CrownPeak™

TECHNICAL DOCUMENTATION 

CROWNPEAK CLASSIC API REFERENCE GUIDE

CROWNPEAK CMS

February 2012

© 2012 CrownPeak Technology, Inc.

All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from CrownPeak Technology.

TABLE OF CONTENTS

INTRODUCTION

This Application Programming Interface (API) Reference Guide provides descriptions of the objects available for you to use within the CMS templates. The CMS API extends ASP classic, which is the foundation of the templating environment.

The various routines encapsulate large groups of functions which simplify and optimize the code. For example, the interaction between the templates and the database have been abstracted.

API calls are there for your convience and have been tested over many implementations.

Note: In theory, you can write your own functions that replicate existing API functionality. However, it is usually unnecessary because a solution most likely already exists within the API.

If you have any questions regarding the API, please refer to the Getting Help section on Page 3 of this guide.

GETTING HELP

ONLINE REFERENCE

The entire contents of this document can be found online via the CrownPeak CMS Help menu. Please refer to the online version for the latest version of the help system. In addition to this help guide, there are several online links to other sources of information, and extensive reporting options.

SUPPORT NUMBERS

CrownPeak Technology can be reached at

5880 West Jefferson Blvd.
Suite G
Los Angeles, CA 90016
(310) 841-5920
support@crownpeak.com

The support@crownpeak.com e-mail address should only be used for emergencies and off hour support requests. In general, please contact your client service director.

SUPPORT POLICY

A certain number of service hours are included in each contract. These hours can be used for:

- Updates to templates.
- Training of additional staff.
- Requests pertaining to issues not caused by the CrownPeak CMS system.
- Requests for advice on site construction, capabilities of the system etc.
- Issues arising from changes to published pages on the target system.

If an issue is determined to be due to an error in the core CMS system (not the implementation). the service hours will not be charged for the work.

New templates will require an extension of the current service agreement.

Any time above the included service hours will be billed at an hourly rate.

CrownPeak may change the interfaces as part of the upgrade process. New documentation will be provided with each interface update. While these changes may require a certain amount of re-learning, they are often created as a result of making the interfaces more efficient, more secure etc.

For more information regarding support, please refer to the troubleshooting guide.

CROWNPEAK CMS TEMPLATE FILE API

Most of the CrownPeak CMS is configurable via the browser interface. There are many times, however, when GUI-based configuration and manipulation are not sufficient. To allow customization at a lower level, the CMS uses several template event files to accomplish certain tasks related to the manipulation of assets. (Note that the term “assets” refers to both files and folders.) These files are strictly grouped and comprise a template. A template is defined by the event files that codify actions for a particular type of document (for example, press release, job listing, etc.).

The most basic template usually includes two of these files, input.asp and output.asp, which define the data entry form and the data presentation layout respectively. The following is a list of possible template files and their purpose:

- new.asp – used to initialize variables when a new page is created for the first time.
- input.asp – used to define the HTML form layout for content entry.
- output.asp – used to define the HTML presentation or other layout code.
- preview.asp – similar to output.asp but is only used within the CMS to view files and is not published. Often used to remove navigation not needed when previewing the files within the CMS.
- frame.asp – used in conjunction with Preview/Output to define a framed based layout.
- post_input.asp – used to process data submitted using the input.asp file before entry into the CMS.
- post_save.asp – used to process data submitted from the input.asp file AFTER the information has been saved to the database as opposed to post_input.asp which is called before saving the information.
- filename.asp – used to alter the filename for a particular document prior to publishing or deployment.
- url.asp – used to process hyperlinks in a page before publishing or deployment.
- assetfilename.asp – used to process the filename for uploaded assets prior to publishing or deployment.

- asseturl.asp – used to process any urls to media assets within an HTML pages.
- upload.asp – used to process uploaded assets within the input.asp file. Often used for image scaling or manipulation.
- post_publish.asp – called after the asset has been published to the Web site.
- delete.asp – called when a template page is deleted
- copy.asp – called when a template page is copied, pasted, branched or cloned (_cmsAction)
- smtp_import.asp – called when email is available for the template
- ftp_import.asp – used to import remote FTP files into the CMS by polling remote FTP sites.
- odbc_import.asp - used to import the results of ODBC SQL queries into the CMS.
- import.asp – used when importing content via urls into the CMS
- http_delete.asp
- http_update.asp
- http_insert.asp – used to create the HTTP post statement prior to HTTP publishing/deploy.
- smtp_delete.asp
- smtp_update.asp
- smtp_insert.asp – used to create the email text prior to an SMTP tramission.
- odbc_delete.asp
- odbc_update.asp
- odbc_insert.asp – used to define the SQL statement that is issued against the remote DB.
- soap_delete.asp
- soap_update.asp
- soap_insert.asp – used to create the XML SOAP request statement prior to a SOAP publish/deploy.
- login.asp - located in /System/login.asp - executed on each login into the CMS. Can be used to change group permissions, etc on login.

Templates can be created using as many or as few of these files as needed. They only extend the functionality of the system and are not required for the CMS to operate (with the exception of the output.asp file).

Each of these template files are script files (VBScript or JScript) and are available within a template's folder. While they can be edited using an external editor (such as Visual Interdev), an editor is provided within the browser that offers specific CMS capabilities. The script files allow most VBScript and JScript statements as in any Microsoft ASP page, but are limited in their ability to perform insecure actions such as random access to the hosting machine's filesystem, database, services, registry, etc.

Within each of these template files, several objects are defined and available for use. They provide functionality specific to the CMS. These objects are:

- Input – provides functions that are primarily used in the input.asp file to create form based structures.
- Asset – provides functions that are used to access information from the CMS files and folders.
- Content – holds the current assets content as entered from the input.asp file.
- List – provides listing/looping mechanisms for iterating through data.
- Util – various utility functions.
- Clock – provides additional date functions above the ASP Date(), Month(), etc.
- User – provides an interface to user objects within the CMS.
- Response – the regular ASP response object.
- Request – the regular ASP request object.
- System – various system level routines such as creating objects, including files, etc.
- Image – provides some basic image operations (scaling, overlaying, etc).

All objects are accessed using the '.' notation. For example, the most common action is to print out content entered by the user:

```
<%= content.item("title") %>
```

Where 'content' is the object, "item" is a function within that object that accesses its data and "title" is the data field that you wish to access. Note

the familiar shorthand "=" used to print out (response.write) any text. This is the most common type of statement seen in the output.asp file.

Each object has several functions that provide the functionality of the object. Each function within an object requires certain default parameters to function correctly. Due to the limitation that VBScript routines cannot expand their parameters and do not allow polymorphism without breaking existing code (unlike Java), the setParam routine allows the user to set parameters that can be used by successive routines. This allows routines to be upgraded and provide additional capabilities without requiring a rewrite of existing code.

CONTENT OBJECT

The content object always contains the current data fields (and other metadata) about the current asset being worked on and is automatically populated before any template file is executed.

↗ Asset Variables

- _cmsBaseModelId - The ID of the asset that was used as the model for the current asset.
- _cmsBaseStatus - The status numeric ID of the asset that was used as the model for the current asset.
- _cmsBaseStatusColor - The status color of the asset that was used as the model for the current asset.
- _cmsBaseStatusText - The status text (live, stage, etc) of the asset that was used as the model for the current asset.
- _cmsBranchId - The asset's ID that this document is a branch of or the current asset's id if it is not a branched asset.
- _cmsChangeDate - The date the current asset was last changed by either content or configuration modifications.
- _cmsChangeUserId - The user's ID that last changed the current asset by either content or configuration modifications.
- _cmsCheckoutDate - The date the current asset was checked out.
- _cmsCheckoutUserId - The user's ID that checked out the current asset.
- _cmsCommentsCount - The number of comments associated with the current asset.

- _cmsCreateDate - The date the asset was first created.
- _cmsCreateUserId - The user that created the asset.
- _cmsFilename - The label of the current asset.
- _cmsFolder - The full folder path that contains the current asset. (/folder:1/folder:2/folder:3/etc)
- _cmsFolder:1 - The first folder part of the current asset. (/folder:1/folder:2/folder:3/etc)
- _cmsFolder:2 - The second folder part of the current asset. (/folder:1/folder:2/folder:3/etc)
- _cmsFolder:3 - The third folder part of the current asset. (/folder:1/folder:2/folder:3/etc)
- _cmsFolder:4 - The fourth folder part of the current asset. (/folder:1/folder:2/folder:3/etc)
- _cmsFolderCount - The number of folders in the full folder path (i.e. the folder depth) of the current asset.
- _cmsFolderId - The folder's numeric id that contains the current asset.
- _cmsId - The current asset's numeric ID.
- _cmsIdPath - The numeric folder path that contains the current asset.
- _cmsLabel - The label of the current asset.
- _cmsLayout - The current layout being used to render the current asset.
- _cmsLiveDate - The date the current asset was last published to any site.
- _cmsLiveUserId - The user's ID that last republished the current asset.
- _cmsModifiedDate - The last date the current asset's content was modified.
- _cmsModifiedUserId - The user's ID that last modified the current asset's content.
- _cmsPath - The folder+label of the current asset.
- _cmsPropertiesFilename - The filename value from the publishing properties page currently being used to publish the current asset.
- _cmsPropertiesFolder - The folder value from the publishing properties page currently being used to publish the current asset.
- _cmsRetireDate - The date the current asset was removed from a site.

- _cmsRetireUserId - The user's ID that removed the current asset from a site.
- _cmsRootFolder - The root folder of the current asset (/rootFolder/folder:2/folder:3/topFolder/).
- _cmsRootFolder - the bottom folder in the path to this asset.
- _cmsRootFolderId - The root folder ID of the current asset (/rootFolderId/folderId:2/folderId:3/topFolderId/).
- _cmsShortcutId - The numeric ID of the asset that the current asset is a shortcut of, otherwise the current asset's ID.
- _cmsSize - The size in bytes of the current asset.
- _cmsStatus - The status numeric ID of the current asset.
- _cmsStatusColor - The status color of the current asset.
- _cmsStatusDate - The date the status was last changed for the current asset.
- _cmsStatusText - The status text (Live, Stage, etc) for the current asset.
- _cmsStatusUserId - The user's ID that last changed the current asset's status.
- _cmsSystem - The URL to the CMS system including querystring information.
- _cmsTemplateId - The numeric ID of the current asset's template.
- _cmsTemplateName - The name of the current asset's template.
- _cmsTemplatePath - The path of the current asset's template (typically /system/templates/name).
- _cmsTopFolder - The top folder name for the current asset (/rootFolder/folder:2/folder:3/topFolder/).
- _cmsTopFolderId - The top folder numeric id for the current asset (/rootFolderId/folderId:2/folderId:3/topFolderId/).
- _cmsType - The type (file, folder, mount) of the current asset.
- _cmsUrl - The URL to the CMS system.
- _cmsWorkflowId - The workflow numeric ID that the current asset belongs to.
- _cmsWorkflowName - The workflow name that the current asset belongs to.

The functions available in the content object are:

↗ setParam(name, value)

Sets an extended parameter for use by other functions.

```
<% setParam "sort_order", "asc" %>
```

↗ getParam(name)

Gets the current value of a previously set parameter.

```
<%= getParam("sort_order") %>
```

↗ delParam(name)

Deletes a parameter.

```
<% delParam "sort_order" %>
```

↗ text item(fieldname)

Accesses a particular field within the content object.

```
<%= content.item("article_body") %>
```

↗ Text itemAt(fieldname, index)

Returns the value of the fieldname located at the requested index. Returns empty quotes if the field does not exist, or index is out of bounds.

```
<%= content.itemAt "article_body",5 %>
```

↗ text defaultItem(fieldname, default_value)

Accesses a particular field within the content object, which if not found (or empty value) will return the default value instead.

```
<%= content.defaultItem("fullname", "My name here")
%>
```

↗ text escapeItem(fieldname)

Similar to "item" but escapes all HTML tags. This function escapes out any extended HTML characters (often these characters can be entered by cut/paste from MS Word), translates newlines to
 and tabs to <DD> to ensure that the text entered within a text field is rendered as close as possible to the entered text. It is often used to prevent someone from adding in HTML text within an input field and altering the presentation of that text.

```
<%= content.escapeItem("description") %>
```

↗ Text escapeItem2(text)

This function is similar to escapeItem except it does not escape &#xxxx;

```
<%= content.escapeItem2(“description”) %>
```

↗ setPrefix(text)

Sets the default content item prefix that is added to each item reference before accessing the field. This is used when multiple fields differ only by their prefix, such as in language translations.

```
<% content.setPrefix("french_") %>
<% ' prints out value of french_title
<%= content.item("title") %>
<% content.setPrefix("spanish_") %>
<% ' prints out value of spanish_title
<%= content.item("title") %>
```

↗ boolean exists(fieldname)

Determines if a data field exists in the content object. Note that a data field can exist in the content object even if its value is empty.

```
<% if content.exists("show_name") then %>
<%= content.item("name") %>
<% end if%>
```

↗ Bool hasValue(fieldname)

Returns true if fieldname is defined within the local instance of the content object otherwise returns false. It checks if fieldname is not null, 0, false, or off. It works on single fields and arrays.

```
<% if content.hasValue(“title”) then %>
  <% response.write “The content contains title” %>
<% else %>
  <% response.write “The content does not contain
title” %>
<% end if %>


<% if content.hasValue(“counter:2”) then %>
  do something
<% end if %>
```

↗ boolean isTrue(fieldname)

Determines if a data field contains a true value. A true value is defined as a non null value, not equal to "", "0", "false", or the boolean False.

```
<% if content.isTrue("is_release") then %>
Press Release
<% end if %>
```

↗ add(fieldname, fieldvalue)

Adds a name and value pair into the content object. This does not necessarily save that data to permanent storage unless explicitly saved. The post_input.asp file is an exception: any data within content manipulated within the process file will be automatically saved into permanent storage.

```
<% create a fullname field from user entered
firstname and lastname %>
<% (this statement is assumed to be in
post_input.asp) %>
<% content.add "fullname",
content.item("firstname") & "," &
content.item("lastname") %>
```

↗ rename(current_fieldname,new_fieldname)

Replaces current_fieldname, which previously existed in the content, with the new_fieldname. Rename will not automatically update all the fieldnames within a list, (counter:1, counter:2, . . .), but it can be done manually.

