ASSET MANAGER BUILDING BLOCK 9.5 TECHNICAL GUIDE

Asset Manager Building Block 9.5 Technical Guide

February 22, 2011

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

Chapter 1: Asset Manager building block basics

The Asset Manager Building Block 9.5 is used register asset types and support type extension through custom type extension handlers.

The Asset Manager building block provides following features:

- the ability to register event handlers around pre- and post-CRUD events on any asset.
- the ability to associate any number of extended properties with an asset type.
- framework to register and launch the editor UI for a given asset type

The Asset Manager building block includes these components:

Asset Manager building block UI The Asset Manager building block provides multiple Flex® UI components which can be used and stitched together to create the UI for your solution. The Asset Manager building block UI uses the Assembler services for interacting with the underlying repository, Adobe® LiveCycle® Content Services 9. This component provides the capability to execute pre-configured actions, such as Create, Edit, and Delete on assets. The various assets and their corresponding actions can be configured declaratively and the same will show up on the UI. It also provides advanced search capabilities on the metadata associated with the assets. The Asset Manager building block UI can be configured to launch other user interfaces, on click of respective action buttons.

The editor UIs in the Correspondence Management Solution Accelerator 9.5 solution template are configured to work with relevant Asset Manager building block actions by default. The Asset Manager building block UI component provides a configurable search pod and search result UI based on asset definitions and configurations.

Security The overall application security is managed and intercepted via a Spring security-based module that connects to Adobe User Management.

Building block structure

The Asset Manager building block components are installed in the following folder: c:\Adobe\Adobe LiveCycle ES2\sa_resources\SA_SDK_9.5\BuildingBlocks\assetmanager\.

Component location	Description
\client\flex\adobe-amg-flex-presentation_rb.swc	Resource bundles SWC for the Asset Manager building block. This SWC file contains bundles for all supported locales and is used as a compile-time dependency for adobe-amg-flex-presentation.swc
\client\flex\adobe-amg-flex-domain-model.swc	Domain library for the Asset Manager building block. It contains various domain classes encapsulating business data and behavior, and can be leveraged to build Flex views with less effort.
\client\flex\adobe-amg-flex-presentation.swc	Presentation SWC for the Asset Manager building block. This SWC contains the presentation logic and the Flex views built atop the Asset Manager Flex domain model.
\client\flex\adobe-amg-flex-presentation_styles.swc	Styles SWC for the Asset Manager building block. This SWC file contains the style sheets and various assets (such as animations, icons, and skins) that are used by the Asset Manager building block.

Component location	Description
\client\flex\src\adobe-amg-flex-presentation_rb-src.zip	Zip file that contains the project for resource bundles of Asset Manager building block Flex components.
\client\flex\src\adobe-amg-flex-presentation_styles-src.zip	Zip file that contains the project for the style sheets and various assets (such as animations, icons, and skins) that are used by Asset Manager building block.
\client\flex\src\adobe-amg-flex-presentation-src.zip	Zip file that contains the project for presentation logic and the Flex views that encompass various UI components such as the Search Pod and Search Reults Viewer in the Asset Manager building block.
\client\java\adobe-amg-asset-manager-api	Client jar for the Asset Manager building block.
\services\adobe-amg-assembler.jar	For internal use only.
\services\adobe-amg-asset-manager-impl.jar	For internal use only.
\services\adobe-amg-cms-client-api	For internal use only.
\services\adobe-amg-cms-client-impl	For internal use only.

In addition, a file named cs-provider.amp is copied to the c:\Adobe\Adobe LiveCycle ES2\deploy folder. This is the Content Services provider AMP for the Asset Manager building block interfacing with Content Services for internal use only.

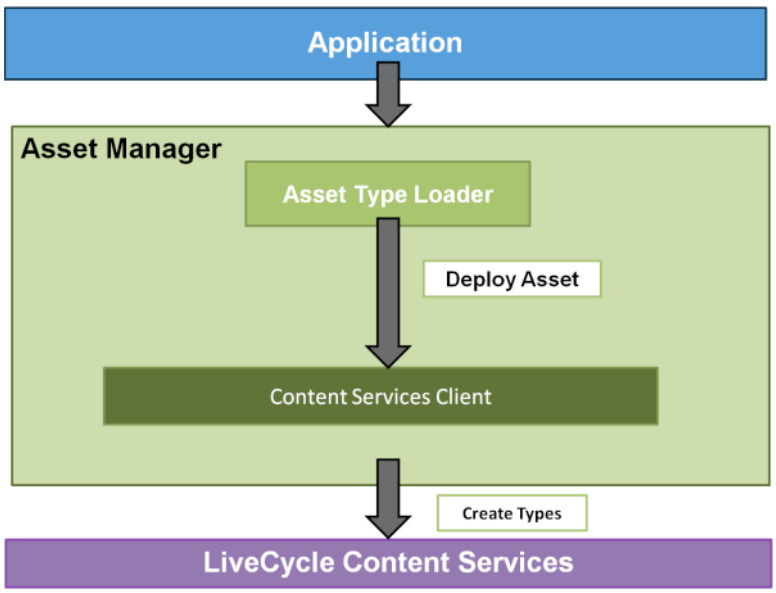
Chapter 2: How the Asset Manager building block works

