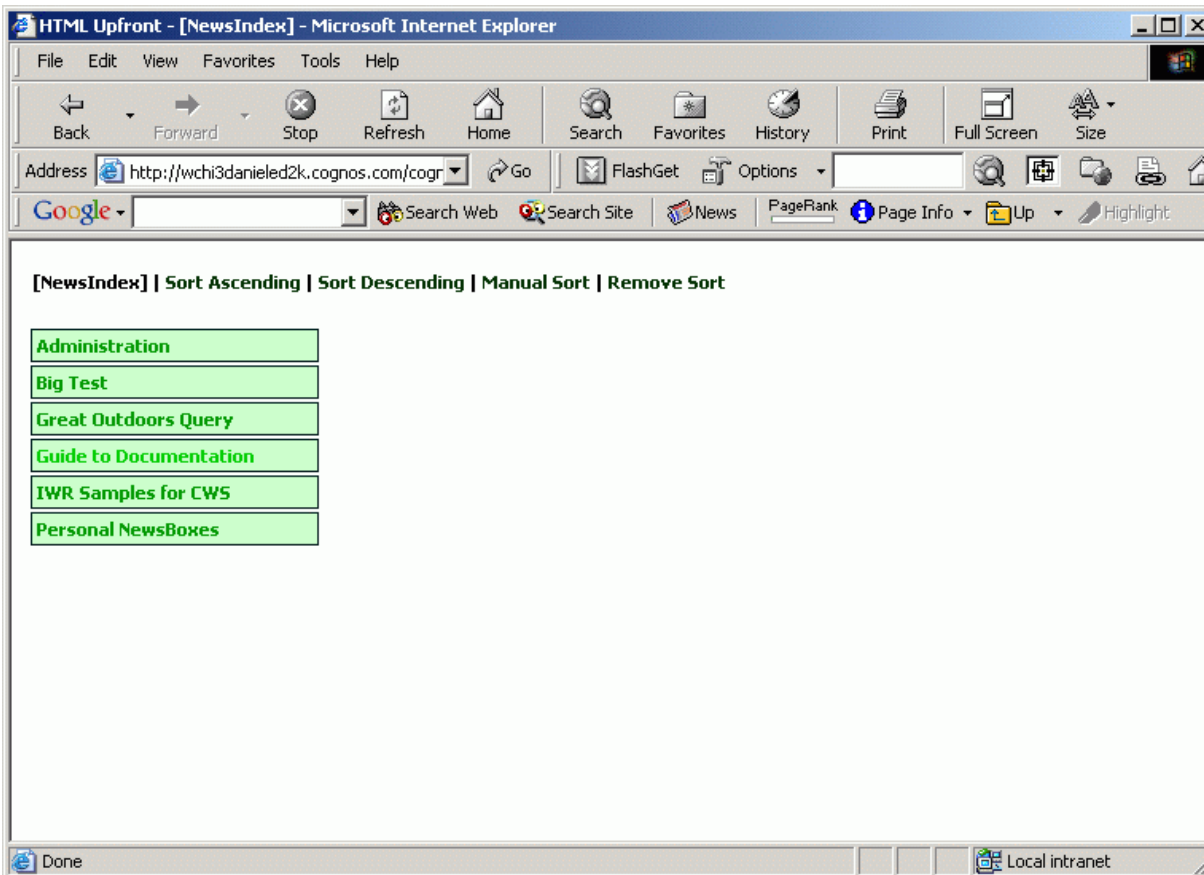
In the second scenario, when using the xsl:apply-templates, the context is automatically changed to the element being processed, so we obtain its value by using select=".", which simply reads "select current item". I am pretty sure that by now you will be able to dissect the formatUpfrontItem template yourself.