Single fieldname:

```
<% content.rename "event_title", "event_name" %>
<% content.item("event_name")
```

Multiple fieldnames:

```
<%
  for i = 1 to content.listSize("counter")
    content.rename "counter:" & i, "new_counter:" &
i
  next
  set list = content.createList("new_counter")
%>
```

↗ remove(fieldname)

Removes a data field from within the content object. Removing a field does not necessarily delete that field from the permanent storage.

```
<% content.remove "title" %>
```

↗ removePrefix(fieldname)

Removes one or more data fields whose name begins with a particular phrase.

```
<% ' add in some content %>
<% content.add "title_1", "my new book" %>
<% content.add "title_2", "my second book" %>
<% ' remove both title_1 and title_2 %>
<% content.removePrefix "title_" %>
```

↗ removeValuePrefix(fieldname)

Removes one or more data fields whose value begins with a particular phrase.

```
<% ' add in some content %>
<% content.add "title_1", "my new book" %>
<% content.add "title_2", "my second book" %>
<% ' remove both title_1 and title_2 %>
<% content.removeValuePrefix "my " %>
```

↗ dictionary getTable()

Returns the inner dictionary (Hashtable) of the content object. This is the same object that is typically created using Server.CreateObject("Scripting.Dictionary"). Modification of this object will modify data fields within the content object.

```
<% ' list all the fields within the content object
%>
<% set fields = content.getTable() %>
<% for each itm in fields %>
<%   response.write itm & "<br>" %>
<% next %>
```

↗ dictionary cloneTable()

Returns a copy of the inner dictionary (Hashtable) of the content object. This is the same object that is typically created using Server.CreateObject("Scripting.Dictionary"). Modification of this object will NOT modify data fields within the content object.

```
<% ' list all the fields within the content object
%>
```

```
<% set fields = content.cloneTable() %>
<% for each itm in fields %>
<%   response.write itm & "<br>" %>
<% next %>
```

↗ Text getHidden(fieldname)

Returns a hidden HTML input tag with the name attribute set equal to the given fieldname, and the value attribute set equal to the value mapped to the given fieldname.

```
<%=content.getHidden("price") %>
```

Output:

```
<input type="hidden" name="price" value="200.00"/>
```

↗ void clearData()

Removes all the content data items in the content object. This does NOT remove any CMS-specific variables (denoted with a _cms prefix) from the content object OR and variable that begins with an underscore (_).

```
<% ' remove all data after a post_input.asp has
used the information %>
<% asset.setContent(7423, content) %>
<% ' remove all the content from the content object
to ensure no data is %>
<% ' saved in this asset %>
<% content.clearData() %>
```

↗ bool hasContent()

Checks to see if the content object has any content in it (for example, fieldnames with values that do not begin with an underscore).

```
<% ' check if an asset has content in it %>
<% set fields = asset.getContent(7423) %>
<% if fields.hasContent() then %>
There is content in asset id 7423
<% end if %>
```

↗ dump()

Prints out the current data items of the content object in a table format. Mainly used for debugging purposes.

```
<% content.dump() %>
```

↗ number size()

Returns the number of data items (fieldnames that do not begin with an underscore) that are within the content object.

```
The content object has <%= content.size() %> data
items in it.
```

↗ text pageItem(page_number)

Returns the text that corresponds to appropriate page number specified. This function is used in conjunction with the WYSIWYG page break icon. If the page break icon is used this function will correctly return the appropriate page and setup pagination variables for multiple pages. Since this publishing loop will be set up automatically, this template will be called multiple times – once for each page in the asset.

```
<table>
<% ' show the current page, and the title at the
top of the first page
  totalPages = content.pageCount("txt_body")
  currentPage = content.pageNum
  if currentPage = 1 then %>
  <tr>
    <td colspan="2"><%=
content.escapeItem("long_title") %></td>
  </tr>
  <%end if%>
  <tr>
    <td colspan="2" height="2">
      <%= content.pageitem("txt_body", currentPage)
%><p>
    </td>
  </tr>
  <tr>
    <td colspan="2" height="2">
<% ' show the page numbers as links, and a next
icon
   ' similar code can be used for the prev link
      if totalPages > 1 then
        For i = 1 to totalPages
          if currentPage <> i then
            asset.setParam "args", "_pagenum=" & i
```

```
            response.write "<a href="" &
asset.getLink(content.item("_cmsId")) """>" & i &
"</a> |"
          else
            response.write "<b>" & i & "</b> |
"
          end if
        next
        if currentPage < totalPages then
          asset.setParam "args", "_pagenum=" &
currentPage+1
          response.write "<a href="""" &
asset.getLink(content.item("_cmsId")) & """>" &
"&gt;&gt;</a>"
        end if
      end if
%>
    </td>
  </tr>
</table>
```

The filename.asp also needs some extra code since the pagination will call the page template for each page:

```
<%
if content.item("_pageNum") <> "" and
content.item("_pageNum") <> "0" then
  content.add "_cmsPublishPath",
util.filterText(strPath & "-" &
content.item("_pageNum") & ".asp", "filepath")
else
  content.add "_cmsPublishPath",
util.filterText(strPath & ".asp", "filepath")
end if
%>
```

↗ list_object createList(fieldname)

Creates a list object based on the specified fieldname as the iterator. The fieldname is assumed to be one from a panel or have been created as a list item. The list object created contains all the fields associated with that fieldname variable corresponding to the current entry (1, 2, 3, etc).

```
<%
set list = content.createList("subject")
```

```
do while list.nextPanel()
%>
  <input type="text" value="<%=
list.escapeItem("subject") %>"><br>
  <input type="text" value="<%=
list.escapeItem("message") %>"><br>
<% loop %>
```

↗ list_object createListPrefix(fieldname, alt_fieldname)

Creates a list from a single list entry based on the prefix of fieldnames contained in the current entry. The alt_fieldname provides a uniform way of accessing the values once the list is created.

```
<% set fields =
asset.getContent("/path/myasset.txt") %>
<% ' asset contains sel_1=545, sel_2=332,
sel_3=777, etc
<% set list = fields.createListPrefix("sel_",
"selected") %>
<% do while list.nextEntry() %>
  <% response.write list.item("selected") & "<br>"
%>
<% loop %>
```

↗ Number listSize(fieldname)

Returns the size of a list created using the provided fieldname. listSize will return 0 if fieldname does not exist. This function is great for checking the size of list without actually needing to create it.

```
<% if content.listSize("counter") > 0 then %>
   <% set list = content.createList("counter") %>
   <!--Process List-->
<% end if %>
```

↗ addToList(fieldname, value)

Appends the given value to the end of the list denoted by fieldname. If the list is undefined within the content, a new list will be created.

```
<%
'list as it appears the content properties
'counter:1 "This is the start."
'coutner:2 "This is the middle."
```

```
'counter:3 "This is the end."


'The following %>
<% content.addToList "counter", "This is the new
end of the list." %>


<%
'results in
'counter:4 "This is the new end of the list."
%>
```

↗ array createArray(fieldname)

Creates a VBScript array that contains the fieldname values. The results can be used similar to the List Object but are instead in a traditional VB Array.

```
<%
myArray = content.createArray("subject")
for j = 0 to ubound(myArray)
  response.write myArray(j) & "<br>"
next
%>
```

↗ number intItem(fieldname)

Similar to the "item" function, the "intItem" returns a numerical interpretation of the value of the specified fieldname. For example, if the value is "4" then 4 is returned. If the value is "the5number" then the digit 5 is returned.

```
<%= content.intItem("count") %>
```

↗ Text getMeta(name)

Returns the value of the given meta data key. Meta data tags begin with an underscore. Lists of available meta data can be found on page 6.

```
<%=content.item("_cmsId") %>
```

ASSET OBJECT

The asset object contains functions that allow templates to interact with the CMS' files and folders. The asset object allows a template to pull in information from any other asset or set any information in any other asset.

↗ Text getBinaryLink(path or id)

Much like getLink returns the a link to the provided path or asset id, except the link returned is a binary link and not a CMS preview link. Will return empty quotes if the provided asset does not exist in the CMS.

```
<% path = "/Site/Home Page/Index" %>
<% path_id = asset.getId(path) %>
<% =asset.getBinaryLink(path_id) %>
```

Returns:

/cpt_downview/Site/Home Page/Index

asset.getLink(path) would return:

/[CMS_INSTANCE_NAME]/default.asp?_task=preview&_path_ids=/828/859/&_assetid=87093&_layout=output.asp&_view=

↗ Text getLabel(path or id)

Returns the value of _cmsLabel, for the provided asset id, or asset path. Will return empty quotes if the provided asset does not exist in the CMS.

```
<% my_id = content.item("_cmsid") %>
<% my_label = asset.getLabel(my_label) %>
<% response.write "The asset's label is " &
my_label & "<br />"
```

↗ Folders and Files

Most of the asset functions require an asset to be specified as one of the parameters. All functions allow either an asset ID (number) or a path to be specified. Using an asset ID allows for that asset to move throughout the CMS without other assets losing track of it. A path is similar to folder/file specification of any popular operating system and is in the form /folder/folder/file. The wildcard characters ".", "..", "?" and "*" are respected when appropriate.

Examples of paths:

```
/Archive/Articles/today.html
../Articles/today.html
./today.html
/Archive/Article?/today.html
/Archive/*/*.html
```

SHORTCUTS

Shortcuts within the CMS allow for a more flexible structure that does not need to be strictly hierarchical. When an asset is a shortcut of another asset they share the same data fields but can exist in different folders, use different templates, use different workflows, etc.

Shortcuts are often used to allow content to be categorized into different folders but centralize the data editing of that content. This case allows folders to become category containers that can be used to create indexes within the Web site, and allow content to be edited directly from multiple locations within the CMS.

BRANCHES

Branches within the CMS allow for updates to be made to files that eventually overwrite the branch origin during the workflow process. Since a "Live" file cannot be moved back to draft without being removed from the Web site (a non-desirable situation), a branch allows a copy to be made of the "Live" asset for editing. Once the newly branched document moves through the workflow to the live state, it will force the current "Live" document to be retired (depending on the workflow specification). It will then proceed to become the current "Live" document.

ASSET FUNCTIONS

↗ setParam(name, value)

Sets an extended parameter for use by other functions.

```
<% setParam "sort_order", "asc" %>
```

↗ text getParam(name)

Gets the current value of a previously set parameter.

```
<%= getParam("sort_order") %>
```

↗ delParam(name)

Deletes a parameter.

```
<% delParam "sort_order" %>
```

↗ content_object getContent(id or path)

Returns a content object that contains all the content of the specified asset.

```
<% set fields =
asset.getContent("/Company/MyCompany") %>
<%= fields.item("location") %>
<%= fields.item("number_of_staff") %>
<%= fields.item("address") %>
```

↗ text getContentField(id or path, fieldname)

Similar to **getContent** but just returns the data for a single field. This is a convenience function that should not be used to collect multiple fields as getContent is more efficient for that purpose.

```
<%= asset.getContentField("/Company/MyCompany",
"location") %>
```

↗ content_object getHeader(id or path)

Returns the header data about the specified asset. An assets header contains information about the asset as it exists in the CMS. See above (Asset Header Variables) to see what variables are populated.

```
<% set fields = asset.getHeader(312) %>
The document with id 312 was last modified on <%=
fields.item("_cmsModifiedDate") %>.
```

↗ content_object getFile(id or path)

Returns the contents of a single file within the specified folder. This is useful if you need to access a single file in a folder that contains many files based on the file status.

filter_status: "Draft" or "Pending" or etc. – specifies what status to filter on when returning a file. If more than one file exists with the specified status the most recent file is returned.

```
<% asset.setParam "filter_status", "Live" %>
<% set fields = asset.getFile("/Images/Cover/") %>
Current image: <img src="<%= fields.item("img_url")
%>">
```

↗ List_object getList(id or path)

Returns a list object that contains the contents (files and folders) of a specified folder. Useful for creating indexes of folder contents.

- sort_order: "fieldname ASC" or " fieldname DESC" – specifies how the contents should be sorted.

- is_recursive: 1 - specifies that all folders within the specified folder are also serarched.
- filter_status: "Draft" or "Pending" or etc. – specifies what status to filter on when returning files/folders.
- limit: only return the specified number of results.

If you want to exlcude certain folders or files, use the setParam "exclude", name & vbCRLF & name2 etc to exclude those files/folders from the list.

To sort the list in a particular order, use setParam "sort_by", "fieldname ASC".

To limit the number of results returned use setParam "limit", 5.

See below for examples:

↗ List_object getFileList(id or path)

Similar to getList but only returns files within the specified folder.

```
<% asset.setParam "sort_by", "_cmsLabel asc" %>
<% asset.setParam "limit", 10 %>
<% asset.setParam "exclude", "index" & vbCRLF &
"_menu" %>
<% set list = asset.getFileList("/Articles/") %>
<% do while list.nextEntry() %>
<%= list.item("_cmsLabel") %><br>
<% loop %>
```

↗ list_object getFolderList(id or path)

Similar to getList but only returns folders within the specified folder.

```
<% asset.setParam "filter_status", "Live" %>
<% set list =
asset.getFolderList(content.item("_cmsId")) %>
<% do while list.nextEntry() %>
<%= list.item("_cmsLabel") %><br>
<% loop %>
```

↗ list_object createListFromFolder(id or path, list_label, list_id)

Similar to getFolderList, but replaces fieldnames _cmsLabel, and _cmsid with list_label and list_id, respectively.

```
<% asset.setParam "filter_status", "Live" %>
```

```
<% set
list=asset.createListFromFolder(content.item("_cmsF
olderId"),"new_label","new_id") %>
<% do while list.nextEntry() %>
<%= list.item("new_label") %><br>
<% loop %>
```

↗ list_object createFileListFromFolder(id or path, list_label, list_id)

Similar to createFolderListFromFolder but it only returns files from within the specified folder.

```
<% asset.setParam "filter_status", "Live" %>
<% set
list=asset.createListFromFolder(content.item("_cmsF
olderId:2"),"new_folder_label","new_id") %>
<% do while list.nextEntry() %>
<%= list.item("new_folder_label") %><br>
<% loop %>
```

↗ list_object createFolderListFromFolder(id or path, list_label, list_id)

Creates a list from both the list_id and the specified folder. This function is normally used in conjunction with providing a user a list of files and allowing them to order that list manually. The order is then stored as content in an asset and used to reorder the files that are retrieved from the CMS. If a file is deleted from the CMS that file is removed from this list; likewise, if a new file is created it is added to the list returned from **createListFromFolder**.

- CreateFileListFromFolder only returns files within the specified folder. CreateFolderListFromFolder only returns folders and ignores any files.

Arguments:

Folder – the folder to create the list from

- List_label – the name of the variable that contains the label that represents the name of the file/folder in the specified folder
- List_id – the name of the variable that contains the ID of each file/folder

```
<% set list =
asset.createListFromFolder("/Indexes/", "label",
"id") %>
<% list.setParam "panel_style", "mini_move" %>
<% do while list.nextPanel() %>
```

```
<input type="hidden" name="id" value="<%=
list.item("id") %> ">

<input size=40 type="text" name="label" value="<%=
list.item("label") %>" onClick="blur()"
onSelect="blur()">

<% loop %>
```

↗ setContent(id or path, content_object)

Sets the data fields of the specified asset to the values in content. The values in content will be added to the specified asset. If the field already exists in the asset, it will be overwritten. If the field does not exist, it will be inserted. If a field exists in the asset but not in content, there is no change to that field.

```
<% set fields =
asset.getContent("/Company/MyCompany") %>

<% fields.add "another_location", "Chicago" %>

<% fields.add "another_address", "1234 Pane Street"
%>

<% asset.setContent "/Company/MyCompany", fields %>
```

↗ setContentField(id or path, fieldname, fieldvalue)

Similar to setContent but only sets one field of the specified asset. This is a convenience function that should only be used to set single fields. Use setContent to modifiy multiple fields.

```
<% asset.setContentField "/Company/MyCompany",
"location", "Chicago" %>
```

↗ deleteContentField(id or path, fieldname)

Removes the specified fieldname from the specified asset. This is a convenience function that should only be used to remove single fields. Use setContent to delete multiple fields.

```
<% asset.deleteContentField "/Company/MyCompany",
"location" %>
```

↗ setHeader(id or path, content_object)

Allows setting of the header information for a particular asset.

Warning! Directly setting header information can corrupt the functioning of the CMS and may result in data loss. Use this only if you understand the implications of what you are doing.

The following variables are updated in an asset depending on the values within content:

- _cmsSize
- _cmsType
- _cmsLabel
- _cmsWorkflowId
- _cmsTemplateId
- _cmsStatus
- _cmsStatusDate
- _cmsStatusUserId
- _cmsFolderId
- _cmsModifiedUserId
- _cmsPublishDate
- _cmsPublishUserId
- _cmsCreateDate
- _cmsCreateUserId
- _cmsCheckoutDate
- _cmsCheckoutUserId

See the Content Object for the field descriptions.

```
<% set header =
asset.getHeader("/Company/MyCompany") %>
<% header.add "_cmsTemplateId", 12 %>
<% asset.setHeader("/Company/MyCompany", header) %>
```

↗ boolean setLabel(id or path, text)

Sets the specified asset's label to the supplied text. This can also be done in setHeader but specifically sets the asset's label only.

```
<% asset.setLabel content.item("link_to"), "New
label" %>
```

↗ text getLink(id or path)

Returns a link to the specified asset. When clicked in preview mode this link will cause the linked to asset to show its output/preview. When the asset is published out to the destination Web site the link is transformed according

to where the linked to asset is published. By linking to an asset based on its ID, the link will not be broken if that asset is moved. If a folder is linked to the CMS, it will chose an asset to link to based on the status of the assets within that folder.

