opentext™

D2 Smartview SDK

version: 23.4.0

opentext™

Table of contents:

opentext™

opentext™

Documentum D2 Smartview SDK - 23.4.0

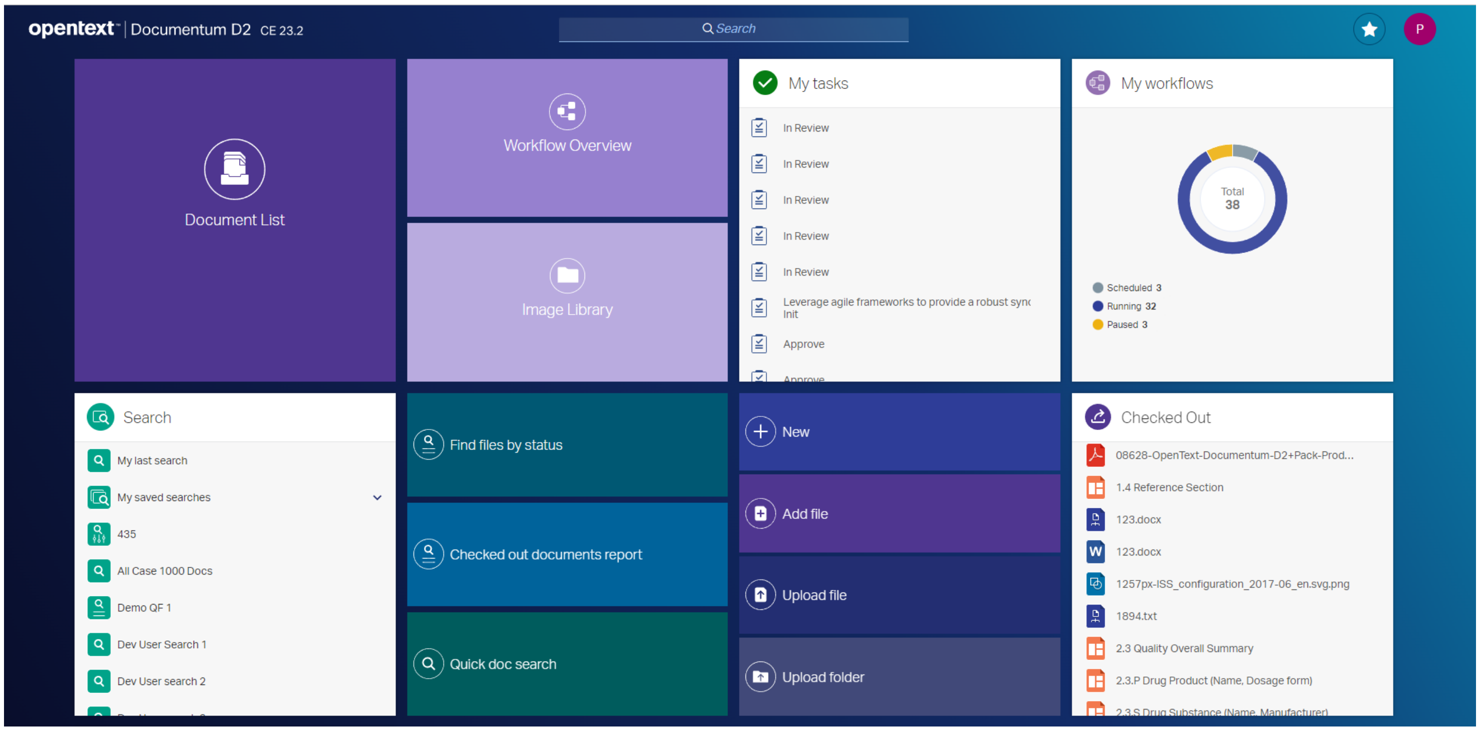
The D2 Smart View SDK consists of sources, binaries, documentation, and samples for -

- D2 Smartview UI extension enviornment.
- D2-REST services extension enviornment.
- D2 plugin development enviornment.

It also includes a few tools to create and maintain a development workspace.

With the D2 Smart View SDK you can build enterprise-ready software components for Documentum D2 Smartview runtime to cater custom business needs.

Out of the box, D2 Smart View landing page looks like:



How to prepare and start with the development environment

1. Download developer tools
2. Install developer tools

3. Create the development workspace
4. Get familiar with SDK tools
5. Create a plugin project
6. Start coding

1. Download the developer tools for your OS:

```
JDK         - JDK is required to compile Java code present within a development
workspace.
              Use JDK 17 or later.
              See https://openjdk.java.net
Maven       - Apache maven is the secondary build tool used in this SDK development
workspace.
              Recommended version is 3.8.2. A different version may not be fully
compatible.
              See https://maven.apache.org
NodeJS      - JavaScript VM to execute the SDK tools, build tools and to run the
development web server for UI code.
              Recommended version is 16 LTS. A different version may not be fully
compatible.
              See http://nodejs.org.
Grunt       - JavaScript task runner for building and testing UI code.
              See http://gruntjs.com. Nothing to be downloaded from this URL though.
```

2. Install the developer tools for your OS

```
JDK         - Run installer. Set the JAVA_HOME path variable to point to the JDK root
directory.
Maven       - Unzip & extract to a directory. Set MAVEN_HOME environment variable
pointing to the directory.
              Update PATH variable accordingly so that Maven commands can be executed
from command-line/terminal.
NodeJS      - Install the package for your OS. Set NPM_HOME path variable pointing to
the NodeJS
              installation directory. Update PATH variable so that Node & NPM commands
can be executed from
              command-line.
              It is recommended to avoid installing NodeJS under 'Program Files' as
```

```
doing that has been known to create
              problem some times.
NPM         - Update the NPM module management tool to the latest version:
                npm install -g npm@latest

Grunt       - Install the command line task runner client as a global NPM module
                npm install -g grunt-cli
```

3. Create the development workspace:

```
# 1. Extract the SDK
# 2. Open a command prompt at the extracted folder

# Execute batch script ws-init.bat
>ws-init.bat

# It will take a while to fully initialize the workspace.
# Once initialization completes successfully, the workspace assistant starts
automatically. Select "Check out documentation" option to open documentation in default
browser.
#
# The directory where SDK was extracted becomes the root of the development workspace.
# It doesn't require to run ws-init.bat inside the initialized workspace again, unless
some other instructions specifically says to do so.
# If you want to run the workspace assistant anytime later, open a command
prompt/terminal at workspace root directory and run
>npm start

# Select "Nothing", to terminate the workspace assitant, if wanted.

# To access the documentation without the workspace assistant, you can run the following
command in a command prompt/terminal at the workspace root.
>npm run documentation
```

4. Get familiar with Workspace assistant

Check out the Workspace assistant. It's a good idea to familiarize yourself with the general aspects of the SDK, this can be done later though.

opentext™

5. Create a plugin project:

```
# Open command prompt at workspace root and run
>npm start

# Select "Create a new plugin project" from the workspace assistant options.
# Follow on-screen instruction and anser questions to create your first plugin project.
# Once done, type and run-
>npm run build

# Or, alternatively run the workspace assistant again and select "Build all plugins in
this workspace" option.
# This will build all projects in the workspace
```

6. Getting started with SDK development

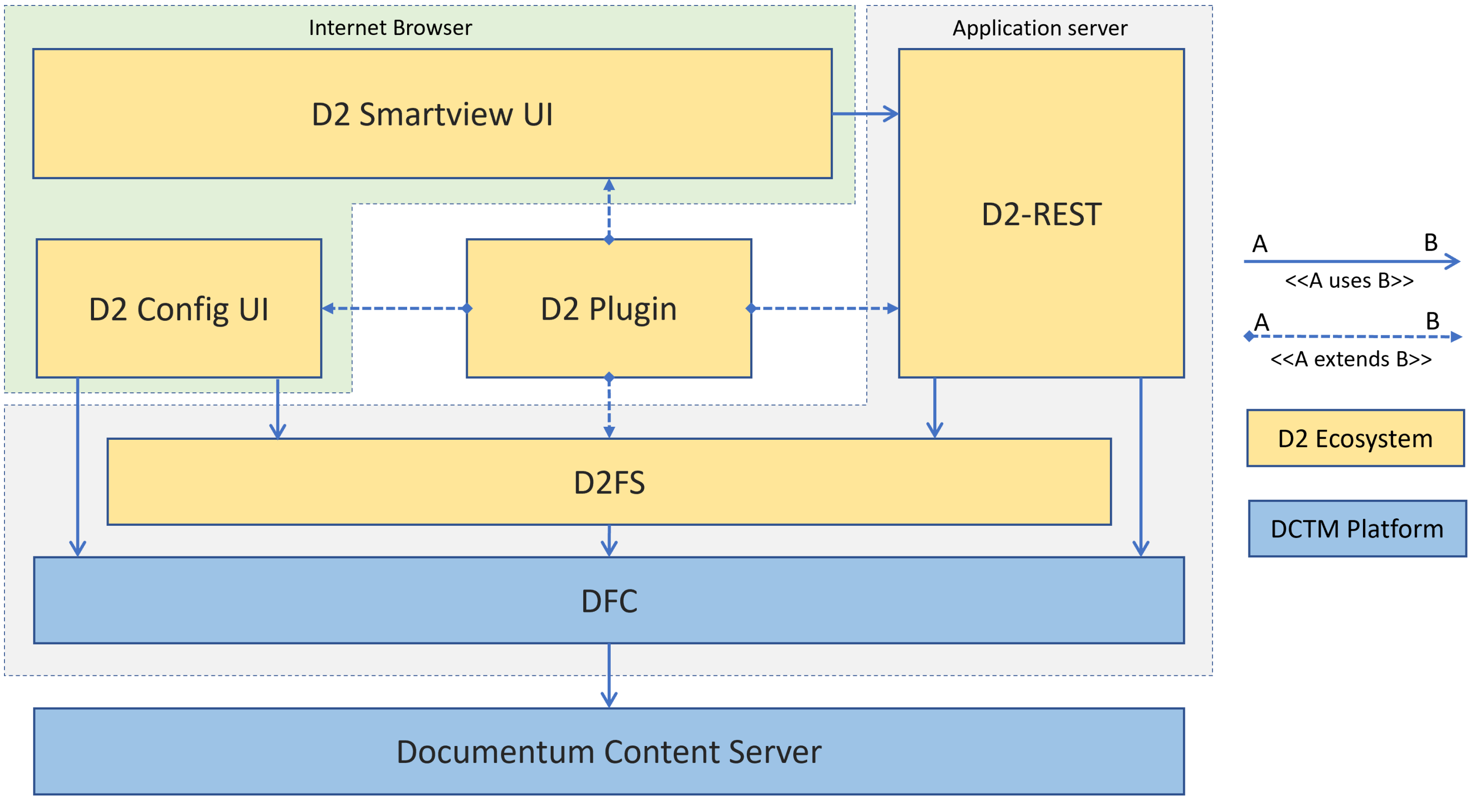
If you are a new SDK developer, you can check out this documentation to get started.

7. Start coding

Check out the API documentation and start coding as per business requirement.

Architecture

A simplified representation of the D2 Smartview(D2SV) runtime.



A central component of the runtime is D2 Plugins a.k.a D2FS Plugins. A D2 Plugin is loaded dynamically in the D2SV Ecosystem and it can primarily augment functions of D2 Foundation Services(D2FS).

The D2SV SDK API is built around the same D2 Plugins architecture and additionally it can augment functions of D2SV UI & D2FS-REST runtime.

The SDK deals with hybrid Maven + NodeJS project which has both Java & Javascript code along with other static resources organized in a certain structure. Upon build, the SDK compiles and packages the built output into a Jar. This Jar file can then be dropped inside the `lib` folder of a D2 Smartview runtime. The D2SV runtime loads the pluggable components from within the Jar dynamically.

Technology

The D2 Smartview is a web application and requires a hybrid middleware runtime. JVM runs the Java written back-end code and an Internet browser's Javascript VM runs the front-end. All the communication between FE & BE happens through AJAX request-response.

- The Java based back-end uses the Spring WebMVC Framework, configured to run in an application container, along with other proprietary and open-source libraries.
- The Javascript front-end uses Backbone & Marionette UI framework along with jQuery, RequireJS, Underscore, Handlebars etc. libraries.

Setup SDK workspace in IDE

Preface

All D2SV plugin projects are made of Java & Javascript source codes. Naturally the project setup also has to be hybrid to compile and package all parts of the source code. This is the reason the SDK uses a mixed tooling approach towards the same.

All the plugins inside the workspace, are layed out in a Maven project structure where the `pom.xml` found at the workspace root directory serves as the aggregator-parent and each plugin is linked to it as Maven module.

The NodeJS specific portion does not require a parent-child relationship between the plugins and the workspace. However to optimize dependency management, the workspace declares itself as a NodeJS project through its

`package.json` and each plugin in turn uses directory shortcuts to refer to the same set of dependencies even though the plugin declares a separate NodeJS project for its Javascript code through `package.json` found in its `src/main/smartview` directory.
While building, Maven is used as the primary tool to trigger it. Internally Maven uses `maven-antrun-plugin` to execute NodeJS script through shell and that builds the Javascript portion.

Any IDE that works with Maven projects, can recognize this hybrid setup. All that it takes is the support to be able to import a Maven project from existing source code.

TIP

Before trying any of the following steps to setup IDE, make sure to either create a plugin project or extract a packaged sample in the SDK workspace by running either Create a plugin project or Checkout some samples option from Workspace Assistant.

Steps to setup workspace in IntelliJ IDE

- Select **File -> New -> Project from Existing Sources** from menu.

  Note. If starting with a fresh IDE installation select **Import Project** option from welcome screen.

- In **Select File or Directory to Import** dialog, locate and select `pom.xml` from the workspace root.
- In **Import Project from Maven** dialog, keep the default values and deselect *Search for projects recursively* checkbox if it is selected. Then, click **Next** button to go to next screen.
- In the current screen select the checkbox against a group-id and artifact-id combination that correctly represents the workspace root pom. By default it may look something like *com.opentext.d2.smartview:D2-Plugin-Projects:1.0.0*. After selection, click **Next** button to proceed to the next screen.
- Select an available JDK to use for the imported projects and click **Next** to proceed.
- In current screen keep default values for **Project name** and **Project file location** input fields and click **Finish** to start import.

  Note. It might ask whether to open the project in current window or new window, please select an appropriate option. **New window** is could be a preferred option.

- After the project import completes, it might take a while for the IDE to index files from the workspace. To cut short on the indexing time, it is advised to mark all directories from workspace root except **plugins**(or whichever directory you chose to store plugins) as **Excluded**.

INFO

As you keep adding/removing new plugin projects in the workspace using the workspace assistant, the IDE automatically catches up with the change.

Steps to setup workspace in Eclipse IDE

- Select **File -> Import** from menu.
- In **Import** dialog, expand **Maven** and select **Existing Maven Projects** and click **Next**.
- In **Import Maven Projects** dialog, Click **Browse...** button beside the **Root Directory** field.

- In **Select Root Folder** file-selection dialog, navigate to SDK workspace root directory and click **Select Folder** button to go back to **Import Maven Projects** dialog.
- In **Import Maven Projects** dialog, click **Select All** to select all discovered projects. Then click **Finish** to close the dialog and start importing the projects.
- Once the project import completes, right-click on root project's **pom.xml** and select **Run as -> Maven build** from the menu to open **Edit Configuration** dialog.
- In **Edit Configuration** dialog type `clean install` in **Goals** field then click **Run** button to start building the plugins in the workspace.

CAUTION

Eclipse is unable to automatically detect plugin projects added to/removed from the workspace using the workspace assistant.

To detect newly added project, in Eclipse Project Explorer

- On the root project, **right click -> Refresh**.
- Select the **plugins** folder(or whichever folder you're using to store plugins) and select **Right click -> New -> Project**
- In **New Project** dialog, select **General -> Project** and click **Next**.
- In **New Project** dialog, deselect **Use default location** checkbox and click **Browse**
- In **Select Folder** dialog, navigate inside the root folder of newly created plugin project on disk and click **Select Folder** button
- In **New Project** dialog, copy the last part of path value from **Location** field and paste it into **Project name** field.
- Select the chekcbox **Add project to working sets** and select value **D2-Plugin-Projects** for the field **Working sets** then click **Finish** to close the dialog.
- Once the project creation completes, in **Project Explorer** of Eclipse, expand the **plugins**(or whichever folder you're using) folder and gesture **Right-click -> Configure -> Convert to Maven Project** to finish setting up the new plugin project.

To detect removed project, in Eclipse Project Explorer

- Gesture **Right-click -> Maven -> Update Project** on the root project's **pom.xml**
- In the **Update Maven Project** dialog, deselect every project except the root one and click **OK**
- If the above steps do not automatically remove the plugin project entry, then you can safely **Right-click -> Delete** the project.

