CrownPeak™

TECHNICAL DOCUMENTATION

DEVELOPER REFERENCE GUIDE FOR CLASSIC API

CROWNPEAK CMS

February 2012

ENSURING YOUR WEB CONTENT SUCCESS

TABLE OF CONTENTS

INTRODUCTION

This Developer Reference Guide is intended to provide a reference for CMS developers making updates to existing templates or creating new ones.

Readers of this document should have some familiarity using the CMS as a regular end user. You should have logged into your custom CMS instance and used the CMS to create, edit, and publish content to your live Web site. As an end user, you should have read the CrownPeak CMS User Guide, which explains the interface and concepts of CMS's in general.

Once you are familiar with the uses of the CMS, you should review the Administrator and Developer documentation. The Developers Training Guide is especially helpful. It is a step by step tutorial in template development and system configuration. Additionally, you should review the Admin Guide, The Design Patterns document, and the API Reference Guide. Being familiar with these guides will make concepts in this reference manual easier to digest. Also, the Common Actions document has tips on development tasks we have encountered regularly when implementing CMS instances.

All of this documentation should be available in the Help menu of your CMS instance. If a guide is not available to you, please contact your CrownPeak Administrator to make it available.

At CrownPeak Technology, we take great pride in the CrownPeak CMS development environment. It is the same interface that we use to develop all of our projects, so we welcome any suggestions for improvement!

TEMPLATES AND TEMPLATE FILES

Any code that is written in CrownPeak CMS lives in template files and uses VBScript with many higher-level CMS API functions. VBScript was chosen since it is easy to learn, has widespread adoption, can be tested with no explicit compilation, and has a nice sandbox component available so that multiple developers on the same machine are protected from themselves.

The rest of the system administration is managed via the GUI interface, so there are no configuration files needed.

DEFINITION OF A TEMPLATE

With CrownPeak CMS, the definition of a Template is a collection of template files that define the input form, the output forms, and optional event specific code. Templates are normally found in the /System/Templates folder, and each Template is a folder since it may contain several template files.

From the /System/Templates folder, you can use the File > New > Template menu to make a new Template folder. Inside a Template folder, the File > New menu will list a majority of the available template files.

TEMPLATE FILES

Within a Template, there are one or more template files. Each contains HTML and CMS code to define the input forms for content, the final output forms, and any event code needed to support the Template. None of the template files are required, and different types of Templates may use different template files. For example, a Press Release Template might have both input.asp and output.asp template files, while the Press Release Index Template would only have an output.asp file (since the input is all done at the Press Release level). The Index is generated by gathering a list of individual Press Releases.

Of course, you can also have mixed pages with some automatic content and some fielded content. A good example is a typical home page template where the input.asp input form defines a message of the day, while the rest of the page is drawn from recent additions or slotted material from other parts of the CMS.

This example also highlights an important point: content is not limited to one template. Templates can use content in any asset or folder, and assets can be managed by several Templates. Template files, then, are grouped by typical usage, but many variations are possible.

The following is a list of the available template files, what they are used for, and typical code samples:

↗ input.asp

The input.asp template file is used to define the input forms for entering content. It is mostly HTML, defining rows of field name/value pairs. The CMS code involved is used to populate any fields with existing values or default values since this same form is used for creating new as well as editing existing assets. DHTML can be used in this form to show/hide fields or provide other functionality if required.

Much of the code for the input fields can be automatically generated using the Code Wizard in the Template Editor. The Code Wizard is invoked by clicking on the "sorcerer's hat" icon, which is the last icon in the second row of buttons in the Template Editor.

Note that the input.asp file is already in a HTML and <TABLE> context, so only table rows are required in your code.

↗ output.asp

The output.asp template file is used to define any final content that is published via FTP (or other protocols as configured) to the Web server. These are typically HTML files with embedded field display code, but could be CSV data files, XML etc. They may also include any kind of server side code (PHP, ASP, server-side includes, etc.) being passed through (and optionally modified).

Since the CMS uses VBScript, any server side ASP code needs to be escaped by replacing all <% and %> tags with <$ and $> tags. All other server-side languages can be simply placed inline like the HTML, although they will not be parsed by the CMS, and thus should probably be wrapped in the CMS "isPublishing" conditionals.

The output.asp file will also be used as the preview if no preview.asp is defined.

↗ preview.asp

The preview.asp template file is used to define the preview of a Template that is separate from the output.asp. This is typically used if you want to focus the author's attention on a specific portion of a page (for example, just the poll, not the whole page), or to remove any extraneous DHTML or server-side code that makes the preview look messy.

Another use is to have the preview plus some internal information for author use only, like a more fielded view of the content.

The contents of preview.asp template files are typically very similar to output.asp.

↗ post_input.asp

The post_input.asp template file is used to process the input.asp form submissions before the data is saved. It is triggered when the user clicks on "Save", “Save and New”, “Save and Edit”, or "Save and Preview" on an input form. Common uses include performing field checking/returning error messages and setting the _CMSLabel for the asset.

It is also often used to create derived fields or process existing fields.

Another use is to update assets in other folders, but remember that the workflow is set on the current asset only, so when you modify a separate asset, workflow is essentially being circumvented.

This file is mostly VBScript code, and has no visible component, and thus will not contain any HTML, except as coded into the error messages.

Typical code for field validation:

```
<%

 if content.item("contact_phone") = "" then
content.add "_CMSError", "Please enter an Event
Contact Phone."

 if content.item("organizer") = "" then content.add
"_CMSError", "Please choose an Event Sponsor."

%>
```

This code will return errors one by one. To return them all as a group, you would do something like this:

```
<%

  if content.item("contact_phone") = "" then
content.add "_CMSError", content.item("_CMSError")
& "<br>" & "Please enter an Event Contact Phone."

  if content.item("organizer") = "" then
content.add "_CMSError", content.item("_CMSError")
& "<br>" & "Please choose an Event Sponsor."

%>
```

↗ filename.asp

The filename.asp template file is used as the last stop in naming an asset as it gets published. In many cases, the Publishing Properties in the system can be used to create the final path and file name. However, just before an asset gets published, this template file is called to allow the file name and path to be further modified if required.

Usually your assets will either rely on the Publishing Properties configuration or the filename.asp template file, but not both. The Publishing Properties for the asset (View > Properties > Publishing) will display a message notifying you if the asset's template has a filename.asp that might override your settings.

Typical uses of the filename.asp template file are to use dates for press release filenames rather than asset ids, making sure a file extension is added to file names, etc.

Some typical examples:

```
<% ` use the CMS id as the filename to avoid
namespace collision

content.add "_CMSpublishPath",
content.item("_CMSpropertiesFolder") &
content.item("_CMSRelativeFolder") &
content.item("_CMSId") & ".html"

%>
```

```
<% ‘ remove spaces from filenames
content.add "_CMSpublishPath",
replace(content.item("_CMSpublishPath"), “ “, “”)
```

The system variables useful in this context are documented in the API reference, as well as later in this document. In addition to the standard system variables like _CMSLabel, some other key variables are:

_CMSPublishPath: The final path and file name for the asset. This is the value that you are writing to.

_CMSRelativeFolder: If the folders in the CMS are useful in the Website path, you can use this variable to indicate the folder the asset is in. If you want to use a portion of the CMS path, you can parse the directories out of this variable.

_CMSPropertiesFolder: This variable gives access to the path info in the Publishing Properties information panels.

↗ assetfilename.asp

If assets have attached files, this template file is used just like the filename.asp template file to provide the ability to rename assets. This is not normally needed since the CMS will automatically name and link the files. However, it can be useful in cases where a template might be managing content to be drawn into a Flash file, where the naming of images must match the Flash file’s configuration.

↗ url.asp

The url.asp template file is used in cases where the file name and URL don’t match up on the Webserver. This is most often the case when the FTP root does not match the Webroot. The url.asp is used when other assets make links to the current asset, and need to know what the actual final URL will be. This is rarely used, and is similar in concept to the filename.asp template file (usually just a few lines of VBScript).

```
<%

content.add "_CMSPublishUrl",
content.item("fooWebRoot") & "/events/" &
content.item("_CMSId") & ".html"

%>
```

In this case, the fooWebRoot variable is set in the System>Configure>Variables screen, which is where general purpose global variables can be set.

↗ asseturl.asp

Similar to the url.asp template file, asseturl.asp allows you to rename links to asset files associated with the current asset.

```
<%

content.add "_CMSPublishUrl",
content.item("fooWebRoot") & "/events/images/" &
content.item("_CMSRemoteFilename")

%>
```

↗ upload.asp

The upload.asp template file is called when an asset is uploaded in an input form. This is typically used to resize or make thumbnails of any images being uploaded by the user.

```
<%

 Dim value

 value = content.item("_CMSUploadValue")

 if right(lcase(value), 4) = ".jpg" or
right(lcase(value), 5) = ".jpeg" or
right(lcase(value), 4) = ".png" then

  ' update the var to the scaled image


  if content.item("_CMSUploadVariable") =
"upload#image" then

   content.add "_CMSUploadValue",
image.scaleImage(value, "", 100, 100)

   ' create a tmp variable that contains its
thumbnails ... if you set it here already

   ' then canceling after an upload will cause the
variable to be updated when

   ' you did not want it to.

   content.add "image_thumbnail_tmp",
image.scaleImage(value, "_thumbnail", 50, 50)

  end if

 end if

%>
```

↗ smtp_import.asp

The smtp_import.asp template file is used to process content submissions via e-mail, similar to the input.asp and post_input.asp file. In the following example, the e-mail imports use XML to define the numerous fields. XML is not a requirement; for simpler schemas, a custom syntax can be easily used, though the XML DOM provides for easy parsing of the data.

E-mail imports are configured via the System > Configure > Import > SMTP screen.

```
<%
  Dim txt
  Dim fieldStart, fieldEnd
  Dim xml
  Dim fields
  Dim ltxt
  Dim key
  Dim value

  txt = trim(content.item("_CMSEmailBody"))
  ltxt = lcase(txt)

  fieldStart = inStr(ltxt, "<root>")
  fieldEnd = inStr(ltxt, "</root>")
  if fieldStart >= fieldEnd then
   content.add "_CMSError", "Invalid email format.
No XML data detected."
   exit Sub
  end if

  ' skip the end /root
  fieldEnd = fieldEnd + 6
  set xml = system.createXML()
  if not xml.loadXML(mid(txt, fieldStart,
fieldEnd)) then
   content.add "_CMSError", "Invalid XML format."
   exit Sub
  end if

  set fields =
xml.selectSingleNodeAsContent("root")
  if not isObject(fields) then
   content.add "_CMSError", "Invalid xml format.
Missing root /root node."
   exit Sub
  end if

  if isObject(fields) then
   for each key in fields
    value = fields.item(key)
    ' do any renaming/processing here
    if left(key, 8) = "category" then
     key = "categories" & mid(key, 9)
    end if

    ' skip over token
    if key <> "token" then
     content.add key, value
    end if
   next
  end if
%>
```

↗ delete.asp

The delete.asp template file is called when an asset is deleted from the CMS, at which time the functions specified in this file are run.

One possible use is in a scenario where a list of assets is displayed in an index, but the ID of each asset has been written to the index. In this case, if an asset was deleted, you would use the delete.asp template file to run a function that would remove the ID from the index.

↗ copy.asp

This is called when an asset is copied, cloned, etc. It allows you to modify or remove selected fields for the copied asset.

new.asp

This is used for initializing variables when a new page is created.

↗ frame.asp

This is used in conjunction with output.asp or preview.asp to define a frame-based layout.

↗ post_save.asp

Post_save.asp runs after the post_input.asp file is executed. The difference between the two is that **post_input.asp is executed before** the data is saved to the CMS database, where **post_save.asp is executed after** the data has been saved to the CMS database.

↗ post_publish.asp

This is called after the asset has been published. One possible use is to create an archived shortcut of an asset once it has gone live.

↗ ftp_import.asp

This is called when an import is triggered via the System>Configure>Import>FTP settings. An FTP import retrieves a file from a remote FTP server, and then processes that file to import the data into the CMS.

CSV or XML files are the typical formats used. CMS instances that require translation or the import of product data typically use FTP imports to bring that data in.

↗ smtp_insert.asp, smtp_update.asp, smtp_delete.asp, http_insert.asp, http_update.asp, http_delete.asp, odbc_insert.asp, odbc_update.asp, odbc_delete.asp, soap_insert.asp, soap_update.asp, soap_delete.asp

These files all relate to their respective protocols for publishing, and can be found under the **System** > **Configure** > **Export** menu. The ODBC and SOAP options are rarely used. The SMTP and HTTP methods are most often used for working with third-party e-mail providers.

↗ login.asp

Located in /System/login.asp, this is executed on each login to the CMS. It can be used to change group permissions, etc. on login.