This routine can also be used to create links to perform CMS-related tasks. Often preview pages are used to create specialized interfaces that are not published to any Web site. Within these interfaces links to create, edit, approve, and reject assets can be set up using getLink and several parameter specifications.

- link_type
    1. create, model_id – creates a new document
    2. edit – edits a document
    3. workflow – executes a workflow command
    4. delete – deletes the document
    5. clone – clones the document
    6. branch – branches the document
    7. folder – shows the folder listing
    8. include – includes a document (e.g., JS file)
    9. binary - links to the binary field of a template fiew (similar to /cpt_downview)

```
Regular Link
<a href="<%=
asset.getLink(content.item("product_id"))
%>">Product</a>


Create Document Link
<% asset.setParam "link_type", "create" %>
<% asset.setParam "model_id", "567" %>
<a href="<%= asset.getLink("/products/")) %>">New
Product</a>


Approve Link
<% asset.setParam "link_type", "workflow" %>
<% asset.setParam "link_command", "approve to
stage" %>
<a href="<%= asset.getLink(content.item("_cmsId"))
%>">Approve</a>
```

```
Reject Link
<% asset.setParam "link_type", "workflow" %>
<% asset.setParam "link_command", "reject" %>
<a href="<%= asset.getLink(content.item("_cmsId"))
%>">Reject</a>

Include Link
<% asset.setParam "link_type", "include" %>
<script language="JavaScript"
src="<%=asset.getLink("802")%>"></script>
```

↗ text getBinaryLink(id or path)

Returns a link to the specified binary asset. When linking to a template that contains a field that is to be used as the “image” of the template, you can use getLink with a link_type of "binary" or use getBinaryLink instead. This is only needed when you want to link to an image/pdf/doc etc within a templated asset, as opposed to the resulting page.

```
Regular Link
<a href="<%=
asset.getLink(content.item("product_id"))
%>">Product</a>

Binary Link
<img src="<%= asset.getBinaryLink("/img/photo.jpg")
%>">
```

↗ text getFolder(id or path)

Returns the folder path of the specified asset.

```
<a href="<%=
asset.getFolder(content.item("product_id"))
%>">Product Folder</a>
```

↗ text getAbsoluteName(id or path)

Returns the /folder/label of the specified asset.

```
The asset is <%=
asset.getAbsoluteName(content.item("link_id")) %>.
```

↗ List_object getComments(id or path)

Returns a list containing the comments associated with the specified asset.

The following variables are available in the list object:

- _cmsComment – the comment text
- _cmsCommentDate – the date the comment was added
- _cmsCommentName – the full name of the user who added the comment
- _cmsCommentUserId – the userid of the user who added the comment
- _cmsCommentUserAvatar – the avatar of the user who added the comment
- _cmsCommentUsername – the username of the user who added the comment
- _cmsCommentId – the current comments ID

```
Comments:<br><br>
<% set list =
asset.getComments(content.item("_cmsId")) %>
<% do while list.nextEntry() %>
  <b><%= list.item("_cmsCommentName") %></b><br>
  <%= list.item("_cmsComment") %>
<% loop %>
```

↗ show(id or path)

Similar to “include” but instead evaluates another asset preview/output within its own context as opposed to the current asset context. This is often used to combine different templates to create a composite template.

```
<!--basic homepage template-->
<table>
  <tr>
    <td colspan=2><%=
asset.show(content.item("banner_id")) %></td>
  </tr>
  <tr>
    <td><%= asset.show(content.item("nav_id"))
%></td>
    <td><%= asset.show(content.item("feature_id"))
%></td>
  </tr>
  <tr>
    <td colspan=2><%= asset.show("/Footer.html")
%></td>
```

```
  </tr>
</table>
```

↗ create(label, in_folder, use_model, content_fields)

Creates a new file asset in the CMS.

- Label - the label of the new asset
- In_folder – the location of the new asset.
- This field is passed ByRef. If a string path is passed, the cms will automatically convert it to the corresponding cmsid, which means the value of in_folder may change after the function has been called. If you need to preserve the string path, it is highly recommended that it is passed in via a temporary variable.
- Use_model – specifies the file to use as the model for the new file. Using the a model file allows the new file to automatically have all the acls, templateId, workflowId, publish/deploy, etc configuration information set.
- Content_fields – the data fields of the new asset.

```
<-- Create an archived version of the current asset
-->
<% asset.create "archive-" &
content.item("_cmsLabel"), "/Archive/",
"/System/Models/Archive", content %>
```

↗ addDependencyTo(path)

Creates a dependency between the current asset and the asset located at the provided path.

```
<% path = "/site/Home Page/Index" %>
<% asset.addDependencyTo(path) %>
```

↗ publish(id or path)

Forces a manual publish of the specified asset to the current Web site. The routine will delay until the pubishing is complete. This delay will depend on network connection and can vary.

```
<% asset.publish content.item("link_to") %>
```

↗ publishLater(id or path)

Forces a manual publish of the specified asset to the current Web site. The routine will return immediately and publish the document within the next 5 minutes.

```
<% asset.publishLater content.item("link_to") %>
```

↗ delete (id or path)

Deletes the specified asset.

```
<% asset.delete content.item("link_to") %>
```

↗ move(id or path, newFolder)

Moves the specified asset to a new folder location.

```
<% asset.move content.item("link_to"), "/Archive/"
%>
```

↗ rename(id or path, newLabel)

Renames the specified asset.

```
<% asset.rename content.item("link_to"), "New
Label" %>
```

↗ id branch (id or path)

Creates a branch of the specified asset. The new branched asset has a copy of all the fields of the supplied asset but the initial workflow state is set to Draft. All other configuration information remains the same.

```
<% newId = asset.branch(content.item("_cmsId")) %>
or updating the branch in email_import.asp when an
update email arrives for an asset.
<%
  ' we want to check if a branch in draft already
exists ... if so update that one
  asset.setParam "filter_status", "Draft"
  newId = asset.getId(content.item("_cmsId"))
  ' newId = 0 when no branch exists in draft state
  if newId = "0" then
    ' so create a new branch
    newId = asset.branch(content.item("_cmsId"))
  end if
  ' and finally update its page_text field as
needed.
  asset.setContentField newId, "page_text", "this
is a branched doc " & Now()
```

```
%>
```

↗ id copy (id or path)

Creates a copy of the specified asset. The new asset has a copy of all the fields of the supplied asset but the initial workflow state is set to Draft. All other configuration information remains the same.

```
' create a copy of an asset in the same folder

<% newId = asset.copy(content.item("_cmsId"),
content.item("_cmsFolderId")) %>

<% asset.setLabel newId, "Copy of new asset" %>
```

↗ boolean exists(id or path)

Checks if the specified file exists or not.

```
<% if asset.exists(content.item("link_to")) then %>

<% asset.rename content.item("link_to"), "New
Label" %>

<% end if %>
```

↗ createShortcutIn(id or path, folder)

Creates a shortcut to the specified asset in the specified folder. See above for a description about shortcuts.

```
<% asset.createShortcutIn content.item("link_to"),
"/Archive/" %>
```

↗ boolean isShortcutIn(id or path, folder)

Searches in specified folder to see if a shortcut to the specified asset exists.

```
<% if not
asset.isShortcutIn(content.item("link_to"),
"/Archive/") then %>

<% asset.createShortcutIn content.item("link_to"),
"/Archive/" %>

<% end if %>
```

↗ text getUploadAsText(id or path)

Returns the specified asset as text. This is often used to gain access to the contents of a file that was uploaded by a user and needs to be modified or processed in some way.

```
<!--Displaying an upload csv file within an HTML
table -->

<table>
```

```
<%
  txt =
asset.getUploadAsText(content.item("csv_file"))
  rows = split(txt, vbCRLF)
  if isArray(rows) then
    for i = 0 to ubound(rows)
     response.write "<tr>"
      cols = split(rows(i), ",")
      if isArray(cols) then
        for j = 0 to ubound(cols)
          response.write "<td>" & cols(j) & "</td>"
        next
      end if
      response.write "</tr>"
    next
  end if
%>
</table>
```

↗ filter_object createFilter()

Creates a filter object that can then be used to return a list of content (using the getFilterList function) that matches the criteria assigned to the filter. The filter criteria can vary greatly, and usually is a mix of comparisons on system and template variables.

The following fields are supported for the filter (in addition to any content fields):

- _cmsId
- _cmsCommentsCount
- _cmsFolderId
- _cmsWorkflowId
- _cmsCreateDate
- _cmsCreatorUserId
- _cmsCreatorUser
- _cmsModifiedDate
- _cmsModifiedUserId
- _cmsModifiedUser

- _cmsChangeDate
- _cmsChangeUserId
- _cmsChangeUser
- _cmsCheckoutDate
- _cmsCheckoutUserId
- _cmsCheckoutUser
- _cmsFolder
- _cmsLabel
- _cmsTemplateName
- _cmsTemplateId
- _cmsBaseModelId
- _cmsShortcutId
- _cmsPublishUserId
- _cmsPublishDate
- _cmsStatusIds
- _cmsStatus
- _cmsType

The type comparisons for string, int, number(float), and date are:

- string_contains
- string_not_contains
- string_within
- string_not_within
- string_starts_with
- string_ends_with
- string_greater_than
- string_greater_equal_than
- string_less_than
- string_less_equal_than
- string_equals
- string_not_equals

- string_not_null
- string_is_null

- int_greater_than
- int_greater_equal_than
- int_less_than
- int_less_equal_than
- int_equals
- int_not_equals
- int_not_null
- int_is_null
- int_greater_than
- int_greater_equal_than

- number_less_than
- number_less_equal_than
- number_equals
- number_not_equals
- number_not_null
- number_is_null

- date_greater_than
- date_greater_equal_than
- date_less_than
- date_less_equal_than
- date_equals
- date_not_equals
- date_not_null
- date_is_null
- date_range

```
<% set ffilter = asset.createFilter() %>

<% ffilter.add "_cmsTemplateName", "string_equals",
"Document" %>

<% ffilter.add "_cmsBaseModelId", "int_not_equals",
0 %>

<% ' Define scope of assets for filter to search:
in this case, entire CMS. %>

<% ffilter.add "_cmsBaseFolder", "",
content.item("_cmsRootFolderId") %>

<% ffilter.add "cat_type_select", "string_equals",
content.item("cat_type_select") %>

<% ffilter.add "doc_type_select", "int_equals",
content.item("doc_type_select") %>
```

↗ list_object getFilterList(filter_object)

Returns a list object of results based on the execution of the filter against the content specified in the filter's configuration.

```
<% set auto_list = asset.getFilterList(ffilter) %>
```

↗ number getFolderId(path or id)

Returns the folder ID of the specified asset.

```
<%= asset.getFolderId(content.item("_cmsID")) %>
```

↗ text getTopFolder(path or id)

Returns the name of the top folder for the specified asset (/rootFolder/folder:2/folder:3/topFolder/).

```
<%= asset.getTopFolder("1234") %>
```

↗ text getTopFolderId(path or id)

Returns the ID of the top folder for the specified asset (/rootFolder/folder:2/folder:3/topFolder/).

```
<%= asset.getTopFolderId(content.item("_cmsID")) %>
```

↗ text getRootFolder(path or id)

Returns the name of the root folder for the specified asset (/rootFolder/folder:2/folder:3/topFolder/).

```
<%= asset.getRootFolder(content.item("_cmsID")) %>
```

↗ text getRootFolderId(path or id)

Returns the ID of the root folder for the specified asset (/rootFolder/folder:2/folder:3/topFolder/).

```
<%= asset.getRootFolderId(content.item("_cmsID"))
%>
```

↗ number getIdInFolder(path or id, label)

Returns the CMS ID of the specified asset.

```
<%=
asset.getIdInFolder(content.item("_cmsFolderID"),
"_Menu") %>
```

↗ numebr getId(path or id)

Returns the CMS ID of the specified asset.

```
<%= asset.getId("/Site/Homepage/Homepage") %>
```

↗ Text getIdPathInFolder(path, label)

Works much like getIdInFolder, except it returns the full id path to the asset. If the path exists, but the label does not, the id path to the folder will be returned. Returns 0 if the path does not exist. In some cases getIdPathInFolder can return 0/.

```
<%= asset.getIdPathInFolder("/Site/Home
Page/","Index") %>
```

Possible Result:

/828/859/87093

↗ Number getTemplateId(path or id)

Returns the value of _cmsTemplateId for the given asset. Will return 0 if the asset provided does not exist within the CMS.

```
<% path = "/Site/Press Release/" %>
<% pr_template_id = 421337
<% set list = asset.getFileList(path) %>
<% do while list.nextEntry() %>
   <% 'only display assets using the press release
template id %>
   <% if asset.getTemplateId(list.item("_cmsid") =
pr_template_id then %>
      display press release
   <% end if %>
<% loop %>
```

↗ text getStatusText(path or id)

Returns the workflow step that the asset is currently in. This will typically be LIVE, STAGE, etc. Will return 0 if the asset provided does not exist within the CMS.

```
<% path = "/Site/Home Page/Index" %>
<% =asset.getStatusText(path) %>
```

Possible Result:

DRAFT

↗ text getStatusColor(path or id)

Returns the color of the current workflow step in hexadecimal. Returns 0 if the asset provided does not exist within the CMS.

<% path = "/Site/Home Page/Index" %>

<% =asset.getStatusColor(path) %>

Possible Result:

8DBEDA

↗ replaceContentFieldInAll(fieldname, text_to_find, text_to_replace_with)

Replaces all fieldnames that have a specific value to another value. This allows field values to be converted to other values. This function operates at a global level and changes ALL occurences of the fieldname with the specified value.

```
<% asset.replaceContentFieldInAll("Subject", "My
Subject", "My New Subject" %>
```

↗ setSchedule(path or id, schedule_name, schedule_date)

Allows you to set a scheduled workflow transition (usually a publish or retire) for the specified asset. You must provide the schedule name as configured in the workflow. The schedule date must include a date and time.

```
<% Call asset.setSchedule(content.item("_cmsID"),
"Live Date", content.item("feature_date") &
"00:11:00") %>
```

↗ clearSchedule(path or id, schedule_name)

Clears the schedule date set for the specified asset. The schedule name must match with a schedule name in the workflow for the asset.

```
<% asset.clearSchedule(content.item("_cmsID"),
"Retire Date") %>
```

↗ route(path or id, step subject or step number or status name or status id)

Provides you with the ability to transition the specified asset into another step in the asset's current workflow.

```
<% ' Route example using the step subject %>

<% if asset.route(content.item("_cmsID"), "Pending
State") = "" then %>

            <% content.add "_cmsError", "Unknown
Route State" %>

<% end if %>

<% ' Route example using status ID %>

<% if asset.route(content.item("_cmsID"), 781) = ""
then %>

            <% content.add "_cmsError", "Unknown
Route State" %>

<% end if %>

<% ' Route example using status name %>

<% if asset.route(content.item("_cmsID"),
"PENDING") = "" then %>

            <% content.add "_cmsError", "Unknown
Route State" %>

<% end if %>

<% ' Route example using step number %>

<% if asset.route(content.item("_cmsID"), 2) = ""
then %>

            <% content.add "_cmsError", "Unknown
Route State" %>

<% end if %>
```

↗ text getUploadAsHex(filename)

Encodes the uploaded asset as a hex string.

```
<%=
asset.getUploadAsHex(content.item("feature_image"))
%>
```

↗ saveUploadFromHex(ByRef extension, ByRef txt)

Saves the hex string as an attached asset.

```
<%= asset.saveUploadFromHex("dat",
"ff0034b5d2a40567") %>
```

↗ text getUploadSize(filename)

Returns the size of the referenced asset. The size returned is relative to the formatted size of the asset. Thus, a 147K image will return as 147K, a 1.8MB file will show as 1.8MB, etc.

```
<%=
asset.getUploadSize(content.item("feature_image"))
%>
```

↗ access_object getAccessFields(path or id)

Returns the access fields for the specified asset. The access object contains two functions to set and get acls for the asset. See access object for further details.