The last step that needs explanation is how do folder's elements get sorted and stored within the custom property. We have provided the prospective admin with four different links to sort and un-sort the elements: Ascending, Descending, Manual and Remove. The first two use the same JavaScript function, with different flags, that indicate the order of elements. When the HTML is being rendered, a JavaScript table called els is build, and contains the IDs and Names of all children of the current folder. It is a two dimensional table, that contains the IDs in the first column, so we cannot use the default Array.sort() JavaScript method, without providing the correct set of functions for comparison. In JavaScript, it is possible to treat a function as any other object, so using a method call like els.sort(compareAsc) is perfectly legal, and it basically tell the interpreter to sort the els array while using the compareAsc object, in our case a function, for comparison purposes. The requirement for the comparison function is to return -1 if element one is smaller then element two, 1 if it is greater, and 0 if they are equal. You may take even more advantage of it and sort folders elements by name and date for example.

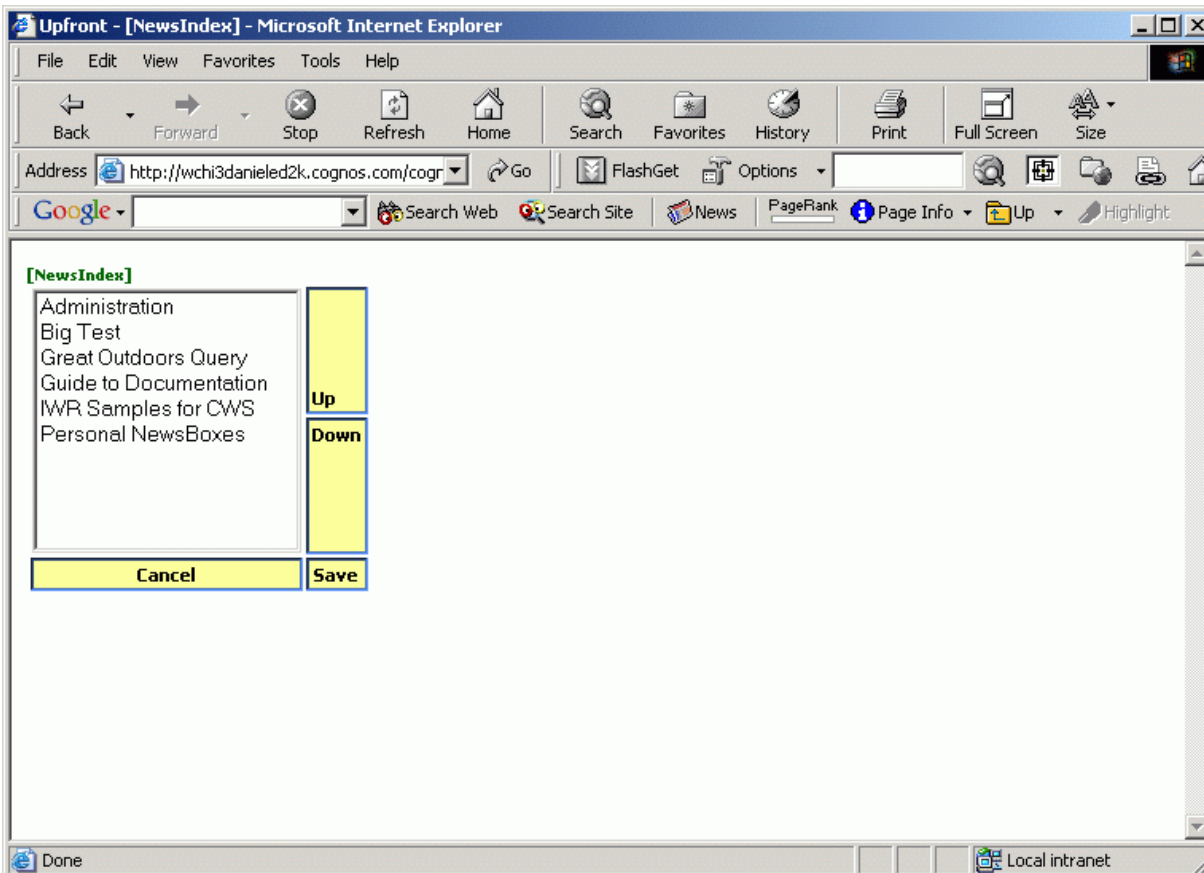
After sorting the array, the sortOrder function prepares an XML command that will be send to Upfront to store the sorting order as a new or existing Property element. The removeSort function simply removes the SortOrder property from the list of folder's properties. The manual sort is more complicated and requires a separate template to process. You may find the source of that template in *Listing 2*. It should be save to a file called manualsort.utml.