INPUT FORMS

SPECIFYING FIELDS

If you need to design a more complex input form, filling out the following table may help to identify the required information:

Field Label	**Field Name**	**Field Type**	**Size & Options**	**Default Value**	**Validation**
The label used on the input form in front of the field.	*Optional:* Internal database field name (used if there is any data export in the system)	See chart below.	See chart below.	Use only when necessary.	Required: Y/N Also any data rules. Use only when necessary

Field Types and Required Options Chart

Field Type	**Size and Options**
Single Line Text ("Textbox")	SIZE – width of field in characters. This does not limit the data entry, just the display size. To limit the data entry, set the MAXLENGTH attribute to the number of characters you want to allow.
Text Area	ROWS, COLS – size in characters. Set the WRAP attribute to "virtual" to allow for text wrapping in the field.
WYSIWYG Text	WIDTH, HEIGHT – in pixels. WIDGETS – a list of what to include or not, depending on which is smaller. Widgets are configured in the User Group admin
Radio Buttons	NAME, VALUE – for each button. These can be the same if the names are short enough. The user sees the name, and the data holds the value. The NAME for each radio button in a group must be the same for them to function properly. Adding the word CHECKED to a radio button will select that value by default. A group of radio buttons should always have one value checked by default.
Check Boxes	NOT RECOMMENDED (see discussion in next section)
Select (drop down/pull down, multiple select) Menu	A list of values, or data source, contained within the SELECT tag. Each item in the list is contained in an OPTION tag, with a VALUE associated. Adding the word SELECTED to an OPTION will select that value by default. A select menu should always have one value selected by default. Adding the word MULTIPLE to the SELECT tag will display the list as a multiple select box, allowing the selection of multiple options simultaneously. A select menu

	(single or multiple) will always be as wide as its longest entry.

UNSUPPORTED FIELDS AND WORKAROUNDS

↗ Check Boxes

You can get check box processing code from the CMS developer environment Code Wizard. The code provided allows for the proper processing of check box data within your template. However, there is an alternative method to dealing with them if needed.

Check boxes are allowed in HTML, but the form submission by browsers does not indicate whether the user made a selection or chose the unselected state. Instead of using check boxes, you should use radio buttons in pairs or a single select menu with multiple options.

To cover this functionality, CrownPeak CMS has a graphical check box which can be implemented like this (see the API reference for details):

```
<% input.checkbox "home_page", "y",
content.item("home_page") %>
```

You may also use post_input to check and set.

If you need to use a check box, you should **always use the CMS check box** generated by the Code Wizard.

↗ Multiple Selects

Similar to check boxes, if the user unselects all the items and submits, the browser sends no data. Thus, if you previously had some items selected, the data will be unchanged rather than cleared out. To explicitly remove all the previous selects, use code like this in the post_input template file:

```
If content.item("audience:1") = "" then content.add
"audience", ""
```

If there is no form submission, the addition of the empty string will remove all parts of the data with that name in the asset.

↗ Combo Box

Combo Boxes (a pull-down menu where an option may be selected or manually typed in) are not supported by HTML. Instead, you can use a drop-down menu and an edit box. Use DHTML to show/hide the extra box.

PANELS

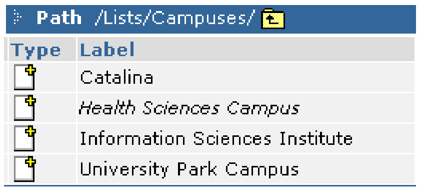
Many kinds of assets can be automatically ordered for index pages or lists by either random sort, by date, or order entered. In some cases, though, a more manual sort order is preferred. The Panels construct and associated support routines allow for quick creation of this type of interface.

BASIC MANUAL SORT PANELS

↗ Sorting Panels and Folder Storage

There is often a need to sort the assets in a folder manually to support index pages. One common way to implement this is to add a sorting section to each of the assets, but have that sort order common to all of the assets. That way, you can adjust the order while you are working on any of the assets. The sort order data in this case, is kept in the folder rather than each asset since the folder is one thing that is shared by all the assets.

Here is a sample list. The goal is to have each campus asset maintain a list of venues on that campus. There is also a need to order the campus names for display. This sort of hierarchical concept is a great use for folder storage.

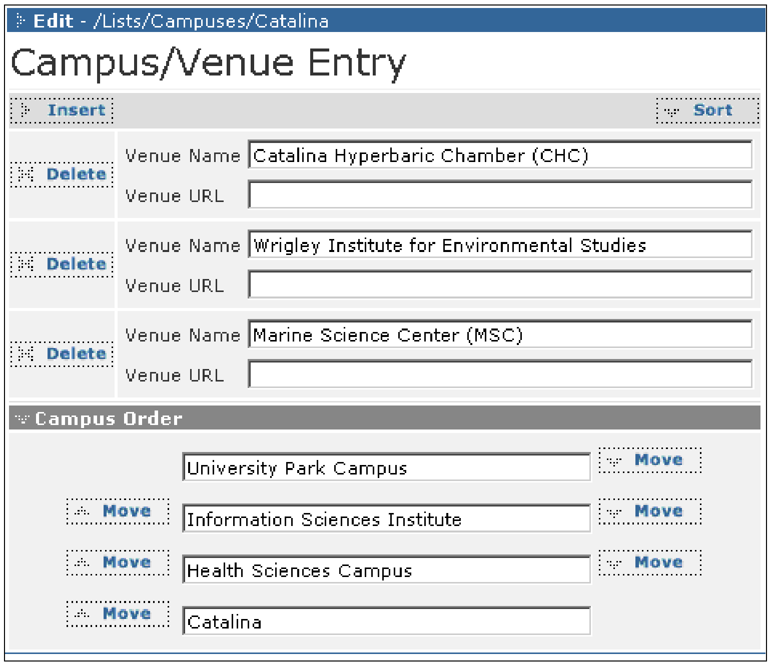


Path /Lists/Campuses/

Type	Label
	Catalina
	Health Sciences Campus
	Information Sciences Institute
	University Park Campus

When you edit any of these assets, the following input.asp template file is shown, containing two panels:

- The first is a panel to hold the venue name and ID pairs for the campus. This is a sort panel with a manual sorting button. This approach was chosen to allow quick data entry while keeping track of the new items. At the end of a batch of changes, the sort button is clicked once to reorder. This approach also avoids a popup panel or page reload for each entry.
- The second panel is the asset sorting panel, and is the same for each campus since the sort data is stored in the campus folder itself. Note that there are no insert or delete buttons since the goal is just to order the assets in the campus folder.



Edit - /Lists/Campuses/Catalina

Campus/Venue Entry

Insert | Sort

Delete	Venue Name	Catalina Hyperbaric Chamber (CHC)
	Venue URL	
Delete	Venue Name	Wrigley Institute for Environmental Studies
	Venue URL	
Delete	Venue Name	Marine Science Center (MSC)
	Venue URL	

Campus Order

	University Park Campus	Move
Move	Information Sciences Institute	Move
Move	Health Sciences Campus	Move
Move	Catalina	

The input.asp template file code follows, and shows how the sorting panel is created by using the setParam function. The second panel shows how the “mini_move” panel is created by combining the list stored in the folder with the current file list, which will contain any new or deleted files.

```
<TABLE class="tabletext">
<tr><td colspan=2><font size="+1">Campus/Venue
Entry</font></td></tr>


 <TR align=justify bgColor=#f0f0f0 vAlign=top>
  <TD colSpan=2>
   <% set list = content.createList("venue_name")
%>
   <% list.setParam "panel_style", "sorted" %>
   <% list.setParam "sort_name", "venue_name" %>
   <% do while list.nextPanel() %>
    <TABLE class="tabletext">
     <TR>
      <TD>Venue Name</TD>
      <TD>
       <INPUT name="venue_name" size=50 value="<%=
list.item("venue_name") %>">
      </TD>
     </TR>
     <TR>
      <TD>Venue URL</TD>
      <TD>
       <INPUT name="venue_url" size=50 value="<%=
list.item("venue_url") %>">
      </TD>
     </TR>
    </TABLE>
   <% loop %>
  </TD>
 </TR>


<!--Link Navigation Goes Here ->
<tr align="justify" valign="top" bgcolor="#808080">
 <td colspan=2><img
src="/cpt_media/arrow_white_open.gif"
hspace="3"><B><font color="#FFFFFF">Campus
Order</font></B></td>
</tr>
<tr>
 <td colspan=2 bgcolor="F0F0F0" align=center>
 <table class=tabletext align="justify"
valign="top" bgcolor="F0F0F0">
 <% set newlist =
asset.createFileListFromFolder(content.item("_CMSFo
lderId"), "coe_label", "coe_id")%>
 <% newlist.setParam "panel_style", "mini_move" %>
 <% do while newlist.nextPanel()%>
 <input type="hidden" name="coe_id" value="<%=
newlist.item("coe_id") %>">
 <input size=40 type="text" name="coe_label"
value="<%= newlist.item("coe_label") %>"
onClick="blur()" onSelect="blur()">
 <%loop%>


 </table>
 </td>
</tr>


</TABLE>
```

The post_input.asp file needs to remove the data from that asset, and replace the folder data with the new ordering:

```
<% Dim folder_content
```

```
  set folder_content =
asset.getContent(content.item("_CMSFolderId"))

  folder_content.removePrefix "coe_" 'delete all
exisitng lists


  set list = content.createList("coe_id")

  do while list.nextEntry()

   folder_content.add "coe_id:" & list.itemIndex,
list.item("coe_id")

   folder_content.add "coe_label:" &
list.itemIndex, list.item("coe_label")

  loop

  content.removePrefix "coe_" 'delete lists from
asset

  asset.setContent content.item("_CMSFolderId"),
folder_content

%>
```

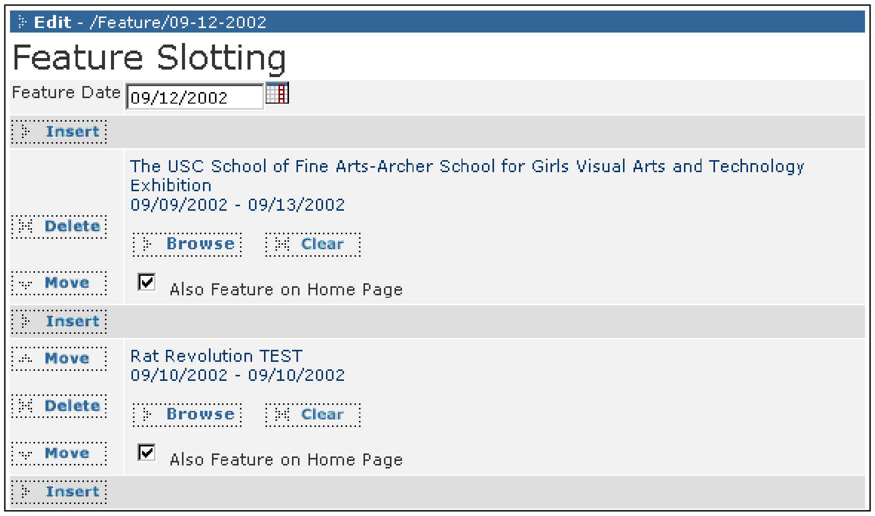
Using the sorted data now requires getting the list back from the folder as in the following output.asp template file (see createFileListFromFolder in the API Guide):

```
<% set list =
asset.createFileListFromFolder("/Lists/Campuses/",
"coe_label", "coe_id")%>

<% do while list.nextEntry()%>

  <option value="cam_<%= list.item("coe_id")
%>"><%= list.item("coe_label") %></option>

<%loop%>

<option value="offcampus">Off Campus</option>
```

Note that nextPanel() is used in the input.asp template file while nextEntry() is used in the post_input.asp and output.asp template files. nextPanel() will always return at least one item even if the list is empty. So for output template files, nextEntry() should always be used.

