AxCMS.net

Upgrading to AxCMS.net ver8.1.2

Versioning

It is possible to configure your system, that it creates minor snapshots (xml) for every change made on a page or a document.

There is a new key in appSettings of your web.config of management system:

<!-- switch versioning between None / OnlyMajor / Full -->

<add key="Versioning" value="Full" />

None = no versions are created

OnlyMajor = a html and xml snapshot is created when a page or a document is published

Full =

1. a html and xml snapshot is created when a page or a document is published

2. for each change on a page (label, image, dynamic element etc.) a xml snapshot is created

3. if a document gets updated (the file), a snapshot is created

Default setting is "Full", so if nothing is defined in web.config, your system will use VersioningMode.Full.

Please remove "versioning" attribute in publisher section, if you used it there, it has no effect anymore.

Please also check in your CreateApplication if AxVersionHistory table gets deleted in delete-funtion and please add AxVersionHistory sequence in sequence reset function.

Export definitions

In export definitions it is now possible to define if the export result should be saved to file with save dialog or opened in a new window and also extension of the export result file.

Complete element definition looks like this:

```
<Xslt Type="Axinom.AECMS.WebControls.AxPage"
MimeType="text/plain"
Serializer="Axinom.AECMS.page.PageContentSerializer, AxCMS.BL"
FileName="admin/ExportDefinitions/PageContentToCsv.xslt"
AxID="AxPageContentToCsv"
SaveDialog="true"
Extension=".csv"
Description="Page Content as CSV" />
```

Html Rules

HtmlRules for texts in CMS are now defined in CMSSite.xml, all tables and overview/detail pages about HTML-Tags and HTML-Attributes are deleted.

There is a new section in CMSSite.xml <HtmlRules>. An example for a default rule is:

```
<HtmlRules>
  <HtmlRule axid="default" tags="strong,b,br,ol,ul,li,hr,u">
    <Tag name="A" attributes="href" styles="default">
     <Attribute name="target" values="_blank, _top, _self, _parent, _media" />
     <Attribute name="track" values="0, 1" />
    </Tag>
    <Tag name="P" attributes="align" styles="default" />
    <Tag name="SPAN" styles="default" />
    <Tag name="FONT" attributes="style, color" />
```

```
 </HtmlRule>
 <StyleDefinition axid="default" values="highlight, attention, highlightbold,
attentionbold, articleheadline, linkbold, error">
  <Value value="headline" property="Details.Caption" />
  <Value value="source" property="Details.Origin" />
  <Value value="description" property="Details.Description" />
  <Value value="validon" property="Details.ValidOn" />
  <Value value="validfrom" property="Details.ValidFrom" />
  <Value value="validuntil" property="Details.ValidTill" />
  <Value value="unittestheadline" property="Details.Caption" />
 </StyleDefinition>
</HtmlRules>
```

Your rule has an unique axid. All tags that have no attribute can be added directly in "tags" attribute of <HtmlRule> element. Other tags can be defined as children of <HtmlRule>. They have a name= the tag itself. If they have attributes, that have no defined values, they can be added directly to "attributes"-attribute of <Tag>-tag. If the <Tag>-tag has an attribute "styles", then also "class" is allowed. Inside the "styles"-attribute the according axid of a StyleDefinition is inserted.

All attributes which have defined values can be added as children of <Tag>-tag. They have as name the attribute value and as "values" the allowed values for that attribute. You can also add a "default"-attribute which defines the default value for this attribute.

Styles have an own Element "StyleDefinition". It is needed as style classes can be assigned to certain properties of a page. Each StyleDefintion has an unique axid. All styles which have no assigment can be added in "values"-attribute. The others can be defined as children of <StyleDefinition> element. Please use as property the name of your property. If you need to access a subobject like those from Details-object, separate the names with ".". In this approach you can also add your own developed properties there to be filled automatically.

You can define of course as much rules as you like. There should be one with axid "default", as this is applied by default. If you now need a special rule for a special placeholder, create it and in PlaceholderDefinition add its axid like this

<PlaceholderDefinition axid="myplaceholder" description="mydescription" htmlrule="myrule">

To convert your existing rules from DB to xml for CMSSite.xml call page:
http://cms.yoursystem.com/installation/HtmlRuleTransformation.aspx
PLEASE DO THIS BEFORE RUN DB CHANGE SCRIPTS!

Create Application changes

Please delete following functions in your Create Application in AxCMSApplication.cs:

DefineHTMLTags

DefineHTMLAttribute

DefineAttribute

CreateMappings

In your application class:

CreateHTMLRules

Also delete call in DeleteAll function:

DeleteTables("AxFormatValue,AxFormatRelation,AxFormatAttribute,AxFormat", _msConnectionManager);

CodeTemplates

It is now possible to define special kind of text that the editor should be able to add in text editor in CMS, e.g. usecase is in mailtemplates where you want to define your custom placeholder.

Therefore a new section in CMSSite.xml is introduced:

```
<HtmlCodeTemplates>
 <CodeTemplate axid="CompanyName" image="/templates/images/codeicon.gif" hint="Company name">
  <Code>
   <![CDATA[ <b>Test GmbH & Co. KG</b>]]>
  </Code>
 </CodeTemplate>
</HtmlCodeTemplates>
```

You can then define in your placeholder definition which code template is used there (comma separated)

```
<PlaceholderDefinition axid="BasePlaceholder" description="Base Placeholder" codetemplates="CompanyName">
  <ContentRules></ContentRules>
  <ElementTemplates>
    <ElementTemplate axid="20" atLeast="0" atMost="100"></ElementTemplate>
  </ElementTemplates>
 </PlaceholderDefinition>
```

Your codetemplate should have a unique name, the path to icon should be absolute to root folder and you can add a hint. The codetemplate is then added as own button in text editor. When the user clicks the button, the text defined in <Code>-tag is added at the end of the current text.

Reports

Reports categories are replaced by service tasks. Each report represents a service task running on background with the specified schedule.

For report subscription new link is introduced on user detail page. All classes inherited from AxReport in all assemblies available will be shown for subscription. To provide friendly name for your reports, use translation.

Report subtree is deleted with DB change scripts. To convert your report subscriptions to service tasks use this page:

http://cms.yoursystem.com/installation/ReportSubscription.aspx

Axinom.AECMS.GenerateReportsActivity service task is no longer used. AxReportFactory class is deleted. If you classify user to report categories during create process, please replace it with service task scheduling.

Checkpoints

Old approach to call Check for checkpoints is obsolete:

```
(new AxElementCheckPoint(ElementCheckPointID.PageApprove)).Check(user, element);
```

New approach:

```
(new AxElementCheckPoint(ElementCheckPointID.PageApprove, element)).Check(user);
```

This allows moving common functionality from all checkpoints to base checkpoint class.

Alerts

Users may now subscribe to e-mail alerts on various CMS activities. To add alert to your CMS activity, override AlertOnActivity property and set it to true. This will display your activity on Reports & Alerts page of user details and let users to subscribe to alerts when activity is done on element in selected categories. To provide friendly name for your activities, use translation.