Debugging D2 Smartview UI

D2 Smartview UI being written purely in Javascript, HTML, CSS has the benefit of debugging its client-side source code directly from an Internet Browser. Debugging can happen in either mode

1. with the source code as is
2. with compiled & minifed version of the source code

In distribution, D2 Smartview front-end and back-end is packaged as a single web archive however these two parts are very loosely coupled and communicates via HTTP request-responses.

This loosely coupled nature helps keep the front-end and back-end clearly separate even upto an extent where these two parts are hosted and served by two different application servers. This technique is recommended and also used by us to debug D2 Smartview UI such as we setup and configure D2 Smartview UI to make it talk to a running Smartview instance as backend and then host only the UI part on a lightweight NodeJS server.

> **INFO**
>
> The following method only allows debugging client-side Javascript code. If a plugin has some server-side component like REST-Controller, D2FS dialog, D2FS service plugin, menu confguration etc. then the plugin has to be built and deployed on the D2 Smartview application server followed by a server restart in order to make them available to use for the corresponding client-side code.

How to setup?

For D2SV plugin developers, the heavy lifting is already done for you. All you need is to -

1. Open to edit the `server.conf.js` from `src/main/smartview` directory of your plugin.

2. For `APP_SERVER_URL` property, set an appripriate URL to a running instance of D2 Smartview

   E.g. http://my.domain.com:port/D2-Smartview

> **TIP**
>
> The D2-Smartview installation, which is being referred to by the URL, must be setup to produce either relative linkrels by setting `rest.use.relative.url=true` in its rest-api-runtime.properties file, or alternatively it should be configured to allow Cross-Origin requests by setting other appropriate properties(refer to rest-api-runtime.properties.template file from D2-Smartview distribution).

3. Save `server.conf.js`

4. Open a Terminal/Command Prompt in the same `src/main/smartview` directory of the plugin.

5. Execute command `npm start`

6. Navigate to URL `http://localhost:6989/ui/pages/debug/app.html` in a browser to start debugging code in as-is format.

   Or, alternatively navigate to `http://localhost:6989/ui/pages/release/app.html` to debug the compiled & minified code. Please remember that command `grunt compile` from `src/main/smartview` or `npm run build` from workspace root directory has to be executed before you can debug the compiled and minified code.

opentext™

Extending/Overriding D2FS service through Service Plugin

If developer wants to create a customization which needs to override/extend the existing functionality of a D2FS service, the developer can create custom class with the **"(D2FS Services Name)Plugin.java'** which extends the D2FS service class and implements **ID2fsPlugin** class. For an example -

```
package com.opentext.d2.smartview.d2svdialogs.webfs.dialog;

import com.emc.d2fs.dctm.web.services.ID2fsPlugin;
import com.emc.d2fs.dctm.web.services.dialog.D2DialogService;
import com.emc.d2fs.models.context.Context;
import com.emc.d2fs.models.attribute.Attribute;
import com.emc.d2fs.models.dialog.Dialog;
import java.util.List;

public class D2DialogServicePlugin extends D2DialogService implements ID2fsPlugin {
    ...
    public Dialog validDialog(Context context, String id, String dialogName,
List<Attribute> parameters) throws Exception {
        //custom logic

        //If the following line is executed during an invocation then it becomes an
extension, otherwise it becomes an override.
        return super.validDialog(context, id, dialogName, parameters);
    }
    ...
}
```

Following services can only be extended/overriden for the supported methods. Below table lists both overidable and non-overridable methods for D2-Smartview

Services	Overridable Methods	Non-Overridable Methods

Services	Overridable Methods	Non-Overridable Methods
D2CreationService	applyVdTemplate getVDTemplates setTemplate getTemplates getTemplates updateTemplatesListwithFilter getConvertStructureConfig createTemplateFromServer createProperties getRecentlyUsedVDTemplates getImportStructureConfigs getRecentlyUsedTemplates	hasAnyAttachments removeAttachments getUIMaxSize hasAttachments hasAttachments getFilteredTemplates isAFolderOrACabinet getTemplateFilterOptions isNoCreationProfile isNoContentAuthorized
D2DialogService	getOptions getDialog validDialog cancelDialog	getTaxonomy getLabels isMemberOfGroup getImportValuesUrl getSubforms getExportValuesUrl
D2PropertyService	dump saveProperties saveProperties	getProperties
D2WorkflowService	rerunAutoActivity resumeTask pauseTask setTaskPriority updateWorkflowSupervisor getWorkflowTemplatesByWidgetName removeWorkflowSupportingDocuments	isTaskAcquired getTaskMode acquireTask getTaskPermissions canRejectTask canForwardTask canDelegateTask

Services	Overridable Methods	Non-Overridable Methods
	isTaskQueueItemRead addWorkflowSupportingDocuments getWorkflowUsersByWidgetName getWorkflowWorkingDocumentsCount getWorkflowSupportingDocumentsCount completeAutoActivity getWorkflowAttachments checkLifeCycle pauseWorkflow processTask addNoteToTask launchWorkflow abortWorkflow canAddTaskNote delegateTask setTaskReadState resumeWorkflow updatePerformer fetchWorkflowConfig checkAttachmentLockState getWorkflowStatusSummary verifyEntryCriterias verifyEntryCriteria launchScheduledWorkflow acquireTaskAndGetState getUnreadTasks delegateTaskEx addNoteToWorkflow doAutoTaskAction	checkPropertyPage checkPropertyPage getTaskFolderLabel canAbortWorkflow getConfigurationsNames getWorkflowDisplayName checkWorkflowAliases delegateTaskOnError isManualAcquisitionTask

opentext™

Services	Overridable Methods	Non-Overridable Methods
D2CheckoutService	checkout cancelCheckout testCheckout testControlledPrint	cancelCheckoutAll checkoutAsNew getNumberOfCheckoutDocument
D2CheckinService	getCheckinConfig	checkin
D2DownloadService	checkin getUploadUrls getUploadUrls getUploadUrls getPageServingUrl getCheckinUrls getDownloadUrls getExtractUrls getFile setFile hasAnyValidC2ExportAndRenditionConfig getDefaultServerInfo checkinAndLifeCycle extractDcoumentProperty getImportStructureUrls setDocumentProperty extractProperties getDownloadObjectId	getFileInfo getObjectsDownloadUrls canPrintControlledView getDispatchDownloadUrl getDownloadFileDetails isProtectedInControlledView addRendition

INFO

In case any unsupported methods are overriden by a plugin, it will be shown as warning in `D2-Smartview.log` during startup of the application

Custom Widget Type

Developers can define custom widget types if the default set of widgets provided from OOTB D2. This helps in developing custom views and business operation to perform

Custom shortcut type in D2

D2-Smartview landing page configuration elements like `<tile>` requires a `type` attribute to be set. By default all "Widget type" found in D2-Config are accepted as valid values. However, while defining new shortcuts for the landing page tile one might need to use a value that is not a "Widget type" at all or in other words the value is not pre-defined. Additionally, while defining new application scope perspective one might need to declare a default widget name for the corresponding perspective to use when a direct URL based navigation is taking place in the UI.

To facilitate this, an SDK developer can create a properties file **{Plugin}/resources/strings/config/U4Landing.properties**. There are two properties that can be defined in the file -

- *default_widget_tags*: This allows declaring new valid XML tag names to go under the `<default-widgets>` section in the D2SV landing page file. Multiple comma separated values could be specified.
- *shortcut_types*: This allows declaring valid values for \`type\` attribute of `<tile>`. Multiple comma separated values could be specified.

Example

```
default_widget_tags=abcd,defg
shortcut_types=custom1,custom2
```

If this is the content of the **U4Landing.properties** file, then the following hypothetical landing page structure is accepted as valid configuration

```
<root>
    <space>
        <flow-layout-container>
            <content>
                <tile-container size="full">
                    <tile name="one" type="custom1">
                        <theme color="shade1" type="stone1"/>
                    </tile>
                    <tile name="two" type="custom2">
                        <theme color="shade1" type="indigo1"/>
                    </tile>
                </tile-container>
            </content>
        </flow-layout-container>
    </space>
    <default-widgets>
        <abcd>SOME_NAME</abcd>
        <defg>ANOTHER</defg>
    </default-widgets>
</root>
```

The developer needs to register the custom widget type if needed to create widgets which can driven through the D2-Config context matrix evaluation. To do this, developer needs to create a properties file **{Plugin}/resources/strings/config/WidgetSubtypelist.properties**. This will include all the widget type created in the format **{Widget Type Name}=true**

Example

```
CustomWidgetType=true
```

If developer wants the widget to inherit the properties of some other OOTB D2 widget types, then developer needs to prefix the parent widget type name in the format **{Parent Widget Type Name}.{Widget Type Name}=true**

Example

```
DocListWidget.CustomWidgetType=true
```

If the developer want to have custom parameters as part of the custom widget type, developer needs to add those in the properties file **{Plugin}/resources/strings/config/WidgetSubtype.properties**. This will include the parameters for all the widget type created for plugin in the format **{Widget type Name}.{{Parameter name}={Default value}**.

Example

```
CustomerCustomType.sample1 = Text1
CustomerCustomType.sample2 = Text2
```

In order to provide custom labels for the custom parameters which are created needs to be add in the properties file **{Plugin}/resources/strings/actions/config/modules/widget/WidgetDialog/WidgetDialog_en.properties**. Entries will be in the format **label_{parameter name}={display label}**.

Example

```
label_sample1 = Sample text field 1
label_sample2 = Sample text field 2
```

opentext™

Delta Menus in D2

Smart View is driven via menu configuration for operations and these menus are shown in the D2-Config under the 'Menu Smart View' configuration. Each of the menu configuration consists of various menu context. Developers can change default behaviour of the menu items in the various menu contexts.

Developers can add/modify or delete each of the menu items in the respective menu context.

TIP

using the ws assistant, developers can try to create a new menu for the custom d2fs dialogs. Add D2FS dialog to a plugin)

If the developer wants to change OOTB default menu, then developer need to create a delta menu xml to define those changes in the following path and format. `resources/xml/unitymenu/<menu_context_name>Delta.xml`.

For example: If the we want to add a new menu item in the *Action Toolbar* then we need to create a delta file as follows: `resources/xml/unitymenu/MenuContextDelta.xml`.

Following are the currently supported menu contexts in Smart View. Each of the context are provided with the OOTB Menus and their corresponding internal 'id' references needed for appending.

INFO

Details regarding each of the menus can be found in the D2 admin guide

MenuContextDetailRelations(Relations)

Menu item name	Menu Id
Delete relationship	menuContextRelationsDestroy

opentext™

MenuContextDetailRenditions(Renditions)

Menu item name	Menu Id
Export rendition	menuContextRenditionsExport
Delete rendition	menuContextRenditionsRemove

MenuContextDetailVersions(Versions)

Menu item name	Menu Id
Properties	menuContextProperties

MenuContext(Action Toolbar)

Menu item name	Menu Id
Properties	menuContextProperties
Copy link	menuContextCopyLink
Share	menuContextShare
Edit	menuContextEdit
Cut	menuContextCut
Copy	menuContextCopy
Paste	menuContextPaste

Menu item name	Menu Id
Paste as link	menuContextPasteLink
Add version	menuContextAddVersion
Checkout	menuContextCheckout
Cancel checkout	menuContextCancelCheckout
Permissions	menuContextPermissions
Print	menuDocumentPrint
Download	menuContextDownload
Export properties	menuContextExport
Delete	menuContextDestroy
Add to collection	menuContextAddToCollection
Create relation	menuContextRelationCreate
View native content	menuDocumentViewNative
Convert to virtual document	menuContextConvertToVD
Convert to virtual document	menuContextConvertFolderToVD
Display outline	menuNewOpenVD
Display snapshot	menuNewOpenSVVDSnapshot

Menu item name	Menu Id
Lifecycle	menuContextDocumentLifeCycle
Send to workflow	menuContextDocumentWorkflow
Mass update	menuToolsMassUpdate

MenuNewObject(+ Menu)

Menu item name	Menu Id
Add file	menuNewObjNewDocument
Upload file	menuNewObjImportDocument
Add folder	menuNewObjNewFolder
Add cabinet	menuNewObjNewCabinet
Upload folder	menuNewObjImportFolderStructure

MenuUser(User Menu)

Menu item name	Menu Id
User settings	menuUserSettings
Help	menuHelpContents
About D2	menuHelpAbout

opentext™

Menu item name	Menu Id
Sign out	svMenuUserLogout

MenuContextVD(VDoc Action Toolbar)

Menu item name	Menu Id
Download	menuContextVDDownload
Edit	menuContextVDEdit
Checkout	menuContextVDCheckout
Check in	menuContextSVVDCheckin
Cancel checkout	menuContextVDCancelCheckout
Add child	menuContextVDAddChild
Create snapshot	menuContextSVVDSnapshotCreate
Convert to virtual document	menuContextVDConvertToVd
Convert to simple document	menuContextVDConvertToSimpleDoc
Set to version	menuContextSVVDSetBinding
Replace	menuContextSVVDReplaceChild
Remove	menuContextSVVDRemoveChild

Menu item name	Menu Id
Lifecycle	menuContextDocumentLifeCycle
Send to workflow	menuContextDocumentWorkflow

MenuContextTasksList(Tasks)

Menu item name	Menu Id
Acquire	menuContextAcquireTaskSV
Accept	menuContextForwardTaskSV
Reject	menuContextRejectTaskSV
Pause task	menuContextPauseTask
Resume task	menuContextResumeTask
Rerun	menuContextRerunTask
Complete	menuContextCompleteTask
Delegate	menuContextDelegateTaskSV
Update performers	menuContextUpdatePerformer
Manage supporting files	menuContextManageAttachmentsSV
Change task priority	menuContextTaskPriority

Menu item name	Menu Id
Mark as unread	menuContextTaskUnread
Mark as read	menuContextTaskRead
Add note	menuContextAddTaskNoteSV
Email to performer	menuContextSVEmailPerformer
Abort workflow	menuContextAbortWorkflow

MenuContextTaskDocument(Task Documents Actions Toolbar)

Menu item name	Menu Id
Properties	menuContextProperties
Go to location	menuContextGotoLocation
Copy link	menuContextCopyLink
Download	menuContextDownload
Export properties	menuContextExport
Edit	menuContextEdit
Add to collection	menuContextAddToCollection
Add version	menuContextAddVersion

Menu item name	Menu Id
Cancel checkout	menuContextCancelCheckout

MenuContextWorkflowOverview(Workflow Actions Toolbar)

Menu item name	Menu Id
Start workflow	menuContextLaunchScheduledWorkflow
Pause workflow	menuContextPauseWorkflow
Resume workflow	menuContextResumeWorkflow
Change supervisor	menuContextChangeSupervisor
Update performers	menuContextUpdatePerformer
Email to supervisor	menuContextEmailSupervisor
Email to performers	menuContextEmailPerformer
Manage supporting files	menuContextManageAttachmentsSV
Abort workflow	menuContextAbortWorkflow

MenuContextCollection(Collections)

Menu item name	Menu Id
Add items	menuContextAddToCollectionItems

Menu item name	Menu Id
Rename	menuContextRenameCollection
Delete	menuContextDestroy

MenuContextCollectionItems(Collections Items Action Toolbar)

Menu item name	Menu Id
Properties	menuContextProperties
Copy link	menuContextCopyLink
Edit	menuContextEdit
Add version	menuContextAddVersion
Checkout	menuContextCheckout
Cancel checkout	menuContextCancelCheckout
Permissions	menuContextPermissions
Print	menuDocumentPrint
Download	menuContextDownload
Export properties	menuContextExport
Remove	menuContextRemoveFromCollection

Menu item name	Menu Id
Add to collection	menuContextAddToCollection
Create relation	menuContextRelationCreate
View native content	menuDocumentViewNative
Convert to virtual document	menuContextConvertToVD
Convert to virtual document	menuContextConvertFolderToVD
Display outline	menuNewOpenVD
Display snapshot	menuNewOpenSVVDSnapshot
Lifecycle	menuContextDocumentLifeCycle
Send to workflow	menuContextDocumentWorkflow
Mass update	menuToolsMassUpdate

Sub menu contexts which are used for lifecycle/workflows/mass update/vdoc actions

MenuDocumentLifeCycle (Lifecycle sub menus)

Menu item name	Menu Id
Dynamic display D2 lifecycle start state	dynamicMenuDocumentD2LifeCycleAttach
Dynamic display of D2/DCTM state	dynamicMenuDocumentD2LifeCycleNextStates

Menu item name	Menu Id
Dynamic display of DCTM lifecycle	dynamicMenuDocumentLifeCycleAttach
Detach	menuDocumentLifeCycleDetach
Promote	menuDocumentLifeCyclePromote
Demote	menuDocumentLifeCycleDemote
Suspend	menuDocumentLifeCycleSuspend
Resume	menuDocumentLifeCycleResume

MenuDocumentWorkflow (Workflow sub menus)

Menu item name	Menu Id
Dynamic display of D2 workflow	dynamicMenuDocumentD2Workflow

MenuToolsMassUpdate (Mass update sub menus)

Menu item name	Menu Id
Dynamic display D2 mass updates	dynamicMenuToolsMassUpdate

MenuContextVDAddChildOption(Add Child submenu in VDoc action toolbar)

Menu item name	Menu Id

Menu item name	Menu Id
Browse	menuSVVDAddChildBrowse
Create	menuVDAddChildCreate
Upload	menuSVVDAddChildImport
From template	menuSVVDAddChildTemplate

MenuContextVDReplaceChildOption(Replace Child submenu context in VDoc action toolbar)

Menu item name	Menu Id
Browse	menuVDReplaceChildBrowse
Create	menuVDReplaceChildCreate
Upload	menuSVVDReplaceChildImport

MenuContextTaskPriority(Change task priority sub-menu in Tasks toolbar)

Menu item name	Menu Id
Highest	menuContextTaskPriorityHighest
High	menuContextTaskPriorityHigh
Normal	menuContextTaskPriorityNormal

Menu item name	Menu Id
Low	menuContextTaskPriorityLow
Lowest	menuContextTaskPriorityLowest

Defining the delta changes in the xml

Menu item in the D2 will follow the below structure

```
<menuitem id="MenuId">
     <dynamic-action class="ClassName"/>
     <condition class="ClassName" equals="value"/>
  </menuitem>
```

Here we have mainly 3 part for the menu as follows

- **menuitem**: define the menu item which will have a id attribute. The id attribute can be used to define label and is also a unique identifier.
- **dynamic-action**: developer can define a class which is extended from the `IDynamicAction`. This tag is used to define the action that has to be performed when the menu item is clicked.
- **condition**: developer can define condition which has to be extended from `ICondition`

TIP

Depending on the class custom attributes can be passed to the tag for both `dynamic-action` and `condition`

In order to define the menu item we need to understand the root tag as `delta` tag. Following are the operations that can done on the menu item

- **insert** - This will be used to insert a new menu item. This tag mandates attributes such as `position-before` or `position-after` which will define menu id before or after which the current menu item has to be placed.