↗ Asset Selection Panels (Slotting)

A slotting panel allows assets to be selected and ordered manually. A common use for this is to select a few key articles, etc. for the home page. The screen shows an interface to choose feature events using a panel to manually order them:



Edit - /Feature/09-12-2002

Feature Slotting

Feature Date 09/12/2002

Insert

The USC School of Fine Arts-Archer School for Girls Visual Arts and Technology Exhibition
09/09/2002 - 09/13/2002

Delete | Browse | Clear

Move | ☑ Also Feature on Home Page

Insert

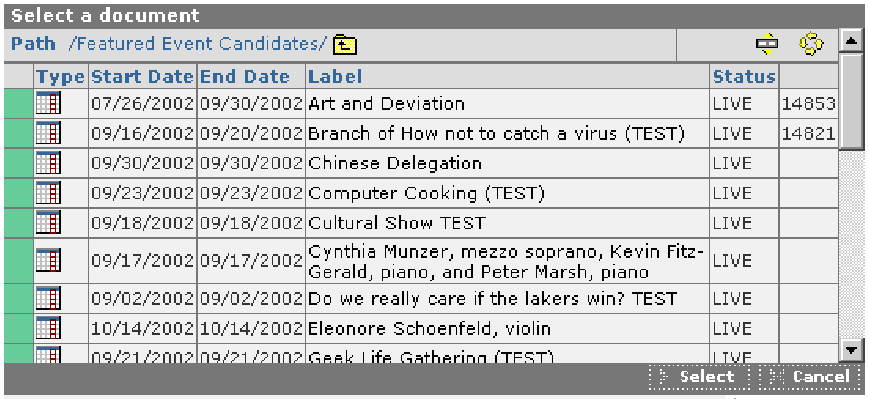
Move | Rat Revolution TEST
09/10/2002 - 09/10/2002

Delete | Browse | Clear

Move | ☑ Also Feature on Home Page

Insert

- Click Browse to bring up the standard asset selection popup window:



Select a document

Path /Featured Event Candidates/

Type	Start Date	End Date	Label	Status	
	07/26/2002	09/30/2002	Art and Deviation	LIVE	14853
	09/16/2002	09/20/2002	Branch of How not to catch a virus (TEST)	LIVE	14821
	09/30/2002	09/30/2002	Chinese Delegation	LIVE	
	09/23/2002	09/23/2002	Computer Cooking (TEST)	LIVE	
	09/18/2002	09/18/2002	Cultural Show TEST	LIVE	
	09/17/2002	09/17/2002	Cynthia Munzer, mezzo soprano, Kevin Fitz-Gerald, piano, and Peter Marsh, piano	LIVE	
	09/02/2002	09/02/2002	Do we really care if the lakers win? TEST	LIVE	
	10/14/2002	10/14/2002	Eleonore Schoenfeld, violin	LIVE	
	09/21/2002	09/21/2002	Geek Life Gathering (TEST)	LIVE	

Select | Cancel

Once the user selects an asset, the name appears on the panel interface.

The input.asp template file code shows how to setup this asset selection interface. The key code is the span, and the input.showSelectDocument command and associated parameters:

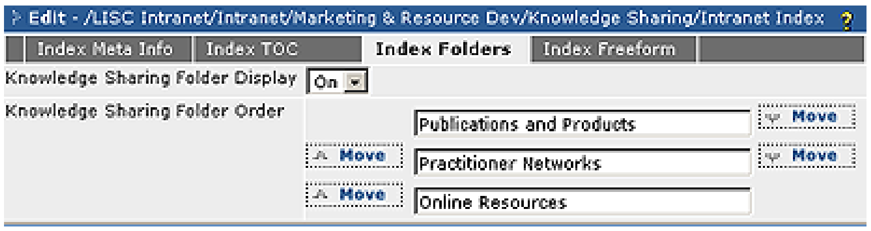
```
<tr><td colspan=2><font size="+1">Feature
Slotting</font></td></tr>


<TR align="justify" valign="top" bgcolor="F0F0F0">
 <TD>Feature Date</TD>
 <TD><% input.showSelectDate "feature_date",
content.item("feature_date") %></TD>
</TR>


<% set list =
content.createList("feature_calendar") %>
<% do while list.nextPanel() %>
<table class="tabletext" width="100%">
```

```
 <tr>
  <td>
   <font color="#003366">
   <% set fields = 
asset.getContent(list.item("feature_calendar")) %>
   <span id="title_text">
    <%= fields.item("name") %><br>
    <%= fields.item("start_date") %> - <%= 
fields.item("end_date") %>
   </span>
   <br>
   <br>
   </font>
   <% input.setParam "default_folder", "/Featured 
Event Candidates/" %>
   <% input.setParam "label_span", "title_text" %>
   <% input.showSelectDocument "feature_calendar", 
list.item("feature_calendar") %>
  </td>
 </tr>


 <tr>
  <td>
    <% input.checkbox "home_page", "y", 
list.item("home_page") %>
    Also Feature on Home Page
  </td>
 </tr>


</table>
<% loop %>
```

TABS AND DIVIDERS

↗ Tabs

A collection of tabs can be used to separate groups of related fields. Clicking on a tab label will make the fields associated with that tab visible, while hiding the fields for the previously visible tab.


Edit - /LISC Intranet/Intranet/Marketing & Resource Dev/Knowledge Sharing/Intranet Index

Index Meta Info | Index TOC | Index Folders | Index Freeform

Knowledge Sharing Folder Display: On

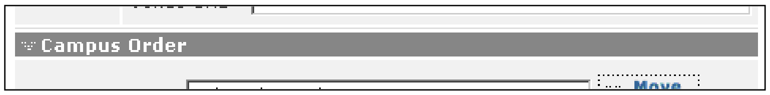
Knowledge Sharing Folder Order: Publications and Products; Practitioner Networks; Online Resources

The advantage of tabs is that they allow for the separation of fields, without all fields needing to be viewable simultaneously. This reduces clutter in the interface, as well as the need for excessive scrolling.

```
<% input.startTabbedPanel "Index Meta Info, Index
TOC, Index Folders, Index Freeform" %>
… Index Meta Info code here …
<% input.nextTabbedPanel %>
… Index TOC code here …
<% input.nextTabbedPanel %>
… Index Folders code here …
<% input.nextTabbedPanel %>
… Index Freeform code here …
<% input.endTabbedPanel %>
```

↗ Dividers

A conventional divider is used as a horizontal separator between groupings of fields:



The code uses a simple HTML table row with a stock image (included with the CMS), and coloring. This is also available in the Code Wizard should you need it.

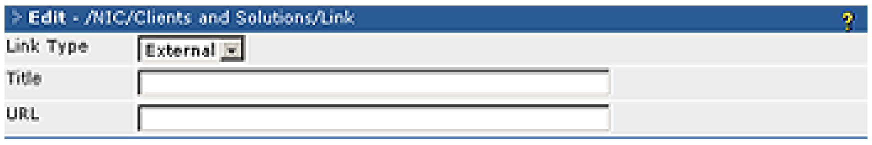
```
<td colspan=2><img
src="/cpt_media/arrow_white_open.gif"
hspace="3"><B><font color="#FFFFFF">Campus
Order</font></B></td>
```

USING JAVASCRIPT TO SHOW/HIDE FIELDS

Most of the functionality in the following section is now encapsulated in the CMS, via the JavaScript function cpt_ShowHideDropDown() and the util.visibility API call. These can be seen at work by creating a Related Link Panel from the Code Wizard. The description below, however, is still useful for understanding methods in which JavaScript can be used to modify and extend the CMS interface.

For certain types of Input Forms, it may be advantageous to use extra DHTML and/or JavaScript in your input.asp file to control the display of certain fields. One example of this is for a template that creates Link objects within a site, where the link could be external or internal. Rather than present the fields for creating both internal and external links simultaneously, some DHTML and JavaScript can be added that displays or hides the fields, based on the type of link the user wants to create.

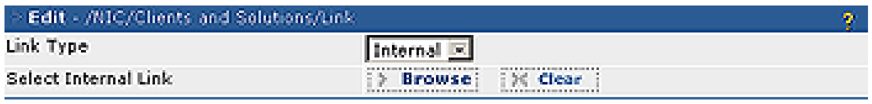
Here is the Link Input Form:



Edit - /NIC/Clients and Solutions/Link

Link Type	External
Title	
URL	

Using the drop-down menu to select “Internal” displays the Internal link fields while hiding the External link fields:



Edit - /NIC/Clients and Solutions/Link

Link Type	Internal
Select Internal Link	Browse Clear

To provide this functionality, a simple JavaScript is added to the input.asp template. The JavaScript searches for the occurrence of the drop-down menu’s value in the ID attribute of a <TR> tag. Based on this, the visibility of that row is toggled on or off.

```
function showLinkType() {

var currentValue = event.srcElement.value
var oParent = event.srcElement
var oChild = ""

while ((oParent != null) && (oParent.tagName !=
"TR")) {
  oParent = oParent.parentNode;
}

oChild = oParent.nextSibling;

for (i = 1; i < oParent.parentNode.children.length;
i++) {
  if (oChild.id != "") {
    if (oChild.id.indexOf(currentValue) == -1) {
      oChild.style.visibility = "hidden";
      oChild.style.display = "none";
    } else {
      oChild.style.visibility = "visible";
      oChild.style.display = "block";
    }
    oChild = oChild.nextSibling;
  }
}

}//end showLinkType
```

Here is the code for the input fields. The onChange event handler in the <SELECT> tag triggers the JavaScript. The VBScript ensures that the correct group of fields (External or Internal) is displayed when the Asset is opened for editing, based on the current state of the Asset's data.

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
 <td>Link Type</td>
 <td>
  <SELECT name="link_type"
onChange="showLinkType()">
  <OPTION value="ext" <% if
content.item("link_type") = "ext" then%>SELECTED<%
end if %>>External</OPTION>
  <OPTION value="int" <% if
content.item("link_type") = "int" then%>SELECTED<%
end if %>>Internal</OPTION>
  </SELECT>
 </td>
</tr>

  <% Dim row_vis
    if content.item("link_type") <> "" then
     if content.item("link_type") = "ext" then
       row_vis = 1
     else
       row_vis = 2
     end if
    else
     row_vis = 1
    end if

  %>


<tr id="ext_title" <% if row_vis = 2
then%>style="visibility: hidden; display: none;"<%
end if %> align="justify" valign="top"
bgcolor="F0F0F0">

 <td>Title</td>

 <td><input type="text" name="link_title"
value="<%= content.escapeItem("link_title") %>"
size="50"></td>

</tr>



<tr id="ext_url" <% if row_vis = 2
then%>style="visibility: hidden; display: none;"<%
end if %> align="justify" valign="top"
bgcolor="F0F0F0">

 <td>URL</td>

 <td><input type="text" name="link_url" value="<%=
content.escapeItem("link_url") %>" size="50"></td>

</tr>


<TR id="int_sel" <% if row_vis = 1
then%>style="visibility: hidden; display: none;"<%
end if %> align="justify" valign="top"
bgcolor="F0F0F0">

 <TD>Select Internal Link</TD>

 <TD>

   <span name="link_select_span"
id="link_select_span">

   <% if content.item("link_select") <> "" then %>

    <%=
asset.getContentField(content.item("link_select"),
"title") %>

   <% else %>

    <%=
asset.getContentField(content.item("link_select"),
"title") %>

   <% end if %>

   </span>

   <% input.setParam "default_folder",
content.item("_CMSFolder") %>

   <% input.setParam "label_span",
"link_select_span" %>
```

```
   <% input.showSelectDocument "link_select",
content.item("link_select") %>
 </TD>
</TR>
```

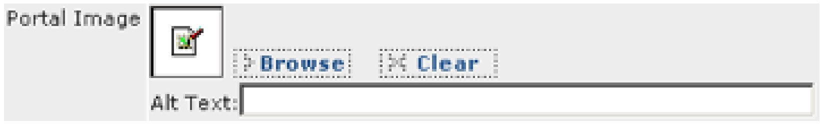
For additional information, a secondary example of this technique is shown later in this document.

FILE/IMAGE UPLOAD AND SELECTION

Most Web pages are not just text; many display informative graphics and link to associated files. For example, a press release might link to a PDF version that users can download and print. Or a company's client listing may include the client's logo next to each reference.

To assist in the association of these files with a template, we have the File/Image Upload and Selection interface. As you can see in the code examples below, a list of extensions can be provided to constrain what types of files are allowed for upload.

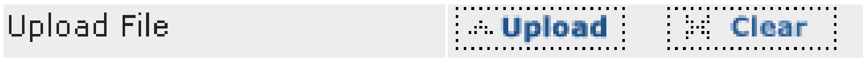
The interface is usually invoked through this method for images (note the addition of an ALT text field as well):



```
<TR align="justify" valign="top" bgcolor="F0F0F0">
  <TD>Portal Image</TD>
  <TD>
   <% if content.item("portal_image") = "" then %>
   <img src="/cpt_media/default.gif">
   <% else %>
   <img src="<%= content.item("portal_image") %>">
   <% end if %>
   <% input.setParam "default_folder",
"/Clients/Images/" %>
   <% input.setParam "extensions", "png gif jpeg
jpg" %>
   <% input.setParam "show_browse", "Link" %>
   <% input.setParam "show_upload", "Attach" %>

   <% input.showAcquireImage "portal_image",
content.item("portal_image") %>
```

```
   <br>Alt Text:<input type="text"
name="portal_image_alt" size=50 value="<%=
content.item("portal_image_alt") %>">


  </TD>

 </TR>
```

And this is the usual method for files:



