Customizing Notifications
Path-Specific Notifications in KM

Applies To:

NetWeaver Enterprise Portal version EP6 or later.

Summary

The Knowledge Management component of the NetWeaver Enterprise Portal generates various e-mail notifications associated with certain events such as subscribed document changes and workflow. The existing mechanism may be customized and may be extended, for example, to allow different notifications to be generated depending on the path of a subscribed resource. This article demonstrates some of the possibilities for customization of subscriptions. Modifying the standard files, however, burdens the customer with maintaining this customization.

By: Darin Krasle

Company: SAP AG

Date: 25 Jan 2006

Overview

The notification service uses a mixture of technologies. The handling of the eventing and generation of e-mails is performed within Java classes. The content of the notification is generated using XSL and XML. This article examines some of the customization possibilities for subscriptions available by changing the XSL and XML without changing any Java. The content of notifications can be changed to show different information than they currently do and this information can be made dependent on the properties related to the event and associated resource that the Java classes deliver to the XSL transform as parameters. There are several levels of customization possible depending on the extent to which you are willing to make changes. For instance, it is possible to change the messages and properties included therein by merely changing the XML files. The set of properties made available from the Java side is fixed, but rich enough to allow interesting extensions such as making notifications path-dependent if one is willing to modify both the XML and XSL. Unfortunately, because the XSL only generates the content of the e-mail body, properties of the e-mail notifications such as the subject line cannot be influenced by the XSL. The examples described relate to notifications generated “on every event” rather than the periodic summaries, also known as “bundled notifications”, that are composed somewhat differently. These are discussed in a separate section. In any case, modifying any of the subscription files creates maintenance issues. Applying upgrades could overwrite any changes you have made. Hence, it is important to keep track of any customizations you have made and check that they are still intact after any upgrades.

XML / XSL

The content of the subscription notifications are generated using an XML / XSL transformation utilizing some additional parameter input provided by Java. There is an XML file for each supported language which contains the language-dependent texts. The texts are organized in “textspan” tags which allows them to be enhanced with document-specific information. There are 2 basic types of texts, those that are general and those that are event-specific. The general texts are fragments that may appear in various portions of the notification, such as the reminder at the bottom that the user should not reply to the automated mail. The event-specific textspans specify attributes which allow them to be properly associated with the event type and properties of the subscribed resource. For instance, a notification for a changed folder may need to be different than for a changed document. Hence, separate textspans are specified for these. A textspan may include references to properties associated with the subscribed resource and event. Some references may also insert HTML links which allow actions to be performed, such as calling up the Details page for the subscribed resource. Options for customizing the XML are discussed in the next section.

The XSL stylesheet contains the “smarts” for generating the message and consists of 3 main parts: the parameter block, the message structure, and the parameter rendering library. It makes sense to first describe the XSL in order to understand the XML. The XSL message structure part is the most interesting candidate for customization and is described last. Note that it is also possible to do lots of customization by only changing the XML, but it is important to first understand a bit about how the XSL works.

XSL Parameter Block

The parameter block defines which resource-specific data gets passed into the transform as parameters. The data defined in the parameter block is the only resource-specific data that can come across the API since this is hard-coded on the other side. There is no point in changing this unless the calling code is changed to deliver more parameters. Hence, notifications can only contain the properties listed here and no additional properties are available. When customizing, it will be necessary to test your stylesheet changes. It is easy to test stylesheet changes without using the portal, but you will need to either have a HTML test page with JavaScript to set the sample data (hint: use the addParameter method of the XSL processor) or you have to manually enter sample data in the parameter block in this way:

```
<xsl:param name="resourcename" select="'resourcename'" />
```

Testing the modified files in the portal is also not difficult since you should be able to replace the existing files without having to do a restart. Manually generating the notifications by performing the associated actions on resources is the tedious part. If you test with a client page, you may have to comment out the output encoding tag while testing.

Library

The rendering library provides a set of little templates that can be referenced to insert parameter values and links into textspans in a manner similar to markup. The library provides both "primitive" templates which render just the text value of a parameter, as well as formatted templates that will formulate HTML links. Since we wanted to use the same XML textspans for HTML and non-HTML channels, we built in a switch in the rendering which will control the formatting based on whether the output format is HTML or not. The use of this is described later. The following table shows the most recent list of templates:

Primitive Templates	Associated Formatted Template
resourcename	resourcelink
contentaccessurl	newresourcelink
resourceDescription	
parenturl folderaccessurl	parentlink
notificationtext	
eventid	
eventname	
lastmodified	
lastmodifieduser	
subscriptionname	subscriptionlink
subscriptiondetailsurl	subscriptiondetailslink
administrationurl	administerlink
deleteurl	deletelink
mailto	mailtolink
detailsurl	detailslink

The meaning of most of the templates are relatively self-explanatory. Some templates are only relevant to specific event types. For instance, the newresourcelink template only makes sense when a resource is moved or renamed. As time goes by the templates may be changed or extended. If your installation is older, may have fewer available than are listed here.