Summary

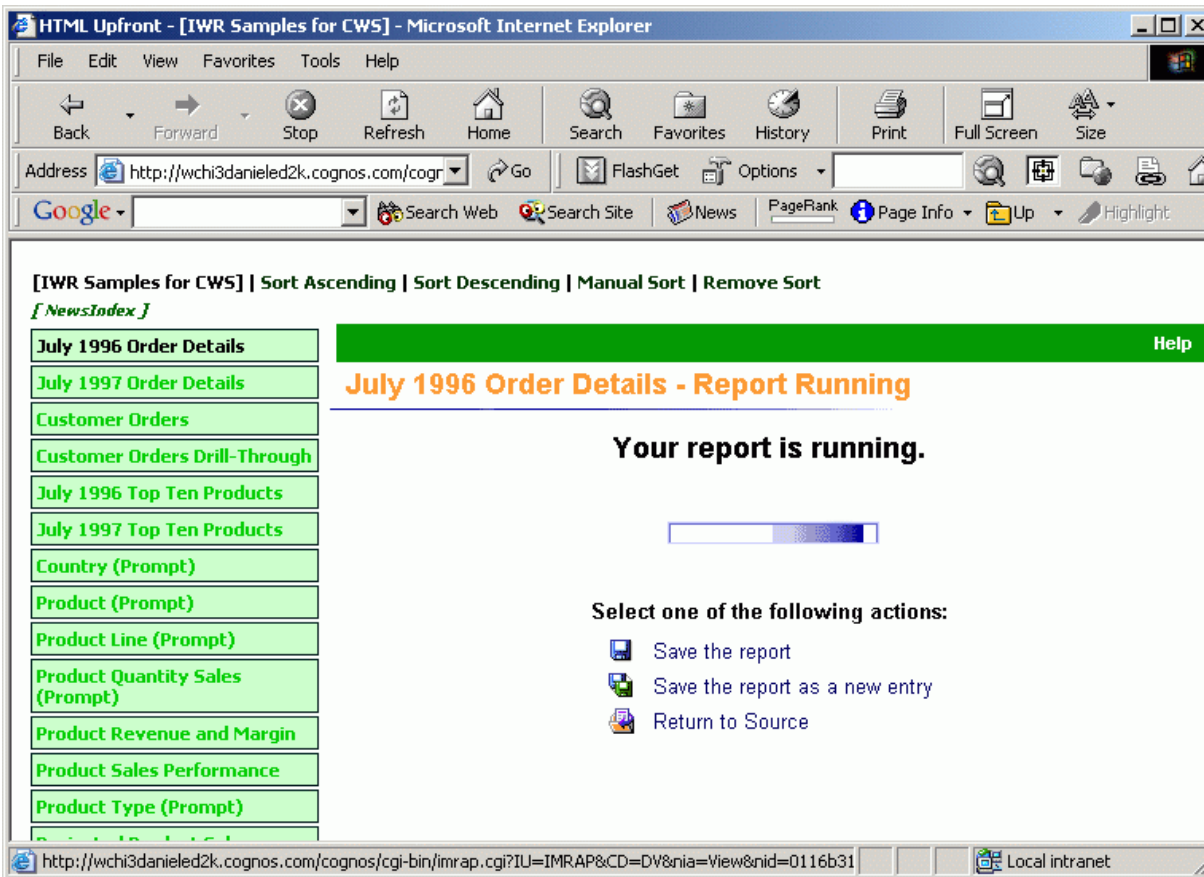
Here are screen shots of what the result should look like (after you update the styles.css file):