The Asset Manager building block is a specialized repository that allows type-based management of assets. It is a generic framework that eases application creation, as most things are taken care of by the Asset Manager building block. This allows you to concentrate effort on application-specific components. As the first step, you must register all of the assets used by your correspondence management application with the Asset Manager building block. Based on the information collated in the registration step, the Asset Manager building block provides the following features and functionality to your correspondence management application:

- Provides the ability to register event handlers around pre- and post- CRUD events on any asset.
- Provides the ability to associate any number of extended properties with an asset type. The Asset Manager building block not only saves, retrieves, and deletes these extended properties (along with asset CRUD), but also allows searching the assets based on its values. This is a very powerful feature as it can be achieved at runtime without writing any custom code.

Asset Manager building block components

The following diagram describes the high-level architecture of the Asset Manager building block.



Asset Manager building block architecture

Asset Type Loader This component is responsible for registering asset types and asset handlers (use to extend asset CRUD and deploy) with the Asset Manager building block. This component uses a native provider to connect to Content Services to deploy the various asset types in Content Services. Type registration is important because it provides metadata that Content Services stores and uses to understand the asset type.

Chapter 3: Understanding the Asset Manager building block

Use the following steps to configure and get started with the Asset Manager building block:

- Asset Repository Registration
- Asset Type Registration
- Custom Asset Handler Registration
- Asset Manager CRUD and Search Interface
- Asset Manager Assembler
- Overriding CredentialProvider
- Asset Manager Flex Components

For additional information about the Asset Manager building block services, see these resources:

- For information about the Flex APIs, see the *ActionScript® 3.0 Reference for the Adobe® Flash® Platform*.
- For information about the Java® APIs, see the *Solution Accelerators API Reference*.

Asset Repository Registration

Your application must register the Content Services repository that will be used to manage the assets. The repository must be defined in an XML file similar to this one:

```
<RepositoryList>
    <RepositoryInfo>
        <!--  This element contains URL endpoint for Asset Manager Repository -->

<repositoryUrl>http://icc_server:8080/contentspace/wcservice/adobe/cms/webscript</repository
Url>
        <!--  A unique identifier for the repository (to be specified) -->
        <dataSourceId>icc_repo</dataSourceId>
        <!--  Path of the root folder in repository inside which all asset data will go-->
        <rootPath>/DataStore</rootPath>
    </RepositoryInfo>
</RepositoryList>
```

A Content Services repository is uniquely identified by either its dataSourceId or repositoryUrl.

The following elements must be provided by your application:

repositoryUrl URL endpoint used to connect to Content Services, for example, http://<*server*>:<*port*>/contentspace/wcservice/adobe/cms/webscript.

dataSourceId A unique identifier for the asset repository. This can be any string.

rootPath A root folder under which all assets will be stored. Defaults to /DataStore if not specified.

The correspondence management application specifies the location of this configuration file through Spring's `org.springframework.beans.factory.config.PropertyPlaceholderConfigurer` bean. This would normally be done through the module.properties file associated with the application.

Asset Type Registration

Before your application can perform CRUD operations on assets, you must use the AssetTypeLoader Spring bean provided by Asset Manager building block to register a type definition for all assets.

This bean has a String property, `typeDeploymentMode`, that controls the deployment behavior. The possible values of this property are:

- unchanged: No changes will be allowed on existing entities. New entities can be added.
- update: Only incremental updates will be allowed to existing entities. New entities can be added. This is the default behavior.
- recreate: Existing entities will be dropped if present with the same name. New entities will be added.
- delete: Existing types will be deleted.

Registration of asset types

Use the AssetTypeLoader bean to register asset definitions with the repository. Your application must ensure that type registration is done only when the asset repository is up and running, otherwise the registration will fail when AssetManager tries to deploy the types to the asset repository.

These properties must be specified by applications using this bean:

dataSourceId This is the identifier of the repository where asset types will be registered and later managed through CRUD. This is the identifier specified in the repository configuration XML discussed previously.

templates This is a map containing the assets that need to be registered with the Asset Manager building block. Every element of this property is a map entry of the form `<Asset_Name, Fully_Qualified_Asset_Class_Name>`, where `Asset_Name` is a unique name assigned to every asset and `Fully_Qualified_Asset_Class_Name` is a fully qualified class name for the asset class. When registering an asset class, all referred asset classes must also be specified in the templates. The Asset Manager building block will not be able to register an asset class whose referred class definitions are not present.