I would not recommend altering these templates unless you know your XSL well, since there are a lot of consistency points that you have to observe.

XSL Message Structure

This is the meat of the XSL, and is the ONLY part that is really interesting to change. This portion defines the basic structure of the message, leaving the language-dependent portions and property references to the XML. In the standard e-mail notification, it may look as if everything is doubled - well it is! That is because it includes both a plaintext and an HTML version. This ensures that it will be somewhat readable in a non-HTML mail client. When processing the textspans, the outputFormat XSL parameter is used to pick the formatting:

```
<xsl:with-param name="outputFormat" select="'plaintext'"/>
```

For formatted templates, using "plaintext" will cause the URL to be listed in parentheses. "plaintextsms" will omit the URL entirely.

In the message structure, the available parameters may be used in logic expressions to do special stuff depending on event and resource-specific values. You can see this where a different message is generated depending on whether the isOwner parameter is set or not. Conceivably, you could program a different message for every parameter combination. You could even add attributes to the textspans in the XML to implement a more complex matching scheme as an advanced customization.

The XSL message structure is the main area for customizations since you have so much flexibility over the HTML generated. It is important to make sure that if you introduce any new text strings that need to be localized, that you also define them in the XML – this means adapting each language XML file.

You could potentially add all kinds of HTML and inline CSS and whatnot to make the notification prettier. We kept this to a minimum to try to keep the standard delivery simple. Be careful about adding links to images and stylesheets, since these may not be available to the client machine if it is not connected to the network. Many popular mail clients may not display images you reference due to security settings. If you do choose to reference images, it is advisable to reference them as a CSS background image (hint: use “url()”), since this may prevent the ugly red X if it cannot be loaded. Another gotcha is that if you define CSS classes in the header tag, it may be hard to test with a simple HTML page doing the transform on the client since you may only be replacing the body tag.

Quick and Easy Customization of the XML

The texts that are shown in the message are broken into tidbits in the XML. Some of these texts may be shown in every subscription notification, regardless of the event type. Additionally, for each individual event case and supported language, a text is defined. Both types of texts may incorporate resource-specific information, using a scheme that resembles HTML markup. Different languages can also reference different information. For instance, if all the resource descriptions are in English, you could have them only shown in English notifications. The texts are encoded into textspan tags in the XML. You can insert property values into the texts by referencing any of the templates listed in the templates table in the previous section. A reference is formulated as a tag and the tags work in different ways. Let us look at a concrete example to make this clear. When a subscribed document gets renamed, this textspan gets used:

```
<textspan isCollection="false" pEventid="rename"><resourcename/> in folder
<parentlink/> renamed as <newresourcelink/>.
(<detailslink>Details</detailslink>)</textspan>
```

The <resourcename/> tag will insert the former name of the resource into the message as plain text. In contrast, the <parentlink/> tag will insert an link, allowing the user to call up the containing folder. Some tags allow static text to be used for the link text, such as the use of “Details” in the <detailslink> tag shown here. In the German XML file, <detailslink>Eigenschaften</detailslink> would be used since this is the translated version. The tags that allow this are detailslink, deletelink, subscriptiondetailslink, administerlink and mailtolink. Only simple text is allowed in these tags; they may not be nested.

To continue with our example, let’s say we wanted to add information about who renamed the document. We can do this by adding in the <lastmodifieduser/> tag in the following manner:

```
<textspan isCollection="false" pEventid="rename"><resourcename/> in folder
<parentlink/> renamed as <newresourcelink/> by <lastmodifieduser/>.
(<detailslink>Details</detailslink>)</textspan>
```

For the other language XML files, we could have used a different sequence to better fit the target language grammar. With respect to the various language files, if we only change the content of the textspans, it does not matter which properties we reference or in what order. Technically, these need not be consistent across the languages.

It is also worth noting that the XSL will not be able to process all of the named entities one knows from HTML. For instance, trying to use   will cause problems. There are generally numeric substitutes for these such as   for .