- **append** - This will be used to append menu item to the end of the menus.
- **modify** -Requires id attribute which is a reference to any existing menu item id which has to be modified. This can can be used to `delete`, `insert`, `append` and `insert-before`.
- **delete** - This requires an id attribute which will refer to the menu id that has to be deleted.
- **insert-before** - This can be used along with the `modify` to add new conditions.
- **move**- This can be used to move an exiting menu by using the menu id with the attributes `position-before` or `position-after`.

TIP

1. `position-before` and `position-after` attributes contains the menu id of other menus
2. menu id, class names for the dynamic actions and conditions can be discovered by creating menu items in D2-Config and exporting the menus.

Find below some of the ways to use delta menus:

1. Insert new menu

```
<delta>
  <insert position-before="menuToolsMassUpdate">
    <menuitem id="menuContextViewPermission">
      <dynamic-action class="com.emc.d2fs.dctm.ui.dynamicactions.actions.U4Generic"
eMethod="getPermissions" eMode="SINGLE" eService="PermissionActionService"
rAction="${pluginNamespace}/dialogs/permissions/permissions.dialog:showPermissions"
rType="JS"/>
      <condition
class="com.emc.d2fs.dctm.ui.conditions.interfaces.IsMultipleSelection"
equals="false"/>
      <condition class="com.emc.d2fs.dctm.ui.conditions.options.IsPluginActivated"
value="${pluginName}"/>
    </menuitem>
  </insert>
  <insert position-before="menuToolsMassUpdate">
    <separator/>
  </insert>
</delta>
```

2. Modify an existing menu with new conditions

```
<delta>
    <modify id="menuDocumentEdit">
        <insert-before>
            <condition class="com.emc.d2fs.dctm.ui.conditions.typeparts.IsObjectType"
value="d2c_bin_deleted_document¬d2c_bin_deleted_folder¬d2c_bin_deleted_folder_dump¬d2c_
equals="false"/>
        </insert-before>
    </modify>
</delta>
```

Overriding the default post action behavior

There are basically 3 types of post action that can be performed on the selected objects after the completion of the dialog service operation. Those operation can be set as an attribute on the new custom menus created which will define as the default behavior

1. **Locate content and refresh state upon action** : This will locate the object and update the state of the object selected. For example, if you are performing some operation which will move the selected object from one location to another. Then this attribute will help the user to identify where it is located as well as refreshing the state of the current container. Attribute used for this is `locateAndRefresh`
2. **Refresh state** : This will refresh the state of the object selected post dialog operation. For example, if you have performed a lifecycle operation or a property update as part of the **validDialog(.)** then if you want to have the updated menus as well as values in the selected item in the widget we would need to set this value as part fo the result. Attribute used for this is `refreshCheckoutState`
3. **Refresh widget** : This will reload the widget post the operation. Attribute used for this is `refreshWidget`

Sample Code

```
      <menuitem id="menuRefreshDialog">
          <dynamic-action
class="com.emc.d2fs.dctm.ui.dynamicactions.actions.U4ShowDialog"
            dialog="RefreshDialog"
            locateAndRefresh="false"
            refreshWidget="false"
            refreshCheckoutState="true"/>
        </menuitem>
```

INFO

1. Supported values are "true"/"false".
2. Default value if not set is false.
3. this value updated in D2-Config will override the default value given in the delta xml.

Icons in D2 Smartview

Like any other application, D2 Smartview also uses different icons for visual context of data, infromation or action.

D2 Smartview uses Scalable Vector Graphic(svg) images to implement an icon. All the icons used in D2SV can be broadly classified into

- Non interactive

  Used to attach a context to a piece of data or information.

- Interactive a.k.a action icons

  Used to represent an action

Action icons are different semantically compared to the other type in the sense that action icons are reactive and respond to keyboard focus, blur or mosue events. Implementation wise, the SVG behind an action icon has a certain element structure whereas the other type don't have any such restriction.

Specification

Being SVG, a D2SV icon can be upscaled or downscaled to any size to fit in a UI element's boundary, however we recommend and follow that each SVG is defined w.r.t a view box of size `32px x 32px`.

Action icons must have 3 svg sub-elements with id `state`, `metaphor` and `focus`. The `state` element reacts to mouse position w.r.t to icon itself and changes its color accent. The `metaphor` element holds the visual graphic of the icon. And the `focus` element reacts to keyboard focus gain, giving itself a highlighted border.

How to use an icon

- For non-interactive icons, place the svg file at any source folder location and refer to it as `background-image` property in any CSS file using relative(from css file location to svg file locaiton) url.

We follow a convention where an svg file is placed inside `impl/images` folder w.r.t to CSS file's location where the svg image will be used.

- For interactive icons, drop the svg file inside `utils/theme/action_icons` folder w.r.t the smartview source code directory. Then run `grunt compile` in the smartview source code directory. And finally, at the place of use (which is mostly inside node.actions.rules module), set property `iconName: '<pluginNamespace>_<svg_file_name_without_extension>'` after replacing the format with appropriate values.

Sample non-interactive icon

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Generator: Adobe Illustrator 19.1.0, SVG Export Plug-In . SVG Version: 6.00 Build
0)  -->
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
     viewBox="-379 291 32 32" enable-background="new -379 291 32 32"
xml:space="preserve">
<g>
    <circle fill="#7E929F" cx="-363" cy="307" r="16"/>
</g>
<circle fill="#FFFFFF" cx="-363" cy="315" r="2"/>
<path fill="#FFFFFF" d="M-361,309c0,1.1-0.9,2-2,2l0,0c-1.1,0-2-0.9-2-2v-11c0-1.1,0.9-
2,2-2l0,0c1.1,0,2,0.9,2,2V309z"/>
</svg>
```

Sample action icon

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Generator: Adobe Illustrator 24.0.1, SVG Export Plug-In . SVG Version: 6.00 Build 0)
<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
```

```
     viewBox="0 0 32 32" style="enable-background:new 0 0 32 32;" xml:space="preserve">
<style type="text/css">
    .st0{fill:none;}
    .st1{fill-rule:evenodd;clip-rule:evenodd;fill:#333333;}
    .st2{fill:none;stroke:#2E3D98;}
</style>
<circle id="state" class="st0" cx="16" cy="16" r="14"/>
<path id="metaphor" class="st1" d="M22.4,12.5H9.6c-0.331,0-0.6-0.336-0.6-
0.75S9.269,11,9.6,11h12.8c0.331,0,0.6,0.336,0.6,0.75
    S22.731,12.5,22.4,12.5z
M23.53,16.5H9.685C9.307,16.5,9,16.164,9,15.75S9.307,15,9.685,15H23.53c0.378,0,0.684,0.336,0
    S23.908,16.5,23.53,16.5z M9.628,20.5h10.785c0.346,0,0.628-0.336,0.628-
0.75S20.759,19,20.412,19H9.628C9.281,19,9,19.336,9,19.75
    S9.281,20.5,9.628,20.5z"/>
<circle id="focus" class="st2" cx="16" cy="16" r="15.5"/>
</svg>
```

TIP

For a list of built-in action icons refer to the Catalog

opentext™

Upgrading Documentum D2 Smartview SDK to the latest version

With Documentum D2 Smartview SDK, new features will be added and delivered in each release. So,the SDK workspace need to be updated for consuming newly delivered feature.

D2 Smartview SDK provides an automated way to update the current workspace to the latest version of SDK.

Steps to update existing workspace to new version SDK.

- Close any IDE opened in existing workspace root.
- Unzip D2SV-SDK-23.4.0.zip
- Copy d2sdk-maven-plugin-23.4.0.jar from following path in zip.
  - D2SV-SDK-23.4.0 (Archive root)/tools/
- Paste the copied d2sdk-maven-plugin-23.4.0.jar in following path.
  - d2sv-sdk-23.2.0(Existing SDK workspace root)/tools/
- Execute the following command in D2SV-SDK-23.2.0(Existing SDK workspace root) using Command Prompt. With this command execution, current SDK workspace will be updated with latest D2-SmartView dependencies and changes.
  - ws-update.bat
- Wait for Success message for SDK update.

Testing workspace upgrade and update all plugins in the workspace

- Execute **npm start** in workspace root folder in terminal application.
- Choose option **Build all plugins in this workspace** to compile and build all plugins in the workspace. This operation will restore all symbolic folder links in the workspace plugin. Check the generated plugin jars in **dist** folder in workspace root.
- Choose option **Check out the documentation** to open the SDK documentation and verify the version in the documentation.

- Deploy the generated plugin in server validating UI code and Backend code is compatible with new version of D2 Smartview SDK.
- Run the plugins in local node server for validating UI code is compatible with new version of D2 Smartview SDK.
- Open the updated SDK workspace in IDE.

Effect of workspace update

- Current workspace will be updated with latest 23.4.0 dependencies for UI and REST.
- Documentation will be updated with latest D2 Smartview 23.4.0.
- SDK infrastructure will be updated for root SDK and all plugin modules in workspace.
- Backup will be created for root pom.xml and package.json with version number suffix. (package_23.2.0.json.bak and pom_23.2.0.xml.bak)
- Backup will be created for existing workspace as zip file in same directory where workspace exist. If workspace root folder name D2SV-SDK-WORKSPACE then zip name will be D2SV-SDK-WORKSPACE_23.4.0_bak.zip.

Troubleshooting SDK Update

- **Restoring Dependencies**
    - As part of SDK upgrade, workspace root **pom.xml** and **package.json** will be regenerated with new dependencies.
    - If SDK developer has added any new java dependency in pom.xml or new npm dependency in package.json in older version, then those dependencies need to be restored again in new version pom.xml and package.json.
    - For SDK developer's reference, backup created for the older version pom.xml and package.json file.
- **Restoring workspace with older version of SDK**
    - When upgrade SDK fails, D2 Smartview SDK developer can restore the SDK workspace to older version.

- As part of SDK upgrade, complete workspace backup is created as zip file in same directory where workspace exist.
- So that, SDK developer can restore the workspace with older version of SDK.
  - **Steps to restore the older version of workspace**
    - Unzip the backup zip file.
    - Delete the new version of **d2sdk-maven-plugin.jar** in **tools** folder and keep old version of the same jar.
    - Update current root folder's path in **lib.path** property in root pom.xml for pointing to older dependency. Since this zip is back of original workspace, the original workspace path will be available in **lib.path** property.
    - Execute the **ws-init.bat -fo** (ws-init with force overwrite argument) command in extracted folder using Command Prompt.
    - With this command execution, current SDK workspace will be restored with older version of D2-SmartView dependencies and changes.
    - Later same workspace can be upgraded to new version of SDK.

- **Handling Java and Spring upgrade**
  - In 23.4.0 Documentum and D2-Rest has migrated to the new version of Java v17 and Spring v6.
  - Tomcat server need to be upgraded to version 10.
  - If there was any Java code uses older Java or Spring dependencies in any of the plugin, then build will be failing for those plugins.
  - SDK developer need to manually fix the compilation issues, which is caused due to Java and Spring upgrade.
- **Handling SDK upgrade for excluded modules**
  - If some plugin modules are excluded from root pom.xml, then those plugins are not considered for SDK upgrade automation.
  - The upgrade for the excluded plugins in the workspace need to be done manually.
  - For such excluded plugins, create a new plugin and migrate the source code changes manually.

Set form mode for D2FS dialog

Forms in D2FS dialogs can be rendered in either editable(`create`) or readonly(`read`) mode.

When rendering a dialog form on D2 Smartview UI, this is determined by checking for the form_mode attribute of dialog xml.

When creating a D2FS dialog, the workspace assistant prompts user to select the form_mode. The selected value from workspace assistant is set in the dialog definition xml file `<Plugins Directory>\src\main\resources\xml\dialog\<Dialog Name>.xml`.

```
<dialog auto_smartview_edit_mode="false" buttons_right="false" focus="" height="500"
id="SomeDialog" initial_invalid="false" resizable="true" signoff_creation="false"
signoff_edit="false" signoff_import="false" signoff_intention_dictionary=""
signoff_intention_required="false" width="400" form_mode="create">
  <content>
    <text advancedView_required="true" condition_required="true" control="true"
id="name_field"/>
  </content>
  <buttons>
    <button action="validDialog()" id="buttonOk" isPrimary="true"/>
    <button action="cancelDialog()" id="buttonCancel"/>
  </buttons>
  <signoff_fallback_message>
    <message locale="en" value=""/>
  </signoff_fallback_message>
</dialog>
```

But this can be overwritten later either by editing the dialog definition xml file or by setting `form_mode` attribute for `result` XmlNode in `buildDialog` method of `<Plugins Directory>\src\main\java\<Maven Group Id>\smartview\<Plugin Name>\dialogs\<Dialog Name>.java`

```
public class SomeDialog extends AbstractDialog implements ID2Dialog {
    ...
    @Override
    public XmlNode buildDialog(D2fsContext context, List<Attribute> attributes) throws
Exception {
        ...
        XmlNode result = super.buildDialog(context, dialogFile, labelsBundle,
context.getFirstObject(), defaultValues);

        if(result == null) {
            result = super.buildDialog(context, attributes);
        }

        // Custom logic to determine formMode at run time
        String formMode = 1 == 0 ? "read" : "create";

        result.setAttribute("form_mode", formMode);

        return result;
    }
    ...
}
```

In case of chained dialogs, if the chained dialog is returned as result from `validDialog` method of `<Plugins Directory>\src\main\java\<Maven Group Id>\smartview\<Plugin Name>\dialogs\<Dialog Name>.java`,

`form_mode` attribute could be set for `result` XmlNode before returning it.

```
public class SomeDialog extends AbstractDialog implements ID2Dialog {
    ...
    @Override
    public XmlNode validDialog(D2fsContext context) throws Exception {
        ...
        XmlNode result = super.validDialog(context);

        // Custom logic to determine formMode at run time
        String formMode = 1 == 0 ? "read" : "create";

        result.setAttribute("form_mode", formMode);

        return result;
    }
    ...
}
```

The `form_mode` attribute for chained dialog could also be set from `validDialog` method of `<Plugins Directory>\src\main\java\com\opentext\d2\smartview\webfs\dialog\D2DialogServicePlugin.java` before returning the dialog.

```
public class D2DialogServicePlugin extends D2DialogService implements ID2fsPlugin {
    ...
    public Dialog validDialog(Context context, String id, String dialogName,
List<Attribute> parameters) throws Exception {

        // Custom logic

        Dialog dialog = super.validDialog(context, id, dialogName, parameters);

        XmlNode xmlDialog =
XmlUtil.loadFromString(dialog.getXmlContent()).getRootXmlNode();

        // Custom logic to determine formMode at run time
        String formMode = 1 == 0 ? "read" : "create";

        xmlDialog.setAttribute("form_mode", formMode);

        dialog.setXmlContent(xmlDialog.toString());

        return dialog;
    }
    ...
}
```

opentext™

Custom Dialogs

The custom dialog is not a widget but more of a dialog to do a particular business operation.

What it can crete from workspace assistant

Using the workspace assistance developer can create custom dialog for the plugins created by the developer.