Note: *"Folder" and "Document" are reserved words, and cannot be used for Asset_Name.*

A request for an asset that is already registered is silently handled by the Asset Manager building block and mapped to the already existing asset.

Asset Class Definition

Keep these important considerations in mind when defining an Asset Class.

Java bean

Every asset class must be defined as a Java bean. There should be getters and setters corresponding to every field, and they must follow the Java bean semantics. These are some constraints while defining the Java bean properties:

ID field An ID Property must be present in every asset class. The name of the property can be `id` or `ID` and it must be of type `java.lang.String`. This is required to uniquely identify an asset object.

Transient field Any field that is declared as transient will be ignored in the asset type definition used by the Asset Manager building block.

Content field There can be no more than one content field in an asset object. This field will hold any blob or content stream associated with the asset. The permissible types for this field are `byte[]`, `Byte[]` and `java.io.InputStream`.

Collection field When a field must hold a collection of primitive types or a collection of other asset class types, then it can be declared either as a `java.util.List`, `java.util.Collection`, `java.util.Set` or as an array, for example `int[]`, `Form[]`.

Extended properties In order to pass extended properties in the VO, there should be a property named `extendedProperties` and its type should be `java.uti.Map<String,Object>`. This map will contain entries of type `<extended_prop_name, extended_prop_value>`.

Mandatory field Any property that needs to be made mandatory should be annotated with an `@Mandatory` annotation.

Custom Asset Handler Registration

The Asset Manager building block allows registration of handlers for performing custom actions on pre- and post-CRUD (create, read, update, or delete) and Deploy (type registration) events on an asset type. These handlers are pieces of custom code that are automatically invoked when the event for which they are registered happens. The Asset Manager building block supports five types of handlers:

- CREATE: Handlers to be invoked before and after creation of an asset type
- READ: Handlers to be invoked before and after retrieval of an asset type
- UPDATE: Handlers to be invoked before and after an update of an asset type
- DELETE: Handlers to be invoked before and after deletion of an asset type
- DEPLOY: Handlers to be invoked before and after deployment (registration of an asset type). This handler is used to extend the type definition of an asset class by registering extended properties with an asset.

All handlers in the Asset Manager building block are Java classes registered as Spring beans. All handlers must implement the `preProcess()` and `postProcess()` methods of `com.adobe.livecycle.assetmanager.TemplateHandler` interface that is invoked before and after the event for which they are registered.

Before the asset type deployment, the `preProcess()` method of the extension handler will be invoked. The implementing class will get a List of `EntityPropertyInfo` in the preProcessMap identified by `EXTENSION_MAP_KEY`. Extension handlers can add new properties as `EntityPropertyInfo` objects in the same list and they will be used as extended or extra properties while deploying the corresponding asset class definition to the asset repository.

Defining handler information in a TemplateHandlerInfo bean

A TemplateHandlerInfo bean defines the following:

- **templateType:** Name of the asset class for which this handler is to be registered. This is the same name with which the asset class was registered in AssetTypeLoader bean. A value of '*' would mean that the handler(s) has to be registered for all assets.
- **targetOperation:** Name of the operation on the asset for which this handler is registered. Possible values are DEPLOY, CREATE, READ, UPDATE and DELETE. A value of '*' would mean that the handler(s) need to be registered for all operations.

- **templateHandlers:** A list of TemplateHandler objects that needs to be registered for the templateType and targetOperation. The handlers will be invoked in the same order in which they are registered.

Registering handlers with the TemplateHandlerRegistry

All the TemplateHandlerInfo Spring beans are required to be registered with a TemplateHandlerRegistry (Spring bean).

Overriding CredentialProvider

Your application must override the l`c.content.umCredentialProvider` bean to inject the application's CredentialProvider into the Asset Manager building block.

It should also override the lc.content.adminCredentialProvider bean and instead use a credential provider that provides access to a user credential with administrator level privileges.

By default the lc.content.adminCredentialProvider bean uses `PreConfiguredCredentialProvider` with username and password values picked up through Spring's `org.springframework.beans.factory.config.PropertyPlaceholderConfigurer` bean defined by the application to inject properties to Spring beans. This would normally be done through the module.properties file associated with the application.

Asset Manager Flex Client Components

The Asset Manager building block provides a client-side API and widgets to enable your correspondence management application to connect to the Asset Manager building block back end and search and view the assets. To enable this communication, your application must fulfill the following requirements:

- Expose server-side Data Services DataService endpoints on the assets through the AssetManagerAssembler.
- Create an Asset Definition System data dictionary corresponding to each search-enabled asset, and save that in the Asset Manager building block.