```
<TR align="justify" valign="top" bgcolor="F0F0F0">

  <TD>Upload File</TD>

  <TD>

   <% if content.item("id_upload") <> "" then %>

   <%

     Dim f_ext, d_ext, down_text

     f_ext = Right(content.item("id_upload"),"3")

     Select Case f_ext

      Case "pdf"

       d_ext = "pdf2.gif"

       down_text = "Adobe Acrobat"

      Case "doc"

       d_ext = "doc2.gif"

       down_text = "Microsoft Word"

      Case "xls"

       d_ext = "xls2.gif"

       down_text = "Microsoft Excel"

      Case "ppt"

       d_ext = "ppt2.gif"

       down_text = "Microsoft PowerPoint"

     End Select

   %>

   <a href="javascript:openPopup('<%=
content.item("id_upload") %>', 640, 480);">VIEW<img
src="/images/icons/<%= d_ext %>" width="16"
height="16" alt="Download this document in <%=
down_text %> format" hspace="2" vspace="1"
border="0"></a>

   <% end if %>
   <% input.setParam "extensions", "pdf doc xls ppt
jpg jpeg gif png psd" %>

   <% input.setParam "default_folder", "/" &
content.item("_CMSFolder") & "/" %>

   <% input.setParam "show_upload", "attach" %>

   <% input.setParam "show_browse", "link" %>

   <% input.showAcquireDocument "id_upload",
content.item("id_upload") %>

  </TD>

 </TR>
```

↗ Modalities: Browsing and Uploading, Linking and Attaching

For the inclusion of files (images, PDFs, Word docs, etc.) in an asset, there are two primary considerations: the method and the action.

- Method: Will you browse to locate the file from within the CMS or will you upload the file from my local computer?
- Action: Will you be linking to the file from the asset or attaching the file to the asset?

A related concern is how the method and action tie together to build the proper experience for the users.

For the method, there are three main interface types that can be presented: Browse and Upload, Upload only, or Browse only. Examples of these interfaces are included below.

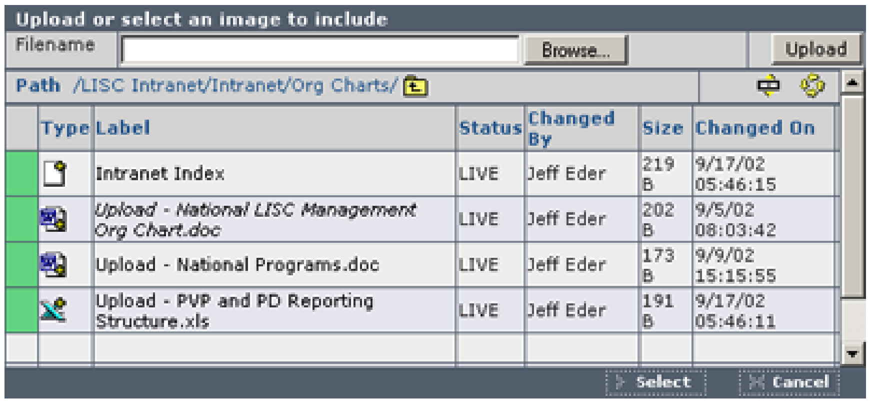
To upload a file from your local system:

1. Click **Browse** to bring up the standard Windows file browser.
2. Select the file and click **Upload** to complete the operation.

To select the file from the CMS:

1. Select from the currently opened folder in the interface, or use the Path area to navigate through the CMS structure to find the desired file.
2. Once you see the file you want to use, click on its label to select it, and then click **Select**.
3. In either case, click **Cancel** to exit from the upload and selection interface.

This is the Upload and Browse interface:



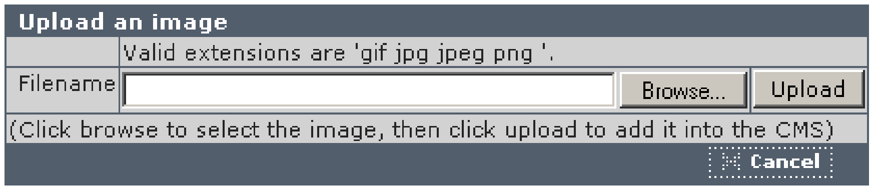
Upload or select an image to include

Filename

Path /LISC Intranet/Intranet/Org Charts/

	Type	Label	Status	Changed By	Size	Changed On
🟩		Intranet Index	LIVE	Jeff Eder	219 B	9/17/02 05:46:15
🟩		*Upload - National LISC Management Org Chart.doc*	LIVE	Jeff Eder	202 B	9/5/02 08:03:42
🟩		Upload - National Programs.doc	LIVE	Jeff Eder	173 B	9/9/02 15:15:55
🟩		Upload - PVP and PD Reporting Structure.xls	LIVE	Jeff Eder	191 B	9/17/02 05:46:11

This is the Upload-only interface:



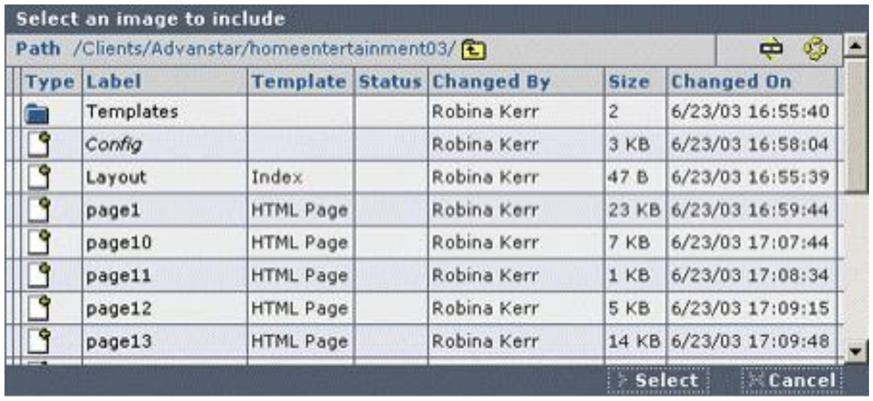
Upload an image

Valid extensions are 'gif jpg jpeg png '.

Filename

(Click browse to select the image, then click upload to add it into the CMS)

This is the Browse-only interface:



Select an image to include

Path /Clients/Advanstar/homeentertainment03/

Type	Label	Template	Status	Changed By	Size	Changed On
	Templates			Robina Kerr	2	6/23/03 16:55:40
	Config			Robina Kerr	3 KB	6/23/03 16:58:04
	Layout	Index		Robina Kerr	47 B	6/23/03 16:55:39
	page1	HTML Page		Robina Kerr	23 KB	6/23/03 16:59:44
	page10	HTML Page		Robina Kerr	7 KB	6/23/03 17:07:44
	page11	HTML Page		Robina Kerr	1 KB	6/23/03 17:08:34
	page12	HTML Page		Robina Kerr	5 KB	6/23/03 17:09:15
	page13	HTML Page		Robina Kerr	14 KB	6/23/03 17:09:48

With the method interface selected, the next task is to map it to the action you wish to be performed: linking or attaching. It is good to note that the end result of each action is the same: The desired file will be published to the live site and referenced by the current asset. However, the exact nature of each action requires some additional explaining.

Linking is as basic as creating a link to the desired file. This means that the asset and the file linked to remain as separate entities in the CMS. Each item is moved through workflow (if applicable) separately, and published separately (although a dependency might cause one to pull the other along). Linking is best used for files that are uploaded separately into the CMS. Linking also allows for more explicit, separate control over the file and the asset.

Attaching means that the file is now part of the data of the current asset. With the exception of the controls made available in the asset's input.asp template file, the attached file cannot be managed separately. As such, the file relies on the asset's Workflow and publishing information. Attaching is similar to pasting images into a Word document, since the images pasted in will now follow the Word file wherever it goes.

With uploading, browsing, linking, and attaching, there are no set rules. The Code Wizard "Image Upload" command will provide an interface to help configure this as you construct your templates. However, the most common configuration is to allow a user to browse the CMS, link to files, and upload files that are attached to the current Asset.

FILE NAMES ON THE TARGET WEB SERVER

In traditional Web development, developing a file naming convention and directory structure is a substantial task. From an end-user perspective, file names directly translate into the URLs through which content is accessed. For developers, the convention provides a key method of defining how a site may grow and it serves as the guide for them to understand how the content is organized. Additionally, issues such as namespace collisions and intelligibility of names are a factor.

Within a CMS environment, many of these concerns are alleviated because the CMS provides its own methods for interacting with and growing the content, as well as for keeping track of unique names and other variables. The CMS provides several ways to specify this information, from completely manual systems to dynamic systems that can enforce specified logic for file name and path creation.

HOW A FILE NAME IS DEFINED

There are three methods that can define the file name and path for an asset:

- Default method
- Publishing properties method
- File name template method

Note: Regardless of the method used, the Package setting (as defined in the Publishing Properties method below) must be defined for each asset, or in a higher level folder from which the asset will inherit its Package settings. This defines what server(s) the asset will be published to. If this is not set, the asset will not be published.

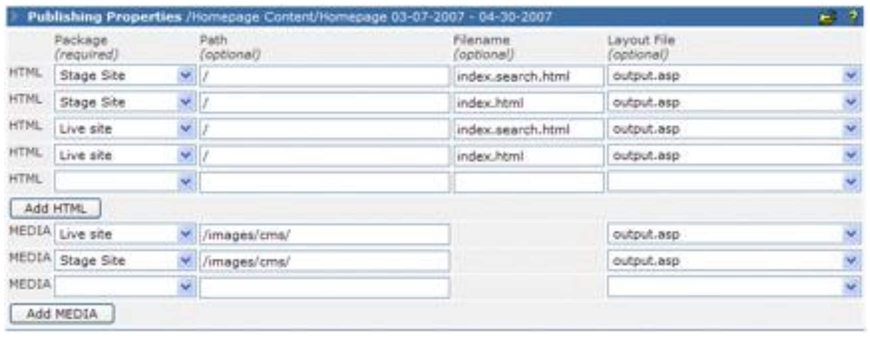
↗ Default Method

The default method is to use the existing label and location for the asset. The asset's label becomes the file name and the folder path to the asset's location becomes the path to the published file.

Any spaces in the label or folder path are translated into their hexadecimal equivalent to allow for proper publishing and linking. This method can be achieved by simply setting the Publishing Package in the top level CMS folder to where the content is located. From there, the CMS will automatically configure the paths and file names.

↗ Publishing Properties Method

The file name and path info for an asset can be specified using the Publishing Properties. To access this interface, select an Asset and go to View>Properties>Publishing.



Publishing Properties /Homepage Content/Homepage 03-07-2007 - 04-30-2007

	Package *(required)*	Path *(optional)*	Filename *(optional)*	Layout File *(optional)*
HTML	Stage Site	/	index.search.html	output.asp
HTML	Stage Site	/	index.html	output.asp
HTML	Live site	/	index.search.html	output.asp
HTML	Live site	/	index.html	output.asp
HTML				
Add HTML				
MEDIA	Live site	/images/cms/		output.asp
MEDIA	Stage Site	/images/cms/		output.asp
MEDIA				
Add MEDIA				

Example: Publishing Properties for a homepage with additional layout files.

Using this interface, the path info and file name can be manually entered. For each entry, a Publishing Package must be defined. The Package represents the location (usually a Web server) and protocol (usually FTP) that the Asset is published under. The available Packages are configured elsewhere in CrownPeak CMS.

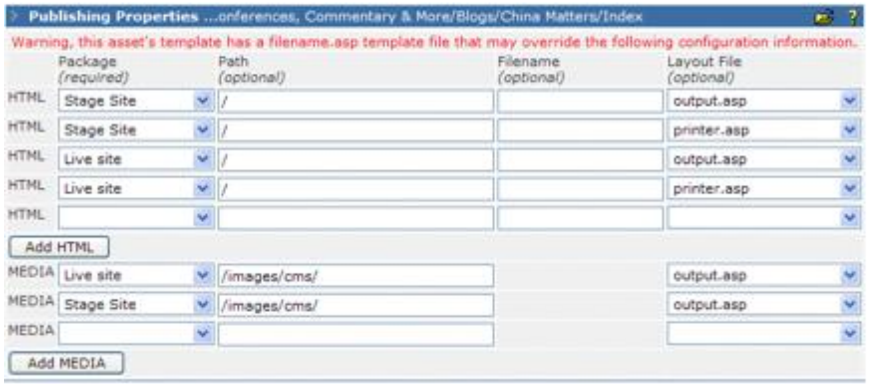
Separate rows are provided for configuring HTML and Media publishing information. The HTML rows define publishing information for the most of the data in the Asset. The only data not covered by this row are any uploaded media files (e.g., images, PDFs, Word docs, etc.) associated with the Asset; these are defined by the Media rows. Upon saving any configuration, a new blank row will be added to the HTML and Media sections. This allows an Asset to have several different publishing configurations, depending on the number of Packages that must be supported.

If the Asset has an associated template, there will be an option to specify a Layout File. The Layout File allows a single Asset to be published in multiple formats, with a Layout File corresponding to each desired format. This could be as simple as allowing for the publishing of a normally formatted HTML file and a "printer friendly" HTML file, or as complex as publishing an Asset in HTML, XML, and WML formats, all to separate servers (based on the Package configuration).

↗ File Name Template Method

The third method of specifying the file name and path info is through the filename.asp template file. By setting the _CMSPublishPath value in this file, you can programmatically define the path and file name. These values may be set for the various Layout Files as well by analyzing the value of

_CMSLayout. Of all the methods, this is the most robust and flexible in terms of the end result. One reason for using the Filename Template would be to ensure that a file name is unique by appending the Asset's ID to it.



Publishing Properties ...onferences, Commentary & More/Blogs/China Matters/Index

Warning, this asset's template has a filename.asp template file that may override the following configuration information.

	Package *(required)*	Path *(optional)*	Filename *(optional)*	Layout File *(optional)*
HTML	Stage Site	/		output.asp
HTML	Stage Site	/		printer.asp
HTML	Live site	/		output.asp
HTML	Live site	/		printer.asp
HTML				
Add HTML				
MEDIA	Live site	/images/cms/		output.asp
MEDIA	Stage Site	/images/cms/		output.asp
MEDIA				
Add MEDIA				

Example: Publishing Properties for an asset that uses a Filename template.

CHOOSING A FILE NAMING SCHEME

Based on the above methods, there are several different ways of defining a file naming scheme. Any scheme must balance the needs of end user intelligibility with ease of growth/maintenance, while ensuring that there are no namespace collisions.

Most implemented schemes rely on a combination of the above methods. For example, a site's homepage would likely have its publishing information specified in the Publishing Properties interface (because it will always publish to index.html), while supporting pages (which will vary in number as site content grows) would utilize the Filename Template method to dynamically generate a name.

For additional information, see the Best Practice Filenaming Scenarios section under the Tips section later in this document.

SYSTEM VARIABLES

The following system variables show up in the field pick list in the CMS Template Editor. You will see these variables in addition to any variables defined in any of your open assets. Note that all system variables are prefixed with a "_CMS" – for example an Asset's ID would be specified as _CMSID.

Long Name	CMS Variable	Description	Example Value
Base Model Id	_CMSBaseModelId	The ID of the model that created this asset.	0
Base Model Id	_CMSBaseModelId	The ID of the model that created this asset.	0
Base Status	_CMSBaseStatus	The numeric ID of the status of the asset that was used as a model for the current asset.	1234
Base Status Color	_CMSBaseStatusColor	The status color of the asset that was used as a model for the current asset.	FFCC33
Base Status Text	_CMSBaseStatusText	The status text (DRAFT, LIVE, etc.) of the asset that was used as a model for the current asset.	DRAFT
Branch ID	_CMSBranchId	The ID of the asset that the current asset was originally branched from.	906
Changed By	_CMSChangeUserId	The user ID of the person that performed the last action on this asset.	11
Changed Date	_CMSChangeDate	The last time an action was performed on this asset.	11/3/2003 12:13:08 PM
Checkout Date	_CMSCheckoutDate	The date the current asset was checked out.	11/3/2003 12:13:08 PM
Checkout User ID	_CMSCheckoutUserId	The ID of the user that checked out the current asset.	11
Comments Count	_CMSCommentsCount	The number of comments associated with the current asset.	4
Created By	_CMSCreateUserId	The user ID of the creator of this asset.	11
Created Date	_CMSCreateDate	The date this asset was created.	11/3/2003 12:11:37 PM
Filename	_CMSFilename	The name of the current asset (same as _CMSLabel), filtered for special characters.	Section Index
Folder	_CMSFolder	The folder path to the current asset.	Clients/My Client/Press Release
Folder Id	_CMSFolderId	The numerical ID for the folder that contains this asset.	8584