Methods exposed as part of the dialog

The custom dialog exposed below three method for building and handling various action of the dialog.

- **buildDialog(..)** : This method is responsible for building the Dialog.
- **validDialog(.)** : validdialog is used to either chain to different dialog or to perform a business action and return the success.
- **cancelDialog(.)** : this method is called when user perform invalid action.

How to return Success message

There are 2 types of interactions that can be provided for the success message.

1. **Alert** : if the developer wants to show an alert message popup post operation.

```
        XmlNode result = super.validDialog(context);
        XmlNode msg = new XmlNodeImpl("message");
        msg.setValue("Processed Successfully");
        msg.setAttribute("type", "alert");
        msg.setAttribute("title", "Success");
        result.appendXmlNode(msg);
```

*__Sample Output__ :

```
    "page-content": {
         "success": {
             "message": {
                 "type": "alert",
                 "order": 1,
                 "content": "Processed Successfully"
             }
         }
     }
```

2. **Toast** : if the developer wants to show a toast message on top of the screen.

```
        XmlNode result = super.validDialog(context);
        XmlNode msg = new XmlNodeImpl("message");
        msg.setValue("Processed Successfully");
        msg.setAttribute("type", "toast");
        msg.setAttribute("title", "Success");
        result.appendXmlNode(msg);
```

**Sample Output* :

```
    "page-content": {
        "success": {
            "message": {
                "type": "toast",
                "order": 1,
                "content": "Processed Successfully"
            }
        }
    }
```

Where

- `message` is child node of success, and it is mandatory which is having their content as Value.
- `type` is also a mandatory key which allow either `alert/toast` as value.
- `title` is an optional parameter for the `type='alert'` which will be displayed as the title for the alert popup.

Overriding the default post action behavior

There are basically 3 types of post action that can be performed on the selected objects after the completion of the dialog service operation. Those operation can be set as an attribute to the result before the returning in the case of **validDialog(.)** and **cancelDialog(.)**

1. **Locate content and refresh state upon action** : This will locate the object and update the state of the object selected. For example, if you are performing some operation which will move the selected object from one location to another. Then this attribute will help the user to identify where it is located as well as refreshing the state of the current container. Attribute used for this is `locateAndRefresh`
2. **Refresh state** : This will refresh the state of the object selected post dialog operation. For example, if you have performed a lifecycle operation or a property update as part of the **validDialog(.)** then if you want to have the updated menus as well as values in the selected item in the widget we would need to set this value as part fo the result. Attribute used for this is `refreshCheckoutState`
3. **Refresh widget** : This will reload the widget post the operation. Attribute used for this is `refreshWidget`

Sample Code

```
        XmlNode result = super.validDialog(context);
         result.setAttribute("locateAndRefresh", "true");
         result.setAttribute("refreshCheckoutState", "true");
         result.setAttribute("refreshWidget","true");
         return result;
```

INFO

1. Supported values are "true"/"false".
2. Default value if not set is false.

opentext™

Workspace & Assistant

What is a workspace?

A workspace is the collection of tools, libraries, documentations put together in a specific directory structure that acts as a **Development Environment** and facilitates creation, management and build of D2 Plugin projects. D2SV SDK distributable comes in the form of a zip file named as `D2SV-SDK-<Version>.zip`. After extracting the zip and subsequently executing `ws-init.bat` successfully in the extracted folder turns it into a workspace.

What is the Assistant?

The workspace assistant is a command line utility that lives within a workspace & provides the functional aspects of the workspace with help of `NodeJS` & `Apache Maven` runtimes. While running, the assistant presents users with options to get something done within the corresponding workspace.

To run the assistant, open a command prompt/terminal in a workspace root directory and run

```
$npm start
```

Things that the workspace assistant is capable of doing are:

- Create a new plugin project
- Add smartview application scope perspective
- Add smartview UI support to an existing plugin
- Remove a plugin from workspace
- Add D2-REST controller to a plugin
- Build all plugins in the workspace
- Check out the documentation
- Check out some samples

- Add smartview shortcut behavior
- Add smartview list tile
- Add smartview shortcut tile
- Add D2FS dialog to a plugin
- Add new metadata view to plugin
- Add new task details view to plugin

Creating a plugin

This option is used to initiate a fresh D2SV plugin project. A plugin project is basically a maven project with all its dependencies pre-declared from the workspace `lib` folder. Based on selected options a plugin project may optionally have a Smartview UI component. If there is a D2SV UI component in a plugin, it requires NodeJS runtime to compile and package that specific component. All the relevant NodeJS and Javascript dependencies will be initialized upon plugin project creation.

To create a new plugin, a developer has to choose the **Create a new plugin project** option from the D2SV SDK workspace assistant.

Upon selecting the specific option in workspace assistant, a developer has to answer a few questions before the assistant can create and initialize the plugin project. For some of these questions asked, the workspace assistant will provide a meaningful contextual default answer based on usage, the default answer is enclosed within a pair of parentheses `()`, to choose the default value, one has to only press *Enter* key on the keyboard. These questions are self-explanatory however, here are a list of those questions and their meaning -

- Directory name to save this plugin project in

  Where to save the newly created project, defaults to \`plugins\` directory within the workspace.

- Maven group-id of the plugin

  Since all the plugin projects are maven projects, each project requires a group-id to be specified.

> **CAUTION**
>
> Known Issue: D2SV-SDK-23.2 allowed maven group id to be left empty. This causes error when building the plugin. Maven group id is validated in D2SV-SDK-23.4 to avoid this.

- Name(maven artifact-id) of the plugin

  Artifact identifier of the maven project to uniquely identify this plugin within the provided maven group ID.

CAUTION

Known Issue: D2SV-SDK 23.2 allowed plugin name to start with numeric character. This causes error when building the plugin. Plugin name is validated in D2SV-SDK-23.4 to avoid this.

- Version of the plugin

  Version of the plugin project.

- One liner description

  Used as the name and description for the underlying maven project. This is also shown as part of installed plugin data in D2 runtime.

- Package namespace

  A unique name used as prefix/suffix for generating the source code & properties in the maven project. The lowercase version of the given package name is used as part of the Java package name and also used as unique identifier for the Smartview UI code in the project, if any. For an example, if a Plugin project is created with Maven group-id `a.b.c` and it is given a package name `MyPlugin` then the base package for all Java source code becomes `a.b.c.myplugin` and the Smartview specific UI code is identified by `myplugin`.

- Include support for D2SV UI

  Whether to include D2 Smartview UI specific code infrastructure in the created plugin project. This question should be answered with an `Yes (Y)` only if the plugin is meant to develop, override or complement any D2SV front-end functionality. However, even for a plugin project initially created to develop or complement back-end oriented functionality, D2SV UI support can be added later through Add smartview UI support to an existing plugin project option.

opentext™

Add Smartview application scope perspective

A perspective in D2 Smartview is loosely defined as something similar to a web-page in case of multi page web application. A perspective renders a view of semantically similar data with relevant UI controls and interactions to operate on the data. D2 Smartview switches from one perspective to another based on user interaction.

D2 Smartview has several perspective implementations like Landing, Doclist, Virtual Documents, Tasks & Workflows etc. Out of all these the **Landing persepctive** is the default and shown immediately after user login, unless the URL in browser's address bar points to a different hint.

The Landing perspective shows a collection of widget & shortcut tiles, some of them, upon click, opens up another persepctive with a more specialized representation of the corresponding data. The semantic associated with the data in such perspective is generally termed as **application scope**. All such application scope perspectives (tied to different widget tiles in Landing perspective) visually look similar irrespective of the data semantic that they represent.

This **Add smartview application scope perspective** option of workspace assistant, helps create the boilerplate if we want to define a new data semantic in addition to those already defined by D2 Smartview.

CAUTION

Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- This perspective goes to plugin

  Select a plugin project, inside which this boilerplate will be created.

- Name of the widget type to associate

  Which landing page widget this perspective will associate itself with. Each landing page widget tile is configured with a `type` attribute, the possible values that can go against the `type` attribute can be used here.

- Name of the default smartview widget type(from landing config) to associate

  The Smartview landing page has a bunch of default widget configuration name defined. When an end-user navigates directly to an application scope using URL, the widget configuration data might be missin in the URL, in thsoe cases the default widget configuration is used to resolve the metadata requirement while opening the perspective.
  Answer to this question specifies which default widget configuration from the landing page is to be associated with the application scope being defined.

- Application scope of this perspective(also used as URL fragment)

  Name of this application scope. Also used as base part of the URL when this perspective is activated.

- Directory name where generated code will be put into

  Relative location of the boilerplate code w.r.t selected plugin projects widget source directory

- Default title of this perspective

  Default displayable label when this perspective is active.

- How would you describe this perspective?

  Description associated wtih this perspective

opentext™

Add smartview UI support to an existing plugin project

This options of the workspace assistant can be used to add Smartview UI support to an existing D2SV plugin project, if the project was created initially without Smartview support.

CAUTION

Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Add D2SV UI to plugin

  Select the D2SV plugin project, from the list of options, where Smartview UI code will be injected.

- Selected plugin seems to have D2SV UI support enabled already, overwrite it ?

  Confirm whether to overwrite the boilerplate code related to enabling D2SV UI support for the plugin. This question is asked only when the assistant detects that the selected plugin project already has D2SV UI support enabled in it.

Remove a plugin from workspace

This option is used to remove a plugin project from the corresponding workspace.

CAUTION

Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Select plugin to remove from workspace

  Select which plugin project is to be removed from the workspace.

- Remove it completely from file system?

  Whether to remove the project completely from filesystem. In case of a soft removal, the project is left intact on the disk, however its entry from the aggregator POM in workspace is removed to exclude it from build order.

opentext™

Add D2-REST controller support to a plugin project

If a D2SV plugin intends to deploy new D2-REST endpoints in addition to the factory endpoints, then this option of the workspace assistant comes in handy.

Upon successful execution, this option, can add a boilerplate REST controller definition to the plugin such that the controller handles HTTP GET(Retrieve), POST(Create), DELETE(Delete) operation on the specified endpoint resource matching part of the CRUD style transaction.

The complete URL path of the endpoint created is represented as -

```
/D2-Smartview/repositories/<repo_name>/<group_name>/<endpoint_name>
```

CAUTION

Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Select plugin to add REST support

  Specify the plugin project where to add boilerplate for this controller

- Group name of controller

  Group name for the service that this REST controller implements. Used as `<group_name>` part of the URL format mentioned above. Usually same group name is used across multiple services if they all happen to be correlated.

- Endpoint name of controller

  A meaningful name that should uniquely identify this endpoint in the group of correlated services. This name is used as `<endpoint_name>` part of the URL format mentioned above.

- Service name to use against the controller

  Name of the service that this endpoint represents. This name is used to form the name of Java classes and interfaces which are finally going to implement the service.

opentext™

Build all plugins in the workspace

This option builds all the plugin projects in the workspace whose entries are found in the aggregator `pom.xml` file in the workspace.

Basically it runs `mvn clean package` command in the workspace root directory. As part of this option, final build output from each of the plugin project is collected in the `dist` directory right under the workspace root directory.

CAUTION

Use this option when at least one plugin project exists in the workspace.

Checkout documentation

This option starts the embedded documentation server and opens the home page of it in the default browser.

Checkout samples

The D2SV SDK packs a few working samples to demonstrate the building blocks of D2SV runtime APIs. This option of the workspace assitant is used to unpack a sample into the workspace such that subsequently it can be built and deployed on a running D2 Smartview or simly the source code could be checked out as tutorial.

Upon selecting this option, the only additional question to be answered is to pick the sample that is to be extracted.

opentext™

Add smartview shortcut behavior

D2 Smartview landing perspective can be configured to have a number of shortcut tiles. Each of these shortcut tiles can do something specific and different from others based on its type, however their visual representations are same irrespective of what they do.

The function behind a particular type of shortcut is defined by a shortcut behavior attached to the shortcut type.

> **CAUTION**
>
> Use this option when at least one plugin project exists in the workspace.

This assistant option lets define a new shortcut behavior that can be attached to a shortcut type.

Associated questions and their meanings -

- This shortcut goes to plugin

  Select the plugin project where to create the boilerplate for the shortcut behavior

- Type of the target shortcut

  Declare the type of shortcut tile that are to be associated with this particular behavior

Add smartview list tile

Using this option of the assistant, a new list type of tile a.k.a widget tile definition could be added.

All widget tiles shown in D2 Smartview landing perspective are visually similar, however they are backed by different type of widget conifiguration and display data evaluated in the context of that widget.

CAUTION

Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- This shortcut goes to plugin

  To select the plugin project where to create boilerplate associated with creating a widget tile.

- Name of Widget type to associate

  Specify which type of widget configuration will drive this list tile data.

- CSS class name for the icon

  CSS class selector to put in the HTML element hosting the icon for this widget tile. Later this same class name could be used to render a specific icon for this tile.

- Directory name where generated code will be put into

  Relative location of the boilerplate code w.r.t selected plugin projects widget source directory

- Default title of this tile

  Specifies an optional default name for the tile which is shown at the header region of this tile incase the underlying widget configuration is not able to provide a name for it.

- How would you describe this tile?

Description metadata for this tile.

- Will it expand to own perspective?

  Whether this tile will expand to its own application scope perspective. If anwered with `yes` then a bunch of followup question related to the application scope perspective are asked. Add smartview application scope perspective can be referred for meaning of such questions.

opentext™

Add smartview shortcut tile

Using this option of the assistant, a shortcut type of tile definition could be created.

D2 Smartview has a bunch of shortcut tile implemenation on its own. Visually all the shortcuts look similar except their representation icon. However, each type of shortcut has its own mechanism to react to click event.

> **CAUTION**
>
> Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- This shortcut goes to plugin

  To select the plugin project where to create boilerplate associated with creating a widget tile.

- Type of this shortcut

  Declare the type of this shortcut tile definition. Creating custom widget type can be referred for more information.

- CSS class name for the icon

  CSS class selector to put in the HTML element hosting the icon for this shortcut tile. Later this same class name could be used to render a specific icon for this tile.

- Does it require a widget config?

  Shortcut tiles can optionally be backed by a widget config(E.g. Doclist type of shortcut requires a `DoclistWidget` config). Answering with `yes` associates this shortcut with a widget config.

- What this shortcut should do on click?

  Defines what happens when end user clicks on this tile. Can be one of -

- Execute inline handler
  A JS callback function will implement the action.
- Execute a behavior
  The action is delegated to a shortcut behavior implementaion. The actual definition of the beahvior is to be coded into the boilerplate created.
- Expand to perspective
  The action is to open an application scope perspective. Further questions are asked for creating the application scope perspective. Add smartview application scope perspective can be referred for meanings of such questions.

Add D2FS dialog to a plugin

If a D2SV plugin intends to define a property-page like form then this assistant option could be used to create the boilerplate associated with such form, which is also known as D2FS dialog.

> ⚠ **CAUTION**
>
> Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Select plugin to add new D2FS dialog

  Specify the plugin project where to add boilerplate for this dialog

- Enter name of the dialog

  A unique name of the dialog. It is also used as an ID to refer to the dialog from other part of D2SV runtime.

> ⚠ **CAUTION**
>
> Known Issue: D2SV-SDK 23.2 allowed dialog name to start with lowercase or number and have '_' & '-' in them. This causes server error when opening the dialogs on Smartview UI. Dialog name is validated in D2SV-SDK-23.4 to avoid this.

- Title of the dialog

  The title to be displayed when the dialog is visible on screen. It defaults to the given name of the dialog.

- Select form mode for the dialog

  Whether to show the dialog in `read-only` or `editable` mode.

- Select view mode for the dialog

  Whether to show the dialog in `center` or `side` panel mode.

- Create a menu to open the dialog?

  Whether to also define a menu item in D2SV context menu so that clicking that menu would show the dialog. Defaults to `Yes`.

- Label for the menu

  Define an English label for the menu. This question is asked only if the previous question was answered with an `Yes`.

- Pick a toolbar to add this menu to

  D2-Smartview UI shows different type of menu bars depending on the application context. By answering this option we can specify the menubar where this menu is going to be added. This question is asked only if 'Create menu to open the dialog?' was answered with `Yes`.

- Specify selection mode for the menu

  Whether the menu should support `single` or `multiple` object selection.

Add new metadata view to plugin

If a D2SV plugin intends to define new views like properties, versions, permissions, task performers then this assistant option could be used to create the boilerplate code associated with such views, which is also known as metadata panel.

> ⚠ **CAUTION**
>
> Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Select plugin to add new metadata view

  Specify the plugin project where to add boilerplate code for the new metadata view.

- Enter name for the view

  A unique name for the new metadata view.

  This will be the option name shown in the dropdown by default.

Associated generated boilerplate files and their use -

- The generated files would be present under `<Plugins Directory>\<Selected Plugin>\src\main\smartview\src\widgets\metadata\panels\<View Name>`

- `impl` folder

  This folder contains the handlebar template file and the style sheet file used by the view.

