ViewFramework for CommonSpot

Goals

The goal of this simple framework/toolkit is to give developers the ability to create “views” when developing a CommonSpot site.

Thinking in terms of MVC architecture, ViewFramework provides a clean implementation of the “view” pattern by 100% decoupling display code from CommonSpot. It lets a developer create a custom element and output the data without writing anything but the display code.

Basics

XML configuration data tells CommonSpot what “Page Types” are available. These Page Types are the names of custom elements whose value is chosen in a meta-data form select list.

Each custom element has a render handler, conceptually the “view”, which asks ViewFramework for its data by calling:

```
Application.ViewFramework.getPageEvent(
        elementInfo = attributes.elementInfo,
        elementName = attrubutes.elementName
        )
```

ViewFramework then returns the PageEvent object populated with the correct data.

Functionality Overview

- Encapsulates several CommonSpot request scoped variables and sets them into a “PageEvent object”.

- Supplies custom element and meta-data select lists with “dynamic expression” variables to populate with configuration data.

- Uses rh_obj.cfc (renamed and re-factored to RenderHandlerService.cfc) to get a pages custom element data.

- Provides an easy solution to render HTML headers and footers.

- Supplies a CSS ID selector for a 1st level sub-site AND OR any particular sub-site if defined in configuration XML files.

- Has a DataService “sub application/toolkit” (configurable On or Off) that decorates the PageEvent with query data for a page.
  - The DataService has 3 types of caching (configurable)
    - HARD variables scoped cache – *default*
    - Soft java cache – set it and forget it
    - XML file cache of all custom queries – always runs
  - The DataService can also rebuild the cache from the stored XML cache files if any data sources, remote db’s for example, are unavailable.

Request Life Cycle Explained

Here is a birds eye view of the life cycle that renders a page:

1. template-basepage.cfm calls an “invoke-dynamic-cfml” module which then calls **Application.ViewFramework.buildPageEvent()**. This is where all the action happens.
    a. Values are set from the request scope to the PageEvent object which persists data for a request.
    b. Some logic sets other variables from XML configuration files into the PageEvent object.
    c. Decorates the PageEvent object with the correct queries from the **DataService.decoratePageEvent()** method.
2. template-basepage.cfm then sets a CSS ID selector in a wrapper DIV by calling **Application.ViewFramework.getPageEvent().getCSSClass().** This drives the look and feel for each sub site via a CSS cascade.
3. template-basepage.cfm then includes the viewstatecontroller.cfm file which instantiates the correct custom element based on variables set in the including parent template-basepage.cfm file.
4. Then the custom elements render handler calls **Application.ViewFramework.getPageEvent(elementInfo = attributes.elementInfo, elementName = attrubutes.elementName)** which returns the PageEvent object with the correct data to render the page.

API's

ViewFramework API

buildPageEvent()

This method is called from within a CommonSpot "invoke-dynamic-cfml" module in the template-basepage.cfm. ViewFramework is light, usually adding no more than 100 to 130 ms per request when a page is cached at the template level.

If NO Page Type render handlers use CommonSpots "invoke-dynamic-cfml" module **buildPageEvent()** should be called directly from template-basepage.cfm so it gets cached by CommonSpot at the template level.

configure()

Use this method to reload XML configuration data in to the framework without having to reload the full framework. Useful for development and first time deployments of ViewFramework.

deployFiles()

When using the ViewFramework index.cfm configuration file, an interface is provided to copy files from the displaypackages directory to the root or any sub-site. This function is useful if the developer has not access to the server and wants a controlled deployment of files.

getAvaiablePageTypes()

Used to populate a select list in the Page Type meta-data form. This is the value that determines what custom element will be used to render the page.

getConfigurationBean()

Returns ViewFrameworks Configuration Bean object.

getDataService()

Returns the DataService toolkit. Provided for developers and documentation.

getElementData()

Returns an easy to use array or structure, depending on element type, of the elements data. This method is provided so other render handlers inside the Page Type render handler, or code outside the context of the Page Type render handler can conveniently use ViewFramework.

getLogger()

Returns the frameworks instance of CFLog4J object which allows developers to keep log files within their application.

getMockPageEventOjbect()

Returns a “mock” Page Event object populated so a developer can send a request through the framework without being tied to a CommonSpot server.

getPageEvent()

Returns an object populated with the correct data to render a page. This method can be called from inside or outside the context of the Page Type’s render handler.

It is **Best Practice** to pass the render handlers **attributes** *when called from within the context of the render handler*. This sets the PageEvent’s correct custom element data.

getViewFrameworkVersion()

Returns the current version of ViewFramework.

getViewFrameworkFileContext()

Returns the root relative system file path to ViewFramework.

getViewFrameworkComponentContext()

Returns the application context to the framework.

getListOfOUs() – DEPRECIATED

Used to populate the select list in a custom element.

getCampusList() – DEPRECIATED

Used to populate the select list in a custom element.

getStateList() – DEPRECIATED

Used to populate the select list in a custom element.

getSubSiteList() – DEPRECIATED

Used to populate the select list in a custom element.