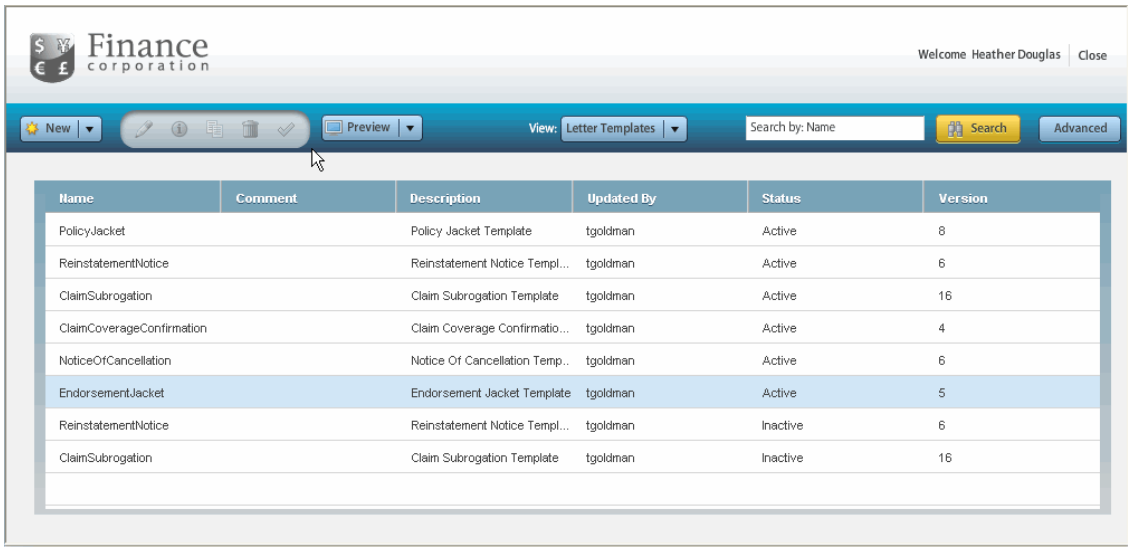
After that is done, you can use the Asset Manager building block widgets in your Flex application to quickly enable search functionality and listing of search results. The search and listing view can be further customized with the help of these system data dictionaries.

The combination of Data Dictionary integration with the Asset Manager building block adds the following important features:

- Creates a UI with a powerful search capability for all assets. Search UI is fully configurable for every asset type. Your application can define which fields should be shown on the Search UI, how they should be ordered and rendered, and so on.
- The Asset Manager building block also provides Search Results Listing View, where the search results for assets are displayed. The listing page is fully configurable for every asset type. It allows you to configure the properties to be displayed for an asset type in the listing view, their order, renderer, and so on.
- Your application can define the actions that can be taken on the assets displayed in the Search Results Listing. You can also specify custom action handlers that can be configured for every asset type.

Chapter 4: Implementing the user stories

The Manage Assets UI uses the Asset Manager Assembler services for interacting with the underlying repository (Content Services). It lists existing assets in the Search Results Viewer and provides the capability to execute pre-configured actions such as Create, Edit, and Delete on assets. It also provides advanced search capabilities on the properties of the assets.



Manage Assets screen

The toolbar displays actions that can be taken on a given asset. Actions are enabled only when the user who is logged in has the appropriate permissions. The toolbar dispatches an event when the user selects an action. AssetHandler listens to the event and carries out the action. The Manage Assets UI can register its own custom AssetHandler.

The Search Results Viewer displays the search results when the user performs a basic or advanced search. You can configure which columns are shown in the search results.

The Search Pod is displayed when the user performs an Advanced search. You can customize the search functionality.

The Manage Assets UI can also be configured to launch other user interfaces when a user clicks an action button.

The Correspondence Management Solution Accelerator includes a sample implementation of the Manage Assets UI, which uses all four building blocks. Before you do any customization work, you must set up your development environment. For information, see Setting up your development environment in the *Correspondence Management Solution Accelerator Solution Guide*.

Configuring the Search Results Viewer

Preferences related to the Manage Assets UI can be configured with the help of the Asset Type Definition. The Asset Type Definition is essentially a System data dictionary representing the asset type.

By default, the Manage Assets UI that is part of the Correspondence Management Solution Template delivers such preconfigured data dictionaries for correspondence management-specific asset types as FML files in the WEB-INF/assetDefinitions folder in the correspondence management EAR file. You can change these FML files, repackage and redeploy the EAR file, and then bootstrap the Correspondence Management solution to deploy the Asset Type Definitions.