Folder:n	_CMSFolder:n (_CMSFolder:1)	The name of a single folder from the _CMSFolder value, where n is a number corresponding to a folder's position in the path.	My Client
Folder Count	_CMSFolderCount	The number of folders in the _CMSFolder variable.	3

Long Name	CMS Variable	Description	Example Value
ID	_CMSId	The numerical ID of the current asset.	9461
ID Path	_CMSIdPath	The path of folder ID's to the current asset.	/828/3940/8584/
Label	_CMSLabel	The label of the current asset.	Section Index
Modified By	_CMSModifiedUserId	The user ID of the person that last modified this asset's data or content.	11
Modified Date	_CMSModifiedDate	The last time the data or content in this asset was modified.	11/3/2003 12:13:08 PM
Path	_CMSPath	The full path to the current asset, including the asset name.	/Clients/CPTtest/cpt test 4/test upload 031103
Published By	_CMSLiveUserId	The user ID of the user that sent this asset live.	11
Published Date	_CMSLiveDate	The scheduled live date for this asset.	11/3/2003 12:13:08 PM
Retired By	_CMSRetireUserId	The user ID of the user that retired this asset.	1568
Retired Date	_CMSRetireDate	The scheduled retire date for this asset.	11/3/2003 12:13:08 PM
Root Folder	_CMSRootFolder	The name of the first folder in the path to the current asset.	Site
Root Folder ID	_CMSRootFolderId	The ID of the first folder in the path to the current asset.	89
Shortcut ID	_CMSShortcutId	The ID of the asset that the current asset is a shortcut of.	1449
Size	_CMSSize	The size of the current asset.	1
Status	_CMSStatus	The ID of the current status for this asset.	32887
Status Text	_CMSStatusText	The name of the current status for this asset.	DRAFT
Status Color	_CMSStatusColor	The color (hexadecimal) for the current status.	8DBEDA
Status Date	_CMSStatusDate	The date of the last status change for the current asset.	10/3/03 15:32:37
Status User ID	_CMSStatusUserId	The ID for the user that last changed the status on the current asset.	11
Template Id	_CMSTemplateId	The ID for the template for this asset.	35389
Template Name	_CMSTemplateName	The name of the template for this asset.	Press Release
Template Path	_CMSTemplatePath	The path to the template for the asset.	/System/Templates/Press Release/
Top Folder	_CMSTopFolder	The name of the current folder that the asset is in.	Homepage
Top Folder ID	_CMSTopFolderId	The ID for the current folder that the asset is in.	8584
Type	_CMSType	The type for the current asset (will be either file or folder).	File
Workflow ID	_CMSWorkflowId	The ID for the current workflow being used by this asset.	10
Workflow Name	_CMSWorkflowName	The name for the current workflow	Full Workflow

		being used by this asset.	
Long Name	**CMS Variable**	**Description**	**Example Value**
Status	_CMSStatus	The ID of the current status for this asset.	32887
Status Text	_CMSStatusText	The name of the current status for this asset.	DRAFT
Status Color	_CMSStatusColor	The color (hexadecimal) for the current status.	8DBEDA
Status Date	_CMSStatusDate	The date of the last status change for the current asset.	10/3/03 15:32:37
Status User ID	_CMSStatusUserId	The ID for the user that last changed the status on the current asset.	11
Template Id	_CMSTemplateId	The ID for the template for this asset.	35389
Template Name	_CMSTemplateName	The name of the template for this asset.	Press Release
Template Path	_CMSTemplatePath	The path to the template for the asset.	/System/Templates /Press Release/
Top Folder	_CMSTopFolder	The name of the current folder that the asset is in.	Homepage
Top Folder ID	_CMSTopFolderId	The ID for the current folder that the asset is in.	8584
Type	_CMSType	The type for the current asset (will be either file or folder).	File
Workflow ID	_CMSWorkflowId	The ID for the current workflow being used by this asset.	10
Workflow Name	_CMSWorkflowName	The name for the current workflow being used by this asset.	Full Workflow

MANAGING IMAGES

Images are a special type of binary asset on the Web since they are embedded in pages, but treated as separate files. They are often different than other binary assets (such as PDF, DOC, XLS, PPT etc.) because they are viewed inline in Web pages, are usually part of a site's navigation, and are usually more numerous (both on a single page and across a site).

Given the intrinsic need to manage images, the CMS offers several management modes—depending on whether the context is in the templates, or is content entered by an author.

MANAGING IMAGES IN CONTENT: TWO APPROACHES

When content authors edit pages, they will often need to upload an image to be used in the page as a portion of content. There are two ways to treat these per-page images:

↗ 1. Embedded in the Asset

In this case, the image is handled like an image in MS Word. The author inserts the image (either through a template function or using the WYSIWYG interface), and it shows up in the content. However, the author doesn't see or worry about the image file. They view the document as a single asset.

The CMS handles this by keeping track of the image, but moving it along with the document automatically. This mode is best used for images that only appear on one or a small number of pages.

This approach may be counter-intuitive to Web developers since they have become used to making multiple references to a single image. But it may make the most sense to authors since its the way that almost every application works (except for the Web).

↗ 2. Upload the Image as an Asset, then Link to it

In this approach, the author uploads an image to an asset folder in the CMS and then makes a link to it while authoring a document. This method is ideal if a single image will appear in many pages, or the images need separate management (for example a marketing graphics repository). This also makes sense if a shared image needs to be updated frequently.

For one-off images, this approach is not recommended since the author has to perform two steps. A variation on this approach is to have a template to manage assets where the image is uploaded and supporting meta information is added.

MANAGING IMAGES IN TEMPLATES: THREE COMMON APPROACHES

An image in a template is part of a site or page, but not under author control. These images are mostly static on a site and rarely change (for example, images that provide part of the site navigation design). When they are changed, it is usually the job of the Web developer, as opposed to content authors, to carry out the task.

There are three ways to treat images in templates:

↗ 1. Embedded in the Template

Similar to scheme 1 in the previous section, the site images are embedded in the template asset. This approach is used by the page import utility and wizard. The images are available in the template design mode and, when an asset belonging to the template is published out, the images are published to the server.

Since the images are saved in one place, you can update the site image by editing the template (in design mode) and then republishing.

↗ 2. As External Links

Images can also be left as external links in the CMS so that they are not managed by the CMS at all. The advantage of this approach is that the URLs in the templates are not changed and the images can be updated directly on the Web server (assuming there is a reason NOT to have the images managed by the CMS).

This approach may also be useful in sites where the CMS is only used for part of the content, since the CMS and non-CMS pages will share the same image files. It is also easy to setup, especially for existing sites.

To make the reviewing go faster, the images should also be copied to the stage folder in the CMS so that they are available locally.

Note: This will require the CMS repository of images to be manually updated anytime an image is changed on the live site.

↗ 3. As CMS managed Assets

Similar to scheme 2 in this section, the images are loaded into an asset folder and the template calls them with the asset.getLink command. This approach is more difficult to setup, but results in a CMS where the site-wide images are explicitly managed and easy to individually change and republish.

CONTENT PLACEMENT AND ORGANIZATION

One of the largest decisions in configuring a CMS is how to organize the content. Because the CMS decouples the relationship of the content to the final published files on the Web server, the organization systems of each set of content can vary greatly. While this does introduce some complexity to the organization process, it also provides the flexibility to ensure that your organization system is optimally tuned to the current environment: CMS or Web server.

While there is no single “correct” method for organizing and placing your content within the CMS, there are some organization methods that may be more advantageous than others. The use of any specific organizational scheme will vary depending on the exact nature of your content and how you want to structure the work of your content creators and editors.

Below are some guidelines on possible organization systems. It is likely that your CMS implementation will use a hybrid approach, incorporating aspects of both methods, as well as some new concepts.

APPROXIMATE SITE STRUCTURE

One possible organization system is to approximate the existing site structure. This may be particularly useful if you are implementing a CMS for your existing site, with developers that maintain the existing site that is being transitioned into using the CMS. In this respect, you can transfer their understanding of the existing structure into the CMS.

At the top level, the CMS structure would begin with having folders corresponding to the existing Web root folders. Files would exist in the same folders that they exist in on the live site.

One decision to be made about this structure would be about the exact naming of the files and folders. By using the exact same names as those on the existing site, the configuration of publishing would be greatly streamlined and continuity is maintained. However, this would mean that CMS users would possibly be dealing with potentially arcane names for assets, which could be confusing for a user, and would not make the best use of the capabilities in the CMS.

Another decision is whether or not some elements would be better off in a different area. For example, the homepage of a site is typically called “index.html” and always resides in the Webroot folder. However, perhaps the homepage is updated weekly, and new versions of the homepage are created in advance to preview the changes. Within the CMS, it would be possible to have the homepage reside in its own folder (along with the soon to be launched versions) rather than have it clutter the root directory. The homepage can still be previewed and published properly from this separate folder, without adding confusion to other areas of content.

Sites that utilize a well developed hierarchy might benefit from this method. For these types of sites, the CMS Models can be set up in a recursive nature so that a single model can allow the user to create additional nodes in the hierarchy at any level. If a common organization system for the CMS and live Web site is desired, this is a good way to maintain it.

FUNCTIONAL GROUPINGS

Sometimes it is useful to group content in the CMS in a completely different structure than what is present on the live Web site. In these cases, content is usually aggregated into functional groupings. A functional grouping would mean that assets with the same purpose (and probably the same template) are kept together, separated from other assets with different templates.

For sites that have deep hierarchies, using functional groupings can serve to reduce the overhead of traversing several hierarchy nodes in order to edit or manage content. This may also help in maintaining increased access control over assets. For example, perhaps a specific user group is limited to modifying a specific content type. By keeping those content types together, the chance of the user group accessing forbidden content is reduced.

One possible implementation of functional groupings would be to conceptually abstract the content of the site into the concept of a content repository. In this case, one area (essentially, a folder) in the CMS could be the content repository, where all the content is created and managed. This repository would be one of the functional groupings. The remainder of the site could be conceptually thought of as a series of indexes of this content. These indexes would reside separately from the content repository, forming a separate functional grouping. The indexes would then be used to select desired content to be displayed on the site.

There are also drawbacks to this method. The publishing configuration is usually more complex, and thus requires more time to setup properly. The abrupt change in organizational systems between the CMS and the live site, or the lack of a more developed hierarchy, might be confusing for some users.

LISTS

List variables (often generated by panels) are stored as foo:1, foo:2, foo:3, etc. The list processing routines take care of adding the :# (where "#" = a number) and iterating through these.

If you want to add an ID or similar tag for JavaScript in a panel, just follow the identifier name with a colon, and the system will add the panel index in automatically.

If you want to add a list in a post_input.asp template file, add the :# explicitly. For example, storing the asset order in a folder (see later section), the following code would go in the post_input.asp template file:

```
<% ' save the sort order back in the folder - each
asset shares the same data
  set toc =
asset.getContent(content.item("_CMSFolderId"))

  'delete all the current toc entries in the folder
  toc.removePrefix "toc_"

  ' make a list of the new ones from this asset
  set toc_list = content.createList("toc_id")
  do while toc_list.nextEntry()
    toc.add "toc_id:" & toc_list.itemIndex,
toc_list.item("toc_id")
    toc.add "toc_label:" & toc_list.itemIndex,
toc_list.item("toc_label")
  loop

  ' save the new toc in the folder
  asset.setContent content.item("_CMSFolderId"),
toc
%>
```

Notice that when you use createList, it will make a list based on the key field you provide, and there may be some cases, where the field may have no values and be left out of the list. If this happens, you can add a hidden field that always has a value to ensure you get all the rows:

```
  <% set list = content.createList("counter") %>
  <% do while list.nextPanel() %>
   <input type=hidden name="counter" value="<%=
list.itemIndex %>">
```

```
   <TABLE width="100%" class="tabletext">
... and the rest of the fields
```

Note: During development, you may get a variable foo, and then foo:1, foo:2 if you transition from a single val to multiple vals. You will need to remove that data element using the View > Properties > Content screen, or delete and recreate the asset.

OTHER EXAMPLES

FOLDER STORAGE

Within the CMS, content related data is typically stored within the asset that contains it. However, it is also possible to store data within a folder, which can provide additional flexibility in the administration of your content.

One of the more common examples of data that requires folder storage is navigation list data. The templates for an index of a content section or a table of contents for a group of content will likely include a sorting panel that provides for the ordering of its elements. By saving this ordering data in the asset's folder rather than the asset itself, you can use this data more efficiently in multiple areas.

For example, you could add an ordering panel to the templates for all content in a section (including the index), allowing a user to adjust the order from any asset. This way, if a user needs to edit an asset and make an ordering adjustment, this may be accomplished by editing a single asset rather than editing an asset (and later the index).

The critical code for saving folder storage of data is contained in the post_input template file. The code below accomplishes folder storage by adding submitted data to a folder data dictionary ("toc"), removing the data from the submitted asset (to ensure that data isn't duplicated), and finally setting the contents of the data dictionary to the folder.

Note the different methods of processing for list and standard data:

```
<%
Dim toc


set toc =
asset.getContent(content.item("_CMSFolderId"))


toc.removePrefix "paper_"


set toc_list = content.createList("paper_id")


do while toc_list.nextEntry()
  toc.add "paper_id:" & toc_list.itemIndex,
toc_list.item("paper_id")
  content.remove "paper_id"
  toc.add "paper_label:" & toc_list.itemIndex,
toc_list.item("paper_label")
  content.remove "paper_label"
loop

toc.add "papertitle", content.item("papertitle")
content.remove "papertitle"

toc.add "author", content.item("author")
content.remove "author"

toc.add "toclabel", content.item("toclabel")
content.remove "toclabel"

asset.setContent content.item("_CMSFolderId"), toc
%>
```

Folder storage can also be helpful when creating new versions of an asset that should share some (but not all) of the data from the previous version. For example, the home page template for a site might handle ordering of the site navigation and stylistic configuration (perhaps a site-wide background color) in addition to specific home page content. When a new version of the home page is created, the navigation and stylistic information should remain the same, with the home page content being the area that will change. By using folder storage for the navigation and stylistic information, the information can be pulled in by the new home page without needing to transfer it from the old version.

Please note that while folder storage can be useful in the management of your content, it does have some drawbacks as well. The primary issue is that the folder is not locked like an asset is when its content is being edited by a template. So, if you have a configuration where multiple assets can modify folder data, it is possible that multiple users could change the data at virtually the same time, causing unwanted results.

Also, if a user makes a copy of an asset and pastes it in a new location, the folder data will not move with it. Depending on the template configuration, this data could become orphaned if the template cannot determine the path to the original folder is from its new location. Or, if the template can determine the location of the data, then the newly moved asset may be saving data in an area that is inconsistent with its location.

Despite the drawbacks, the benefits of folder storage typically outweigh the risks. Maintaining a good awareness of your CMS configuration and the manner in which your users work with the CMS can help reduce any of the issues related to folder storage.

HOME PAGE WITH SLOTTING, INDEXING, AND CONTENT

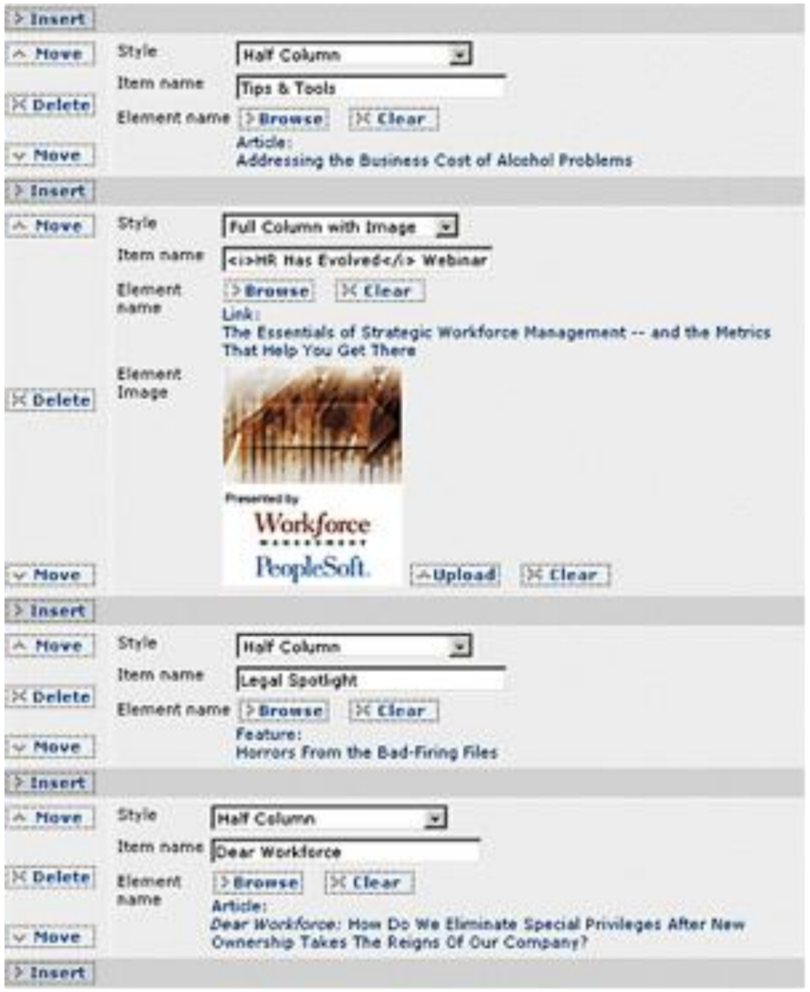
Perhaps more than most other pages, the home page is unique in its form and function within both the live site and the CMS. It has a duty to be informative and provide new content to users. It also serves as the primary point of entry for the site, thereby making it the launching pad for all site navigation. As a result, the Home Page template can be comprised of a mixture of publishing controls, providing the best combination of functionality suited to various content needs.

Within the CMS, there are three main types of controls that might apply to the home page:

- Content entry
- Slotting
- Indexing

Content entry controls would include text boxes, text areas, and WYSIWYG fields. This allows for freeform content that is specific to the home page, and thus stored and managed by the Home Page template.

Slotting controls allow you to draw content from other sources to add to the content of your home page. Here is a look at a Slotting interface:



Style: Half Column
Item name: Tips & Tools
Element name: Article:
Addressing the Business Cost of Alcohol Problems

Style: Full Column with Image
Item name: <i>HR Has Evolved</i> Webinar
Element name: Link:
The Essentials of Strategic Workforce Management -- and the Metrics That Help You Get There
Element Image

Style: Half Column
Item name: Legal Spotlight
Element name: Feature:
Horrors From the Bad-Firing Files

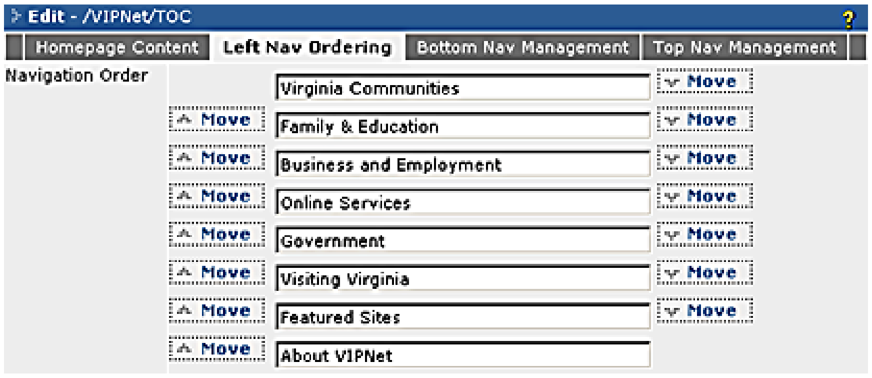
Style: Half Column
Item name: Dear Workforce
Element name: Article:
Dear Workforce: How Do We Eliminate Special Privileges After New Ownership Takes The Reigns Of Our Company?

Technically, the interface is just using the Panel construct within the template. So, the "Slotting" nature of it comes more from the details of how it is implemented.

The key item in the above interface is the "Element Name" which is using the Select Document function. By using this, other documents in the CMS can be referenced from the current asset (in this case, the home page). The ID of that selected document is then stored in the home page. By having the ID for other assets stored within it, the home page display code (preview.asp and/or output.asp) can then locate that content and bring in selected elements to fill out the overall home page layout. So, content has been "slotted" into available "spaces" on the home page.

While slotting content could be as simple as just providing the Select Document function, the above example shows a more robust version where specific style and labeling information can be supplied with the slotted content.

Indexing is the third method that a home page might use to manage content. Similar to a Press Release index, the home page might need to bring in and link to a set list (or “index”) of other content. The home page might also be used to manage the ordering of the top-level navigation. Either of these techniques would utilize the “mini-move” Panel interface.



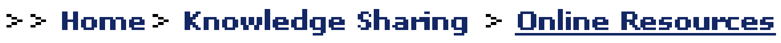
Virginia Communities
Family & Education
Business and Employment
Online Services
Government
Visiting Virginia
Featured Sites
About VIPNet

In this instance, a list of content has been pulled in from a location in the CMS for manual ordering within the home page. Again, all that is stored in the Home Page is the ID of each item and its ordering, but that is sufficient for getting any information you might need about the content.

Note that it would also be possible to bypass this step, and pull a list of content in directly to the preview.asp or output.asp file, if you did not want someone to edit or re-order the list.

BREADCRUMBS

Breadcrumbs are now a common navigational element that displays a linked path to the current content being viewed, listing each parent node in a hierarchy that is then made linkable. This provides an added ability to navigate around the structure of a site, with the added benefit of breadcrumbs occupying a minimum of screen real estate.



Using the CMS, breadcrumbs can be easily added to a template, thereby providing your site with an enhanced navigational system. There are many ways of implementing breadcrumbs. The approach illustrated below assumes a fairly hierarchical file/folder structure in your CMS implementation.

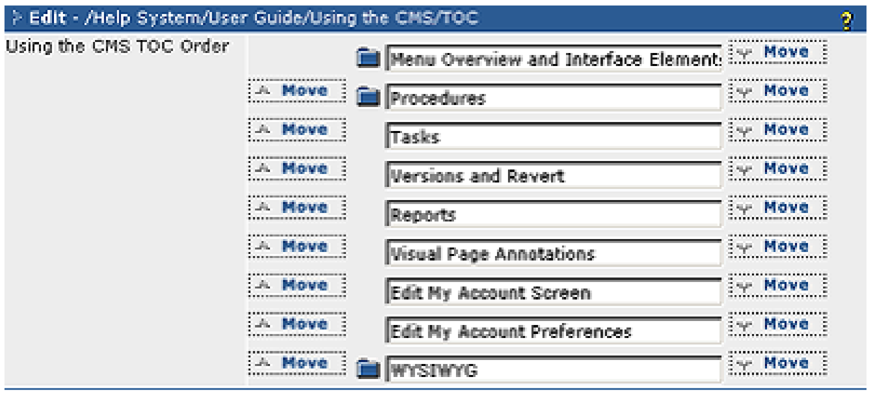
```
<!-- BREADCRUMB NAV BEGIN ->