↗ setAccessFields(path or id, access_object)

Saves the access object back onto the asset's acl structure. To modify an assets ACL list first get the access object, make modifications to the object and then set the access back to the asset.

```
<% set acl =
asset.getAccessFields(content.item("_cmsId")) %>
<% acl.setAccess "Authors", "view", false
<% asset.setAccessFields(content.item("_cmsId"),
acl) %>
```

↗ boolean hasFiles(path or id)

This function checks to see if any files could be linked to in the specified folder. This is useful to prevent printing out header information if no files would be linked to.

```
<% if asset.hasFiles("/Assets/") then %>
There are files to link to!
<% end if %>
```

↗ list_object getContentFieldFunction(path or id, fieldname, function, fieldType)

Provides a way to gather field values in a functional way. The function can be one of of min, max, average, sum, or count. The fieldType can be either a "date" or a "number."

```
<% minDate =
asset.getContentFieldFunction(content.item("sysvar_
events_folder"), "start_date", "min", "date") %>
```

↗ getDateOverlap(path or id, startDate, endDate, startFieldname, endFieldname)

Returns back a list of assets that contain the specified startFieldname and endFieldname whose date range includes, starts or stops within the range specified by startDate and endDate. This function is primarily used for calendaring applications that need to determine if an event range falls on a specific day, week or month.

By default, getDateOverlap only looks in the folder specified for date ranges. To seek below the specified folder, use setParam "recursive", 1

```
Todays events:
<%
  set list =
asset.getdateoverlap(content.item("sysvar_events_fo
lder"), Date(), DateAdd("d", 1, Date()),
"start_date", "end_date")
  do while list.nextEntry()
    response.write list.item("event_name") & "<br>"
  loop
%>
```

↗ dictionary getMonthOccurrence(path or id, dateFieldname, startField, ByRef endField)

Used to determine if there are events whose date range falls on a specific day of the month. The month is specified by dateFieldname. startField and endField specify the fieldnames where the corresponding start and end date for an event are held. Using this routine, a calendar month can quickly determine if events within a given month are present for a particular day within that month.

```
<table width="142" cellpadding="2" cellspacing="0"
border="0" align="center" class="calendar"
bgcolor="#FFFFFF">
  <tr>
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">S</td>
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">M</td>
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">T</td>
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">W</td>
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">T</td>
```

```
    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">F</td>

    <td align="right" class="sidebar" width="20"
height="12" valign="bottom">S</td>

  </tr>

  <tr>

  <% curDate = Date()

     curMonth = Month(currentDate)

     'skip days from the previous month

     for i = 1 to Weekday(curDate) - 1 %>

     <td class="sidebar" align="right" width="20"
height="12" valign="bottom"> </td>

     <% next

     set days =
asset.getmonthoccurrence(content.item("sysvar_event
s_folder"), currentDate, "start_date",
"end_date_internal")

     do while Month(curDate) = curMonth

     ' if it's a Sunday, but not if day 1 is a
Sunday, start a new row

     if Weekday(curDate) = 1 and Day(curDate) <> 1
then %>

       response.write "</tr><tr>"

     end if


     'show the day with a link if any events occur
on that date

     if days.exists(Day(curDate)) then

       asset.setParam "args", "_the_current_date="
& curDate

       strLink =
asset.getLink(content.item("sysvar_calendar_folder"
)&"Daily")

       calDay =Day(curDate)%>

       <td class="sidebar"
id="<%="d"&Day(curDate)%>"
name="<%="d"&Day(curDate)%>" align="right"
width="20" height="12" valign="bottom"><a href="<%=
strLink %>" title="Click for more info"
class="sidebar"><%=Day(curDate)%></a></td>

  <% else %>

       <td class="sidebar"
id="<%="d"&Day(curDate)%>"
name="<%="d"&Day(curDate)%>" align="right"
```

```
width="20" height="12"
valign="bottom"><%=Day(curDate)%></td>
  <% end if
     curDate = DateAdd("d", "1", curDate)
   loop

   'skip days to finish the month
   if Weekday(curDate) <> 1 then
   for i = 1 to 8 - Weekday(curDate) %>
      <td class="sidebar" align="right" width="20"
height="12" valign="bottom"> </td>
 <%next
   end if %>
  </tr>
</table>
```

↗ getNearestMatch(path or id, list, listFieldname, confidence)

Given a list of asset IDs (list) the getNearestMatch routine will return the ID within that list that is nearest to the first parameter (path or ID). This routine can be used to determine which menu item in an expanding menu should be "on" by supplying the routine with the current linked to pages from the navigation and which asset ID you are currently in. The confidence return value is a number that specifies how many folders are in common between the supplied ID list and the ID that you are currently in.

```
<%
  ' we are in /Site/Company/Bios/index.asp
  ' create a list of ids (this is usually created
by a panel interface
  set list = system.createList()
  ' add /Site/Press/index.asp
  list.addEntryByString("link" & vbCRLF &
"/cpt_internal/246")
  ' add /Site/Infrastructure/index.asp
  list.addEntryByString("link" & vbCRLF &
"/cpt_internal/1223")
  ' add /Site/Company/Contact/download.pdf
  list.addEntryByString("link" & vbCRLF &
"/cpt_internal/8883")
  nearestId =
asset.getNearestMatch(content.item("_cmsId"), list,
"link", conf)
```

```
  ' nearestId would be equal to 8883 since we're in
the Contract folder.
  ' Conf = 2 since /Site/Company match the current
doc and 8883
%>
```

↗ text getLastComment(path or id)

Returns the last comment made for the specified asset.

```
<%= asset.getLastComment(content.item("_cmsID")) %>
```

↗ addComment(pathName, comment)

Allows you to add a comment to the specified asset.

```
<% asset.addComment(content.item("_cmsID"), "This
is my new comment" %>
```

number getPercentChanged(pathName, startDate, endDate, mode)

Returns a percentage of change (defined by the mode) for the specified asset between the selected date range. The mode can be "maximum", "minimum", "average", or "total."

```
<%= asset.getPercentChanged(content.item("_cmsId"),
Date-47, Date, "maximum" %>
```

↗ array getUniqueFields(pathName, fieldname)

Returns an array of values correlating to all unique instances of the specified fieldname. The search begins from the specified path. An array is returned (as opposed to a list or content field) for performance purposes: This routine is more ideal for quickly searching a large set of data.

```
<% asset.setParam "is_recursive", 1 %>
<% myarray = asset.getUniqueFields("/", "title") %>
<%= join(myarray, "|") %>
```

↗ array getFileUniqueFields(ByVal pathName, ByRef fieldname)

Similar to getUniqueFields, but only searches through file assets.

```
<% myarray = asset.getFileUniqueFields("/",
"title") %>
```

↗ array getFolderUniqueFields(ByVal pathName, ByRef fieldname)

Similar to getUniqueFields, but only searches through folder assets.

```
<% myarray = asset.getFileUniqueFields("/",
"title") %>
```

LIST OBJECT

The list object is primarily used with the content and asset objects to provide looping mechanisms over files and data. The list object is similar to the iterator or enumerator classes in other languages.

↗ setParam(param_name, param_value)

Sets an extended parameter for use by other functions.

```
<% list.setParam "sort_order", "asc" %>
```

↗ Param(param_name)

Gets the current value of a previously set parameter

```
<%= getParam("sort_order") %>
```

↗ delParam(param_name)

Deletes the given parameter.

```
<% delParam "sort_order" %>
```

↗ load(array)

Loads the list iterator with the specified array. The array is assumed to be an array of Dictionaries (Hashtables) that have the appropriate fields specified in that dictionary.

```
<%
  Dim array(2)
  Dim fields1, fields2
  Dim list

  set fields1 = system.createDictionary()
  fields1.add "title", "The new book"
  set fields2 = system.createDictionary()
  fields2.add "title", "The other new book"
  set array(0) = fields1
  set array(1) = fields2
  set list = system.createList()
  list.load array
```

```
%>
```

↗ loadArray(id)

Returns the color of the current workflow step in hexadecimal. Returns 0 if the asset provided does not exist within the CMS.

```
<% path = "/Site/Home Page/Index" %>
<% =asset.getStatusColor(path) %>

Possible Result:
8DBEDA
```

↗ boolean nextEntry()

Moves to the next (or first) entry in the list. After nextEntry returns the data, context of the object list is incremented to the next entry. The "item" routine will then return values based on the current entry index. Once the nextEntry routine gets to the end of the list, it returns a Boolean false to terminate the loop. Once this happens the list object then resets itself to the start of the list and can be looped across again.

```
<%
  set list = content.createList("title")
  do while list.nextEntry()
    response.write list.item("title") & "<br>"
  loop
%>
```

↗ boolean nextPanel()

Similar to nextEntry but will create the appropriate editing controls within the input.asp file. Within output.asp nextPanel works like nextEntry but will default to one panel entry.

↗ getEntryAt(i)

Returns the array from the specified index in the list.

```
<% do while list.nextEntry() %>
<% = list.getEntryAt(list.itemIndex()) %>
<% loop %>
```

↗ insertEntryAt(i, data)

Inserts the given array, data, at the specified index.

```
<% set fields = system.createDictionary() %>
<% fields.add "subject", "my new subject" %>
<% fields.add "message", "the message of the body"
%>
<% list.insertEntryAt 2,fields %>
```

↗ setPanelColors(row_color1, row_color2)

Sets the colors used when displaying the panel control buttons (delete, insert, move, etc).

- row_color1 – the color surrounding the insert row (default "D9D9D9")
- row_color2 – the color inbetween the insert row where the delete and move buttons are. (default "F0F0F0")

```
<% list.setPanelColors "F0F0F0", "FFFFFF" %>
```

↗ reset()

If a nextEntry or nextPanel loop is prematurely terminated (by an "exit do") the list object's context will still be on the entry that was set on the last nextEntry call. To reset the list to the first entry use the "reset" function.

```
<%
  set list = content.createList("title")
  do while list.nextEntry()
    if list.item("title") > "book" then
      exit do
    end if
    response.write list.item("title") & "<br>"
  loop

  list.reset()

  do while list.nextEntry()
    response.write list.item("title") & "<br>"
  loop
%>
```

↗ text item(field_name)

Accesses the information within a list object depending on its current context. (Similar to the content.item routine.)

```
<%= list.item("title") %>
```

↗ boolean exists(field_name)

Determines if the field specified exists within the list object. The field exists if the key (field_name) has been registered in the object. Even if the value of the field might be NULL or "" this routine can still return true.

```
<% if list.exists("title") then %>
The title is <%= list.item("title") %>
<% end if %>
```

↗ sort(sort_string)

Sorts the list given the specified criteria.

Sort string is the desired field to sort by. An optional sort direction ("ASC" or "DESC") can also be specified after the field name separated by a space.

```
<% list.sort "title DESC" %>
```

↗ list_object filter(name, operation, value, case_type)

Filters the list by the specified criteria. Elements that are excluded by the filter criteria are removed from the list.

- name: the name of the field to check
- operation: the type of operation to perform
- value: the value of the field based on mode
- case_type: specifies if letter type (uppercase or lowercase) should be ignored

Mode is one of the following:

- "CONTAINS"
- "NOT_CONTAINS"
- "WITHIN"
- "NOT_WITHIN"
- "STARTS_WITH"
- "ENDS_WITH"
- "GREATER_THAN"
- "LESS_THAN"
- "GREATER_EQUAL_THAN"
- "LESS_EQUAL_THAN"

- "EQUALS"
- "NOT_EQUALS"
- "NOT_IS_NULL"
- "IS_NULL"
- "EXISTS"
- "NOT_EXISTS"

case_type is "1" – case-insensitive, "0" – case-sensitive

```
<% list.filter "title", "contains", "book", "1" %>
```

Or,

```
<% list.filter "book", "within", "title", "1" %>
```

Or,

```
<% list.filter "title", "ends_with", "book", "1" %>
```

Or,

```
<% list.filter "title", "starts_with", "The", "1"
%>
```

↗ paginate(page_num_name, page_num_value, page_size)

Paginates the current list into page-size increments. This routine allows a page to create multiple pages based on a list. Often used in creating lists that require multi-page layouts with previous, next, page hyperlinks, etc.

- page_num_name – the name of the variable that should contain the current page number.
- page_num_value – the value of page_num_name, i.e., the current page we're processing.
- page_size – how many page links should be in a page set.

It is assumed that the list is a list of documents and has _cmsId, _cmsType and _cmsTemplateId populated in each dictionary of lists. This is true for any of the asset.getList routines.

The following variables are defined after calling this routine:

- list.page_start – the start index for the current page
- list.page_end – the end index of the current page
- list.page_links – the links to each of the page sets that will be created

- list.page_size – the current page size on this page
- list.page_num – the current page number
- list.page_next – the link to the next page set. "" if no next page exists
- list.page_previous – the link to the previous page set. "" if no previous page exists.

In addition, each list element is also populated with its corresponding link as "_cmsLink" that can be used to directly link to that page.

```
<%
  ' print out an index list of pages with previous,
next and page number links.


  ' break the list into bunches of 10
  list.paginate "_pagenum",
content.item("_pagenum"), 10


  ' print out the index list of pages (hyperlinked
around title)
  for i = list.page_start to list.page_end
     response.write i+1 & " "
     response.write "<a href=""" &
list.ItemAt(i).item("_cmsLink")  & """>"
     response.write list.ItemAt(i).item("title")
     response.write "</a><br>"
  next


  ' check if multiple page sets exist
  if ubound(list.page_links) > 0 then


    ' if a previous page exists allow the user to
go page backwards
    if list.previous_page <> "" then
         response.write "<a href=""" &
list.previous_page & """>&lt; Previous</a>"
    end if


    ' print out the page index which allows the
user to click on the
    ' page number to view that page.
    for i = 0 to ubound(list.page_links)
```

```
            ' the link of the current page that
we're on is "" so no need to link
      ' to that one.
        if list.page_links(i) <> "" then
          ' write out the pages number hyperlinked
to that page
          response.write "<a href=""" &
list.page_links(i) & """>"
  response.write i+1
  response.write "</b></a>"
       else
         ' this is for the current page we're on …
just leave that unlinked
         response.write "<b>" & i+1  & "</b>"
       end if
    next


    ' if a next page exists allow the user to go
page forwards
    if list.next_page <> "" then
      response.write "<a href=""" & list.next_page
& """>Next &gt;</a>"
    end if


  end if
%>
```

↗ Number pageCount(name)

Returns the total number of pages. Mainly used in conjunction with paginate in order to keep track of the number of pages.

```
<% set list = list.item("counter") %>
<% list.paginate "_pageNum",
content.item("_pagenum"), 5 %>
<%= list.pageCount("_pageNum)%>
```

↗ crop first_index, last_index

Crops the current list to the specified start and end index. This can be used to truncate or limit the size of the list.

```
<%
  ' create a list based on the variable title
```

```
  set list = content.createList("title")
  ' limit the list to 5 entries
  list.crop 0, 4
  ' print out the contents of the list
  do while list.nextEntry()
    response.write list.item("title") & "<br>"
  loop
%>
```

↗ text join(name, delimiter)

Joins all the values contained in the list of the specified name. Each value is delimited by the specified delimiter.

```
<%= list.join("title", ",") %>
```

↗ boolean contains(fieldname, fieldvalue)

Returns true or false if the specified name and value exists in the list object.

```
<% if list.contains("title", "The new book.") then
%>
The book exists.
<% end if %>
```

↗ size()

Returns size of list.

```
<% set list = content.createList("subject") %>
<% if list.size() = 0 then %>
There are no subjects to report.
<% end if %>
```

↗ itemIndex

Returns index of current item in the list.

```
<% set list = content.createList("subject") %>
<% do while list.nextEntry() %>
Subject #<%= list.itemIndex %>: <%=
list.item("subject") %>
<% loop %>
```

↗ setItemIndex(number_index)

Sets the current item index of the list object.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex(5) %>
The fifth subject (if there is one) is <%=
list.item("subject") %>
```

↗ indexOfItem(fieldname, fieldvalue)

Seeks the specified fieldname and corresponding value in the list object.