For example, to show a Letter Post Process Name in the Search Results Viewer, you would open the Letter.fmal and add the following entry under the model/entity tag:

```
<property name="postProcess" type="string">
   <!—postProcess is the name of property on which is to be shown in serach results viewer -->
    <annotation name="DDS">
        <item name="displayName">Post Process </item>
           <!--displayName corresponds to the String that would be shown as column header in
SRV -->
        <item name="visible">true</item>
          <!—visible set to true  is the setting that would cause the post processto be shown
in SRV -->
        <item name="searchable">false</item>
        <item name="columnOrder">2</item>
          <!—columnOrder is the relative order(Relative to other properties in same fml file)
of PostProcess column in SRV-->
     </annotation>
</property>
```

Save and close Letter.fml and repackage the ear, deploy ear on server and run booststrap. To localize Post Process display name such as the one shown in the example above, you can replace the name with a resource key such as `loc.letter.prop.postProcess`, and specify it's localized value in DataDictionaryMessages resource bundle located in the Web-Inf/assetDefinitions/locale folder.

To see a list of other properties that can be changed, see "DataDictionary-level extended properties" on page 9.You can also change the configurations by modifying the asset definition System data dictionary directly from the Manage Assets UI.

For background information on this user story, see Story: Customizing the Manage Assets UI.

DataDictionary-level extended properties

The Asset Type Definition System data dictionary contains extendedProperties map of type `Map<String,String>` at DataDictionary Level and DataDictionaryElement level.The AssetManager UI uses following extended properties:

Property Name	Description

assetActions	XML defining the actions associated with this asset type. These actions are displayed in the toolbar in the Manage Assets UI. XSD:
	
	`<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`
	`<xs:element name="actions">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element ref="action" maxOccurs="unbounded"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`
	`<xs:element name="action">`
	`<xs:complexType>`
	`<!-- name of the action used internally. Should be alpha-numeric -->`
	`<xs:attribute name="name" type="xs:string" use="required"/>`
	`<!-- 16x16 Icon url to show when the action is enabled-->`
	`<xs:attribute name="enabledIcon" type="xs:string" use="required"/>`
	`<!-- 16x16 Icon url to show when the action is disabled-->`
	`<xs:attribute name="disabledIcon" type="xs:string" use="required"/>`
	`<!-- Tooltip for the action icon-->`
	`<xs:attribute name="toolTip" type="xs:string" use="required"/>`
	`<!-- label for the action.-->`
	`<xs:attribute name="label" type="xs:string" use="required"/>`
	`<!-- Is this action enabled by default requirement need for user to select an Item for e.g. Create icon-->`
	`<xs:attribute name="defaultEnabled" type="xs:boolean"/>`
	`</xs:complexType>`
	`</xs:element>`
	`</xs:schema>`

assetActionsRenderer	Fully qualified class name for Custom ContolButtonBar/Menu bar component to display Asset Actions. Used when you want your application to override the default actions bar.
iconURL	16x16 icon image URL of the asset represented by this data dictionary. The prefix used by Flex client component to check whether the user has the permission to take an action on selected asset.
permissionPrefix	The prefix used by Flex client component to check whether the user has the permission to take an action on the selected asset
systemPermissionPrefix	The prefix used by Flex client component to check whether the user has the permission to take an action on the System data dictionary. This is only relevant for Asset Definition data dictionary, which defines DataDictionary Asset.
assetListingOrder	Order of the asset in the View dropdown of AssetManager UI.
viewAssetsTitle	Title (display text) of the asset in the View/Create dropdown of AssetManager UI:

Data dictionary element-level extended properties

These properties correspond are defined for each asset property.

Property	Description
basicSearchEnabled	Specifies whether the property represented by this data dictionary element (DDE) is enabled for basic search. There can only be one such property for an Asset.
searchable	Specifies whether the property represented by this DDE is searchable through the Asset Manager Advanced Search pod.
searchRenderer	Name of the custom search renderer for searching this property on AssetManager.
searchRendererOrder	The order of search renderer of this property on the Advanced Search pod. This is not an exact order but a relative preference.
visible	Whether this property represented by DDE should be displayed in the Search Results Viewer.

Property	Description
columnOrder	The order of the column in the Search Results Viewer. This is not an exact order, but a relative preference. A value of `0` means it has the highest preference to appear as the first column.
customItemRenderer	Qualified class name of the custom renderer for this property in the Search Results Viewer.
assetPropertyType	Specifies whether the property represented by this DDE is a core property of the asset, or an extended property.
extendedPropertyPath	If the property represented by this DDE is an extended property, this is the path of that property in Flex VO.
searchPath	Search path for the property represented by this DDE. This search path is used while searching assets via Data Services. If searchPath is not provided, then *DDElement.path* is used as searchPath.
minValue	The minimum value for the property represented by this DDE. Used in the numeric stepper UI to set minimum value.
maxValue	The maximum value for the property represented by this DDE. Used in the numeric stepper UI to set maximum value.
displayPattern	Display pattern to display data in Search Renderer.