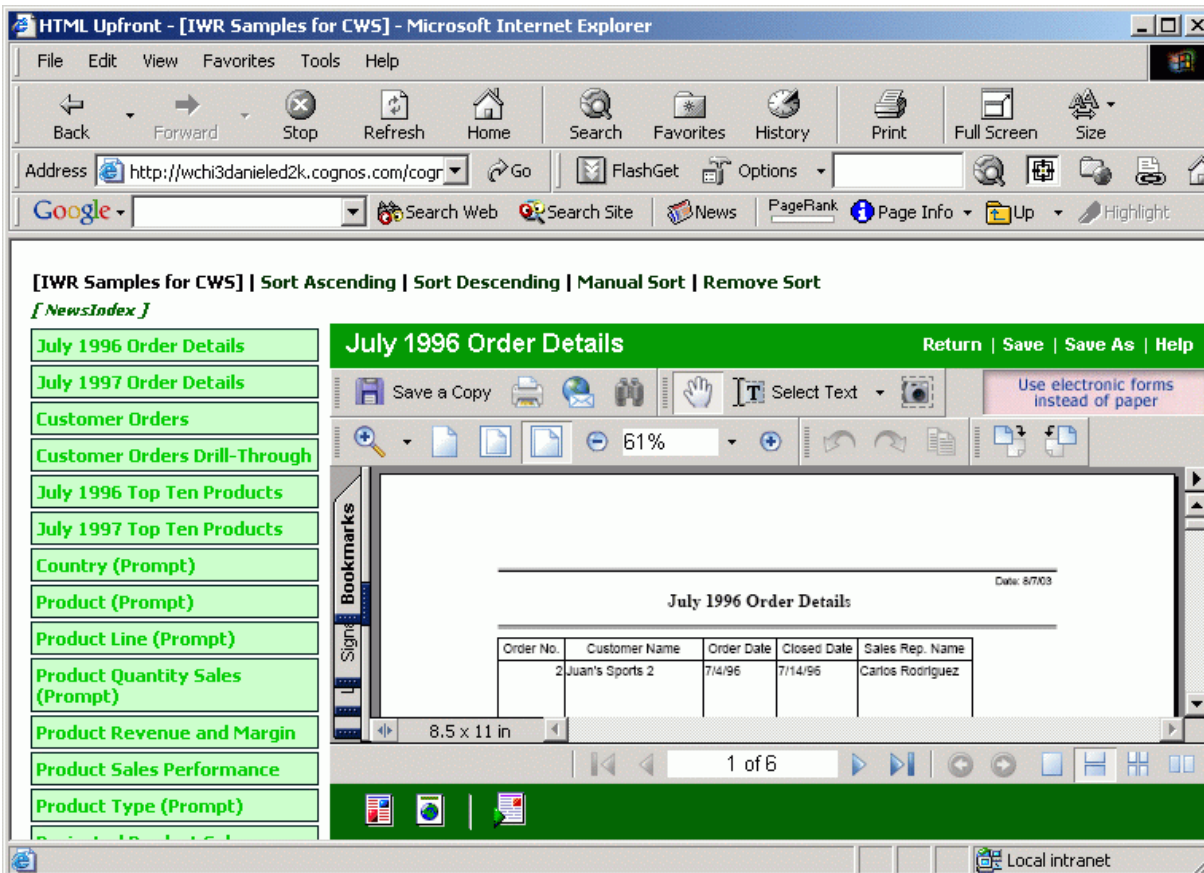
Picture 1: Main page



Picture 2: Manual sorting



Picture 3: Running a report



Picture 4: Rendered report

As you can see, Upfront can be extended in some quite unexpected ways. Hopefully, this will give you even more ideas to further improve the functionality of COGNOS tools.