Path-Specific Notifications

This section describes two methods for producing path-specific notifications. This description is derived in part from the successful implementation of a scheme by Bernd Gerlach-de Campos, SAP Consulting Germany. The first method involves changing only the XSL. This method may be used when the notification e-mail should look different for different content areas, but the text of the notification does not vary. This would be useful for “branding” of different content areas by using different static content at the top of the e-mail. The second method allows completely different e-mails to be generated and involves modifying both the XSL and the XML, including the structure thereof.

Method 1: Changes to the XSL

Method 1 allows the XSL message structure to be path-specific. SUBSCRIPTION_EMAIL.xsl is the subscription XSL we will need to modify. We will need to extend the current scheme to choose which variant of the message structure should be used. The appropriate XSL construct for this is xsl:choose:

```
<xsl:template match="/">
      <xsl:choose>
            <xsl:when test="starts-with($resourceurl, '/prefix1')">
                  [... Message Structure for /prefix1 area ...]
            </xsl:when>
            <xsl:when test="starts-with($resourceurl, '/prefix2')">
                  [... Message Structure for /prefix1 area ...]
            </xsl:when>
            <xsl:otherwise>
                  [... Default Message Structure ...]
            </xsl:otherwise>
      </xsl:choose>
</xsl:template>
```

In this example, the xsl:choose tag was placed immediately within the root match template. This offers the maximum flexibility since each xsl:when contains a complete message structure. Alternatively, the xsl:choose tag could have been placed at a deeper level to reduce duplication within the code. This conditional structure could even be used in several places within the message structure and could also reference additional properties in the when expression. Obviously, the “/prefix1”, etc. need to be replaced with the actual paths to your content areas of interest.

Method 1: Changes to the XML

When using method 1, it is not strictly necessary to change the XML. Only if new texts that need to be localized are added is it necessary to change the XML. This would be in SUBSCRIPTION_en.xml for English. The last 2 characters before the extension indicate the associated language. The most appropriate place to insert new elements would be between the greeting and eventinfo elements. Any changes to the XML , particularly additions, should be propagated for each language that could potentially be used.

Method 2: Changes to the XML

For Method 2, it makes sense to discuss changes to the XML first. SUBSCRIPTION_en.xml is the subscription XML for English. In the current scheme, only a single version of the notification is available and the texts are defined in this XML file. If we want to scale this to allow multiple variants, we will have to add an additional 2 layers to the structure so that we can have more than one variant in the file. Two layers are necessary since there can only be a single root node. We could actually do it by only adding a single level, but we would also have to change all the path references in the XSL. I will leave this optimization as an exercise for the reader. The new structure should look as follows:

```
<root>
<area kmPath="/prefix1">
      <notification>
            [... textspans for /prefix1 area ...]
      </notification>
</area>

<area kmPath="/prefix2">
      <notification>
            [... textspans for /prefix2 area ...]
      </notification>
</area>

<area kmPath="default">
      <notification>
            [... default textspans ...]
      </notification>
</area>
</root>
```

In the example above /prefix1 and /prefix2 are just example placeholders for the actual paths to the content areas. There can be an arbitrary number of these, but one drawback is that different paths that should use the same textspans must duplicate the whole structure.

Method 2: Changes to the XSL

There are two changes in the XSL necessary for method 2. First, we must reformulate the templates to match to the appropriate notification element to reflect the new structure of the XML. This will make the rest of the XSL find the proper textspans in a transparent manner. Second, we will need to put in the logic exactly as in method 1 to choose the appropriate message structure. For the specifics of the second change, see the section “Method 1: Changes to the XSL” above.

For the first change we need to replace this existing code:

```
<xsl:template match="/">
<xsl:output encoding="UTF-8" />
```

With this:

```
<xsl:template match="/root">
  <xsl:output encoding="UTF-8" />
  <xsl:choose>
    <xsl:when test="count(area[starts-with($resourceurl, @kmPath)])>0">
      <xsl:apply-templates select="area[starts-with($resourceurl,@kmPath)]"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="area[@kmPath='default']"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="area">
```

This will cause the XSL processor to locate a matching area in the XML and if none is found, cause it to use the default area.

Aspects to Consider and Frequently Asked Questions

Accessibility

The standard notifications contain a generous number of tab indexes. This is to ensure they will work well when used in conjunction with screen reader software. It is hence important to note that modifications to the notifications may impact accessibility. Improving the accessibility of the notifications was one of the main changes made since EP6.