optionList	XML defining the list of options with label and data. XSD:
	`<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`
	`<xs:element name="domain">`
	`xs:complexType>`
	`<xs:sequence>`
	`<xs:element ref="item" maxOccurs="unbounded"/>`
	`</xs:sequence>`
	`<!-- Currently, supported values of dataType are: string, number, boolean, date with default format MM-DD-YYYY-->`
	`<xs:attribute name="dataType" type="xs:string"/>`
	`</xs:complexType>`
	`</xs:element>`
	`<xs:element name="item">`
	`<xs:complexType>`
	`<!-- UI display value of the option. -->`
	`<xs:attribute name="label" type="xs:string" use="required"/>`
	`<!-- data value of the option that is sent to the server during search/result evaluation. -->`
	`<xs:attribute name="data" type="xs:string" use="required"/>`
	`</xs:complexType>`
	`</xs:element>`
	`</xs:schema>`

Customizing the Manage Assets UI

For detailed information on the components and widgets exposed by the AssetManager SWCs, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

For background information on this user story, see Story: Customizing the Manage Assets UI.

Customizing the toolbar

The com.adobe.livecycle.assetmanager.client.view.actions.ActionBar component is a ViewStack containing IActionsRenderer as children for different asset types. This component is responsible for rendering the toolbar in the Manage Assets UI.

Events thrown

assetActionClick of type com.adobe.livecycle.assetmanager.client.event.AssetActionEvent whenever user selects an action.

Properties

- **defaultRenderer:** Rendererer that is used to render the toolbar by default for all asset types. User can override the default renderer with the assetActionsRenderer extended property.
- **assetType:** Currently selected asset type (AssetTypeDescriptor) for which the toolbar is displayed.

Usage example

```
actions:ActionBar
    id="assetActionBar"
    assetType="{activeAssetType}"
    defaultRenderer="com.adobe.icc.ams.view.components.CustomAssetActionsRenderer"
    />
```

Customizing the Search Results Viewer

The com.adobe.livecycle.assetmanager.client.view.listing.SearchResultsPod component is a ViewStack containing SearchResultsContainer as children for different asset types. This component renders the search results for different asset types.

Events thrown

- assetActionClick of type com.adobe.livecycle.assetmanager.client.event.AssetActionEvent whenever user double clicks on an item.
- selectedItemsChange of type flash.events.Event when user selects items in result data grid.

Properties

- **activeAssetType:** currently selected asset type for which search results are shown.
- **result:** collection containing search results.
- **selectedItems:** array of items currently selected by user. this is readonly property.

Usage example

```
<listing:SearchResultsPod
    id="searchResults"
    height="98.5%" width="98%"
    activeAssetType="{activeAssetType}"
    result="{searchManager.lastResult}"
    selectedItemsChange="onSelectedItemsChange()"
    assetActionClick="onActionSelect(event)"
    />
```

Customizing the search pod

The com.adobe.livecycle.assetmanager.client.view.search.SearchPod component is a ViewStack containing SearchRenderers as children. This component renders the search UI for a given asset type and fires the search query on the given search provider.

Properties

- **assetQueryType:** Identifies the bac kend IQueryService implementation to be used for search queries. Constants are defined in QueryServiceFactory class.
- **searchProvider:** Instance of com.adobe.livecycle.assetmanager.client.ISearchProvider that carries out actual search over Data Services.
- **renderer:** Search renderer can be BasicSearchView or AdvancedSearchView or even a custom implementation written by the solution template. These search renderers should implement the com.adobe.livecycle.assetmanager.client.view.search.ISearchView interface.
- **activeAssetType:** Currently selected asset type for which search the pod is shown.

Usage example

```
<search:SearchPod
    activeAssetType="{activeAssetType}"
    searchProvider="{searchManager}"
    renderer="com.adobe.livecycle.assetmanager.client.view.search.AdvancedSearchView"
    width="100%"height="100%"
/>
```

Service component

The com.adobe.livecycle.assetmanager.client.service.search.IQueryService component is used by the Search Pod to retrieve the data from the back end Data Services destination. This API can be used directly with QueryServiceFactory.

Usage example

```
var queryService:IQueryService =
QueryServiceFactory.getInstance().getQueryService(assetQueryType);
var dataSerice:DataService = ServiceLocator.getInstance().getDataService(lcdsDestination);
var token:AsyncToken = queryService.fill(collection, searchFilterList, dataSerice, path);
    .............
    //once we are done release the managed collection instance from lcds managerment
queryService.releaseSearchResults(resultsListView, dataSerice);
```

Adding custom search attributes to the Advanced Search pod

The Asset Manage building block enables you to add custom search attributes to the Advanced Search pod in the Manage Assets UI.