Darek Danielewski
National BI Advisor

Listing 1

```
<utml:content>
  <DescribeNewsBox RequestId="1" numlevels="1">
    <utml:switch>
      <utml:case test="<%id%>"><Id><%id%></Id></utml:case>
      <utml:default><Id><%SYSTEM.BaseNewsBoxId#safe%></Id></utml:default>
    </utml:switch>
    <Module Name="ProviderTypeDescriptor"/>
    <Module Name="Base"/>
  </DescribeNewsBox>
</utml:content>

<utml:presentation RequestId="1" >
<html>

<head>
<title>Upfront - [<xsl:value-of select="Name"/>]</title>
<style>
  TR { font-family:Tahoma; font-size:11; font-weight:bold; }

  .nbElement { color:#009900; background-color: #CCFFCC; }
  .niElement { color:#00CC00; background-color: #CCFFCC; }
  TD.nbElement { border: 1px ridge #003333; }
  TD.niElement { border: 1px ridge #003333; }
  .nbHeader { font-family:Tahoma; font-size:10; font-weight:bold; color:#006600; }
  A {
    text-decoration: none;
    color: #003300;
  }
  a:hover {
    text-decoration: none;
    color: #000000;
  }
BODY { overflow:hidden; }
</style>
</head>
<body>
<table class="nbHeader" border="0" width="100%">
  <TR>
    <td><xsl:if test="IsOwner='Y'">
      <table border="0" cellpadding="1" cellspacing="0">
        <td>[<xsl:value-of select="Name"/>]</td><td>|</td>
        <td><A href="javascript:setOrder('Asc');">Sort Ascending</A></td><td>|</td>
        <td><A href="javascript:setOrder('Desc');">Sort Descending</A></td><td>|</td>
        <td><A href="javascript:manualSort();">Manual Sort</A></td><td>|</td>
        <td><A href="javascript:removeSort();">Remove Sort</A></td><td></td>
      </table>
    </xsl:if>
    </td>
    <td>&nbsp;</td>
  </TR>
  <TR>
    <TD>
      <xsl:call-template name="build-path">
        <xsl:with-param name="current-node" select="<%UP.FullPath.NewsBoxPath%>" />
        <xsl:with-param name="template" select="'main.utml'" />
      </xsl:call-template>
    </TD>
    <td>&nbsp;</td>
  </TR>
</table>
<TABLE border="0" cellpadding="3" cellspacing="2" style="width:185px;">
<xsl:choose>
  <xsl:when test="Property/Value[../Name='SortOrder']">
    <xsl:call-template name="getListitem">
      <xsl:with-param name="list"><xsl:value-of select="Property/Value[../Name='SortOrder']">
    </xsl:with-param>
    <xsl:with-param name="delimiter" select="'|'" />
  </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates select="(NewsItem|NewsBox)">
```

```

</xsl:apply-templates>
</xsl:otherwise>
</xsl:choose>
</TABLE><IFRAME name="contentDiv" style="position:absolute; top:52px; left:200px;
width:expression(document.body.clientWidth-200); height:expression(document.body.clientHeight-52)"
src="%USER.ThemePath#html%>/common/blank.html" frameborder="0"></IFRAME>

<br/>

<script language="JavaScript">
var els = Array( <xsl:for-each select="(NewsBox|NewsItem)">Array("<xsl:value-of
select="."/Id"/>","<xsl:value-of select="./Name"/>"),</xsl:for-each>Array("", "") );
els.pop();
function compareAsc(a,b){
  if (a[1]<b[1])
    return -1;
  if (a[1]>b[1])
    return 1;
  return 0;
}
function compareDesc(a,b){
  if (a[1]<b[1])
    return 1;
  if (a[1]>b[1])
    return -1;
  return 0;
}
function setOrder(AscDesc){
  if (AscDesc == 'Asc')
    els.sort(compareAsc);
  if (AscDesc == 'Desc')
    els.sort(compareDesc);
  strurl = "<SetNewsBoxProperties><id><%UP.Id%></id>";
  strurl = strurl + "<Property option='set'><Name>SortOrder</Name><Value>";
  for(var i=0; i<((els.length)-1); i++)
    strurl = strurl + els[i][0]+"|";
  strurl = strurl + els[els.length-1][0] + "</Value></Property></SetNewsBoxProperties>";
  actionPost.xmlcmd.value = strurl;
  actionPost.back.value = "/cognos/cgi-
bin/upfcgi.exe?xmlcmd=<GetPage><Template>main.utml</Template></GetPage>&id=<%UP.Id%>";
  actionPost.submit();
}
function getFolder(id){
  actionPost.xmlcmd.value="<GetPage><Template>main.utml</Template></GetPage>";
  actionPost.back.value="";
  actionPost.id.value=id;
  actionPost.submit();
}
function manualSort(){
  actionPost.xmlcmd.value="<GetPage><Template>manualsort.utml</Template></GetPage>";
  actionPost.back.value="";
  actionPost.id.value="<%UP.Id%>";
  actionPost.submit();
}
function removeSort(){
  strurl = "<SetNewsBoxProperties><id><%UP.Id%></id>";
  strurl = strurl + "<Property option='remove'><Name>SortOrder</Name>";
  strurl = strurl + "</Property></SetNewsBoxProperties>";
  actionPost.xmlcmd.value = strurl;
  actionPost.back.value = "/cognos/cgi-
bin/upfcgi.exe?xmlcmd=<GetPage><Template>main.utml</Template></GetPage>&id=<%UP.Id%>";
  actionPost.submit();
}
</Script>
<form name="actionPost" method="POST" action="%SYSTEM.UpfrontPath#html%">
  <input type="hidden" name="xmlcmd"/>
  <input type="hidden" name="back" value="/cognos"/>
  <input type="hidden" name="id" value="<%UP.id%>"/>
</form>
</body>
</html>

<xsl:template name="getListItem">
  <xsl:param name="list" select="''" />
  <xsl:param name="delimiter" select="'|'" />

  <xsl:choose>
    <xsl:when test="string-length($list) = 0" />
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="contains($list, $delimiter)">
          <xsl:call-template name="formatItem">
            <xsl:with-param name="itemValue"><xsl:value-of select="substring-
before($list,$delimiter)" /></xsl:with-param>