```
<% set list = content.createList("subject") %>
The entry index of "my email subject" is <%=
list.indexOfItem("subject", "my email subject") %>
```

↗ itemAt(number_index, fieldname)

Similar to "item" itemAt returns the fieldvalue at the specified index number.

```
<% set list = content.createList("subject") %>
The fifth subject is <%= list.itemAt(5, "subject")
%>
```

↗ dictionary_object entryAt(number_index)

Similar to "itemAt," entryAt returns the entry (dictionary) at the specified index number.

```
<% set list = content.createList("subject") %>
<% set fields = list.entryAt(5) %>
The fifth subject is <%= fields.item("subject") %>
```

↗ dictionary_object currentEntry()

Returns the current entry (aka the dictionary object) at the current index.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex 5 %>
<% set fields = list.currentEntry() %>
The fifth subject is <%= fields.item("subject") %>
```

↗ removeItem(fieldname)

Removes the specified fieldname from the current list index.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex 5 %>
<% ' remove the subject fieldname from the 5th
element %>
<% list.removeItem("subject") %>
```

↗ removeItemAt(index, fieldname)

Removes the specified fieldname from the specified list index.

```
<% set list = content.createList("subject") %>
<% ' remove the subject fieldname from the 5th
element %>
<% list.removeItemAt(5, "subject") %>
```

↗ removeEntry()

Removes the current entry (dictionary) from the specified list.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex 5 %>
<% ' remove the fifth entry from the list %>
<% list.removeEntry() %>
```

↗ removeEntryAt(number_index)

Removes the entry (dictionary) from the specified index.

```
<% set list = content.createList("subject") %>
<% ' remove the fifth entry from the list %>
<% list.removeEntryAt(5) %>
```

↗ setEntryAt(number_index, dictionary_object)

Updates an entry with a new entry (dictionary) given the specified index.

```
<% set list = content.createList("subject") %>
<% set fields = system.createDictionary() %>
<% fields.add "subject", "my new subject"%>
<% fields.add "message", "the message of the
body"%>
<% list.setEntryAt(5, fields) %>
```

↗ addItem(fieldname, fieldvalue)

Adds a fieldname and fieldvalue to the current list entry.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex 5 %>
<% list.addItem "subject_other", "my new subject"%>
<% list.addItem "message_other", "the message of
the body"%>
```

↗ addItemAt(number_index, fieldname, fieldvalue)

Adds a fieldname and fieldvalue to the specified list index.

```
<% set list = content.createList("subject") %>
<% list.addItemAt 5, "subject_other", "my new subject"%>
<% list.addItemAt 5, "message_other", "the message of the body"%>
```

↗ addEntry(dictionary_object)

Adds a fieldname and fieldvalue to the specified list at the end of the list (i.e., the list becomes 1 entry larger).

```
<% set list = content.createList("subject") %>
<% set fields = system.createDictionary() %>
<% fields.add "subject", "my new subject"%>
<% fields.add "message", "the message of the body"%>
<% list.addEntry(fields) %>
```

↗ addEntryByString(fieldname & vbCLRF & fieldValue & ...)

Adds an entry to the specified list using a string to specify the values to be inserted instead of a dictionary (as in addEntry).

```
<% set list = content.createList("subject") %>
<% list.addEntryByString "subject" & vbCRLF & "my new subject" & vbCRLF & "message" & vbCRLF & "the message of the body" %>
```

↗ addList(List_object)

Appends a list onto the end of another list.

```
<% set list = content.createList("subject") %>
<% set list_ext = content.createList("subjects") %>
<% list.addList list_ext %>
The combined list size is <%= list.size() %>
```

↗ rename(fieldname, replace_with_fieldname)

Renames the fieldname to a new fieldname at the current list index.

```
<% set list = content.createList("subject") %>
<% list.setItemIndex 5 %>
<% list.rename "subject", "subject_new" %>
The subject at index 5 is <%= list.item("subject_new") %>
```

↗ renameAll(fieldname, replace_with_fieldname)

Renames all the fieldnames in the current list object with the specified fieldname.

```
<% set list = content.createList("subject") %>
<% list.renameAll "subject", "subject_new" %>
<% do while list.nextEntry() %>
  <%= list.item("subject_new") %><br>
<% loop %>
```

↗ text defaultItem(fieldname, default_value)

Accesses a particular field within the current entry which, if not found (or empty value), will return the default value instead.

```
<%= list.defaultItem("fullname", "My name here") %>
```

↗ text defaultItemAt(number_index, fieldname, default_value)

Accesses a particular field within the specified index which, if not found (or empty value), will return the default value instead.

```
<%= list.defaultItemAt(5, "fullname", "My name
here") %>
```

↗ text escapeItem(fieldname)

Similar to “item,” but escapes all HTML tags. This function escapes out any extended HTML characters (often these characters can be entered by cut/paste from MS Word), translates newlines to
 and tabs to <DD> to ensure that the text entered within a text field is rendered as close as possible to the entered text. Often used to prevent someone from adding in HTML text in an input field and altering the presentation of that text.

```
<%= list.escapeItem("description") %>
```

↗ text escapeItem2(text)

Ignoring HTML numbers, &#xxxx;, removes HTML characters out of the given text. This is helpful because it will not improperly escape the ampersand in an HTML number.

```
<% body = "<h1>War & Peace&#58; A great
read.</h1><br /><p>review text</p>"%>
<%= system.escapeItem2(body) %>
```

Output:

<h1>War & Peace: A great read.</h1>
 review text

↗ text pageItem(fieldname)

Similar to content.pageItem, this function returns the text corresponding to the specified page when used in conjunction with the WYSIWYG page break icon (see content.pageItem for more information).

↗ isEmpty(fieldname)

Returns true if the list item is empty (i.e., does not have any data).

```
<% if not list.isEmpty("subject") then %>
  <%= list.item("subject") %>
<% end if %>
```

↗ isEvenIndex()

Returns true if the item index is even. This is useful for colorizing rows in alternating colors.

```
<% set list = content.createList("subject") %>
<table>
<% do while list.nextEntry() %>
<% if list.isEvenIndex then %>
  <tr bgcolor="#00ff00">
<% else %>
  <tr bgcolor="#ff0000">
<% end if %>
<td><%= list.item("subject") %></td></tr>
<% loop %>
</table>
```

↗ overlayFilesFromFolder(fieldname, label, id)

Overlays files only – (see overlay below for example).

↗ overlayFoldersFromFolder(path or id, label, id)

Overlays folders only – (see overlay below for example).

↗ overlay(srcList, iterator_fieldname)

Overlays one list on top of the other. Using the iterator_fieldname each list entry is consolidated with the corresponding fieldvalue in the second list

entry. This function allows two lists to be merged using a key that defines two entries as being the same.

```
<% set list = content.createList("subject") %>
<% set files = asset.getFileList("/Emails/")
<% files.renameAll "_cmsLabel", "subject" %>
<% list.overlay files, "subject" %>
The id of the subject 'My new subject' is <%=
list.itemAt(list.indexOfItem("subject", "My new
subject"), "_cmsId") %>
```

↗ createList(fieldname)

Creates a list from the specified fieldname. Similar to content.createList, this function creates a sublist from an existing list. This is mostly used to perform nested loops (multi-dimensional arrays) of content.

```
<% set list = content.createList("subject") %>
<% do while list.nextEntry() %>
  <% response.write list.item("subject") & "<br>"
%>
  <% set recip = list.createList("receiver") %>
  <% do while recip.nextEntry() %>
    <% response.write "<DD>" &
recip.item("receiver") & "<br>" %>
  <% loop %>
<% loop %>
```

↗ createListPrefix(fieldname, alt_fieldname)

Creates a list from a single list entry based on the prefix of fieldnames contained in the current entry. The alt_fieldname provides a uniform way of accessing the values once the list is created. (See content.createListPrefix for an example).

↗ clone()

Clones the current list.

```
<% set list = content.createList("counter") %>
<% set new_list = list.clone() %>
```

↗ dictionary_object createLookupDictionary(namefield, valuefield)

Iterates through the entire list object and consolidates a single fieldname into a single dictionary by using the value of the namefield as the key and the value of the value field as the value.

```
<% set list = asset.getFileList("/Files/") %>

<% set dic =
list.createLookupDictionary("_cmsLabel", "_cmsId")
%>

The id for label "Test Asset" is <%= dic.item("Test
Asset") %><br>

The id for label "Another Asset" is <%=
dic.item("Another Asset") %><br>
```

↗ array getArray()

Returns the internal array of dictionary objects that the list object uses.

```
<% myArray = list.getArray() %>

The array size is <%= ubound(myArray) %><br>

The 5th entry is <%= myArray(5).item("subject") %>
```

↗ text getIteratorName()

Returns the name used to create the list from the createList routine in the content or list objects.

```
<% set list = content.createList("subject") %>

the list iterator is <%= list.getIteratorName() %>
which will be "subject"
```

↗ setPrefix(text)

Sets the default content item prefix that is prefixed to each item reference before accessing the field. This is used when multiple fields differ only by their prefix such as in language translations. (See content.setPrefix for an example.)

↗ Text toTableString()

Returns a string representation of the list encapsulated within table cells. Note: The string contains neither “<table>” nor “</table>” tags.

```
<% set list = content.createList(“counter”) %>

<% response.write list.toTableString() %>
```

Output:

<td>counter</td><td>test2</td></tr><tr><td>1</td><td>c
test1</td></tr><tr><td>2</td><td>b

test2</td></tr><tr><td>3</td><td>d
test3</td></tr><tr><td>4</td><td>a test4</td></tr>

↗ Text dump()

Much like the content object's dump function, dump prints out the current data items of the list object in a table format. Mainly used for debugging.

```
<% list.dump() %>
```

INPUT

The input object provides routines specific to creating input interfaces (HTML form input). Many DHTML routines are encapsulated and can be easily used by specifying the appropriate input routines.

↗ setParam(name, value)

Sets an extended parameter for use by other functions.

```
<% setParam "sort_order", "asc" %>
```

↗ getParam(name)

Gets the current value of a previously set parameter.

```
<%= getParam("sort_order") %>
```

↗ delParam(name)

Deletes a parameter.

```
<% delParam "sort_order" %>
```

↗ showSelectDate(name, value)

Provides the popup calendar interface that easily allows the user to enter dates.

The popup calendar allows multiple date selection and range selection. These modes are specified using setParam.

Parameters

multiple – allows user to select multiple dates. Click on existing dates to remove that date from the selection.

range – allows the user to select a date range. Click on a date to select that as the start/end date of the range. Hold down the shift key and select anther date to specify the extent of the range.

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Date</td>
  <td><% input.showSelectDate "date",
content.item("date") %></td>
</tr>
```

↗ showAcquire(fieldname, fieldvalue)

Displays a select and clear graphic button that is used to upload a document into the current template.

- fieldname – the variable name that will contain the uploaded filename
- fieldvalue – the current value of the specified variable name used when editing the page

Parameters

extensions – the allowed extensions that can be specified for upload (gif. Jpeg, etc.) there is no limit to which type of file that can be used.

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Attachment</td>
  <td>
    <% input.setParam "extensions", "doc pdf xls"
%>
    <% input.showAcquire "attach",
content.item("attach"), %>
  </td>
</tr>
```

↗ showAcquireDocument(fieldname, fieldvalue)

Displays a select and clear graphic button that is used to upload a document into the current template.

- fieldname – the variable name that will contain the uploaded filename
- fieldvalue – the current value of the specified variable name used when editing the page

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Attachment</td>
  <td>
    <% input.showAcquireDocument "attach",
content.item("attach"), %>
  </td>
</tr>
```

↗ showAcquireImage(name, value)

Similar to showAcquireDocument but automatically defines the extensions for images (gif, jpeg, jpg, png).

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Portrait</td>
  <td>
    <% input.showAcquireImage "portrait",
content.item("portrait"), %>
  </td>
</tr>
```

↗ showSelectStr(name, value, text_list, value_list)

Utility function to produce a drop-down select HTML form element.

- Name – the name of the select form element
- Value – the value of the select menu as specified in text_list
- Text_list – the list of display items for the select menu (comma delimited list)
- Value_list – optional field of corresponding value items

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Month</td>
  <td>
      <% input.showSelectStr "month",
content.item("month"),
"January,February,March,April,May,June,July,August,
September,October,
November,December" %>
  </td>
</tr>
```

↗ showSelectInt (name, value, text_list)

Produces a select drop-down menu that specifies integer numbers as each selection's value. Unlike showSelectStr, value is always an integer as opposed to a text string. The value specifies the selected index within text_list.

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Month</td>
  <td>
      <% input.showSelectInt "month",
content.item("month")
```

```
"January,February,March,April,May,June,July,August,
September,October,November,December" %>
  </td>
</tr>
```

↗ checkbox(fieldname, default_value, fieldvalue)

HTML check-box input forms do not send back information if the check-box is de-selected (switched off). As no information is transmitted, the CMS does not know that a check-box has been de-sellected unless a process file is used to check for the absence of the value. To remedy this situation, use the check-box routine supplied in this object to provide a check-box that does send back “off” values.

```
<% input.checkbox
 "feature_this", "on", content.item("feature_this")
%>
```

↗ startExpandPanel(title)

↗ endExpandPanel(title)

An expand panel provides a visual way to break up a large input form. The expand panel presents itself as a single gray line that expands out when the user clicks on it. You need to use both startExpandPanel and endExpandPanel for the expand panel to work correctly.

Arguments:

Title – the name to display in the gray area.

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Name</td>
  <td><input type="text" name="name" value="<%=
content.item("name") %>"></td>
</tr>


<% input.startExpandPanel "Other Options – click to
open" %>


<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Option1</td>
  <td><input type="text" name="option1" value="<%=
content.item("option1") %>"></td>
</tr>
<tr align="justify" valign="top" bgcolor="F0F0F0">
```

```
  <td>Option2</td>
  <td><input type="text" name="option2" value="<%=
content.item("option2") %>"></td>
</tr>


<% input.endExpandPanel "Other Options - click to
close" %>
```

↗ startTabbedPanel(titles)

↗ nextTabbedPanel()

↗ endTabbedPanel()

Similar to a windows panel, a tabbed panel (unlike an expand panel) displays a list of tabs that when selected shows the hidden panel.

Arguments:

Titles – tabbed titles separated by a comma.

```
<% input.startTabbedPanel "Option1, Option2,
Option3" %>
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Name</td>
  <td><input type="text" name="name" value="<%=
content.item("name") %>"></td>
</tr>


<% input.nextTabbedPanel %>


<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Option1</td>
  <td><input type="text" name="option1" value="<%=
content.item("option1") %>"></td>
</tr>


<% input.nextTabbedPanel %>


<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Option2</td>
  <td><input type="text" name="option2" value="<%=
content.item("option2") %>"></td>
</tr>
```

```
<% input.nextTabbedPanel %>


<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td>Option3</td>
  <td><input type="text" name="option3" value="<%=
content.item("option3") %>"></td>
</tr>


<% input.endTabbedPanel %>
```

↗ openPopup(filename, width, height, var_list)

Displays a pop-up within the input.asp page. This pop-up makes a callback into the CMS and executes the specified filename. The filename MUST be a file that exists in the current asset's template folder.

Arguments:

- filename – the label of the file to call when opening the pop-up.
- width – width in pixels of the pop-up
- height – height in pixels of the pop-up
- var_list – the list of input form variables to pass to the filename for use in display or processing

input.asp

```
<input type="text" name="date">
<input type="button" onclick="<% input.openPopup
"test.asp", 400, 100, "date" %>">
```

test.asp

```
<html><head></head>
<body
onClick="window.top.main.closeFrame('test.asp');">
Type the text here
<script language="Javascript">
  // can be used to dynamically resize the window
height
  myFrame =
window.top.main.getFrameObj('test.asp').style.heigh
t=50;
</script>
```

```
</body></html>
```

↗ showSelectColor(color_fieldname, color_fieldvalue)

Shows the select color widgit.

```
<TR align="justify" valign="top" bgcolor="F0F0F0">
  <TD>Color</TD>
  <TD><% input.showSelectColor "color",
content.item("color") %></TD>
</TR>
```

↗ addTextAreaButton(button_name)

Adds a WYSIWYG button to the WYSIWYG editor. The buttons will appear in the order specified. If you remove the addTextAreaButton for a specific functionality (for example, "bold"), that function is removed and disallowed from being performed in the editor. (See showTextArea below for an example.)