For background information on this user story, see Story: Customizing the Manage Assets UI.

Adding searchable fields to the Advanced Search pod

To enable search on the asset attributes, which are direct properties of asset pojo, add a property element in the asset fml under the model/entity tag.The name of the property element should be same as the pojo property name. Mark this property searchable through searchable annotation.For example, to enable search based on data dictionary name used in Text (TextModule), add following entry in TextModule.fml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<model xmlns="http://ns.adobe.com/Fiber/1.0">
    <entity name="TextModule">
    .....
        <property name="dataDictionaryRefs" type="string">
            <annotation name="DDS">
                <item name="displayName">Data Dictionary</item>
                <item name="searchable">true</item>
                <item name="searchRendererOrder">10</item>
            </annotation>
        </property>
    ....
    </entity>
</model>
```

Enable full text search

You can enable full text search on asset content) through the Advanced Search pod.

To enable search on the asset content, add a property element in the asset fml under the model/entity tag.The name of the property can be anything, but it is recommended that you keep the same name that is used in Java pojo to hold asset content. Mark this property searchable through the searchable annotation. Also add another annotation `searchRenderer` with value `FullTextSearchRenderer`.For example, to enable full text search on Text (TextModule), add the following entry in TextModule.fml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<model xmlns="http://ns.adobe.com/Fiber/1.0">
    <entity name="TextModule">
    .....
        <property name="TBXXMLBytes" type="string">
            <annotation name="DDS">
                <item name="displayName">Plain Text Content</item>
                <item name="searchable">true</item>
                <item name="searchRendererOrder">10</item>
                <item name="searchRenderer">FullTextSearchRenderer</item>
            </annotation>
        </property>
    ....
    </entity>
</model>
```

Adding custom action buttons

You can use the Asset Manager building block to add new actions to the Manage Assets UI for a particular asset type. For example, you can add an action button that downloads a layout XDP in the Layout toolbar. The rest of this section describes how to implement this example scenario.

For background information on this user story, see Story: Customizing the Manage Assets UI.

Permission creation

Edit the CMSAUserRoleMapping.xml file and add the permission for Layout download. Assign this permission to appropriate role, for example, Correspondence Management Administrator.

Note: *The CMSAUserRoleMapping.xml file is not part of the Asset Manager building block. It is provided as part of the Correspondence Management solution template. The Asset Manager building block on its own does not provide automatic capabilities for creation of permissions for the actions defined in the asset types.*

```
<cmsa>
    <permissions>
        ............
        <permission name="CM Layout Download"
description="$LayoutDownloadPermissionDescription"/>
          ... ... ... .. .
    </permissions>
    <roles>
        <role name="Correspondence Management Administrator"
displayName="$CMAdminRoleDisplayName" description="$CMAdminRoleDescription">
        .........
        <permission ref="CM Layout Download" />
        .........
        </role>
    </roles>
</cmsa>
```

Note: *$LayoutDownloadPermissionDescription is a resource key for a localized string whose value has to be defined in the bootstrap resource bundle that is located parallel to CMSAUserRoleMapping.xml. For example, LayoutDownloadPermissionDescription=Correspondence Management permission to Download Layout.*

Asset FML modification

Modify the WEB-INF/assetDefinitions/Layout.fml to add a new Download action. (See the bold section in the sample below):

```
<model xmlns="http://ns.adobe.com/Fiber/1.0">
<annotation name="DDS">
.....
<item name="assetActions">
    <![CDATA[
    <?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <actions>
         <action name="Create" enabledIcon="assets/icons/LC_New_Md_N.png"
disabledIcon="assets/icons/LC_New_Md_D.png" toolTip="loc.layout.tooltip.create"
label="loc.layout.tooltip.create" defaultEnabled="true"/>
         <action name="Edit" enabledIcon="assets/icons/LC_Edit_Md_N.png"
disabledIcon="assets/icons/LC_Edit_Md_D.png" toolTip="loc.layout.tooltip.edit"
label="loc.layout.tooltip.edit"/>
         <action name="View" enabledIcon="assets/icons/LC_ViewInfo_Md_N.png"
disabledIcon="assets/icons/LC_ViewInfo_Md_D.png" toolTip="loc.layout.tooltip.view"
label="loc.layout.tooltip.view"/>
         <action name="Copy" enabledIcon="assets/icons/LC_Copy_Md_N.png"
disabledIcon="assets/icons/LC_Copy_Md_D.png" toolTip="loc.layout.tooltip.copy"
label="loc.layout.tooltip.copy"/>
         <action name="Delete" enabledIcon="assets/icons/LC_Delete_Md_N.png"
disabledIcon="assets/icons/LC_Delete_Md_D.png" toolTip="loc.layout.tooltip.delete"
label="loc.layout.tooltip.delete"/>
         <action name="Activate" enabledIcon="assets/icons/LC_Activate_Md_N.png"
disabledIcon="assets/icons/LC_Activate_Md_D.png" toolTip="loc.layout.tooltip.activate"
label="loc.layout.tooltip.activate"/>
         <action name="Download" enabledIcon="assets/icons/LC_Download_Md_N.png"
disabledIcon="assets/icons/LC_Download_Md_D.png" toolTip="loc.layout.tooltip.download"
label="loc.layout.tooltip.download"/>
    </actions>
    ]]>
</item>
..........
```