```

```

        </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
        <xsl:call-template name="formatItem">
            <xsl:with-param name="itemValue"><xsl:value-of select="$list" /></xsl:with-param>
        </xsl:call-template>
    </xsl:otherwise>
</xsl:choose>

<xsl:call-template name="getListItem">
    <xsl:with-param name="list" select="substring-after($list, $delimiter)" />
    <xsl:with-param name="delimiter" select="$delimiter" />
</xsl:call-template>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<!-- Actually formats your comma separated stuff -->
<xsl:template name="formatItem">
    <xsl:param name="itemValue"/>
    <xsl:variable name="currentItem" select="*[./Id=$itemValue]"/>
    <xsl:call-template name="formatUpfrontItem">
        <xsl:with-param name="currentItem" select="$currentItem"/>
    </xsl:call-template>
</xsl:template>

<xsl:template match="NewsItem|NewsBox">
    <xsl:variable name="currentItem" select="."/>
    <xsl:call-template name="formatUpfrontItem">
        <xsl:with-param name="currentItem" select="$currentItem"/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="formatUpfrontItem">
    <xsl:param name="currentItem"/>
    <tr>
        <td>
            <xsl:choose>
                <xsl:when test="name($currentItem)='NewsBox'">
                    <xsl:attribute name="class"><xsl:value-of select="'nbElement'"/></xsl:attribute>
                </xsl:when>
                <xsl:when test="name($currentItem)='NewsItem'">
                    <xsl:attribute name="class"><xsl:value-of select="'niElement'"/></xsl:attribute>
                </xsl:when>
            </xsl:choose>
            <A>
                <xsl:choose>
                    <xsl:when test="name($currentItem)='NewsBox'">
                        <xsl:attribute name="href">javascript:getFolder("<xsl:value-of
select="$currentItem/Id"/>")</xsl:attribute>
                        <xsl:attribute name="class"><xsl:value-of
select="'nbElement'"/></xsl:attribute>
                    </xsl:when>
                    <xsl:when test="name($currentItem)='NewsItem'">
                        <xsl:choose>
                            <xsl:when test="$currentItem/ProviderType='CQR'">
                                <xsl:attribute name="href"><xsl:value-of
select="$currentItem/Action/URL[./Name='Run']"/></xsl:attribute>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:attribute name="href"><xsl:value-of
select="$currentItem/Action/URL[./Name='View']"/></xsl:attribute>
                            </xsl:otherwise>
                        </xsl:choose>
                        <xsl:attribute name="class"><xsl:value-of
select="'niElement'"/></xsl:attribute>
                        <xsl:attribute name="TARGET"><xsl:value-of
select="'contentDiv'"/></xsl:attribute>
                    </xsl:when>
                </xsl:choose>
                <xsl:value-of select="$currentItem/Name"/>
            </A>
        </td>
    </tr>
</xsl:template>

<xsl:template name="build-path">
    <xsl:param name="current-node"/>
    <xsl:param name="template"/>
    <xsl:if test="$current-node">
        <I><a class="nbHeader">
            <xsl:attribute name="href">
                %SYSTEM.UpfrontPath#html%
                <xsl:value-of select="'?xmlcmd=<GetPage><Template>'"/>