<sep> specifies that the verticle bar should be shown

 specifies that a new row should be started

↗ showTextArea(fieldname, fieldvalue, width, height)

Displays a WYSIWYG editing area.

The showTextArea takes several parameters for additional configuration:

- stylesheet - specify the stylesheet to use in the editor. Can be an CMS id or path.
- pasteoptions - specifies which paste modes will be supported. Use one or more of:

  "As Is, Word Cleaned, Layout & Styles, Layout Only, Design Only, Text With Breaks, Text with Newlines, Text Only" in any order to offer the user that option when pasting. If only one option is specified the user will not be asked how the paste should be perfomed.
- imagestyle - specify the default stylesheet for images
- image_folder - specify the default image upload folder in the CMS
- image_upload - specify if uploaded images become attached or are linked to from the active document
- image_browse - specify if selected images become attached or are just linked to from the active document.

- linkstyle - default stylesheet for links
- link_folder - specify the default asset link folder in the CMS
- link_upload - specify if uploaded assets to link to become attached to the active document or are placed into the CMS as an asset and linked to from the active document
- link_browse - specify if selected assets become attached or are just linked to from the active document
- body_style - specify the default body text class to use in the editing area.
- designedittabs - specify if the edit and design tabs are shown in the editing area
- defaultToText - specify if the edit area should default to the text edit mode

```
<tr align="justify" valign="top" bgcolor="F0F0F0">
  <td colspan=2>
    <%
      input.setParam "stylesheet",
"/System/stylesheet.css"
      input.setParam "pasteoptions", "Layout Only"
      input.setParam "imagestyle", "text"
      input.setParam "image_folder", "/Images"
      input.setParam "image_upload", "Attach"
      input.setParam "image_browse", "Link"
      input.setParam "linkstyle", "text"
      input.setParam "link_folder", "/"
      input.setParam "link_upload", "Attach"
      input.setParam "link_browse", "Link"
      input.setParam "bodystyle", "text"
      input.setParam "DesignEditTabs", 1
      input.addTextAreaButton "<sep>"
      input.addTextAreaButton "stylesheet"
      input.addTextAreaButton "<br>"
      input.addTextAreaButton "<sep>"
      input.addTextAreaButton "cut"
      input.addTextAreaButton "cuthtml"
      input.addTextAreaButton "crop"
      input.addTextAreaButton "copy"
      input.addTextAreaButton "paste"
```

```
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "undo"
        input.addTextAreaButton "redo"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "internallink"
        input.addTextAreaButton "link"
        input.addTextAreaButton "removelink"
        input.addTextAreaButton "anchorlink"
        input.addTextAreaButton "makeanchor"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "symbol"
        input.addTextAreaButton "find"
        input.addTextAreaButton "replace"
        input.addTextAreaButton "<br>"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "table"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "edittable"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "altertable"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "tableborder"
        input.addTextAreaButton "<br>"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "bold"
        input.addTextAreaButton "italic"
        input.addTextAreaButton "underline"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "forecolor"
        input.addTextAreaButton "backcolor"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "justifyleft"
        input.addTextAreaButton "justifycenter"
        input.addTextAreaButton "justifyright"
        input.addTextAreaButton "<sep>"
        input.addTextAreaButton "unorderedlist"
        input.addTextAreaButton "orderedlist"
        input.addTextAreaButton "<sep>"
```

```
      input.addTextAreaButton "indent"
      input.addTextAreaButton "outdent"
      input.addTextAreaButton "<sep>"
      input.addTextAreaButton "pagebreak"
      input.addTextAreaButton "<sep>"
      input.addTextAreaButton "upload"
      input.addTextAreaButton "Image Caption"

     input.showTextArea "page_body",
content.item("page_body"), 540, 400 %>
</td>
</tr>
```

↗ showSelectList(fieldname, fieldvalues, title, fieldname2, fieldvalues2, title2, width, height, displayNames)

Displays a move selector type widgit that allows a user to move items from one scrolling list box to another, to provide a selection mechanism.

Note: The fieldvalues are list objects.

The last parameter requires a lookup dictionary (name and value pairs) that can be used when the fieldvalues are not usable values and need to be transformed into more readable ones. For example if IDs are used the lookup table can map IDs to text strings for user display but still use IDs as the content stored in the CMS.

```
<TR align="justify" valign="top" bgcolor="F0F0F0">
    <TD>Slideshow Images</TD>
    <TD>
      <% set list =
asset.getFileList("/NCC/Kiosk/Slideshow Images/")
%>
          <% set lookup = list.
createLookupDictionary("_cmsId", "_cmsLabel") %>
      <% input.showSelectList "_cmsId", list,
"Images", "images", content.createList("images"),
"Selected", 140, 150, lookup %>
    </TD>
</TR>
```

↗ showSelectGMT(fieldname, fieldvalue)

Shows a GMT clock selector. This selector ensures that times selected are converted to GMT time from the local time seen by the user. The time zone

of the user is assumed to be correct as it is gathered from the installed browser.

```
<TR align="justify" valign="top" bgcolor="F0F0F0">
    <TD>Enter date</TD>
    <TD><% input.showSelectGTM "date",
content.item("date") %></TD>
  </TR>
```

CLOCK

The Clock object defines some additional date/time manipulation routines in addition to the Date(), Month(), Day(), etc routines that are available in ASP.

↗ formatDate(date, text_format)

Used to print a date based on a format string.

Arguments:

- Date – ASP Date object
- Text_format – the date format string

  The format string is composed of one or more of the following types:

- AM - AM or PM notation
- am - am or pm notation
- MM - month number (2)
- mon - 3 letter month (feb)
- Mon - 3 letter month (Feb)
- MON - 3 letter month (FEB)
- month - full month (febuary)
- Month - full month (Febuary)
- MONTH - full month (FEBUARY)
- mont - per chicago manual of style (feb.)
- Mont - per chicago manual of style (Feb.)
- MONT - per chicago manual of style (FEB.)
- DDD - day in year (62)
- DD - day in month (3)

- D - day in week (4)
- ddd - day in year (62)
- dd - day in month (3)
- d - day in week (4)
- dy - 3 letter weekday (tue)
- Dy - 3 letter weekday (Tue)
- DY - 3 letter weekday (TUE)
- day - full weekday (tuesday)
- Day - full weekday (Tuesday)
- DAY - full weekday (TUESDAY)
- YYYY - 4 digit year (2002)
- YY – year (2)
- yyyy - 4 digit year (2002)
- yy – year (2)
- WW - week in year (10)
- W - week in month (2)
- HH - hours in day (8)
- HH24 - 24 hours in day (20)
- MI - minutes in hour (43)
- SS - seconds in minute (55)

By default, each numerical format has no padding and is displayed as a single or double digit. To ensure “0” padding, add a “0” prior to the format type.

Note: Certain formats are case-sensitive.

```
<%= clock.formatDate(Date(), "0MM/0DD/0YY
HH:0MI:0SS") %>

<%= clock.formatDate(Date(), "0MM/0DD/0YY
0HH:0MI:0SS") %>

<%= clock.formatDate(Date(), "Month DD, YYYY
HH:0MI:0SS") %>
```

↗ Bool isValidDate(date)

Returns true if the provided date is a properly formatted date.

```
<% if clock.isValidDate(content.item("date")) then
%>
  <%= clock.formatDate %>
<% end if %>
```

↗ Bool isFutureDate(date)

Returns true if the provided date is either equal to or occurs after the current date.

```
<%= clock.isFutureDate(now) %>
```

Output:

True

↗ getDateTime()

Shorthand to return the date and time with zero padding and 24 hr.

```
<%= clock.getDateTime() %> - 01/02/2003 20:10:50
```

↗ getDate()

Shorthand to return the date with zero padding.

```
<%= clock.getDate() %> - 01/02/2003
```

↗ Text getMonthName(index)

Given a from 1 to 12, returns the corresponding month name. Will Return an empty string if index is blank, or out of range.

```
<% 'fetching month name for November and December
%>
<%= util.getMonthName(11) & "<br />" %>
<%= util.getMonthName(12) & "<br />" %>
```

Output:

November

December

↗ Text getChicagoName(index)

Given a number 1-12, returns the Chicago Name of the corresponding Month. Returns an empty string if index is empty or the number provided is out of range.

```
<% 'fetching Chicago name from February through
March %>
<%= util.getChicagoName(2) & "<br />" %>
<%= util.getChicagoName(3) & "<br />" %>
<%= util.getChicagoName(4) & "<br />" %>
```

Output:

Jan.

Feb.

Mar.

↗ Text getWeekdayName(index)

Given a number 1-7, returns the name of the weekday.

```
<%= util.getWeekdayName(5) %>
```

Output:

Friday

↗ getTime()

Shorthand to return 24-hr time.

```
<%= clock.getTime() %> - 20:10:50
```

UTIL

The Util object provides the cms templates with a variety of utility functions. Many provide a means to manipulate text such as cropping text and converting HTML to XHTML. The Util object also provides access to HTTP.

↗ setParam(param_name, param_value)

Sets an extended parameter for use by other functions.

```
<% util.setParam "_cmsLayout","output" %>
```

↗ getParam(param_name)

Gets the current value of a previously set parameter.

```
<% util.getParam("_cmsLayout") %>
```

↗ delParam(param_name)

Deletes the given parameter.

```
<% util.delParam("_cmsLayout")
```

↗ URLDecode(text)

Routine to decode a URL encoded string.

```
<%= util.urlDecode("%20%26test%43") %>
```

↗ URLEncode(text)

Routine to encode a URL string.

```
<a href="test.asp?name=<%=
util.urlEncode(content.item("name")) %>">test</a>
```

↗ stripAsp(text)

Removes any ASP tags from the specified text.

```
<%= util.stripAsp( content.item("text")) %>
```

↗ stripHtml(text)

Removes anyHTMLl tags from the specified text.

```
<%= util.stripHtml( content.item("body")) %>
```

↗ filterText(text, mode)

Filters text according to the specified mode.

Arguments:

- Text – the text to filter
- Mode
- Alpha – removes everything except letters from text
- Alphanum – removes everything except letters or digits from text
- Filename – removes illegal filename characters (as defined by FTP) from text
- Filepath – removes illegal filepath characters (as defined by FTP) from text
- Num - removes everything except digits from text

↗ crop(text, length)

Crops the text to the given word length. The routine will attempt to break text on a space if a space exists, otherwise it truncates text. This routine will append "..." if text extends past length.

```
<%= util.crop(content.item("title"),200) %>
```

↗ getRandom()

Returns a random number from 1 to 100000.

```
<%= util.getRandom() %>
```

↗ hasWhiteSpace(text)

Returns true if the text has new lines, carriage returns, tabs and/or spaces.

```
<% if util.hasWhiteSpace("this has many spaces")
then %>

Yup.

<% end if %>
```

↗ stripNoiseWords(text)

Removes noise words from a text string. Noise words are belived not to contribute significantly to the content of a sentence and can be removed to create abbreviated sentences (like ones used for labels).

```
<%= util.stringNoiseWords("This is a test
sentence") %>

<% ' prints out "test sentence" since "this" and
"is" and "a" are noise words %>
```

↗ text getHttp(url)

Returns the HTML of the provided url.

```
<%= util.getHttp("http://www.google.com") %>
```

↗ convertTextToHtml(text)

Converts text to its HTML counter part by converting vbCRLF to
, tabs to <dd>, and any extended characters to HTML counterparts (i.e. smart quotes to quotes, etc).

```
<%= convertTextToHtml("line1" & vbCRLF & "line2")
%>
```

↗ Text convertHTMLToRtf(text)

Converts given HTML string into Rich Text Format.

Template:

```
<% description_text = "<b>This text is bold.</b>
This is <i>italics</i>" %>

<%= util.convertHTMLToRtf(description_text) %>
```

Output:

{/b This text is bold.} This is {\i italics}

↗ text convertHTMLtoXHTML(html_text, namespace)

Converts the given HTML string into XHTML. If a namespace is provided, the xmlns attribute will be set to the provided namespace otherwise it will be removed from html_text.

```
<% html_text = "<p>This html was not xhtml.<br>It
needed to be converted" %>

<% html_text =
util.convertHTMLtoXHTML(html_text,"") %>

<% response.write html_text %>
```

Output:

 This HTML was not xhtml.
it needed to be converted

↗ escapeHtml(text)

Similar to content.escapeItem this function converts text to escaped HTML. This routine will convert quotes to ", less than to <, etc.

```
<%= util.escapeHtml(myTextString) %>
```

↗ Text escapeHTML2(text)

Converts text to escaped HTML ignoring HTML Numbers, for example " ! etc.

```
<% test_html = "&#34;This is a test&#33;&#34;" %>

<%= util.escapeHTML2(test_html) %>
```

Output:

"This is a test!"

↗ createList(dictionary, fieldname_iterator)

Creates a list object from a dictionary. Fields (and the iterator) are assumed to be keyed as name:1, name:2, name:3, etc., to indicate entry indexes.

```
<% set dic = system.createDictionary() %>
```

```
<% dic.add "subject:1", "test" %>
<% dic.add "message:1", "my message" %>
<% dic.add "subject:2", "this" %>
<% dic.add "message:2", "another message" %>
<% set list = util.createList(dic, "subject") %>
```

↗ topFolders(path, number)

Returns the top x folders from the specified path.

```
<%= util.topFolders("/System/Templates/Press
Release/input.asp", 2) %>
<% ' prints out /System/Templates %>
```

↗ regEx(text, pattern, delimiter)

Performs a regular expression against the supplied text and returns matched results delimited by the supplied delimiter.

```
<%= util.regex("this is a test", "([A-Za-z]+)",
"|") %>
<% ' prints "this|is|a|test" %>
```

↗ parseInteger(text)

Parses the specified text for a number and returns just that number. This is similar to content.intValue().

```
<%= util.parseInteger("this678test") %>
<% ' prints "678" %>
```

↗ crypt(text)

Returns the specified text crypted. The crypt function is the same as used by Unix- based systems for .htaccess or related password encryptions. Using this function a user can publish out .htaccess files generated by the CMS. To check for a valid password, the user-entered password is crypted and then compared with the previously stored crypted password. If they are the same, the password is valid.

```
<%= util.crypt("mysecretword") %>
<% ' prints "524LSKTh1sbaE" %>
```

↗ titleCase(text)

Returns the text in title case. As opposed to ucase or lcase (VBscript functions) titlecase will return each first letter of each word as an uppercase letter with following letters in lowercase.

```
<%= util.titleCase("This is a test") %>

<% ' prints "This Is A Test" %>
```

↗ visibility(selected, fieldname)

The visibility routine is used as a convenience routine when hiding parts of the input.asp form to hide or show rows based on content values. This routine is often used by the code wizard.

```
<TR <%=
util.visibility(content.item("has_subject"), "yes")
%> align="justify" valign="top" bgcolor="F0F0F0">

<TD><input type="Text" name="subject" value="<%=
content.escapeItem("subject") %>"></TD>

</TR>
```

↗ wrapText(text, row_size, delimiter_to_use)

WrapText adds delimiters to standard text depending on the specified row size. The function attempts to break each line of text at a word boundary to ensure that text will maintain readability. Often email readers do not automatically wrap text and therefore adding newlines will format the text in a more readable format.

```
<%=util.wrapTExt(util.stripHTML(txtHeader),70,vbCrL
f)%>

<% ' adds newlines approximately every 70
characters %>
```

↗ splitUrl(url_to_split, protocol, domain, path, filename)

Splits an http URL into its components.

```
<% Dim protocol, domain, path, file %>

<%= util.splitUrl("http://www.crownpeak.com

/Content_Management_Solution/Features.asp",
protocol, domain, path, file)%>

<%= protocol & "://" & domain & path & file %>

<% ' prints out the original url %>
```