  It also contains the `nls` folder which contains the lang files used for the translation strings.

  The option name shown in the dropdown could be changed by changing the translation string value for viewName in `lang.js` file under `nls/root` folder.

- `metadata.<View Name>.view.js`

  This is the main view file for the new metadata view created.

  This will have code for a simple helloworld view.

  The template file, style sheet and lang file used by this view are already loaded.

  A wrapper class name for this view, ui, regions and events are defined in this view for reference.

  Modify this view based on the usecase.

  Incase of complex view, break it into smaller independent views and keep them under `impl` folder, specify regions in the main view and show these smaller views using regions.

  By default, this view will be shown for all metadata dropdowns. Add conditions in enabled function to restrict this view based on the usecase.

Add new task details view to plugin

If a D2SV plugin intends to define new views like working files, supporting files, task notes then this assistant option could be used to create the boilerplate code associated with such views, which is also known as task details panel.

> **CAUTION**
>
> Use this option when at least one plugin project exists in the workspace.

Associated questions and their meanings -

- Select plugin to add new task details view

  Specify the plugin project where to add boilerplate code for the new task details view.

- Enter name for the view

  A unique name for the new task details view.

  This will be the name shown for the tab by default.

Associated generated boilerplate files and their use -

- The generated files would be present under `<Plugins Directory>\<Selected Plugin>\src\main\smartview\src\widgets\task.details\panels\<View Name>`

- `impl` folder

  This folder contains the handlebar template file and the style sheet file used by the view.

  It also contains the `nls` folder which contains the lang files used for the translation strings.

  The tab name could be changed by changing the translation string value for tabName in `lang.js` file under `nls/root` folder.

- `task.<View Name>.view.js`

  This is the main view file for the new task details view created.

  This will have code for a simple helloworld view.

  The template file, style sheet and lang file used by this view are already loaded.

  A wrapper class name for this view, ui, regions and events are defined in this view for reference.

  Modify this view based on the usecase.

  By default, this view will be visible in task and workflow tab panel. Add conditions to restrict this view based on the usecase.

opentext™

Packaged Samples

D2SV SDK includes a few sample plugins as part of its distribution. They could be extracted in a workspace by using Checkout some samples option of the workspace assistant.

List of samples

- D2 Admin Groups Sample
- D2SV client to server logging
- D2SV Custom Dialogs(D2FS) sample
- D2SV Read-Only permission display
- D2SV Custom custom widget type sample
- Custom Table Cell Sample
- D2SV Object On Click Sample
- Open a cabinet/folder in Doclist

D2 Admin-Groups Sample

D2 Smartview does not ship with an Out-Of-The-Box(OOTB) group management widget like D2 Classic. However, the D2 Admin-Groups sample plugin fills-in the gap functionally and serves as a complete example of how to use SDK to

- Define a landing page widget tile.
- Define a perspective and stitch it up with the landing page widget.
- Define and use a custom menu type to go with the widget.
- Define a few REST endpoints to serve data to the widget.

Instruction to try out the sample

As this plugin implements a landing page tile as part of it, some configuration changes are required in the D2 Smartview landing page before the tile is made available for the end users. Here are the list of steps required to completely deploy and configure this plugin.

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2-AdminGroups-1.0.0.jar` from 'dist' folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Restart application server on which D2 Smartview is deployed.
- Open D2-Config web application in browser, login and then navigate to **Widget view -> Widget...** from menu.
- Create a new widget configuration, and put "Manage Groups"(or whatever you wish) as **Name** and select "AdminGroupsWidget" for **Widget type** field.
- Fill-in other fields as necessary and save the configuration.
- From the toolbar, click **Matrix** to go to D2-config matrix and enable the widget configuration, you just created, against appropriate contexts.
- Select **Widget view -> Smart View Landing Page...** from menubar to navigate to landing page configurations.
- Select an applicable configuration from the left side and click **Download** to get the structure file locally and then open in notepad to edit.

opentext™

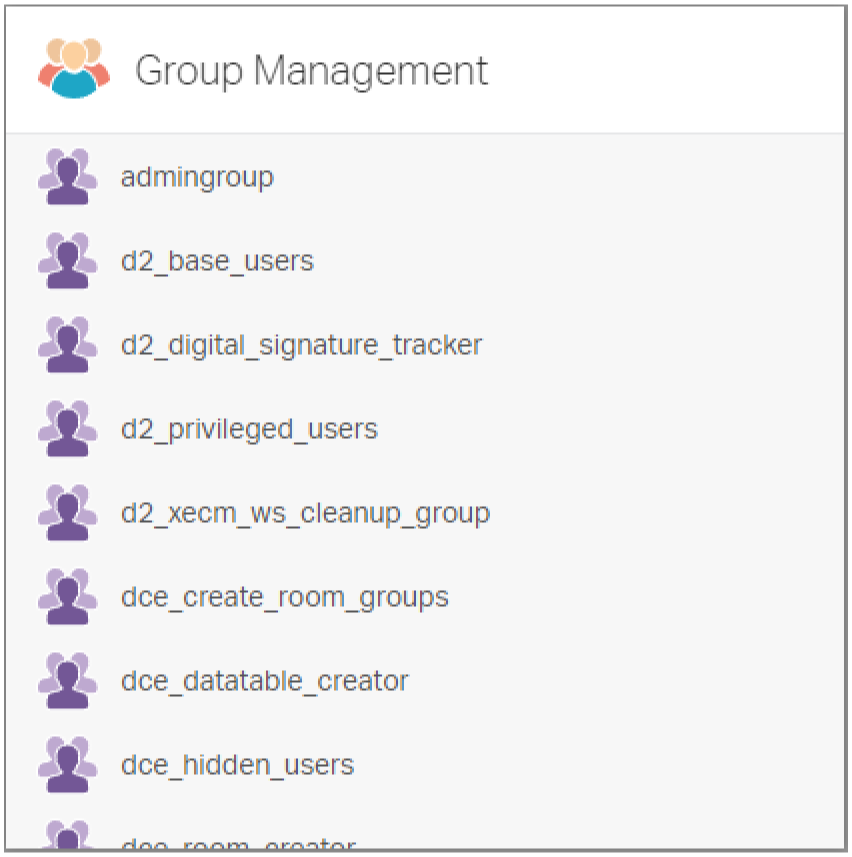
> **TIP**
>
> If a pre-created Smartview landing page configuration does not exist, then refer to D2 Administration Guide documentation to create the same and learn basics of landing page structure file.

- Paste the following piece of xml anywhere right under the `<content>` tag

```
<widget-container>
    <widget name="Manage Groups" type="AdminGroupsWidget"/>
</widget-container>
```

> **TIP**
>
> If you've used a different name while creating the widget configuration, use that name as the value for **name** attribute.

- Save the landing structure xml file and upload it to D2-Config under the same landing page configuration from where we downloaded it before.
- Save the configuration change in D2-Config and click **Tools -> Refresh cache** from menubar.

- Reloading the D2 Smartview at this point should show an additional widget in the landing page, similar to following 

Group Management

admingroup

d2_base_users

d2_digital_signature_tracker

d2_privileged_users

d2_xecm_ws_cleanup_group

dce_create_room_groups

dce_datatable_creator

dce_hidden_users

> **TIP**
>
> Clicking 'More actions' icon for any group, shows a 'Manage' menu, if the logged in user is an Administrator or a Superuser.

Source code structure

```
D2-AdminGroups
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
```

opentext™

```
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_AdminGroups.java
|       |               |
|       |               \---smartview
|       |                   \---admingroups
|       |                       |   AdminGroupsPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       AdminGroupsVersion.java
|       |                       |
|       |                       \---rest
|       |                           |   package-info.java
|       |                           |
|       |                           +---controller
|       |                           |       AdminGroupsController.java
|       |                           |       AdminGroupsMembersController.java
|       |                           |
|       |                           +---dfc
|       |                           |   |   AdminGroupsManager.java
|       |                           |   |
|       |                           |   \---impl
|       |                           |           AdminGroupsManagerImpl.java
|       |                           |
|       |                           +---model
|       |                           |       GroupMembers.java
|       |                           |       GroupModel.java
|       |                           |       UserModel.java
|       |                           |
|       |                           \---view
|       |                                   GroupMembersFeedView.java
|       |                                   GroupsFeedView.java
|       |                                   GroupView.java
|       |                                   UsersFeedView.java
|       |                                   UserView.java
|       |
|       +---resources
|       |   |   admingroups-version.properties
|       |   |   D2Plugin.properties
|       |   |
|       |   +---smartview
```

opentext™

```
|       |   |       SmartView.properties
|       |   |
|       |   +---strings
|       |   |   \---menu
|       |   |       \---PMenuContextAdminGroups
|       |   |               PMenuContextAdminGroups_en.properties
|       |   |
|       |   \---xml
|       |       \---unitymenu
|       |               PMenuContextAdminGroups.xml
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   admingroups.setup.js
|           |   config-editor.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   admingroups-extensions.json
|           |   |   admingroups-init.js
|           |   |   component.js
|           |   |   config-build.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       admingroups-bundle.js
|           |   |
|           |   +---commands
|           |   |   |   manage.group.js
|           |   |   |   node.actions.rules.js
|           |   |   |
|           |   |   \---nls
|           |   |       |   lang.js
|           |   |       |
|           |   |       \---root
|           |   |               lang.js
|           |   |
|           |   +---dialogs
|           |   |   \---manage.group
|           |   |       |   manage.group.dialog.js
|           |   |       |   manage.group.view.js
|           |   |       |
|           |   |       \---impl
|           |   |           |   group.members.form.view.js
|           |   |           |   manage.group.css
|           |   |           |   manage.group.hbs
|           |   |           |
|           |   |           \---nls
|           |   |               |   lang.js
|           |   |               |
|           |   |               \---root
|           |   |                       lang.js
|           |   |
|           |   +---extensions
|           |   |   |   admin.groups.icon.sprites.js
|           |   |   |   admin.groups.perspective.js
|           |   |   |   admin.groups.tile.js
|           |   |   |
|           |   |   \---nls
|           |   |       |   lang.js
|           |   |       |
|           |   |       \---root
|           |   |               lang.js
|           |   |
|           |   +---models
|           |   |       admin.groups.collection.js
|           |   |       group.members.collection.js
|           |   |       group.model.js
|           |   |       member.model.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   +---utils
|           |   |   |   alert.util.js
|           |   |   |   constants.js
|           |   |   |   menu.utils.js
|           |   |   |   startup.js
|           |   |   |
|           |   |   +---contexts
|           |   |   |   \---factories
|           |   |   |           admin.groups.collection.factory.js
|           |   |   |           next.group.factory.js
|           |   |   |
|           |   |   +---perspectives
|           |   |   |       admin.groups.perspective.json
|           |   |   |
|           |   |   \---theme
|           |   |       |   action.icons.js
|           |   |       |
|           |   |       \---action_icons
|           |   |               action_sample_icon.svg
|           |   |
|           |   \---widgets
|           |       +---admin.groups
|           |       |   |   admin.groups.manifest.json
|           |       |   |   admin.groups.view.js
|           |       |   |   toolitems.js
|           |       |   |
|           |       |   \---impl
|           |       |       |   admin.groups.css
|           |       |       |
|           |       |       +---images
|           |       |       |       group-svgrepo-com.svg
|           |       |       |
|           |       |       \---nls
|           |       |           |   admin.groups.manifest.js
|           |       |           |   lang.js
|           |       |           |
|           |       |           \---root
|           |       |                   admin.groups.manifest.js
|           |       |                   lang.js
|           |       |
|           |       \---admin.groups.members
|           |           |   admin.groups.members.view.js
|           |           |
|           |           \---impl
|           |               |   admin.groups.members.css
|           |               |
|           |               \---nls
|           |                   |   lang.js
|           |                   |
|           |                   \---root
|           |                           lang.js
|           |
|           \---test
|                   Gruntfile.js
```

opentext™

```
|                       karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

REST Controller

- src/main/java/com/opentext/d2/rest/context/jc/PluginRestConfig_AdminGroups.java - Declares Spring Bean `AdminGroupsManager` through `AdminGroupsManagerImpl`.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/controller/AdminGroupsController.java - Defines a REST controller with two endpoints to list all the users and groups from Documentum.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/controller/AdminGroupsMembersController.java - Defines a REST controller with two endpoints to list and edit members of a group.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/dfc/AdminGroupsManager.java - Declares the data manager interface used by above REST controllers to get/set the data they deal with.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/dfc/impl/AdminGroupsManagerImpl.java - Data manager that interacts with Documentum through DQL and exchanges data as per `AdminGroupsManager` interface.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/model/GroupMembers.java - Serializable POJO that represents members of a group while editing.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/model/GroupModel.java - Serializable POJO that represents a single group.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/model/UserModel.java - Serializable POJO that represents a single user.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/view/GroupMembersFeedView.java - Spring view used to wrap and serialize a list of group member data. Uses `UserView` in turn to

serialize each individual member.

- src/main/java/com/opentext/d2/smartview/admingroups/rest/view/GroupsFeedView.java - Spring view used to wrap and serialize a list of group data. Uses `GroupView` in turn to serialize each individual group.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/view/GroupView.java - Spring view used to seralize a single group data.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/view/UsersFeedView.java - Spring view used to wrap and serialize a list of user data. Uses `UserView` in turn to serialize each individual user.
- src/main/java/com/opentext/d2/smartview/admingroups/rest/view/UserView.java - Spring view used to serialize a single user data.

Group-manage menu configuration in back-end and its display & execution on the front-end

- src/main/resources/strings/menu/PMenuContextAdminGroups/PMenuContextAdminGroups_en.properties - Labels associated with the dynamically configured menu.
- src/main/resources/xml/unitymenu/PMenuContextAdminGroups.xml - The menu definition file that dynamically adds a new type(PMenuContextAdminGroups) of menu for the D2FS `D2MenuService` to discover and return for D2 Smartview.
- src/main/smartview/src/bundles/admingroups-bundle.js - A portion of this file is used to refer to key RequireJS modules that define the extensions to the toolbar and menu related D2SV UI API.

```
define([
    ...
    'admingroups/utils/startup',
    'admingroups/commands/node.actions.rules',
    'admingroups/commands/manage.group',
    ...
], {});
```

- src/main/smartview/src/commands/manage.group.js - A `CommandModel` that implements the executable logic when a user clicks the `Manage` menu on the UI. It dynamically loads and displays `manage.group.dialog.js` dialog. When the dialog closes(without cancel flag), it makes an AJAX call to group-update related REST endpoint created by the Java code from above and then finally shows a toast message on successful completion.

- src/main/smartview/src/commands/node.actions.rules.js - Defines client side filtering and association logic to attach the above `manage.group.js` command model implementaion to a UI toolbar item.
- src/main/smartview/src/utils/menu.utils.js - Parses the data for `PMenuContextAdminGroups` type of menu into a client-side toolbar definition that in turn is used by landing page tile & perspective.
- src/main/smartview/src/utils/startup.js - Runs as part of D2 Smartview client-side startup flow. This flow is executed everytime endusers reload the D2 Smartview application in their internet browser. As part of this startup hook, an AJAX call is made to get the menu configuration data for type `PMenuContextAdminGroups`, then the response is trasformed into a toolbar definition by `menu.utils.js`.
- src/main/smartview/src/admingroups-extensions.json - A portion of this file registers extensions to the toolbar and menu related D2SV UI API. The corresponding portion is highlighted below

```
"d2/sdk/commands/node.actions.rules": {
    "extensions": {
      "admingroups": [
        "admingroups/commands/node.actions.rules"
      ]
    }
  },
  "d2/sdk/utils/commands": {
    "extensions": {
      "admingroups": [
        "admingroups/commands/manage.group"
      ]
    }
  }
```

Tile on the landing page

- src/main/smartview/src/bundles/admingroups-bundle.js - A portion of this file refers to the the RequireJS modules that implement the landing page tile. These references are only used for RequireJS modules that are otherwise not referenced statically from any other RequireJS module.