```

```

        <utml:escape escape-style="encode-canhtml"><xsl:value-of
select="$template"/></utml:escape>
        <xsl:value-of select="'</Template></GetPage>&id='"/>
        <utml:escape escape-style="encode-canhtml"><xsl:value-of
select="$current-node/Id"/></utml:escape>
        </xsl:attribute>
        [ <utml:escape escape-style="encode-canhtml"><xsl:value-of select="$current-
node/Name"/></utml:escape> ]
        </a></I>
        <xsl:if test="$current-node/NewsBoxPath">
        <xsl:value-of select="' - '"/>
        <xsl:call-template name="build-path">
        <xsl:with-param select="$current-node/NewsBoxPath" name="current-
node"/>
        <xsl:with-param select="$template" name="template"/>
        </xsl:call-template>
        </xsl:if>
    </xsl:if>
</xsl:template>
</utml:presentation>

```

Listing 2

```

<utml:content>
    <DescribeNewsBox RequestId="1" numlevels="1">
    <utml:switch>
        <utml:case test="%id%"><Id><%id%></Id></utml:case>
        <utml:default><Id><%SYSTEM.BaseNewsBoxId#safe%></Id></utml:default>
    </utml:switch>
        <Module Name="ProviderTypeDescriptor"/>
        <Module Name="Base"/>
    </DescribeNewsBox>
</utml:content>

<utml:presentation RequestId="1" >
<xsl:variable name="root" select="." />
<html>
    <head>
        <title>Upfront - [<xsl:value-of select="Name"/>]</title>
        <style>
            TR { font-family:Tahoma; font-size:11; font-weight:bold; }
            .nbElement { color:#009900; }
            .niElement { color:#00CC00; }
            FONT.nbHeader { font-family:Tahoma; font-size:10; font-weight:bold; color:#006600; }
            TD.arrows {
                cursor:hand; background-color:#ffff99;
                border: thin inset #6699FF;
                padding: 2px;
                margin: 2px;
            }
        </style>
    </head>
    <body>
        <font class="nbHeader">
            [<xsl:value-of select="Name"/>]
        </font><BR>
        <TABLE border="0">
            <TR>
                <TD rowspan="2">
<SELECT name="selectionList" size="10" onClick="select_onClick()" >
<xsl:choose>
    <xsl:when test="Property/Value[./Name='SortOrder']">
        <xsl:call-template name="getListitem">
            <xsl:with-param name="list"><xsl:value-of select="Property/Value[./Name='SortOrder']"
/></xsl:with-param>
            <xsl:with-param name="delimiter" select="'|'"/>
        </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
        <xsl:apply-templates select="(NewsItem|NewsBox)">
        </xsl:apply-templates>
    </xsl:otherwise>
</xsl:choose>

</SELECT>
</TD>
<TD valign="bottom" onMouseDown="moveUp()" class="arrows">Up</TD>
</TR>
<TR>
<TD valign="top" onMouseDown="moveDown()" class="arrows" >Down</TD>