↗ list_object createListFromCSV(csv_text, delimiter)

Creates a list object from a CSV text string. The first row is assumed to be the fieldname of each column separated by the specified delimiter. Each row is assumed to be delimited by vbCRLF, vbCR, or vbLF.

```
<%
```

```
  ' create a list object based on a tab delimited
csv datafile

  set list =
util.createListFromCSV(content.item("csv_text"),
chr(9))

  ' now loop through the list object and print out
the company name field

  do while list.nextEntry()

    response.write list.item("companyname") &
"<br>"

  loop

%>
```

↗ text filterUrl(text)

Filters the URL based on the filename configuration located under System > Configure > Filenames. URLs are converted automatically, therefore, this function should only be used to ensure that the changes to the URL conform to the filename configuration.

```
<% txt url = content.item("_cmsPublishPath") %>

<% url = replace(url, "/site/",
"/NEW_Folder_Location/" %>

<% url = util.filterURL(url) %>
```

↗ text filterFilename(text)

Used to process text through the system filters defined in System > Configure > Filenames. When in a filename.asp, url.asp, etc. template file, the only variable that is filtered based on the global system filter is the _cmsPublishPath or _cmsPublishUrl variables. If you construct a filename or URL using _cmsPropertiesFilename, etc., you can use the filterFilename or filterUrl to run the resulting path through the system filters as needed.

```
<%

  Dim myPath

  myPath =
util.filterFilename(content.item("_cmsPropertiesFol
der") & content.item("_cmsRelativeFolder") &
content.item("catId") & "." &
content.item("_cmsRemoteExtension"))

  '... other processing

  content.add "_cmsPublishPath", myPath

%>
```

↗ Text getFileNameFromPath(path)

Returns the filename from the given path.

```
<% fictitious_path = "/Site/Home Page/Index" %>
<% response.write
util.getFileNameFromPath(fictitious_path) %>
```

↗ showEditButton(button_text, left_position, top_position, width, height)

shows an edit button that when clicked upon, opens the current asset in edit mode. The value of button_text, will appear on the button. left_position and top_position are used to define the position of the button using the style attribute. Width and height specify the dimensions of the button. If width and height are equal to "", the width will default to 80 and height will default to 18.

```
<%util.showEditButton "Click Here to
Edit",0,0,"","" %>
```

Output:

↗ showCmsMenu(button_text,cmd,top_position,left_position)

Functions much like showEditButton, but it allows for further customization. cmd defines the action the button will take and can be equal to:

- create – creates a new asset and displays the input template
- edit – edits the current asset
- workflow – edits the workflow attached to the current asset.
- delete – deletes the current asset
- clone – clones the current asset, the resulting new asset keeps the same label as the current one.
- branch – branches the current asset
- folder – returns to the folder view

```
<% util.ShowCmsMenu "Click Here to
Delete","delete",0,0 %>
```

Output:

USER

The user object provides access to user information. The user object is populated with the current user information. Other users (user objects) can be accessed using the getUser routine within the user object.

The user object has the following fields that can be accessed:

- username
- avatar – the current user's avatar image URL
- email
- lastname
- firstname
- description
- title
- department
- location
- phone
- fax
- cell
- pager
- lastLoginDate
- lastLoginAddr

```
The current user is <%= user.firstname & " " &
user.lastname %>.
```

↗ user getUser(username)

Returns a new user object populated with the user specified by username. All fields are set and can be accessed.

```
<% set theUser = user.getUser("Edmund") %>

<%= theUser.firstname & " " & theUser.lastName %>
belongs to <%= theUser.department %>
```

↗ Text getFullName(username or user_id_number)

Given a username or user ID, returns the full name of the user. If the parameter is empty, the getFullName returns empty quotes.

```
<%'Numeric %>
```

```
<%= user.getFullName("fBaggins") %>
<%= user.getFullName(50) %>
```

Output:

Frodo Baggins

↗ boolean inGroup(group_name)

Used to tell if the user belongs to the specified group name.

```
<% if user.inGroup("Admin") then %>
  This user is in the admin group
<% end if %>
```

↗ boolean inGroupId(group_id)

Provides an alternate way of specifying a group. Instead of using the group's name, the group ID can be used instead.

```
<% if user.inGroupId(10007) then %>
  This user is in the admin group
<% end if %>
```

↗ list_object getGroupList()

Returns a list object populated with name and id fields corresponding to the group names and ID that the user belongs to.

```
<% set list = user.getGroupList() %>
<%= list.join("name", ", ") %> or <%=
list.join("id", ", ") %> for group name and group
id
```

↗ content_object getPreferences()

Returns a content object populated with the users preferences. The following preferences are supported:

- user_id = 11
- assignedMyTasks = True
- assignedTasksToMe = True
- completedMyTasks = True
- addComment = True
- dueDateReminder = True
- reminderHours = 0

- startup_folder_id = 856
- annotation_theme_id = 5
- folder_sort = 3
- folder_page_size = 25
- preview_state = 0,780,785,781,783
- hideinfopanel = False
- viewTaskSummary = True
- viewPendingSummary = True
- viewCommentsSummary = True
- viewViewedSummary = True
- viewWorkedSummary = True
- viewCreatedSummary = True
- viewErrorsSummary = True
- viewRejectedSummary = True
- silentdeploy = False
- login_mode = 1
- home_mode = 2
- viewAlerts = False
- workflowTasks = False

↗ setPreferences(content_object)

Resets the users preferences based on the values in the specified content object.

↗ removeGroup(groupId or groupname)

Removes the current user from the specified groupId.

```
<% user.removeGroup("Admin") %>
<% ' removes the user from the Admin group %>
```

↗ addGroup(groupId or groupname)

Adds the user to the specified group.

```
<% user.addGroup("Admin") %>
<% ' adds the user to the Admin group %>
```

ACCESS

The Access Object provides the cms templates with access control information, such as view, edit, and publishing permission.

↗ getAccess(groupid or groupname, ACL name)

Gets values in the Access object returned by asset.getAccessFields. See setAccess for a list of supported ACL names

```
<% set acl =
asset.getAccessFields(content.item("_cmsId")) %>
<% if acl.getAccess("Authors", "view") then %>
  <% acl.setAccess "Authors", "edit", true %>
<% end if %>
<% asset.setAccessFields(content.item("_cmsId"),
acl) %>
```

↗ setAccess(groupid or groupname, ACL name, ACL value)

Sets values in the Access object returned by asset.getAccessFields.

Supported ACL names are:

- checkout
- comments
- content
- copy
- delete
- download
- edit
- general
- history
- import
- inopenmenu
- inpreviewmenu
- move
- newdependency

- newfolder
- newmodel
- newupload
- publish
- rename
- republish
- schedule
- setaccess
- setalerts
- setimport
- setmodel
- setoptions
- setpublish
- settemplate
- setworkflow
- version
- view
- viewdependency

```
<% set acl =
asset.getAccessFields(content.item("_cmsId")) %>
<% if acl.getAccess("Authors", "view") then %>
  <% acl.setAccess "Authors", "edit", true %>
<% end if %>
<% asset.setAccessFields(content.item("_cmsId"),
acl) %>
```

SYSTEM

The System object provides the cms the ability to creat objects not native to ASP Classic, such as DOM, XML and filters. Normally one would create an XML object witht the System object, and utilize the XML Object functions to manipulate the xml. It is also used to extend existing ASP Classic objects to be used in the context of the cms, such as the Dictionary Object.

↗ setParam(name, value)

Sets an extended parameter for use by other functions.

```
<% system.setParam "sort_order", "asc"
```

↗ getParam(name)

Gets the current value of a previously set parameter.

```
<% system.getParam("sort_order") %>
```

↗ delParam(name)

Deletes the given parameter.

```
<% system.delParam("sort_order")
```

↗ include(path or id)

Dynamically includes the specified file into the current file being processed. The included file operates as if it is part of the current document and shares the same namespace, memory, etc.

```
<% system.include "header.asp" %>
```

↗ asp(text)

As the CMS templates are written in ASP VBScript or JScript, it needs to be escaped in order for it to be included in a published page, so it will excute on server instead of the cms. To pass through ASP you can use the asp routine OR you can use the "$" character in place of the "%" character.

```
   <%= asp("Date()") %>
<$= Date() $)
```

Published Output:

<%= Date() %>

<%= Date() %>

↗ virtualssi(path)

Due to the templating system that the CMS uses, virtual includes will not function correctly unless specified by using the virtualssi routines. Specifying a virtual server side include using the virtualssi function will ensure that ssis are resolved in previewing within the CMS and that they are translated to the correct <!-- #include virtual= etc --> specification when published to stage or live sites.

```
<% vistualssi "/access/authorization.php" %>
```

↗ filessi(ByVal code)

Similar to virtual ssi, but specifies a file include instead of a virtual include.

```
<% filessi "/access/authorization.php" %>
```

↗ createContentObject()

Creates another content object that can be used to store data.

```
<% set data = system.createContentObject() %>
```

↗ createList()

Creates an empty list object.

```
<% set list = system.createList() %>
```

↗ createDictionary()

Creates an empty ASP intrinsic dictionary.

```
<% set dic = system.createDictionary() %>
```

↗ wrap(filename, name)

Wraps template with file.

```
<% set dic = system.createDictionary() %>
```

↗ createXML()

Returns a CMS XML object for XML content manipulation.

```
<% set xml = system.createXML() %>
```

↗ createDOM()

Creates a DOM object using the Microsoft asp component.

```
<% set dom = system.createDOM %>
```

↗ createTransform()

Creates a XSLT transform object.

```
<% set transform = system.createTransform()
```

↗ createRegEx()

Returns a new Regular Expression object.

```
<% set regEx = system.createRegEx() %>
```

↗ createFilter()

Returns a filter object to be used in asset.getFilter().

```
<% set myFilter = system.createFilter() %>
```

↗ transform(xml, xsl)

When given an XML file and an XSLT file, it returns the resulting HTML page.

```
<% 'combining the xml_output and xslt_output to
form the transform %>

<% system.transform
"xml_output.asp","xslt_output.asp"  %>
```

↗ setPublishLoop(true or false)

Indicates to the CMS to keep republishing the current page. This is needed if pagination is enable or multiple pages need to be generated from a single page.

```
<% if count < 10 then %>

<% system.setPublishLoop(true) %>

<% end if %>
```

↗ startCapture()

Starts text capture. All output from this point on is recorded and not sent back to the client browser or published.

↗ text stopCapture()

Stops capture mode and returns all the text printed between startCapture and stopCapture. The text can then be processed and/or printed as needed.

```
<% system.startCapture() %>

<%= content.item("subject") %><br>

<%= content.item("message") %><br>

<% txt = system.stopCapture() %>

<% response.write ucase(txt) %>

<% ' prints out contents of subject and message in
uppercase %>

setDependency(true or false)
```

Turns on or off automatic dependencies that the CMS uses to determine which files need to be published as dependant files. This is useful in certain cases to prevent the entire site being published out due to a global dependency such as a navigation include.

```
<% system.setDependency false %>
```

E-MAIL

The e-mail object provides a way to send e-mails from the CMS templates.

Parameters:

- to - who the email is to be sent to
- body - the email message
- subject - the subject of the email
- from - who the email is from
- contenttype - typically text/plain or text/html

↗ setParam(name, value)

Sets an extended parameter for use by other functions.

```
<% email.setParam "subject","this is a subject" %>
```

↗ getParam(name)

Gets the current value of a previously set parameter.

```
<% email.getParam("subject") %>
```

↗ delParam(name)

Deletes the given parameter.

```
<% email.delParam("subject") %>
```

AddAttachment(path or id)

Adds an attachment to the e-mail message.

↗ Send()

Sends the e-mail message.

```
<% set userRec =
user.getUser(content.item("_cmsStatusUserId")) %>

<% system.startCapture() %>

Hi,


The following event just took place for <%=
content.item("_cmsLabel") %>:

<%= content.item("_cmsStatusText") %> <%=
content.item("_cmsStatusDate") %> by
```

```
<%= userRec.firstname %>  <%= userRec.lastname %>.

Go to
http://advantage.crownpeak.com/myname/index.asp?_ta
skvalue=

assetbrowse&_idname=_assetid&_idvalue=<%=
content.item("_cmsFolderId") %> for details.

Thanks,

The CMS System

<%

  body = system.stopCapture()


  email.setParam "subject", " STATUS CHANGE"

  email.setParam "body", body

  email.setParam "to", "user@myname.com"

  email.setParam "from", userRec.email

  email.addAttachment("/System/emails/demo.doc")

  email.send()

%>
```

FILENAME

The Filename object provides the cms templates access to file properties.

↗ getIcon(path)

Returns the CMS path for the icon that best represents the specified path.

```
<IMG BORDER=0 SRC="<%=
filename.getIcon("/path/filename.pdf") %>">

<% ' prints /cpt_icons/pdf.gif %>
```

↗ getExtension(path)

Returns the filename extension for the specified path.

```
<%= filename.getExtension("/path/filename.pdf") %>

<% ' prints "pdf" %>
```

↗ getFilename(path)

Returns the filename of the specified path.

```
<%= filename.getFilename("/path/filename.pdf") %>

<% ' prints "filename.pdf" %>
```

IMAGE

The Image object provides the cms templates with image properties such as height and width, as well as image manipulation functions such as cropping and creating thumbnails.

↗ setParam(name,value)

Sets an extended parameter for use by other functions.

```
<% image.setParam “height”,”100” %>
```

↗ getParam(name)

Gets the current value of a previously set parameter.

```
<% image.setParam(“height”) %>
```

↗ delParam(name)

Deletes the given parameter.

```
<% image.delParam(“height”) %>
```

↗ setQuality(number)

Sets the quality for saving JPEG images. Specifying 1 means save using the best quality but with a large byte size, and with 5 being the worst but least amount of byte size. The default is set to 2.

```
<% image.setQuality 1 %>
<%' updates JPEG save quality to best %>
```

↗ thumbnail(path or id, suffix, width, height)

Creates a thumbnail of the specified image. The size of the created thumbnail is guaranteed to be less wide than the specified width and less high than the specified height. The aspect ratio is preserved so the width and height may be equal to or less than the respective sizes.

```
<%
  ' make sure WYSIWYG images don't get too wide
using upload.asp
  value = content.item("_cmsUploadValue")

ext=filename.getextension(content.item("_cmsUploadV
alue"))
  ' create a medium size thumbnail
```

```
  if ext="gif" or ext="jpg" then
    res = image.thumbnail(value, "", 430, 1000)
    if res = "" then
      content.add "_cmsError", image.errorMsg
      exit Sub
    end if
    content.add "_cmsUploadValue", res
  end if
%>
```

↗ cropImage(path or id, suffix, width, height, x, y, xx, yy)

Crops an image to the specified dimensions and copies the cropped image to a new filename using the suffix as a filename modifier. The suffix is needed if you wish to retain the original uncropped image.

```
<%' in input.asp %>
<TR class=hidden align="justify" valign="top"
bgcolor="F0F0F0">
    <TD>Image</TD>
    <TD>
      <img src="<%= content.defaultItem("image",
"/cpt_media/default.gif") %>">
      <% ' setup image upload with cropping enabled
%>
      <% input.setParam "lasso_width", 360 %>
      <% input.setParam "lasso_height", 270 %>
      <% input.setParam "lasso_box_left",
content.item("image_lasso_box_left") %>
      <% input.setParam "lasso_box_top",
content.item("image_lasso_box_top") %>
      <% input.setParam "lasso_box_width",
content.item("image_lasso_box_width") %>
      <% input.setParam "lasso_box_height",
content.item("image_lasso_box_height") %>
      <% input.setParam "extensions", "png gif jpeg
jpg" %>
      <% input.setParam "show_upload", "Attach" %>
      <% input.showAcquireImage "image",
content.item("image") %>
    </TD>
  </TR>
```

```
<%
  ' in post_input.asp crop and scale the image to
selected sizes
  Dim vals
  ' get the crop sizes and dimensions submitted
from input.asp (above)
  vals = split(content.item("_lasso_my_photo"),
",")
  if isArray(vals) then
    if ubound(vals) = 5 then
      txt = image.cropimage (content.item("image"),
"_thumb", vals(0), vals(1), vals(2), vals(3),
vals(4), vals(5))
      content.add "upload#image_thumb",txt
    end if
  end if
%>
```

↗ getWidth(path or id)

Returns the image width in pixels.

```
<img src="<%= content.item("image") %>" width="<%=
image.getWidth(content.item("image")) %>">
```

↗ getHeight(path or id)

Returns the image height in pixels.

```
<img src="<%= content.item("image") %>" height="<%=
image.getHeight(content.item("image")) %>">
```

↗ number getFormat(filename)

Returns the image format. The image format is determined from the first couple of bytes from the binary file. This is often referred to as the Magic number and is a better way to determine filetype than using the file's extension.