<%

  Dim folder_parts

  folder_parts = Split(content.item("_CMSFolder"),
"/")
```

```
%>
 <tr>
    <td colspan="3">
        <br>
        >> <a href="<%=
asset.getLink("/Intranet/Homepage/Homepage") %>"
class="crumb">Home</a>>

    <% Dim link_path
      for counter = 2 to ubound(folder_parts)
        link_path = link_path & "/" &
folder_parts(counter)
    %>
        <a href="<%=
asset.getLink("/Intranet/Intranet" & link_path &
"/Intranet Index") %>" class="crumb"><%=
folder_parts(counter) %></a> >

    <% next %>

        <a href="<%=
asset.getLink(content.item("_CMSId")) %>"
class="crumb" style="text-decoration:
underline;"><%= content.item("id_title") %></a>

        <br><hr noshade size="1" style="color:
#1F3874;">
  </td>
 </tr>
<!-- BREADCRUMB NAV END ->
```

Note that the above breadcrumb scheme assumes that the current document will be listed as the final item in the breadcrumb list. The presence (or absence) of the current document is simply a matter of your site's navigational design.

UNIFIED FILE AND FOLDER SORTING

Just as separate lists of files and folders may be sorted to provide ordering for content or navigation, a unified list of both files and folders may be sorted as well.


Edit - /Help System/User Guide/Using the CMS/TOC

Using the CMS TOC Order

	Menu Overview and Interface Element:	Move
Move	Procedures	Move
Move	Tasks	Move
Move	Versions and Revert	Move
Move	Reports	Move
Move	Visual Page Annotations	Move
Move	Edit My Account Screen	Move
Move	Edit My Account Preferences	Move
Move	WYSIWYG	Move

With unified lists, it is important to identify the type of each listed item within the input interface, without needing to correlate the labels with their respective type by memory. The visual cue (e.g., a folder icon) requires extra code in the panel sorting routine.

In our example, there is an initial loop through the file/folder list prior to the panel loop. This identifies if there are any folders in the list being retrieved, by checking _CMSType: (important in formatting the list). By doing a preemptive check, the panel loop will not include an invisible spacer in front of files if there are no folders in the list.

```
<% Dim folder_path
  folder_path = Split(content.item("_CMSFolder"),
"/")
%>
<TR ALIGN=justify VALIGN=top BGCOLOR="#F0F0F0">
<TD class="tabletext"><%=
folder_path(ubound(folder_path)) %> TOC Order</TD>
<TD><TABLE class="bodytext" border=0 cellpadding=0
cellspacing=0>
<% Dim folder_check %>
<% folder_check = 0 %>
<% asset.setParam "exclude", "TOC" %>
<% set check_list = asset.createListFromFolder("/"
& content.item("_CMSFolder") & "/", "toc_label",
"toc_id") %>
<% do while check_list.nextEntry() %>
  <% if check_list.item("_CMSType") <> "file"
then%>
    <% folder_check = 1 %>
  <% end if %>
<% loop %>
<% asset.setParam "exclude", "TOC" %>
<% set toc_list = asset.createListFromFolder("/" &
content.item("_CMSFolder") & "/", "toc_label",
"toc_id") %>

<% toc_list.setParam "panel_style", "mini_move" %>

<% do while toc_list.nextPanel() %>

  <input type="hidden" name="toc_id" value="<%=
toc_list.item("toc_id") %>">

  <% if folder_check = 1 then%><% if
toc_list.item("_CMSType") <> "file" then%><img
src="/cpt_icons/folder.gif">&#160;<% else %><img
src="/cpt_media/spacer_empty.gif" height="16"
width="16">&#160;<% end if %><% end if %><input
size=35 type="text" name="toc_label" value="<%=
toc_list.item("toc_label") %>" onClick="blur()"
onSelect="blur()">


<% loop %>

</TABLE></TD></TR>
```

TEXT FIELD AUTO COMPLETION

To create an auto complete, use the following in input.asp:

```
<script language="Javascript">

var words = new Array("test", "this", "thing",
"out");

var words2 = new Array("and", "apple", "acrobat",
"annoymous", "ariel", "anthony");

</script>


<tr>

<TR align="justify" valign="top" bgcolor="F0F0F0">

<TD>Text</TD>

<TD>

<input type="text"
onkeyup="autoComplete(words)"><br>

<input type="text" onkeyup="autoComplete(words2)">

</TD>

</TR>
```

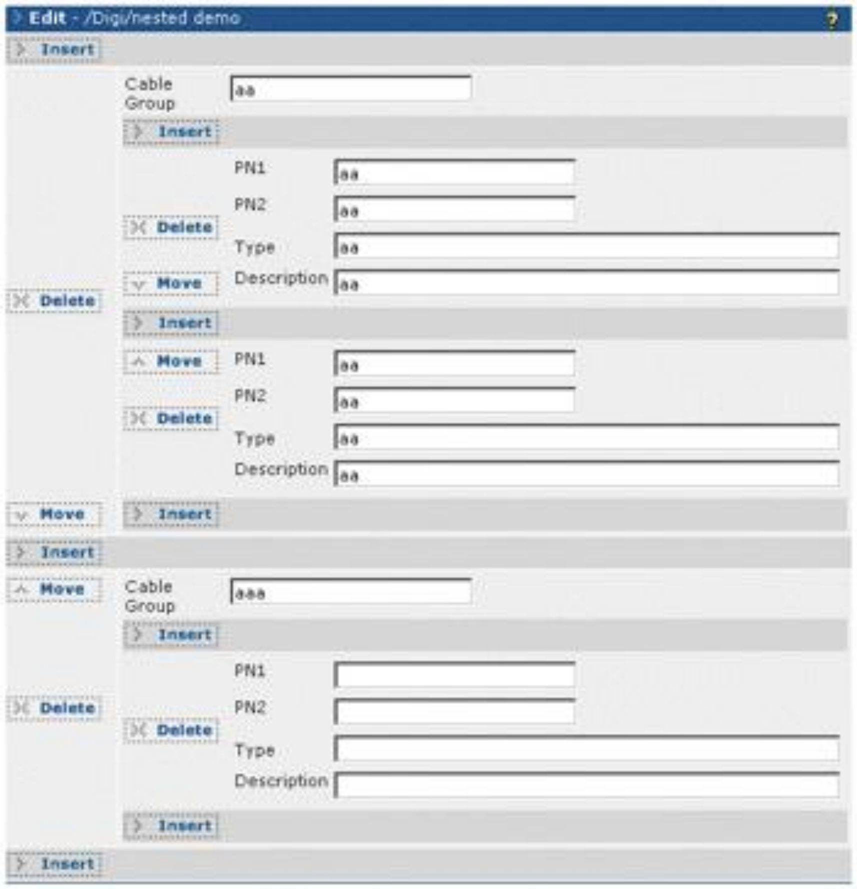
↗ Tips

1. The words do NOT need to be sorted; the JavaScript does that for you.
2. Move the cursor up/down to find additional matches
3. Shift-Cursor Right to add the currently selected text to your typing.

4. You can use the post_input.asp to add the new values to a separate asset so the list will grow with new entries. Ask your CrownPeak production lead for more information.

NESTED PANELS

Nested panels are also possible. Here's one example:



For the above panel, the code in the input.asp template file would look like this:

```
<% set list = content.createList("cable_group") %>
<% do while list.nextPanel() %>
<TABLE WIDTH="100%" CLASS=tabletext>


 <TR ALIGN=justify VALIGN=top BGCOLOR=F0F0F0>
  <TD>Cable Group</TD>
  <TD>
   <INPUT SIZE=25 TYPE=text NAME="cable_group"
VALUE="<% if list.item("cable_group") <> "" then
%><%= list.item("cable_group") %><% end if %>">
  </TD>
 </TR>


 <% index = list.itemIndex %>
 <% set list2 = content.createList("pn1_"& index)
%>
 <% do while list2.nextPanel() %>
  <TABLE WIDTH="100%" CLASS=tabletext>
 <TR ALIGN=justify VALIGN=top BGCOLOR=F0F0F0>
  <TD>PN1</TD>
  <TD>
   <INPUT SIZE=25 TYPE=text indexdepth=2
indexname="pn1_" NAME="pn1_<%= index%>" VALUE="<%=
list2.item("pn1_"&index) %>">
  </TD>
 </TR>
 <TR ALIGN=justify VALIGN=top BGCOLOR=F0F0F0>
  <TD>PN2</TD>
  <TD>
   <INPUT SIZE=25 TYPE=text indexdepth=2
indexname="pn2_" NAME="pn2_<%= index%>" VALUE="<%=
list2.item("pn2_"&index) %>">
  </TD>
 </TR>
  <TD>Type</TD>
  <TD>
   <INPUT SIZE=55 TYPE=text indexdepth=2
indexname="type_" NAME="type_<%= index%>"
VALUE="<%= list2.item("type_"&index) %>">
  </TD>
 </TR>
 <TR ALIGN=justify VALIGN=top BGCOLOR=F0F0F0>
  <TD>Description</TD>
  <TD>
   <INPUT SIZE=55 TYPE=text indexdepth=2
indexname="desc_" NAME="desc_<%= index %>"
VALUE="<%= list2.item("desc_"&index) %>">
  </TD>
 </TR>
 </TABLE>
 <%loop%>
</TABLE>
<% loop %>
```

And here is a corresponding output.asp template file:

```
<table>
 <% set list = content.createList("cable_group") %>
 <% do while list.nextPanel() %>
  <TR>
   <TD COLSPAN=4><B><%= list.item("cable_group")
%></B></TD>
  </TR>
  <% index = list.itemIndex %>
  <% set list2 = content.createList("pn1_"& index)
%>
  <% do while list2.nextPanel() %>
   <TR>
    <TD><%= list2.item("pn1_"&index) %></TD>
    <TD><%= list2.item("pn2_"&index) %></TD>
    <TD><%= list2.item("type_"&index) %></TD>
    <TD><%= list2.item("desc_"&index) %></TD>
   </TR>
   <% loop %>
  <% loop %>
</TABLE>
```

If the internal panel is a file list, you need add a couple of special parameters to support the selected file:

```
<% input.setParam "indexdepth",2 %>
<% input.setParam "indexname","select_file_" %>
```

Here is the full example:

```
<TR ALIGN=justify VALIGN=top BGCOLOR=F0F0F0>
 <TD>Select File</TD>
 <TD><SPAN NAME=select_file_span<%= index %>
ID=select_file_span<%= index %>>
  <% if list2.item("select_file_"&index) <> "" then
%>
  <%=
asset.getLabel(list2.item("select_file_"&index)) %>
  <% end if %></SPAN>
  <% input.setParam "label_span",
"select_file_span" & index %>
  <% input.setParam "extensions", "pdf doc xls pnt
rtf dwg zip" %>
  <% input.setParam "default_folder","/Assets/" %>
  <% input.setParam "indexdepth",2 %>
  <% input.setParam "indexname","select_file_" %>
  <% input.showSelectDocument "select_file_"&index, 
list2.item("select_file_"&index) %>
 </TD>
</TR>
```

REMOVING CONTENT DATA IN THE POST_INPUT.ASP TEMPLATE FILE

There are some occasions where you might need to remove computed content from an asset when it is being saved (in the post_input.asp template file). The system will not delete fields when an asset is saved unless the field is in the input.asp form and has a null value. If the field isn't mentioned in the input.asp file, then the data will remain in the asset. This is an issue for computed field that are added in the post_input.asp template file since you may want to clear out the computed fields and add new ones.

One example of this is in a calendar application where you cache the months and days an event covers in the event so that selection and filtering are easier when building monthly and daily indexes.

The first block of code below removes and existing variables, and the second blocks add the new data:

```
 ' remove any of the previous date caching
  set dbcontent = 
asset.getContent(content.item("_CMSID"))
  set clist = dbcontent.getTable()
  for each name in clist
   if Left(name, 4) = "day_" or Left(name, 4) = 
"mon_" then
    clist.item(name) = ""
   end if
  next
  asset.setContent content.item("_CMSID"), 
dbcontent


  ' cache the months this event covers for faster 
retrieval day_YYYY_MM
  curDate = content.item("start_date")
  if content.item("end_date") <> "" then
   for i = 1 to DateDiff("m", 
content.item("start_date"), 
content.item("end_date")) + 1
```

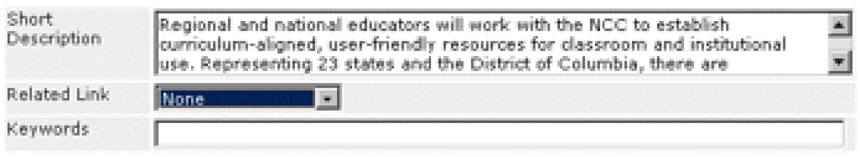
```
    content.add "mon_" & clock.formatDate(curDate, "YYYY_0MM"), "1"
    curDate = DateAdd("m", 1, curDate)
   next
  else
   content.add "mon_" & clock.formatDate(curDate, "YYYY_0MM"), "1"
  end if



  ' cache the days this event covers for faster retrieval day_YYYY_MM_DD
  curDate = content.item("start_date")
  if content.item("end_date") <> "" then
   for i = 1 to DateDiff("d", content.item("start_date"), content.item("end_date")) + 1
    content.add "day_" & clock.formatDate(curDate, "YYYY_0MM_0DD"), "1"
    curDate = DateAdd("d", 1, curDate)
   next
  else
   content.add "day_" & clock.formatDate(curDate, "YYYY_0MM_0DD"), "1"
  end if
```

The other way to do this is to add hidden fields in the input.asp with empty strings for the values for all the variables you want to remove.

HIDING FIELDS BASED ON OTHER FIELDS IN INPUT FORMS, REVISITED

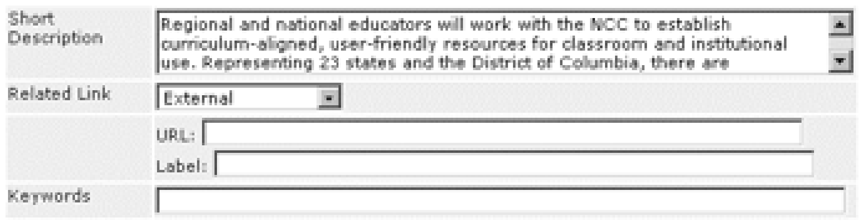
To hide fields in input forms based on drop-down selection or other input, you can use DHTML and JavaScript. For example, a generic link might be internal, external, or from a file upload. This would look like the following in the input form—watch the related link field.

Field with no links:



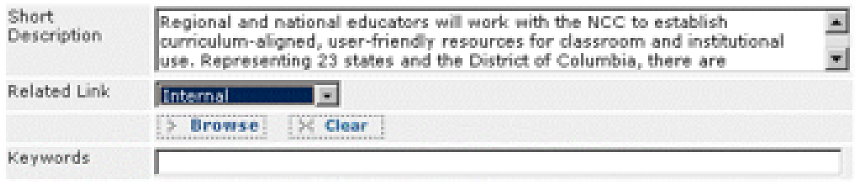
Short Description	Regional and national educators will work with the NCC to establish curriculum-aligned, user-friendly resources for classroom and institutional use. Representing 23 states and the District of Columbia, there are
Related Link	None
Keywords	

Field with external link—shows the URL and label fields:



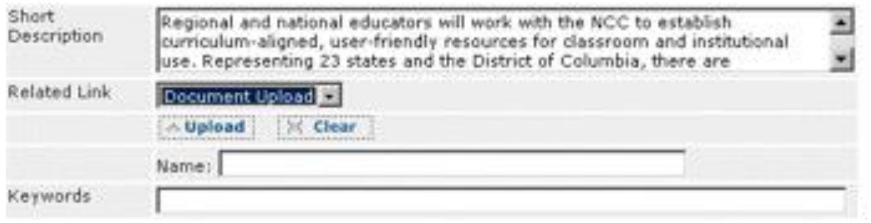
Short Description	Regional and national educators will work with the NCC to establish curriculum-aligned, user-friendly resources for classroom and institutional use. Representing 23 states and the District of Columbia, there are
Related Link	External
	URL:
	Label:
Keywords	

Field with internal link—shows the **Browse** button:



Short Description	Regional and national educators will work with the NCC to establish curriculum-aligned, user-friendly resources for classroom and institutional use. Representing 23 states and the District of Columbia, there are
Related Link	Internal
	Browse Clear
Keywords	

Field with file upload—shows **Upload** button and field for Label:



Short Description	Regional and national educators will work with the NCC to establish curriculum-aligned, user-friendly resources for classroom and institutional use. Representing 23 states and the District of Columbia, there are
Related Link	Document Upload
	Upload Clear
	Name:
Keywords	

To accomplish this behavior, we use DHTML/JavaScript to show and hide the fields based on a selection of the Related Link drop-down menu. Note that like the tabs, all of the fields exist in the page. They are just shown and hidden as needed.

Add the following JavaScript near the top of the input.asp template file. This uses the value of the menu to show and hide table rows with the value as part of their ID.

```
<script language="javascript">
<!--

function showLinkType() {
var currentValue = event.srcElement.value
var oParent = event.srcElement
var oChild = ""

while ((oParent != null) && (oParent.tagName !=
"TR")) {
  oParent = oParent.parentNode;
}

oChild = oParent.nextSibling;

for (i = 1; i < oParent.parentNode.children.length;
i++) {
 if (oChild.id.length > 0) {
  if (oChild.id.indexOf(currentValue) == -1) {
    oChild.style.visibility = "hidden";
    oChild.style.display = "none";
  } else {
    oChild.style.visibility = "visible";
    oChild.style.display = "block";
  }
  }
  oChild = oChild.nextSibling;
}

} //end showLinkType

//->
</script>
```

Here is the input.asp code for the fields themselves:

```
    <TR align="justify" valign="top"
bgcolor="F0F0F0">
     <TD>Related Link</b></TD>
     <TD>
```

```
      <select name="rl_type"
onChange="showLinkType()">

       <option <% if content.item("rl_type") =
"nul" then %>SELECTED<% end if %>
value="nul">None</option>

       <option <% if content.item("rl_type") =
"ext" then %>SELECTED<% end if %>
value="ext">External</option>

       <option <% if content.item("rl_type") =
"int" then %>SELECTED<% end if %>
value="int">Internal</option>

       <option <% if content.item("rl_type") =
"doc" then %>SELECTED<% end if %>
value="doc">Document Upload</option>

      </select>

     </TD>

    </TR>


   <TR id="ext_field:" <% if
(content.item("rl_type") <> "ext") then %>
style="visibility: hidden; display: none;"<% end if
%> ALIGN="justify" VALIGN="top" bgcolor="F0F0F0">

    <TD></TD>

    <TD>URL: <INPUT size=65 type="text"
name="rl_ext" value="<%=
content.escapeItem("rl_ext") %>">

    <br>Label: <INPUT size=65 type="text"
name="rl_ext_lbl" value="<%=
content.escapeItem("rl_ext_lbl") %>"></TD>

   </TR>


   <TR id="int_field:" <% if
content.item("rl_type") <> "int" then %>
style="visibility: hidden; display: none;"<% end if
%> ALIGN="justify" VALIGN="top" bgcolor="F0F0F0">

     <TD></TD>

     <TD>

      <span name="rl_int_span" id="rl_int_span">

      <% if content.item("rl_int") <> "" then %>

       <%= asset.getLabel(content.item("rl_int"))
%>

      <% end if %>

       </span>

      <% input.setParam "label_span", "rl_int_span"
%>
```

```
      <% input.showSelectDocument "rl_int",
content.item("rl_int") %>

     </TD>

   </TR>


   <TR id="doc_field1:" <% if
content.item("rl_type") <> "doc" then %>
style="visibility: hidden; display: none;"<% end if
%> ALIGN="justify" VALIGN="top" bgcolor="F0F0F0">

    <TD></TD>

    <TD>

     <% if content.item("rl_doc") <> "" then %>

      <a href="javascript:openPopup('<%=
content.item("rl_doc") %>', 640, 480);">VIEW</a>

     <% end if %>

   <% 'input.setParam "extensions", "pdf doc xls
ppt jpg jpeg gif png psd" %>

   <% 'input.setParam "default_folder", "/" &
content.item("_CMSFolder") & "/" %>

     <% input.showUploadDocument "rl_doc",
content.item("rl_doc") %>

    </TD>

   </TR>


   <TR id="doc_field2:" <% if
content.item("rl_type") <> "doc" then %>
style="visibility: hidden; display: none;"<% end if
%> ALIGN="justify" VALIGN="top" bgcolor="F0F0F0">

    <TD></TD>

    <TD>Name: <INPUT size=50 type="text"
name="rl_docname" value="<%=
content.escapeItem("rl_docname") %>"></TD>

   </TR>Some sample output.asp code for the above
fields:

 <% if content.item("rl_type") <> "nul" then %>

  <p class="eventsDetail"> <span
class="eventsDate">Related Link</span>:

   <% select case content.item("rl_type")

     case "int" %>

    <a href="<%=
asset.getLink(content.item("rl_int")) %>"><%=
asset.getContentField(content.item("rl_int"),
"headline") %></a><br>

   <% case "ext" %>
```

```
    <% if content.item("rl_ext_lbl") = "" then %>

     <a href="<%= content.item("rl_ext") %>"
target="_blank"><%= content.item("rl_ext")
%></a><br>

    <% else %>

     <a href="<%= content.item("rl_ext") %>"
target="_blank"><%= content.item("rl_ext_lbl")
%></a><br>

    <% end if %>

   <% case "doc" %>

    <a href="<%= content.item("rl_doc") %>"><IMG
BORDER=0 SRC="<%=
filename.geticon(content.item("rl_doc")) %>"> <%=
content.item("rl_docname") %></a>

   <% end select %>

  </p>

 <% end if %>
```

The code for this gets a bit more complex if you want several of these, or if you want them in a panel. Please contact your production lead for sample code to cover more complex cases.

HIDING FIELDS BASED ON USER GROUPS

You can use the User Group affiliations of the current user to show/hide fields if necessary. This is sometimes helpful, but use it sparingly since it makes the template code harder to maintain. The following code would be used in the input.asp template file:

```
<% if user.ingroup("Editors") then %>

 <TR align="justify" style="visibility: hidden;
display: none;" valign="top" bgcolor="F0F0F0">

 <TD>HTML FORM</TD>

 <TD>

  <TEXTAREA rows=50 cols=75 name="html_form"><%=
content.escapeItem("html_form") %></TEXTAREA>

 </TD>

</TR>

<% end if %>
```

HANDLING MULTI-SELECT INPUT FIELDS

For multi-select menu drop-downs, you need to set the selection when an asset is re-edited. The list.contains API function makes this easy as is shown by this sample code from an input.asp template file:

```
<TR align="justify" valign="top" bgcolor="F0F0F0">
  <TD>Grade Level</TD>
  <TD>
   <% set vlist = content.createList("grade_level")
%>
   <SELECT name="grade_level" multiple size=3>
    <option value="1" <% if
vlist.contains("grade_level", "1") then%>SELECTED<%
end if %>>Elementary</option>
    <option value="2" <% if
vlist.contains("grade_level", "2") then%>SELECTED<%
end if %>>Middle</option>
    <option value="3" <% if
vlist.contains("grade_level", "3") then%>SELECTED<%
end if %>>Jr. High</option>
    <option value="4" <% if
vlist.contains("grade_level", "4") then%>SELECTED<%
end if %>>High School</option>
   </SELECT>
   (use <Ctrl>-click to select multiple grades)
  </TD>
 </TR>
```

You can also easily use another list to supply the options if the list is maintained in another asset. Ask your production lead if you would like more complex examples.

5880 West Jefferson Boulevard, Unit G
Los Angeles, California 90016

p. 310-841-5920   f. 310-841-5913

www.crownpeak.com