```

```

</TR>
<TR>
<TD valign="middle" align="center" onMouseDown="cancel()" class="arrows" >Cancel</TD>
<TD valign="middle" align="center" onMouseDown="setOrder()" class="arrows" >Save</TD>
</TR>
</TABLE>
<br/>

<script language="JavaScript">
var selected = "";
function setOrder(){
    strurl = "<SetNewsBoxProperties><id><%UP.Id%></id>";
    strurl = strurl + "<Property option='set'><Name>SortOrder</Name><Value>";
    for(var i=0; i<((selectionList.length)-1); i++){
        strurl = strurl + selectionList.options[i].value + "|";
    }
    strurl = strurl + selectionList.options[selectionList.length-1].value +
"</Value></Property></SetNewsBoxProperties>";
    actionPost.xmlcmd.value = strurl;
    actionPost.back.value = "/cognos/cgi-
bin/upfcgi.exe?xmlcmd=<GetPage><Template>main.utml</Template></GetPage>&id=<%UP.Id%>";
    actionPost.submit();
}
function cancel(){
    actionPost.xmlcmd.value="<GetPage><Template>main.utml</Template></GetPage>";
    actionPost.back.value="";
    actionPost.id.value="<%UP.Id%>";
    actionPost.submit();
}
function select_onClick(){
    selected=selectionList.selectedIndex;
}
function moveUp(){
    if (selected>0){
        var locoptiontxt = selectionList.options[selected-1].text;
        var locoptionval = selectionList.options[selected-1].value;
        var locoptionseltxt = selectionList.options[selected].text;
        var locoptionselval = selectionList.options[selected].value;
        selectionList.options[selected-1].text = locoptionseltxt;
        selectionList.options[selected-1].value = locoptionselval;
        selectionList.options[selected].text = locoptiontxt;
        selectionList.options[selected].value = locoptionval;
        selectionList.options[--selected].selected = true;
    }
}
function moveDown(){
    if (selected<selectionList.length){
        var locoptiontxt = selectionList.options[selected+1].text;
        var locoptionval = selectionList.options[selected+1].value;
        var locoptionseltxt = selectionList.options[selected].text;
        var locoptionselval = selectionList.options[selected].value;
        selectionList.options[selected+1].text = locoptionseltxt;
        selectionList.options[selected+1].value = locoptionselval;
        selectionList.options[selected].text = locoptiontxt;
        selectionList.options[selected].value = locoptionval;
        selectionList.options[++selected].selected = true;
    }
}
</Script>
<form name="actionPost" method="POST" action="<%SYSTEM.UpfrontPath#html%>">
    <input type="hidden" name="xmlcmd"/>
    <input type="hidden" name="back" value="/cognos/" />
    <input type="hidden" name="id" value="<%UP.id%>" />
</form>
</body>
</html>

<xsl:template name="getListItem">
    <xsl:param name="list" select="" />
    <xsl:param name="delimiter" select="|" />

    <xsl:choose>
        <xsl:when test="string-length($list) = 0" />
        <xsl:otherwise>
            <xsl:choose>
                <xsl:when test="contains($list, $delimiter)">
                    <xsl:call-template name="formatItem">
                        <xsl:with-param name="itemValue"><xsl:value-of select="substring-
before($list,$delimiter)" /></xsl:with-param>
                    </xsl:call-template>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:call-template name="formatItem">
                        <xsl:with-param name="itemValue"><xsl:value-of select="$list" /></xsl:with-param>
                    </xsl:call-template>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:otherwise>
    </xsl:choose>

```

```

        </xsl:call-template>
    </xsl:otherwise>
</xsl:choose>

    <xsl:call-template name="getListItem">
        <xsl:with-param name="list" select="substring-after($list, $delimiter)" />
        <xsl:with-param name="delimiter" select="$delimiter" />
    </xsl:call-template>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<!-- Actually formats your comma separated stuff -->
<xsl:template name="formatItem">
    <xsl:param name="itemValue"/>
    <xsl:variable name="currentItem" select=".*[./Id=$itemValue]"/>
    <xsl:element name="OPTION">
        <xsl:attribute name="value"><xsl:value-of select="$itemValue"/></xsl:attribute>
        <xsl:value-of select="$currentItem/Name"/>
    </xsl:element>
</xsl:template>

<xsl:template match="NewsItem|NewsBox">
    <xsl:variable name="currentItem" select="."/>
    <xsl:element name="OPTION">
        <xsl:attribute name="value"><xsl:value-of select="$currentItem/Id"/></xsl:attribute>
        <xsl:value-of select="$currentItem/Name"/>
    </xsl:element>
</xsl:template>

</utml:presentation>

```