```
define([
   ...
   'admingroups/extensions/admin.groups.icon.sprites',
   ...
   'admingroups/extensions/admin.groups.tile',
   'admingroups/widgets/admin.groups/admin.groups.view'
   ...
   'json!admingroups/widgets/admin.groups/admin.groups.manifest.json',
   'i18n!admingroups/widgets/admin.groups/impl/nls/admin.groups.manifest'
], {});
```

- src/main/smartview/src/extensions/admin.groups.icon.sprites.js - Defines the icon to represent a user and a group by means of extension.
- src/main/smartview/src/extensions/admin.groups.tile.js - Declares a new widget type handler for the landing page tiles by means of extension.
- src/main/smartview/src/admingroups-extensions.json - A portion of this file registers extensions to the landing page tile & icon related D2SV UI API. The corresponding portion is highlighted below

```
"d2/sdk/utils/landingpage/tiles": {
    "extensions": {
      "admingroups": [
        "admingroups/extensions/admin.groups.tile"
      ]
    }
  },
  ...
  "d2/sdk/controls/icon.sprites/node.icon.sprites": {
      "extensions": {
        "admingroups": [
          "admingroups/extensions/admin.groups.icon.sprites"
        ]
      }
    }
```

- src/main/smartview/src/models/admin.groups.collection.js - A `BackboneJS` collection that holds all available groups data. Uses `group.model.js` to represent each group in the collection. Makes an

AJAX call to one of the REST endpoint, created by Java code from above, to get the available groups data.

- src/main/smartview/src/models/group.model.js - A `BackboneJS` model that holds data for a single group.
- src/main/smartview/src/utils/contexts/factories/admin.groups.collection.factory.js - A factory to control creation of initialized/uninitialized group collection instances.
- src/main/smartview/src/utils/contexts/factories/next.group.factory.js - A factory to control creation of group-model instances. Purpose of this factory is to create a single instance of group model and use that to reflect current selected group item in the UI. This instance is used by `admin.groups.view.js` to constantly update the selected group data as user clicks an item in the list of visible groups in UI.
- src/main/smartview/src/utils/constants.js - A portion of the file defines a few frequently used constant values in the context of landing tile implementation.
- src/main/smartview/src/widgets/admin.groups/admin.groups.view.js - A `MarionetteJS` view that implements the UI and function for the widget. An instance of this view is dynamically created by the D2SV runtime and this instance manages and renders DOM elements to represent the list of all available groups. It also handles user interaction with the DOM elements.
- src/main/smartview/src/widgets/admin.groups/toolitems.js - The toolbar configuration used by `admin.groups.view.js` to display inline `Manage` menu. An instance of it is manipulated by `menu.utils.js` to dynamically inject the menu items in the toolbar at the time of D2SV UI startup.

The perspective, landing tile expands into

The perspective is defined in a two panel layout where the left-side re-uses the same `admin.groups.view.js` from landing page tile. Apart from the RequireJS modules and other source code resources referred by the landing page tile, here's the other files involved in defining the perspective itself besides the right-side part of it.

- src/main/smartview/src/bundles/admingroups-bundle.js - A portion of this file refers to the the RequireJS modules that implement the perspective and right-side part of it. These references are only used for RequireJS modules that are otherwise not referenced statically from any other RequireJS module.

```
define([
   ...
   'admingroups/extensions/admin.groups.perspective',
   ...
   'admingroups/widgets/admin.groups.members/admin.groups.members.view',
   'json!admingroups/utils/perspectives/admin.groups.perspective.json',
   ...
], {});
```

- src/main/smartview/src/extensions/admin.groups.perspective.js - Declares an application scope handler and associates a perspective definition file with it.
- src/main/smartview/src/admingroups-extensions.json - A portion of this file registers extensions toward application scope perspective related D2SV UI API. The corresponding portion is highlighted below

```
"d2/sdk/utils/perspectives/app.scope.perspectives": {
    "extensions": {
      "admingroups": [
        "admingroups/extensions/admin.groups.perspective"
      ]
    }
  }
```

- src/main/smartview/src/models/group.members.collection.js - A `BackboneJS` collection that holds membership data for a given group. Uses `member.model.js` to represent each member within the group. Makes an AJAX call to one of the REST endpoint, created by Java code from above, to get the members data for a selected group.
- src/main/smartview/src/models/member.model.js - A `BackboneJS` model that holds data for a member within a group.
- src/main/smartview/src/utils/contexts/factories/next.group.factory.js - A factory to control creation of group-model instances. Purpose of this factory is to create a single instance of group model and use that to reflect current selected group item in UI. This instance is used by `admin.groups.members.view.js` to constantly monitor change to the selected group in UI.
- src/main/smartview/src/utils/perspectives/admin.groups.perspective.json - Defines the layout for the perspective and associates `admin.groups` & `admin.groups.members` as the widgets to go on

the left and right side respectively. The D2SV runtime dynamically creates the associated view instances when the perspective comes alive.

- src/main/smartview/src/widgets/admin.groups.members/admin.groups.members.view.js - A `MarionetteJS` view implementation that displays the members of a selected group. An instance of this view is dynamically created by D2SV runtime to show list of members in the perspective. The instance automatically updates itself as a result of end users selecting a group from the left-side by means of constant watch over the group model instance acquired using `next.group.factory.js`.

The side-panel dialog that manages group member

- src/main/smartview/src/bundles/admingroups-bundle.js - A portion of this file refers to the the RequireJS modules that implements the dialog. These references are only used for RequireJS modules that are otherwise not referenced statically from any other RequireJS module.

```
define([
    ...
    'admingroups/dialogs/manage.group/manage.group.dialog',
    ...
], {});
```

- src/main/smartview/src/dialogs/manage.group/manage.group.dialog.js - Uses D2SV UI API to create a side panel and host an instance of `manage.group.view.js` to show the related UI. Also collects information about updated group members and relays that to the caller.
- src/main/smartview/src/dialogs/manage.group/manage.group.view.js - A `MarionetteJS` view that wraps an instance of `group.members.form.view.js` to make it renderable within the side panel and defers the instance creation & rendering until required membership data has been fetched through an instance of `group.members.collection.js`. Also defines utility methods to have a check on whether the membership data has changed from what is loaded initially. Uses `manage.group.hbs` & `manage.group.css` files for HTML templating and CSS styling respectively.
- src/main/smartview/src/dialogs/manage.group/impl/group.members.form.view.js - Uses D2SV UI API to create a statically defined form with multi-select list field to show membership information for the selcted group. Also makes an AJAX call to one of the REST endpoint, created by Java code from above, to get all available users data that serves as the options shown while editing the

membership data. Also defines utility method to get the membership information for the selected group at any time.

D2SV client to server logging

D2SV UI uses `log4javascript` to enable logging for UI components. By default, the library is configured to channel log output to web browser console. In the past while debugging for some issue in D2 Smartview, we felt the need to correlate this client-side log output with server-side log output generated by back-end components and usually saved in "D2-Smartview.log" file. Driven by this need, we've created this sample plugin which re-configures the `log4javascript` and channels the log output to the same server-side log file. Key concepts explored in this plugin are

- REST endpoint with un-conventional input/output.
- RequireJS module re-configuration

Instruction to try out the sample

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2SV-Client2Server-Logging-1.0.0.jar` from "dist" folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Edit D2 Smartview logging configuration file `logback.xml` from `WEB-INF/classes` folder and set the root logging level to `INFO`.
- Edit `rest-api-runtime.properties` from `WEB-INF/classes` folder and add/append pattern `/clientlog` to the value of property `rest.security.anonymous.url.patterns`.
- Restart application server on which D2 Smartview is deployed.
- Reload D2-Smartview application in web-browser with additional query parameter `loglevel=info`.

  > **TIP**
  >
  > Complete URL might look like `https://mydomain.com/D2-Smartview/ui?loglevel=info#d2`

- Open console for the web-browser and check if some INFO level log output is present.
- On the server-side open **D2-Smartview.log** file and search for the same log statements as from web-browser console.

Source code structure

```
D2SV-Client2Server-Logging
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_C2SLogging.java
|       |               |
|       |               \---smartview
|       |                   \---c2slogging
|       |                       |   C2SLoggingPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       C2SLoggingVersion.java
|       |                       |
|       |                       \---rest
|       |                           |   package-info.java
|       |                           |
|       |                           +---api
|       |                           |   |   IClientLogManager.java
|       |                           |   |
|       |                           |   \---impl
|       |                           |           ClientLogManager.java
|       |                           |
|       |                           +---controller
|       |                           |       InboundExternalLogController.java
|       |                           |
|       |                           \---model
|       |                                   HelpModel.java
|       |                                   LogEntry.java
|       |                                   LogLevel.java
|       |                                   LogRequest.java
|       |
|       +---resources
|       |   |   c2slogging-version.properties
|       |   |   D2Plugin.properties
|       |   |
|       |   \---smartview
|       |           SmartView.properties
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   c2slogging.setup.js
|           |   config-editor.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   c2slogging-extensions.json
|           |   |   c2slogging-init.js
|           |   |   component.js
|           |   |   config-build.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       c2slogging-bundle.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |
|           |       \---theme
|           |           |   action.icons.js
|           |           |
|           |           \---action_icons
|           |                   action_sample_icon.svg
|           |
|           \---test
|                   Gruntfile.js
|                   karma.conf.js
```

```
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

REST Controller

- src/main/java/com/opentext/d2/rest/context/jc/PluginRestConfig_C2SLogging.java - Declares Spring Bean `IClientLogManager` through `ClientLogManager`.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/api/IClientLogManager.java - Declares log manager interface for REST controllers to use.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/api/impl/ClientLogManager.java - Log manager implementation that parses and maps input log statements and relays those statements into server side log.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/controller/InboundExternalLogController.java - Defines two REST endpoints, one receives HTTP POST request with log statements as part of request body, the other responds to HTTP GET requests with help information on how to use the first endpoint.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/model/HelpModel.java - Serializable POJO that holds help information.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/model/LogEntry.java - Serializable POJO that represents a single log statement.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/model/LogLevel.java - Enum that represents D2SV client-side log levels.
- src/main/java/com/opentext/d2/smartview/c2slogging/rest/model/LogRequest.java - Serializable POJO that holds a bunch of log statements together.

RequireJS module configuration

- src/main/smartview/src/c2slogging-init.js - This file is used to re-configure module `nuc/utils/log` so that it channels log statements to the endpoint created by Java code from above. It also configures `nuc/lib/log4javascript` to customize the request body format sent to the REST endpoint.

  INFO

  The module `nuc/utils/log` encapsulates the `log4javascript` library and provides managed logging API to D2SV UI components.

opentext™

D2SV Custom Dialogs(D2FS) sample

D2 Custom Dialog sample provide an option to modify the metadata for a document with any available properties page created in D2-Config. As out of the box, the document metadata can be modified only using the properties page which is resolved after configuration matrix against the context.

This sample shows

- How to define a D2 Dialog service plugin which implements ID2fsPlugin.
- How to define a D2FS state method to make dialog chaining as context less. So that the last step Submit will be performed on OOTB property dialog service instead of original dialog service.

Instruction to try out the sample

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2SV-Custom-Dialogs-1.0.0.jar` from 'dist' folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Restart application server on which D2 Smartview is deployed.
- Open D2-Config web application in browser, login and then create few properties page configuration.
- Login into D2-Smartview and navigate to any Doclist widget view.
- Select an object and locate & click menu item `Show advance properties` from the selection toolbar to open `Selective property page view` dialog.
- In the `Select view` dropdown, choose any property page name and click `Show` button.
- The dialog should show selected object's metadata as per the selected property page name.

Source code structure

```
D2SV-Custom-Dialogs
|
|   pom.xml
|
+---src
```

```
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_D2SVDialogs.java
|       |               |
|       |               \---smartview
|       |                   \---d2svdialogs
|       |                       |   D2SVDialogsPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       D2SVDialogsVersion.java
|       |                       |
|       |                       +---dialogs
|       |                       |       SelectivePropertyDisplay.java
|       |                       |
|       |                       +---rest
|       |                       |       package-info.java
|       |                       |
|       |                       \---webfs
|       |                           \---dialog
|       |                                   D2DialogServicePlugin.java
|       |
|       +---resources
|       |   |   D2Plugin.properties
|       |   |   d2svdialogs-version.properties
|       |   |
|       |   +---smartview
|       |   |       SmartView.properties
|       |   |
|       |   +---strings
|       |   |   +---dialog
|       |   |   |   \---SelectivePropertyDisplay
|       |   |   |           SelectivePropertyDisplay_en.properties
|       |   |   |
|       |   |   \---menu
|       |   |       \---MenuContext
|       |   |               MenuContext_en.properties
|       |   |
|       |   \---xml
|       |       +---dialog
|       |       |       SelectivePropertyDisplay.xml
|       |       |
|       |       \---unitymenu
|       |               MenuContextDelta.xml
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   d2svdialogs.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   d2svdialogs-extensions.json
|           |   |   d2svdialogs-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       d2svdialogs-bundle.js
|           |   |
|           |   +---dialogs
|           |   |   \---d2fs
|           |   |           context.less.d2fs.state.method.js
|           |   |
|           |   +---extensions
|           |   |       dialog.state.methods.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |
|           |       \---theme
|            |            |   action.icons.js
|            |            |
|            |            \---action_icons
|            |                    action_sample_icon.svg
|            |
|            \---test
|                    Gruntfile.js
|                    karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

Java Classes

- src/main/java/com/opentext/d2/smartview/d2svdialogs/dialogs/SelectivePropertyDisplay.java - Dialog class which implements ID2Dialog to serve the dialog for selecting the properties page configuration.
- src/main/java/com/opentext/d2/smartview/d2svdialogs/webfs/dialog/D2DialogServicePlugin.java - Dialog service class which implements ID2fsPlugin interface for validating the dialog request.

Dialog form definition

- src/main/resources/xml/dialog/SelectivePropertyDisplay.xml - Defines the form structure for rendering "SelectivePropertyDisplay" dialog. The same file will be processed in "src/main/java/com/opentext/d2/smartview/d2svdialogs/dialogs/SelectivePropertyDisplay.java"
- src/main/resources/strings/dialog/SelectivePropertyDisplay/SelectivePropertyDisplay_en.properties - Label associated with the dialog.

Custom dialog menu configuration in back-end

- src/main/resources/strings/menu/MenuContext/MenuContext_en.properties - Labels associated with the dynamically configured menu.

- src/main/resources/xml/unitymenu/MenuContextDelta.xml - The menu definition file that dynamically adds a new type(MenuContext) of menu for the D2FS `D2MenuService` to discover and return for D2 Smartview.

Dialog state method override

As part of dialog state customization added extension for dialog state methods. This state method will be resolved based on "SelectivePropertyDisplay" dialog name. Intension of having custom dialog state method to override the default behavior of dialog state. With this override dialog state is decoupled between first form and second form.

- src/main/smartview/src/dialogs/d2fs/context.less.d2fs.state.method.js - This is a client side JavaScript file extends "d2/sdk/controls/dialogs/generic/d2fs.state.method". Dialog context is decoupled by having dummy override for "setDialogContextForm()" method.
- src/main/smartview/src/extensions/dialog.state.methods.js - This file is having rule for resolving the dialog state method based on dialog name.
- src/main/smartview/src/d2svdialogs-extensions.json - Adding the rule for dialog.state.method.

```
"d2/sdk/controls/dialogs/generic/dialog.state.methods": {
  "extensions": {
    "d2svdialogs": [
      "d2svdialogs/extensions/dialog.state.methods"
    ]
  }
}
```

D2SV Read-Only Permission View Sample

D2 Read-Only permission view sample plugin fills-in the gap functionally and serves as a complete example of how to use SDK to

- Define a custom action services
- Define a custom menu by default and initiate a dialog
- Define a custom dialog view to display the information and show the form view of the data

Instruction to try out the sample

Developer can extract the sample and build it using the workspace assistant. Once built, the distribution is collected in 'dist' folder as **D2SV-ReadOnlyPermission-View-1.0.0.jar** which can placed in `WEB-INF/lib` directory of a deployed D2 Smartview. The application server needs to be restarted post deployment.

As result of deploying this plugin, it will introduce a new menu in `Doclist` widget as `View Permission`

Source code structure

```
D2SV-ReadOnlyPermission-View
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_D2SVROPView.java
|       |               |
|       |               \---smartview
|       |                   \---d2svropview
|       |                       |   D2SVROPViewPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       D2SVROPViewVersion.java
|       |                       |
|       |                       +---rest
|       |                       |       package-info.java
|       |                       |
|       |                       \---webfs
|       |                           \---custom
|       |                                   PermissionActionService.java
|       |
|       +---resources
|       |   |   D2Plugin.properties
|       |   |   d2svropview-version.properties
|       |   |
|       |   +---smartview
|       |   |       SmartView.properties
|       |   |
|       |   +---strings
|       |   |   \---menu
|       |   |       \---MenuContext
|       |   |               MenuContext_en.properties
|       |   |
|       |   \---xml
|       |       \---unitymenu
|       |               MenuContextDelta.xml
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   d2svropview.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   d2svropview-extensions.json
|           |   |   d2svropview-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       d2svropview-bundle.js
|           |   |
|           |   +---commands
|           |   |   |   node.actions.rules.js
|           |   |   |   view.permission.js
|           |   |   |
|           |   |   \---impl
|           |   |       \---nls
|           |   |           |   lang.js
|           |   |           |
|           |   |           \---root
|           |   |                   lang.js
|           |   |
|           |   +---dialogs
|           |   |   \---permissions
|           |   |       |   permissions.dialog.js
|           |   |       |   permissions.view.js
|           |   |       |
|           |   |       \---impl
|           |   |           |   permission.collection.js
|           |   |           |   permissions.css
|           |   |           |   permissions.hbs
|           |   |           |   table.columns.js
|           |   |           |
|           |   |           \---nls
|           |   |               |   lang.js
|           |   |               |
|           |   |               \---root
|           |   |                       lang.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |
|           |       \---theme
|           |           |   action.icons.js
|           |           |
|           |           \---action_icons
|           |                   action_sample_icon.svg
|           |                   action_view_perms32.svg
|           |
|           \---test
|                   Gruntfile.js
|                   karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

REST Implementations

- pom.xml - Defines the maven project for this plugin.
- src/main/java/com/emc/D2PluginVersion.java - Declares identification information for the entire plugin using `D2SVROPViewVersion` class
- src/main/java/com/opentext/d2/rest/context/jc/PluginRestConfig_D2SVROPView.java - Java configuration for spring components like REST controller, Beans etc.
- src/main/java/com/opentext/d2/smartview/d2svropview/D2SVROPViewPlugin.java - Declares a plugin component for D2FS.
- src/main/java/com/opentext/d2/smartview/d2svropview/api/D2SVROPViewVersion.java - Holder for plugin identification information. Loads relevant data from `d2svropview-version.properties` file resource.
- src/main/java/com/opentext/d2/smartview/d2svropview/rest/package-info.java - Declares package metadata for JDK and IDE.
- src/main/java/com/opentext/d2/smartview/d2svropview/rest/webfs.custom/PermissionActionService.java - Defines a custom service to fetch the basic permissions for the given object id. So the menu will have reference to the method 'getPermissions' which will triggered from the menu action.