Note: ** loc.layout.tooltip.download is a resource key for a localized string whose value has to be defined in the DataDictionaryMessages resource bundle, which is located in the WEB-INF/assetDefinitions/locale folder. For example, loc.layout.tooltip.download=Download. Also, the LC_Download_Md_D.png and LC_Download_Md_N.png files should be at appropriate location in ManageTemplates project.*

Extending Asset Handler

The next step is to extend the existing handler to support the newly added action. There are two requirements:

- Logic for enabling/disabling newly added action based on currently selected items. This is done by overriding the `function set selectedAssets(selectedAssets:Array):void`.
- Actual handling of AssetActionEvent when a user clicks the action. This is done by overriding `function handleAction(event:AssetActionEvent):void`.

Here is the implementation for `com.adobe.icc.ams.handler.CustomLayoutHandler` that extends LayoutHandler:

```
package com.adobe.icc.ams.handler
{
    import com.adobe.consulting.pst.vo.Form;
    import com.adobe.icc.editors.handlers.LayoutHandler;
    import com.adobe.icc.editors.managers.ErrorManager;
    import com.adobe.icc.services.ServiceProvider;
    import com.adobe.icc.vo.DataDownload;
    import com.adobe.livecycle.assetmanager.client.event.AssetActionEvent;
    import com.adobe.livecycle.assetmanager.client.model.AssetAction;

    import flash.net.FileReference;

    import mx.controls.Alert;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;

    public class CustomLayoutHandler extends LayoutHandler
    {
        /**
         * Constant for the name of the action related to creation of an asset.
         */
        static public const ACTION_DOWNLOAD:String = "Download";

        override public function set selectedAssets(selectedAssets:Array):void
        {
            super.selectedAssets = selectedAssets;
            for each(var assetAction:AssetAction in assetActions)
            {
                if(assetAction.name == ACTION_DOWNLOAD)
                {
                    assetAction.enabled = (selectedAssets!=null) && (selectedAssets.length==1);
                        break;
                }
            }
        }

        override public function handleAction(event:AssetActionEvent):void
        {
            if(event.actionName == ACTION_DOWNLOAD)
            {
                downloadLayout();
            }
            else
            {
                super.handleAction(event);
            }

        }

        protected function downloadLayout():void
        {
            var selectedLayout:Form = selectedAssets[0];
            ServiceProvider.getDownloadService().getFormData(selectedLayout.id)
                .addHandlers( layoutDownloadResultHandler,
                                                function(faultEvent:FaultEvent){

ErrorManager.handleFault(faultEvent.fault, selectedLayout);
```

```
                                                    }
                );
        }

        private function layoutDownloadResultHandler(event:ResultEvent):void
        {
            var data : DataDownload = event.result as DataDownload;
            Alert.show("Pick a location to download Layout XDP", "Download Layout",Alert.OK |
Alert.CANCEL, null,
                function (event:*):void
                {
                    if ( event.detail == Alert.OK ) {
                        var fileReference:FileReference = new FileReference();
                        fileReference.save(data.objByteArray ,data.objFileName);
                    }
                }
            );
        }
    }
}
```

Registering the modified asset handler

The final step is to register the CustomLayoutHandler in place of LayoutHandler in AssetHandlerRegistry.This has to be done in ManageTemplates/index.mxml at function `init(event:FlexEvent):void function`:

```
registry.register(new CustomLayoutHandler());
```

Chapter 5: Troubleshooting

You can use the *LiveCycle ES2.5 Error Code Reference* to troubleshoot errors you may encounter when working with the Correspondence Management Solution Accelerator. Errors may be written to log files when you are installing, configuring, or running Adobe® LiveCycle® Enterprise Suite 2.5 (ES2.5). For each error code, there is a probable cause and an action you can take to resolve the error. Error codes are grouped according to component or activity. See the *LiveCycle ES2.5 Error Code Reference*.

You can also consult the guide *Troubleshooting LiveCycle ES2.5* for information about how to troubleshoot installation, configuration and administration issues that may arise within a LiveCycle ES2.5 production environment. See *Troubleshooting LiveCycle ES2.5*.