PageEvent API

getCachKey()

Returns the pages key used to retrieve data from the DataService cache.

getCacheStatus()

Returns whether the DataService returned cached data or had to add data to the cache.

getChildSubsites()

Returns comma delimitated list of the pages current and all children sub-site ID’s.

getCSSClass()

Returns the top-level sub-sites CSS ID selector value.

getCustomElementData()

Returns a structure or an array of structures containing the current pages custom element data.

getCustomElementName()

Returns the name of the Page Type’s custom element.

getFooterEmail()

Returns the administrator email address of the current sub-site.

getImageDirectory()

Returns a root relative path to the sites top level image directory.

getPageCreateDate()

Returns the date the page was created.

getPageID()

Returns CommonSpots PageID for the page.

getPageLastModifyDate()

Returns the date the page was last modified using the “long” mask.

getPageName()

Returns a string of the pages name.

getPageQueryData()

Returns a structure of queries for the page.

getParentSiteDisplayName()

Returns the name of the pages most top level sub-site.

getParentSite() – DEPRECIATED

Returns the name of the pages most top level sub-site. Use **getParentSiteDisplayName()** instead.

getSubSiteDefinition()

Returns a structure of a pages definition as described in MXL configuration.

getRenderHandlerDirectory()

Returns the directory that ViewFramework expects render handlers to be in, usually the default top level render handlers directory for CommonSpot.

getSiteContext()

Returns the file context that the site is running in.

getSiteName()

Returns the name of the CommonSpot site.

getSiteURL()

Returns the URL of the CommonSpot site.

getSubSiteName()

Returns the display name of the pages current sub-site.

getSubSiteCSSClass()

Returns the CSS ID selector for a sub-site if configured in XML.

getSubSiteDefinitions() – DEPRECIATED

Returns a structure containing XML file data flattened as a structure.

getUDFs()

Returns an object with useful UDF’s for display.

Configuration Bean API

The below methods are all of the configuration settings. All values are set in the configuration.ini file.

getAdministratorEmail ()
Returns the sites Administrator email defined in the configuration.ini file.

getCacheMechanism ()
Returns wether DataService is using a “hard” variables scoped cache or a “soft” java managed cache.

getCacheMinutes ()
Returns how many minutes the DataService will wait to reap the cache.

getCacheReapTime ()
Returns how many minutes the DataService will wait to reap the cache.

getCommonSpotDataSource ()
Returns the CommonSpot datasource.

getDataGatewayRuntimeMode ()
Indicates if the DataService is in development mode. A poorly name method.

getDataServiceVersionNumber ()
Returns DataService’s version number.

getDeveloperEmail ()
Returns the developers email address.

getDefaultCSSClass ()
Returns the CSS class used when no other has been defined. Usually used for the root level CSS ID selector.

getFleetMetaDataFieldName ()
Returns the name of the meta-data form field that used by FLEET.

getFLEETDataSource ()
Returns FLEETs datasource.

getFrameworkDeploymentDirectoryList ()
Returns a comma delimitated list of directories that ViewFrameworks deployFiles() method uses.

getFrameworkDeploymentSourceDirectory ()
Returns the root relative path to the deploymentpackages directory.

getFrameworkDisplayTemplateDirectory ()
DEPRECIATED. Returns the name of a directory.

getFrameworkClassContext()
Returns the root relative component context to ViewFramework.

getFrameworkFileContext ()
Returns the root relative file context to ViewFramework.

getFrameworkRederHandlerDirectory ()
Returns the render handler directory. Handy for developers that want a "staging directory" for development.

getFrameworkServerContext ()
Returns the fully qualified UNC path to ViewFrameworks directory.

getFrameworkIncludesDirectory ()
Returns the includes directory used by ViewFramework.

getFleetControlID()
Returns the FLEET control ID..

getImageDirectory ()
Returns the root relative path to the top level image directory.

getImplementDataService ()
Returns whether the DataService is being used by ViewFramework.

getInstance ()
Returns a structure of the ConfigurationBean instance data.

getMailPassword ()
The mail servers password.

getMailServer ()
Returns the address of the Mail server.

getMailUsername ()
Mail servers username.

getProductionServerIPList ()
Returns a list of production servers IP address's. If the code base is running under one of these IP address, the index.cfm file in ViewFramework will not be allowed to execute.

getVersion ()
Returns ViewFrameworks current version. Poorly named method.

getWebSiteContext ()
Returns the sites context, root or not root relative.

getWebmasterEmail ()
Returns the sites web master email defined in the configuration INI file.

getWebSiteServerContext ()
Returns the fully qualified UNC path to web sites directory.

TODOs:

1. The API for the ViewFramework is TOTALLY polluted with NIST centric stuff. Going to:
   a. Add a new method called **getXMLData(name=foo)** which will generalize the API and let developers add any and as many lists or structures as they want or need.
2. Define an XML schema that will let the framework read a directory of XML configuration files to generically add lists, or structures of data to the specific implementation of the framework.
3. Figure out how to make the **Viewframework.dataservice.command.CommandTemplate.decoratePageEvent()** method be generic. This stumps me. Maybe a rule engine based on some XML configuration data?
4. Figure out how to implement Hibernate so ALL the database stuff can be configured via XML.