Approval Notifications

The approval notifications are generated in almost the exact same way as the subscriptions.

Bundled Notifications

There are also two different kinds of notifications: individual and bundled. Bundled notifications are aggregations of notifications occurring over a duration. The mechanism used for creating the bundled notifications is more complicated than for the individual notifications since the XML must be built dynamically by the Java code. The concepts in this article may also be applied to some degree to bundled notifications, but this would be somewhat more complex since it would be probably be desirable to group sets of notifications from certain areas. This would involve adding more logic in the XSL. There are also currently no provisions in the Java for creating separate e-mails based on the paths of the resources.

Hence, without changing the Java, it is difficult to do more than grouping the items within the bundled notification and determining how those are formatted.

Character Encoding

I strongly recommend you do not change character encoding. Changing this may cause things to break that are not obvious - specifically characters that may be expected in other languages. It may be necessary to disable character encoding when testing using a client page. Don't forget to put it back in when you install the files in the portal.

E-Mail Header

This article only describes how to customize the contents of the mail. The Subject line and other properties cannot be influenced by the XSL, and are more or less hard-coded in the Java. There are a couple of other aspects that may be influenced by configuration, but this is outside the scope of this article.

HTML-Compatible Mail Clients

Be careful in customizing the messages to be purely HTML. This may not look very nice on some mail clients. The standard notifications include a header in plaintext equal to the HTML content to try to best support the variety of mail clients. Verfify the assumption that all users have HTML-compatible mail clients before eliminating this.

Links

Be careful when including links in notifications. Targets in the portal require both access to the portal and appropriate permissions. Similarly, Internet links require an active Internet connection. This might cause grief for mobile users. This applies to all kinds of links – both images and CSS stylesheets. This is why the standard notifications links no CSS. Also be aware that mail clients often suppress images for security and privacy reasons. Referencing images as background images via in-place CSS is better than using image tags when the image cannot be displayed.

Prototyping

The best way to customize subscriptions is to use a client-based HTML page for testing. Then you can customize the notifications completely outside the portal and KM on any PC with a suitable browser (supporting XSL transformation). This will involve having to change some stuff in the header that you will have to change back when you put it back in the portal. Specifically, one thing you will need to change is putting in some dummy values for the parameters so they are not blank. You may also need comment out the output encoding, but don't forget to turn it back on afterward. There may also be subtle differences in the client XSL processor you use and that in KM.

If you want to do extensive customization, a client page is the best way to get an efficient dev/test cycle. If you have problems you may try a more recent (or even older) XMLDOM control. Here is a test HTML page you might find useful. Don't forget you have to change the XSL header back and forth if you use this.

```
//This is a sample HTML file for client-based testing
<HTML>
  <HEAD>
    <TITLE>Test for Notification XML/XSL</TITLE>
  </HEAD>

  <XML id="source" src="SUBSCRIPTION_en.xml"></XML>
  <XML id="style" src="SUBSCRIPTION_EMAIL.xsl"></XML>
```

```
  <SCRIPT FOR="window" EVENT="onload">
    xformed = source.transformNode(style.XMLDocument)
    //alert(xformed)
theTimer=setTimeout('xslTarget.innerHTML = xformed',1000);
xslTarget.innerHTML = '<BUTTON
onclick="clearTimeout(theTimer)">Hold</BUTTON><BR/><TEXTAREA
id="TAxslSourceTarget" name="TAxslSourceTarget" cols="80"
rows="100">'+xformed+"</TEXTAREA>";
  </SCRIPT>

<BODY id="xslTarget">
    <P STYLE="font-size:10pt; font-family:Verdana; color:gray">
      <B>This shows the transformed emails generated.</B>
    </P>
    <DIV id="xslTarget1"></DIV><BR/>
  </BODY>
</HTML>
```

I have seen more advanced versions that allow you to choose the transform and XML files, and also allow you to set the parameter values such as the path and event type. Such pages will typically instantiate an ActiveX XSL processor and use the addParameter method to pass the appropriate data. If using a client page is too complicated, the simplest way to test is to temporarily add a link to the XSL to the top of your XML like this:

```
<?xml-stylesheet type="text/xsl" href="SUBSCRIPTION_EMAIL.xsl"?>
```

Then you can just double-click the XML file.

Author Bio

Darin Krasle is a User Interface Design Specialist working in the Knowledge Management area at SAP AG in Walldorf, Germany.

Disclaimer & Liability Notice