The number returned indicates its type.

- Unknown = -1
- BMP = 0
- ICO = 1
- JPEG = 2
- PCX = 10

- PNG = 13
- PPM = 14
- PPMRAW = 15
- TIFF = 18
- PSD = 20
- GIF = 25

```
<%
  if image.getFormat(value) = 2 then
    response.write "Its a JPEG image!"
  end if
%>
```

↗ image_handle load(filename)

Loads an image file into memory in preperation for image operations.

↗ crop(image_handle, x, y, width, height)

Crops a memory image to the specified dimensions.

Note: The first parameter is the result of the load operation as opposed to a filename reference.

↗ scale(image_handle, width, height, mode)

Scales a memory image to the specified dimensions.

- Mode =0 means scale such that the final width and/or height is not larger than the specified width or height.
- Mode =1 means scale such that the final width and/or height is at least the specified width or height.

↗ merge(image_handle, image_handle2, x, y, red, green, blue)

Merges two images into a single image starting at x,y corrdinates. The red, green and blue values specify an RGB value that should be considered transparent, which will cause the image_handle pixels to show through.

↗ saveAsAttachment(image_handle, cmsId, suffix)

Saves a memory image as an attachment into the current document.

↗ setColor(image_handle, color)

Sets the color for subsequant draw operations.

↗ draw(image_handle, image_handle2, x, y, width, height, xx, yy)

Draws image_handle2 image ontop of image_handle at the specified xx, and yy coordinates. The x,y, width, and height determine what area of image_handle2 gets drawn onto image_handle.

↗ drawRect(image_handle, x, y, width, height)

Draws a rectangle with the current color given the specified dimensions.

↗ fillRect(image_handle, x, y, width, height)

Fills a rectangle with the current color given the specified dimensions.

↗ free(image_handle)

Frees the image from memory.

```
<%
  ' in upload.asp
  Dim value
  value = content.item("_cmsUploadValue")
  if content.item("_cmsUploadVariable") =
"upload#image" then
    myImage = image.load(value)
    if myImage <0 then
      content.add "_cmsError", image.errorMsg
      exit Sub
    end if
    image.scale myImage, 100, 100, 1
    image.setQuality 1
    content.add "_cmsUploadValue",
image.saveAsAttachment(myImage,
content.item("_cmsId"),"_thumbnail")
  end if
%>
```

DEBUG

The debug object allows you to log messages generated in template script files to a centralized debugging console. To activate the debugging console, select Debug Console under System > Tools.

↗ write(text)

Used to add a debugging text statement to the log. A timestamp is automatically prepended to the console display.

```
<% debug.write "This text string will show up in
the debug console." %>
```

↗ clear()

Clears the debug log of all messages.

```
<% debug.clear %>
```

↗ startCapture()

Begins logging all text resulting from a template file to the debugging console. Any text characters printed outside of ASP tags or using the response object are logged.

↗ stopCapture()

Terminates logging of all text to the debugging console.

```
<% debug.startCapture  %>
This text will show up in the debug console.
<%= content.item("_cmsFolder") %> This too.
<% response.write content.item("_cmsId") %> and
this aswell.
<% debug.stopCapture %>
But this will not.
```

XML

The XML object provides functions which the cms templates can use to read, parse, and traverse xml data.

↗ createXML()

Creates an XML object.

```
<% set xml = system.createXML() %>
```

↗ loadXML(txt)

Given xml, it will load it into an XML object.

```
<% txt =
asset.getUploadAsText(content.item("xml_file")) %>
<% set xml = system.createXML() %>
<% if not xml.loadXML(txt) then %>
  <% content.add "_cmsError", "Invalid XML format"
%>
  <% exit sub %>
<% end if %>
```

↗ Dictionary selectSingleNodeAsDic(node)

Converts the contents of the given node into a dictionary, including sub nodes. If eventlist were passed to selectSingleNodeAsDic from the XML below, each event node and all of their parts would appear in the dictionary.

<eventlist>

<event>

<title>This is a title</title>

<children>

<child1 />

<child2 />

</children>

</event>

<event>

…

</event>

</eventlist>

```
<% set fields = xml.selectSingleNodeAsDic("event")
%>
```

↗ Text createXmlFromDic(dictionary)

Converts the given dictionary into an XML string. The dictionary key is set to the tag, and value is wrapped by the tag. This function does not support XML attributes. The resulting XML does not include an XML version. For example, <?xml version="1.0" encoding="ISO-8859-1"?>.

```
<% set dictionary = system.createDictionary() %>
```

```
<% dictionary.item("event-title") = "This is a
title" %>
<% dictionary.item("event-start-date") = "20201105"
%>
<%= xml.createXmlFromDic(dictionary) %>
```

Output:

<event-title>This is a title</event-title>

<event-start-date>20201105</event-start-date>

↗ Text selectSingleNodeTypedValue(node)

Extracts all the XML tags from the XML returning and returns the resulting string.

```
<% txt =
asset.getUploadAsText(content.item("xml_file")) %>
<% set xml = system.createXML() %>
<% if not xml.loadXML(txt) then %>
  <% content.add "_cmsError", "Invalid XML format"
%>
  <% exit sub %>
<% end if %>
<% set xml_string =
xml.selectSingleNodeTypedValue("eventlist") %>
```

Sample XML:

```
<eventlist>
  <event id="1" >
  <title>event 1</title>
  <description>This is a description.</description>
  </event>
</eventlist>
```

Output:

1 event 1 This is a description.

STATUS

The status object is primarily used to access the status numbers that represent the workflow states a document can potentially exist in.

↗ getStatusText(status_id)

Given an ID, getStatusText returns the text descrption of that status ID (which is the same as the state file label).

RESPONSE

The ASP response object.

REQUEST

The ASP request object.

EXAMPLES

Listing out all files of a particular template.

```
<%
  set filter = asset.createFilter()
  filter.add "_cmsTemplateId", "4,10"
  set list = asset.getFilterList(filter)
  do while list.nextEntry()
    response.write list.item("_cmsLabel") & "<br>"
  loop
%>
```

Listing out all files of a particular template and date range.

```
<%
  set filter = asset.createFilter()
  filter.add "_cmsTemplate", "", "Event"
  filter.add "startDate", "date_range", (Now()-7) &
vbCRLF & (Now()+14)
  set list = asset.getFilterList(filter)
  do while list.nextEntry() %>
    response.write list.item("_cmsLabel") & "<br>"
  loop
```

```
%>
```

Populating a select tag using contents of a file (panel).

```
<%
  set fields = asset.getContent("/Lists/types.txt")
  set list = fields.createList("item")
  do while list.nextEntry()
    response.write "<option value=""" &
list.item("item") & """ "
    if content.item("type") = list.item("item")
then
      response.write "SELECTED"
    end if
    response.write ">" & list.item("item") &
"</option>"
  loop
%>
```

Populating a select tag using files

```
<%
   set list = asset.getFileList("/Lists/")
   do while list.nextEntry()
     response.write "<option value=""" &
list.item("item") & """ "
     if content.item("type") = list.item("item")
then
       response.write "SELECTED"
     end if
     response.write ">" & list.item("item") &
"</option>"
   loop
%>
```

Displaying a panel interface.

```
<%
  set list = content.createList("item")
  do while list.nextPanel()
    response.write "<option value=""" &
list.item("item") & """ "
```

```
    if content.item("type") = list.item("item")
then
      response.write "SELECTED"
    end if
    response.write ">" & list.item("item") &
"</option>"
  loop
%>
```

List of files with nested panel

```
<%
   set list = asset.getList("/Lists/")
   do while list.nextEntry()
     set innerlist = list.createList("item")
     do while innerlist.nextEntry()
       response.write innerlist.item("item")
     loop
   loop
%>
```

OTHER META VARIABLES

Below is listing of cms template files and the additional Meta variables that are relevant to that file.

↗ post_input.asp

- _cmsVersionComment - setting creates a comment associated with the version just about to be created after a save.

↗ filename.asp, url.asp, assetfilename.asp, asseturl.asp

☐ _cmsRemotePath - the full path to where the file will be published with all the filters, etc already performed should be the same as _cmsPublishUrl and _cmsPublishPath... includes folder, filename and extension

- _cmsRemoteFolder - just the remote folder
- _cmsRemoteFilename - just the remote filename with extension
- _cmsRemoteLabel - the remote filename without extension
- _cmsRemoteExtension - the remote filename's extension

- _cmsLocalPath - same as asset.getAbsoluteName
- _cmsLocalFolder - should be same as _cmsFolder
- _cmsLocalFilename - same as _cmsLabel but with extension and filters performed
- _cmsLocalLabel - same as _cmsLabel but with filters performed
- _cmsLocalExtension - the local filename's extension (usually "")
- _cmsPropertiesFilename - filename publishing info from the publishing properties page
- _cmsPropertiesFolder - folder publishing info from the publishing properties page
- _cmsFtpRoot - the ftp root prepended to the path to create the final filename for FTP - prepended only final transfer of file
- _cmsRemoteHostname - the hostname in the FTP config
- _cmsRemoteUsername - the username in the FTP config
- _cmsRelativeFolder - the path between where the current asset is and where it found publishing information.

↗ email_import.asp

- _cmsEmailRecipient
- _cmsEmailRecipientName
- _cmsEmailRecipientType
- _cmsEmailAttachment
- _cmsEmailAttachmentFilename
- _cmsEmailSender
- _cmsEmailSize
- _cmsEmailTimeSent
- _cmsEmailTimeReceived
- _cmsEmailSubject
- _cmsEmailBody

↗ odbc_import.asp

- _cmsOdbcResults - contains the results the SQL statement execution on a remove database

- _cmsOdbcLabel - the label given to the odbcImport configuration
- _cmsOdbcConnectionString - the connection string used for the import.
- _cmsOdbcSqlStatement - the SQL statement sent to generate the results from the remote database
- _cmsOdbcRecords - the number of records (rows) returned by the SQL command
- _cmsOdbcFields - the number of fields (columns) returned by the SQL command
- _cmsOdbcResults - contains a list object that has the values returned by the SQL command
- _cmsLabel - the label of the asset created during the import process (deleted if _cmStop set)
- _cmsId - the ID of the asset created during the import process (deleted if _cmStop set)

ASSET PARAMETERS

↗ sort_order, _cmsSort, sort_by

Specifies how content should be sorted.

```
  <% asset.setParam "sort_by", "_cmsLabel ASC" %>
        <% asset.setParam "exclude", "index" &
vbCRLF & "_menu" %>
     <% set list = asset.getFileList("/Articles/")
%>
     <% do while list.nextEntry() %>
  <%= list.item("_cmsLabel") %>
```

↗ limit, _cmsList

Only return the specified number of results.

```
        <% asset.setParam "sort_by", "_cmsLabel
ASC" %>
     <% asset.setParam "limit", 10 %>
     <% asset.setParam "exclude", "index" & vbCRLF
& "_menu" %>
     <% set list = asset.getFileList("/Articles/")
%>
```

```
    <% do while list.nextEntry() %>
  <%= list.item("_cmsLabel") %>
```

↗ filter_status

Specifies what status to filter on when returning files/folder. If more than one file exists with the specified status the most recent file is returned.

```
    <% asset.setParam "filter_status", "Live" %>
    <% set fields =
asset.getFile("/Images/Cover/") %>
Current image: <img src="<%= fields.item("img_url")
%>">
```

↗ is_recursive

Specifies that all folders within the specified folder are also searched.

```
      <% asset.setParam "is_recursive", 1 %>
      <% asset.setParam "fieldnames",
"_cmsTemplateName" %>
      <% Set list = asset.getList("/Site/") %>
      <% Do While list.nextEntry() %>
      <%= list.item("_cmsTemplateName") %>
      <% Loop %>
```

↗ exclude

Specifies if you want to exclude certain folders or files from the list.

```
      <% asset.setParam "sort_by", "_cmsLabel ASC"
%>
      <% asset.setParam "limit", 10 %>
      <% asset.setParam "exclude", "index" & vbCRLF
& "_menu"%>
      <% set list = asset.getFileList("/Articles/")
%>
      <% do while list.nextEntry() %>
      <%= list.item("_cmsLabel") %>
```

↗ args

Specifies a URL parameter in a name/value pair that is appended to a URL. If more than one URL parameter exists, each parameter is separated by an ampersand (&).

```
    <% asset.setParam "args", "_the_current_date="
& curDate %>
```

```
    <% strLink =

asset.getLink(content.item("sysvar_calendar_folder"
)&"Daily") %>
    <a href="<%= strLink %>" title="Click for more
    info"><%=Day(curDate)%></a>
```

↗ fieldnames

Queries specific assets's fieldnames.

```
     <% asset.setParam "is_recursive", 1 %>
     <% asset.setParam "fieldnames",
"_cmsTemplateName" & vbCRLF & "_cmsPath" %>
     <% Set list = asset.getList("/Site/") %>
     <% Do While list.nextEntry() %>
     <%   Response.Write list.item("_cmsPath") &
"(" & list.item("_cmsTemplateName") & ")" %>
     <% Loop %>
```

↗ layout

Specifies a different layout to preview or output the content. By default, the CMS uses 'output.asp' when publishing and 'preview.asp' when previewing the content.

```
     <% asset.setParam "layout",
"alternate_output.asp"
     <% asset.show("/Site/Home Page/index") %>
```

↗ link_command

Specifies the name of the workflow command to execute when you create a link using the asset.getLink function.

```
     <% asset.setParam "link_type", "workflow" %>
     <% asset.setParam "link_command", "approve to
stage" %>
     <a href="<%=
asset.getLink(content.item("_cmsId"))
%>">Approve</a>
```

↗ link_type

Specifies the type of link generated by function asset.getLink. Some are used to create links to perform CMS-related tasks. Often preview pages are used to create specialized interfaces that are not published to any web site.

Within these interfaces links to create, edit, approve, and reject assets can be set up using getLink and several parameters specifications:

- **create**: Creates a new document
- **edit**: Edits a document
- **workflow**: Executes a workflow command
- **delete**: Deletes the document
- **clone**: Clones the document
- **branch**: Branches the document
- **folder**: Shows the folder listing
- **include**: Includes a document (ex. JS file)
- **binary**: Link to the binary field of a template file (similar to /cpt_downview).

Create Document Link

```
<% asset.setParam "link_type", "create" %>
<% asset.setParam "model_id", "567" %>
<a href="<%= asset.getLink("/products/")) %>">New
Product</a>
```

Approve Link

```
<% asset.setParam "link_type", "workflow" %>
<% asset.setParam "link_command", "approve to
stage" %>
<a href="<%= asset.getLink(content.item("_cmsId"))
%>">Approve</a>
```

Reject Link

```
<% asset.setParam "link_type", "workflow" %>
<% asset.setParam "link_command", "reject" %>
<a href="<%= asset.getLink(content.item("_cmsId"))
%>">Reject</a>
```

Include Link

```
<% asset.setParam "link_type", "include" %>
   <script language="JavaScript"
   src="<%=asset.getLink("802")%>"></script>
```

↗ model_id

Specifies the id of the model used to create a new asset when using the function asset.getLink to generate a link for this purpose.

```
  <% asset.setParam "link_type", "create" %>
  <% asset.setParam "model_id", "567" %>
  <a href="<%= asset.getLink("/products/")) %>">New
Product</a>
```

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

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

www.crownpeak.com