View permission menu configuration in back-end and its display & execution on the front-end

- src/main/resources/strings/menu/MenuContext/MenuContext_en.properties - Labels associated with the dynamically configured menu.
- src/main/resources/xml/unitymenu/MenuContextDelta.xml - This delta menu will be used to dynamically load the custom OOTB menu to view permissions in the D2 Smartview for the default MenuContext.

```
<delta>
  <insert position-before="menuToolsMassUpdate">
    <menuitem id="menuContextViewPermission">
      <dynamic-action class="com.emc.d2fs.dctm.ui.dynamicactions.actions.U4Generic"
eMethod="getPermissions" eMode="SINGLE" eService="PermissionActionService"
rAction="d2svropview/dialogs/permissions/permissions.dialog:showPermissions"
rType="JS"/>
      <condition class="com.emc.d2fs.dctm.ui.conditions.interfaces.IsMultipleSelection"
equals="false"/>
      <condition class="com.emc.d2fs.dctm.ui.conditions.options.IsPluginActivated"
value="D2SV-ReadOnlyPermission-View"/>
    </menuitem>
  </insert>
  <insert position-before="menuToolsMassUpdate">
    <separator/>
  </insert>
</delta>
```

Here the `dynamic-action` is used to map the method `getPermissions` in `PermissionActionService` when the menuAction is triggered from UI. `dynamic-action` will also have reference to the target UI action to perform using the `rAction`

- src/main/smartview/src/bundles/d2svropview-bundle.js - A portion of this file is used to refer to key RequireJS modules that define the extensions to the toolbar and menu related D2SV UI API.

```
define([
    'd2svropview/utils/theme/action.icons',
    'd2svropview/utils/startup',
    'd2svropview/commands/node.actions.rules',
    'd2svropview/commands/view.permission',
], {});
```

- src/main/smartview/src/commands/view.permission.js - A `CommandModel` that implements the executable logic when a user clicks the `View Permission` menu on the UI. It is an extension of `CallServiceCommand` which is used to take care of the forming the service method request
- src/main/smartview/src/commands/node.actions.rules.js - Defines client side filtering and association logic to attach the above `view.permission.js` command model implementaion to a UI toolbar item.
- src/main/smartview/src/utils/startup.js - Runs as part of D2 Smartview client-side startup flow. This flow is executed everytime endusers reload the D2 Smartview application in their internet browser.
- src/main/smartview/src/d2svropview-extensions.json - A portion of this file registers extensions to the toolbar and menu related D2SV UI API. The corresponding portion is highlighted below

```
  "d2/sdk/controls/action.icons/action.icons": {
    "extensions": {
      "d2svropview": [
        "d2svropview/utils/theme/action.icons"
      ]
    }
  },
  "d2/sdk/commands/node.actions.rules": {
    "extensions": {
      "d2svropview": [
        "d2svropview/commands/node.actions.rules"
      ]
    }
  },
  "d2/sdk/utils/commands": {
    "extensions": {
      "d2svropview": [
        "d2svropview/commands/view.permission"
      ]
    }
  }
```

The side-panel dialog that displays permissions

- src/main/smartview/src/bundles/d2svropview-bundle.js - A portion of this file is used to refer to key RequireJS modules that define the extensions to dialog used for the side panel as part of the response from the menu action.

```
define([
    ...
    'd2svropview/dialogs/permissions/permissions.dialog'
], {});
```

- src/main/smartview/src/dialogs/permissions/permissions.dialog.js - This dialog will be used to show a stepper wizard view. The stepper wizard will be a single step having `permissions.view`.
- src/main/smartview/src/dialogs/permissions/permissions.view.js - The view will be used to render data as table view. Data returned as part fo the response from the `PermissionActionService` be managed by a `MarionetteJS` `Collection` `permission.collection.js`.
- src/main/smartview/src/dialogs/permissions/impl/table.columns.js - Its a `Backbone JS` collection which is used to map the columns information such as the key,title etc

Example:

```
{
      key: 'base_permission',
      column_key: 'base_permission',
      sequence: 3,
      sort: false,
      definitions_order: 3,
      title: lang.colNameBasePermissions,
      type: -1,
      widthFactor: 0.7,
      permanentColumn: true // don't wrap column due to responsiveness into details row
}
```

- src/main/smartview/src/d2svropview-extensions.json - A portion of this file registers extensions toward collections used in handling the list of permissions. The corresponding portion is highlighted below

```
"d2/sdk/models/module.collection": {
    "modules": {
      "d2svropview": {
        "title": "D2SV-ReadOnlyPermission-View",
        "version": "1.0.0"
      }
    }
  }
```

- src/main/smartview/src/dialogs/permissions/impl/permissions.collection.js - Collection is used to parse the unformatted response data from the `PermissionActionService` to collection of models. This collection is included in `BrowsableMixin` to have filter and sorting capability of the result set.

opentext™

D2SV Custom Widget Type Tile

D2SV custom widget type tile plugin will help the developer in solving the following scenarios

- Define a custom widget type
- Define a custom widget type parameter
- Define a custom widget type in the landing pange
- Define a shortcut tile with behavior to access the custom widget type parameter.

Instruction to try out the sample

Developer can extract the sample and build it using the workspace assistant. Once built, the distribution is collected in 'dist' folder as **D2SV-Custom-Widget-Type-1.0.0.jar** which can be placed in `WEB-INF/lib` directory of a deployed D2 Smartview. The same plugin jar file has to be placed in `WEB-INF/classes/plugins` directory of your deployed D2-Config application. Subsequently, edit `WEB-INF/classes/D2-Config.properties` and add an entry like `plugin_<sequence>=plugins/D2SV-Custom-Widget-Type-1.0.0.jar` where `<sequence>` is the next natural number following the existing entries of the same pattern in the file. The application server needs to be restarted post deployment.

After restart -

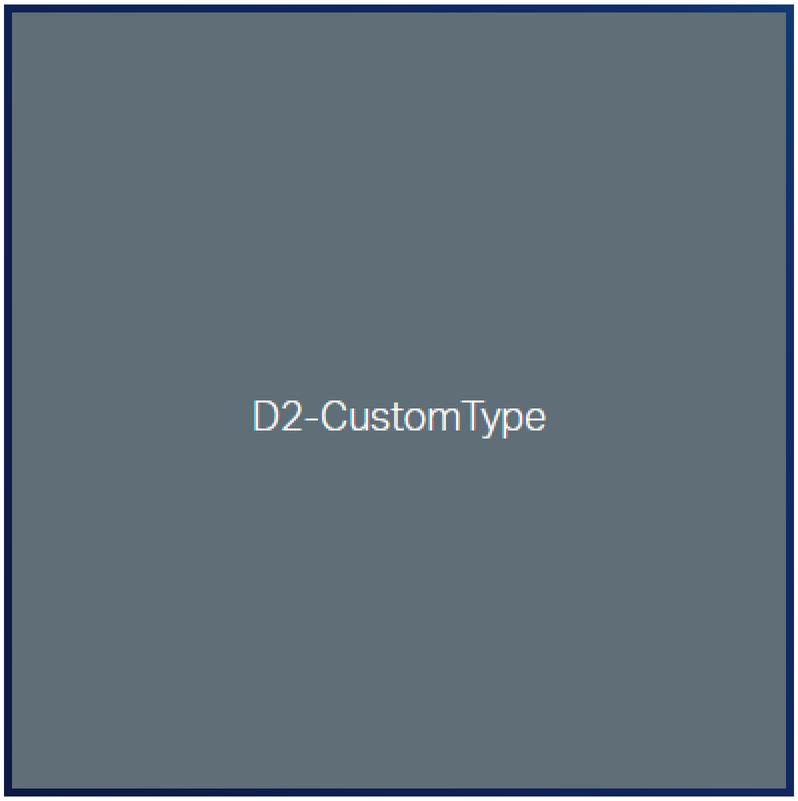
- Login into D2-Config and navigate to `Widget view -> widget` from menubar.
- Create a new widget configuration and let's name it `D2-CustomType` and select value `CustomType` for the dropdown `Widget type`.
- Fill in other fields like `Applications`, `label` etc. as desired.
- Take a note of the value in `Sample text field 1` field.
- Save the configuration.
- From the toolbar, click **Matrix** to go to D2-config matrix and enable the widget configuration, you just created, against appropriate contexts.
- Navigate to `Widget view -> Smart View Landing Page` from menu bar and then download your applicable landing page configuration xml file.

TIP

If a pre-created Smartview landing page configuration does not exist, then refer to D2 Administration Guide documentation to create the same and learn basics of landing page structure file.

- Edit your landing page configuration file and place the following xml anywhere right under `<context>` tag

```
<tile-container>
    <tile name="D2-CustomType" type="CustomType">
        <theme color="shade1" type="stone1"/>
    </tile>
</tile-container>
```

- Save the landing structure xml file and upload it to D2-Config under the same landing page configuration from where we downloaded it before.
- Save the configuration change in D2-Config and click **Tools -> Refresh cache** from menubar.
- Reloading the D2 Smartview at this point should show an additional shortcut in the landing page, similar to following 

- Click on the shortcut to see a browser alert with message `custom parameter sample1 : Hello World`. The value `Hello World` comes from the `Sample text field 1` field in the widget configuration we created above.

As result of deploying this plugin, it will introduce a new widget type in the D2-Config widget types. And the other part of the plugin binds this new widget type to a new shortcut tile in D2-Smartview landing page.

Source code structure

```
D2SV-Custom-Widget-Type
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_CustomWidgetType.java
|       |               |
|       |               \---smartview
|       |                   \---customwidgettype
|       |                       |   CustomWidgetTypePlugin.java
|       |                       |
|       |                       +---api
|       |                       |       CustomWidgetTypeVersion.java
|       |                       |
|       |                       \---rest
|       |                               package-info.java
|       |
|       +---resources
|       |   |   customwidgettype-version.properties
|       |   |   D2Plugin.properties
|       |   |
|       |   +---smartview
|       |   |       SmartView.properties
|       |   |
|       |   \---strings
|       |       +---actions
|       |       |   \---config
|       |       |       \---modules
|       |       |           \---widget
|       |       |               \---WidgetDialog
|       |       |                       WidgetDialog_en.properties
|       |       |
|       |       \---config
|       |               U4Landing.properties
|       |               WidgetSubtype.properties
|       |               WidgetSubtypelist.properties
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   customwidgettype.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   customwidgettype-extensions.json
|           |   |   customwidgettype-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       customwidgettype-bundle.js
|           |   |
|           |   +---extensions
|           |   |       custom.type.tile.behaviors.js
|           |   |       custom.type.tiles.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   +---utils
|           |   |   |   startup.js
|           |   |   |
|           |   |   \---theme
|           |   |       |   action.icons.js
|           |   |       |
|           |   |       \---action_icons
|           |   |               action_sample_icon.svg
|           |   |
|           |   \---widgets
|           |       \---shortcut.tile
|           |               custom.type.shortcut.behavior.js
|           |
|           \---test
|                   Gruntfile.js
|                   karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

Custom widget type references needed for the D2-Config widget configuration and D2 Smartview landing page configuration

- src/main/resources/strings/config/U4Landing.properties - Defines the custom widget type which are supported

```
shortcut_types=CustomType
```

- src/main/resources/strings/config/WidgetSubtypelist.properties - Defines the custom widget type

```
CustomType=true
```

- src/main/resources/strings/config/WidgetSubtype.properties - Defines the custom widget type parameter for the custom widget types mentioned which are supported Here the default value for the paramter can also be provided which can be changed in the D2 Config widget configuration

```
CustomType.sample1 = Hello World
```

- src/main/resources/strings/actions/config/modules/widget/WidgetDialog/WidgetDialog_en.properties - This properties file will contain the labels used for parameters for the custom type

```
label_sample1 = Sample text field 1
```

D2 Smartview UI changes for the plugin

- src/main/smartview/src/bundles/customwidgettype-bundle.js - A portion of this file is used to refer to key RequireJS modules that define the extensions shortcut behavior API and click of the widget tile

```
define([
    'customwidgettype/utils/theme/action.icons',
    'customwidgettype/utils/startup',
    'customwidgettype/extensions/custom.type.tiles',
    'customwidgettype/extensions/custom.type.tile.behaviors'
], {});
```

- src/main/smartview/src/utils/theme/action.icons.js - Defines the default icon for the widgets
- src/main/smartview/src/utils/startup.js - Runs as part of D2 Smartview client-side startup flow. This flow is executed everytime end users reload the D2 Smartview application in their internet browser.
- src/main/smartview/src/customwidgettype-extensions.json - A portion of this file registers extensions to enable the custom shortcut tile for the widget configuration and also to have custom shortcut tile behavior.

```
{
  "d2/sdk/utils/landingpage/tiles": {
    "extensions": {
      "customwidgettype": [
        "customwidgettype/extensions/custom.type.tiles"
      ]
    }
  },
  "d2/sdk/widgets/shortcut.tile/shortcut.tile.behaviors": {
    "extensions": {
      "customwidgettype": [
        "customwidgettype/extensions/custom.type.tile.behaviors"
      ]
    }
  }
}
```

- src/main/smartview/src/extensions/custom.type.tiles.js - This will define the custom widget type to the tile containers in the UI

```
define(['d2/sdk/utils/widget.constants'], function(widgetConstants) {
    'use strict';

    function validateConfigCustomType0() {
        var validation = {
            status: true
        };
        // TODO: Validates widgetConfig. Set validation.status = false if the validation
should fail.
        return validation;
    }
    // List of landing tile definitions
    return [{
        type: 'CustomType',
        icon: 'custom-widget-type',
        isShortcut: true,
        tileConfigType: widgetConstants.TileConfigTypes.WIDGET,
        validateConfig: validateConfigCustomType0
    }];
});
```

- src/main/smartview/src/extensions/custom.type.tile.behaviors.js - This will define the custom shortcut behavior to the custom widget type `CustomType`

```
define(['customwidgettype/widgets/shortcut.tile/custom.type.shortcut.behavior'],
function(CustomTypeShortcutBehavior) {
    'use strict';
    return [{
        type: 'CustomType',
        behaviorClass: CustomTypeShortcutBehavior
    }];
});
```

- src/main/smartview/src/widgets/shortcut.tile/custom.type.shortcut.behavior.js -Define custom shortcut behavior with the onclick action to perform. This will prompt the user with the default parameter value configured for the `CustomType`

```
define([
    'd2/sdk/widgets/shortcut.tile/shortcut.tile.behavior'
], function(ShortcutTileBehaviorImpl){
    'use strict';

    var CustomTypeShortcutBehavior = ShortcutTileBehaviorImpl.extend({
        constructor: function CustomTypeShortcutBehavior() {
            CustomTypeShortcutBehavior.__super__.constructor.apply(this, arguments);
        },
        onClick: function() {
            alert('custom parameter sample1 : '+this.options.widgetParams.sample1);
        }
    });

    return CustomTypeShortcutBehavior;
});
```

Custom Table Cell View sample

Custom Table cell view provides option to render column specific custom cell layout. With this cell view can be visually improved. As out of the box, document modified user column shows information only in text. In this sample modified user information is shown with initials.

This sample shows

- How to define a Custom table cell view implementation.

Instruction to try out the sample

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2-CustomTableCell-1.0.0.jar` from 'dist' folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Restart application server on which D2 Smartview is deployed.
- Login into D2-Smartview and navigate to any Doclist widget view.
- Using the 'Table settings' add column `Modified By` to display, if not already added.
- Check, the `Modified By` column shows with a colored letter-avatar-icon along with the textual username(only username is shown without this plugin).

Source code structure

```
D2-CustomTableCell
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_CustomTableCell.java
|       |               |
|       |               \---smartview
|       |                   \---customtablecell
|       |                       |   CustomTableCellPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       CustomTableCellVersion.java
|       |                       |
|       |                       \---rest
|       |                               package-info.java
|       |
|       +---resources
|       |   |   customtablecell-version.properties
|       |   |   D2Plugin.properties
|       |   |
|       |   \---smartview
|       |           SmartView.properties
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   customtablecell.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   customtablecell-extensions.json
|           |   |   customtablecell-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       customtablecell-bundle.js
|           |   |
|            |   +---table
|            |   |   \---cell
|            |   |       \---modified.by
|            |   |           |   modified.by.view.js
|            |   |           |
|            |   |           \---impl
|            |   |                   modified.by.css
|            |   |                   modified.by.hbs
|            |   |
|            |   +---test
|            |   |       extensions.spec.js
|            |   |
|            |   \---utils
|            |       |   startup.js
|            |       |
|            |       \---theme
|            |           |   action.icons.js
|            |           |
|            |           \---action_icons
|            |                   action_sample_icon.svg
|            |
|            \---test
|                    Gruntfile.js
|                    karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

Custom table cell view

- src/main/smartview/src/table/cell/modified.by/modified.by.view.js - Define the custom cell view implementation.
- src/main/smartview/src/table/cell/modified.by/impl/modified.by.hbs - Handlebar template for custom cell view.

- src/main/smartview/src/table/cell/modified.by/impl/modified.by.css - CSS for styling custom cell view.
- src/main/smartview/src/utils/startup.js - Loaded "modified.by.view.js" in "startup.js".

opentext™

D2SV Object On Click Sample

Object on click actions extension provides option override the default action on the object. With this default action for an object can be controlled with a predicate condition. So default action can be defined based on rules.

This sample shows

- How to customize the default action for an object.

Instruction to try out the sample

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2SV-Object-OnClick-Actions-1.0.0.jar` from 'dist' folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Restart application server on which D2 Smartview is deployed.
- Login into D2-Smartview and navigate to any Doclist widget view.
- Execute default action for PDF file and content less object.
- While executing default action for .txt document, Modal alert will be shown.
- While executing default action for .png document, Toast message will be shown.

Source code structure

```
D2SV-Object-OnClick-Actions
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |
PluginRestConfig_ObjectOnClickActions.javations.javal.java
|       |               |
|       |               \---smartview
|       |                   \---objectonclickactions
|       |                       |   ObjectOnClickActionsPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       ObjectOnClickActionsVersion.java
|       |                       |
|       |                       \---rest
|       |                               package-info.java
|       |
|       +---resources
|       |   |   D2Plugin.properties
|       |   |   objectonclickactions-version.properties
|       |   |
|       |   \---smartview
|       |           SmartView.properties
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   Gruntfile.js
|           |   objectonclickactions.setup.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   Gruntfile.js
|           |   |   objectonclickactions-extensions.json
|           |   |   objectonclickactions-init.js
|           |   |
|           |   +---bundles
|           |   |       objectonclickactions-bundle.js
|           |   |
|           |   +---commands
|           |   |   |   open.modal.alert.js
|           |   |   |   open.toast.message.js
|           |   |   |
|           |   |   \---nls
|           |   |       |   lang.js
|           |   |       |
|           |   |       \---root
|           |   |               lang.js
|           |   |
|           |   +---extensions
|           |   |       object.onclick.actions.rules.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |
|           |       \---theme
|           |           |   action.icons.js
|           |           |
|           |           \---action_icons
|           |                   action_sample_icon.svg
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

D2SV Custom Default Actions Sample

- src/main/smartview/src/extensions/object.onclick.actions.rules.js - Defined ruled for on click action of a document.
- src/main/smartview/src/commands/open.modal.alert.js - Sample command for executing object on-click action with modal alert.
- src/main/smartview/src/commands/open.toast.message.js - Sample command for executing object on-click action with toast message.

Open a cabinet/folder in Doclist

This sample shows how to open the default Doclist widget in D2 Smartview at a specific cabinet or folder. This sample leverages a command implementation named `Browse` from D2 Smartview runtime to do so.

Instruction to try out the sample

- Build the plugin using `npm run build` from SDK workspace root.
- Copy `D2SV-OpenFolder-Doclist-1.0.0.jar` from 'dist' folder in workspace root and paste it inside `WEB-INF/lib` folder of a deployed D2 Smartview application.
- Restart application server on which D2 Smartview is deployed.
- Login into D2-Smartview and navigate to any cabinet or folder of choice in Doclist widget.
- Copy the Documentum Object ID of cabinet/folder from browser address bar.
- Navigate to landing page.
- Click user profile icon and select `OpenFolder` menu item.
- Paste the copied Object ID in `Enter folder ID` field of `OpenFolder` dialog.
- Click `Open` button in the dialog footer to close the dialog and open Doclist at the cabinet/folder identified by the pasted value.

Source code structure

```
D2SV-OpenFolder-Doclist
|
|   pom.xml
|
+---src
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_D2SVOFDoclist.java
|       |               |
|       |               \---smartview
|       |                   \---d2svofdoclist
|       |                       |   D2SVOFDoclistPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       D2SVOFDoclistVersion.java
|       |                       |
|       |                       +---dialogs
|       |                       |       OpenFolder.java
|       |                       |
|       |                       \---rest
|       |                               package-info.java
|       |
|       +---resources
|       |   |   D2Plugin.properties
|       |   |   d2svofdoclist-version.properties
|       |   |
|       |   +---smartview
|       |   |       SmartView.properties
|       |   |
|       |   +---strings
|       |   |   +---dialog
|       |   |   |   \---OpenFolder
|       |   |   |           OpenFolder_en.properties
|       |   |   |
|       |   |   \---menu
|       |   |       \---MenuUser
|       |   |               MenuUser_en.properties
|       |   |
|       |   \---xml
|       |       +---dialog
|       |       |       OpenFolder.xml
|       |       |
|       |       \---unitymenu
|       |               MenuUserDelta.xml
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   d2svofdoclist.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   d2svofdoclist-extensions.json
|           |   |   d2svofdoclist-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       d2svofdoclist-bundle.js
|           |   |
|           |   +---extensions
|           |   |       dialog.actions.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |   utils.js
|           |       |
|           |       \---theme
|           |           |   action.icons.js
|           |           |
|           |           \---action_icons
|           |                   action_sample_icon.svg
|           |
|           \---test
|                   Gruntfile.js
|                   karma.conf.js
|
\---target
```

Files and their purpose

Following are the list of function oriented source files and their purpose. Other source files present within the plugin are part of common infrastructure code and explained in Understanding D2SV plugin project.

Utility to open Doclist widget at a given folder

- src/main/smartview/src/utils/utils.js - Defines a function `openFolder` that in turn executes `Browse` command instance with a given folder ID set as an attribute to a `NodeModel` instance.

  > **TIP**
  >
  > The `Browse` command has a multi-faceted implementation i.e. it behaves diffrently based on type of the object. E.g Instead of cabinet/folder, if a document's object ID is used then it opens the Overview perspective for it.

D2FS dialog to collect folder ID

The remaining part of the sample defines a user profile menu item which shows a D2FS dialog having a single text field to collect cabinet/folder ID.

- src/main/java/com/opentext/d2/smartview/d2svofdoclist/dialogs/OpenFolder.java - Java class file behind the `OpenFolder` dialog.
- src/main/resources/strings/dialog/OpenFolder/OpenFolder_en.properties - Defines labels used in `OpenFolder` dialog.
- src/main/resources/strings/menu/MenuUser/MenuUser_en.properties - Defines label for the menu item in user profile.
- src/main/resources/xml/dialog/OpenFolder.xml - Form definition for the `OpenFolder` dialog.
- src/main/resources/xml/unitymenu/MenuUserDelta.xml - Defines the `OpenFolder` menu item under user profile through delta menu concept.
- src/main/smartview/src/extensions/dialog.actions.js - Defines the code to be executed for `Open` button in the dialog footer. Invokes `openFolder` in `src/main/smartview/src/utils/utils.js` with collected ID.

Action icons catalog

Here is a list of built-in action icons from D2 Smartview runtime and its framework.

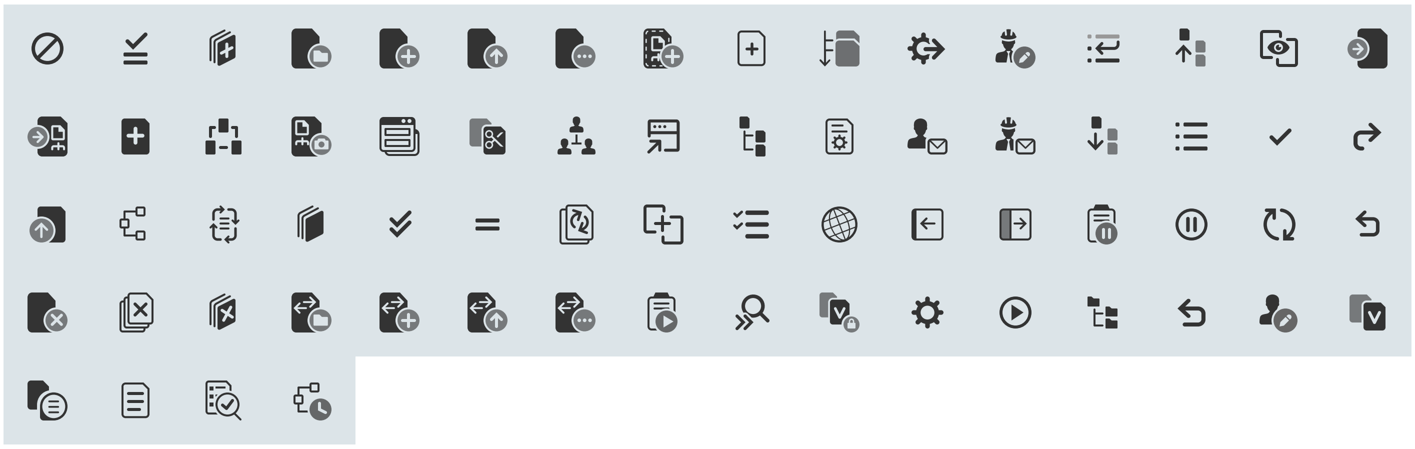
CAUTION

Action icons listed under CSUI & SVF may change without notice.

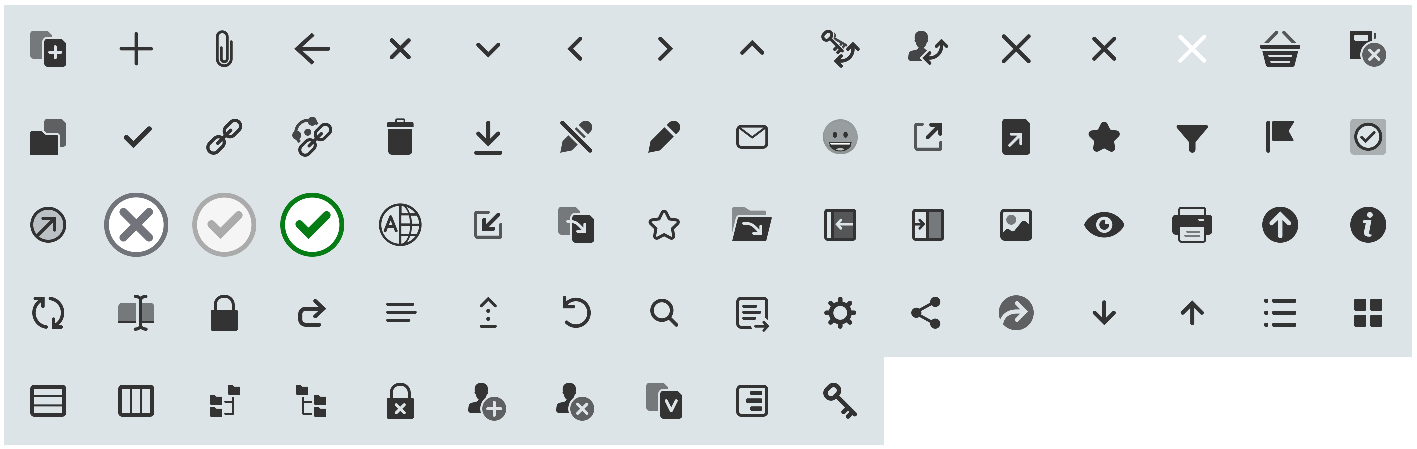
TIP

Hovering with your mouse over an icon below shows the icon's referrable name as tooltip.

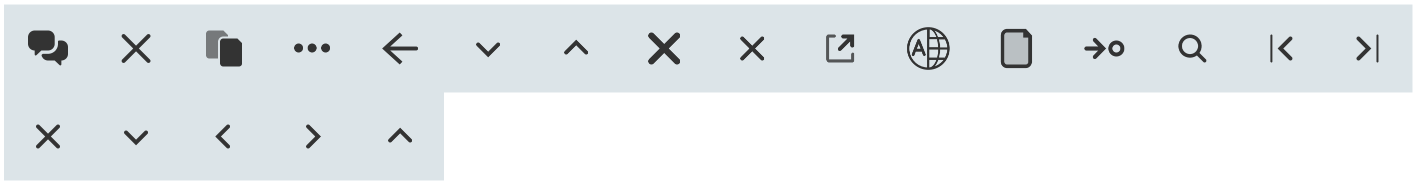
D2 Smartview icons



CSUI icons



SVF icons



opentext™

D2FS REST services developer guide

This document helps to familiarize one with the existing D2FS REST endpoints and learn about the standards and conventions used in developing those endpoints.

```
Cannot GET /bundle/pdf/OpenText%20Documentum%20D2FS%20REST%20Services%20Development%20Guide.p
```

opentext™

Understanding D2SV plugin project

Each D2SV plugin project is a hybrid Maven + NodeJS project having some Java, Javascript and a few static resources as part of its source code. On the outer side, the source code is organized in a Maven project layout and an additional source directory `src/main/smartview` is used to house Javascript source code and directory follows an NPM project structure.

Here we list the directories and files found in a bare-minimum project and outline their purpose. For the purpose of listing, we create a new plugin project by following -

- Execute `npm start` in the workspace root directory to fire up the Workspace Assistant
- Select option **Create a new plugin project**
- Answer the follow-up questions as -
  - Directory name to save this plugin project in: **plugins**
  - Maven group-id of the plugin: **com.opentext.d2.smartview**
  - Name(maven artifact-id) of the plugin: **D2SV-TEST**
  - Version of the plugin: **1.0.0**
  - One liner description of the plugin(shows up everywhere in D2 runtime): **D2SV-TEST**
  - Enter package namespace for the plugin(used as prefix/suffix to generate Java classes & properties, also its lowercase format is used as base Java package name for the plugin & D2SV UI bundle): **D2SVTEST**
  - Include support for D2SV UI: **Yes**

After the assistant runs successfully, it would create a new plugin project in `plugins/D2SV-TEST` directory.

Plugin project layout

```
D2SV-TEST
|
|   pom.xml
|
+---src
```

```
|   \---main
|       +---java
|       |   \---com
|       |       +---emc
|       |       |       D2PluginVersion.java
|       |       |
|       |       \---opentext
|       |           \---d2
|       |               +---rest
|       |               |   \---context
|       |               |       \---jc
|       |               |               PluginRestConfig_D2SVTEST.java
|       |               |
|       |               \---smartview
|       |                   \---d2svtest
|       |                       |   D2SVTESTPlugin.java
|       |                       |
|       |                       +---api
|       |                       |       D2SVTESTVersion.java
|       |                       |
|       |                       \---rest
|       |                               package-info.java
|       |
|       +---resources
|       |   |   D2Plugin.properties
|       |   |   d2svtest-version.properties
|       |   |
|       |   \---smartview
|       |           SmartView.properties
|       |
|       \---smartview
|           |   .csslintrc
|           |   .eslintrc-html.yml
|           |   .eslintrc.yml
|           |   .npmrc
|           |   config-editor.js
|           |   d2svtest.setup.js
|           |   Gruntfile.js
|           |   package.json
|           |   server.conf.js
|           |
|           |
|           +---lib (shortcut to javascript & java libraries)
|           +---node_modules (shortcut to NPM based dependencies used to build/serve the
Javascript portion of code)
|           +---out-debug (directory will contain compiled Javascript code in non-
minified format)
|           +---out-release (directory will contain compiled Javascript code in minified
format)
|           |
|           +---src
|           |   |   component.js
|           |   |   config-build.js
|           |   |   d2svtest-extensions.json
|           |   |   d2svtest-init.js
|           |   |   Gruntfile.js
|           |   |
|           |   +---bundles
|           |   |       d2svtest-bundle.js
|           |   |
|           |   +---test
|           |   |       extensions.spec.js
|           |   |
|           |   \---utils
|           |       |   startup.js
|           |       |
|           |       \---theme
|           |           |   action.icons.js
|           |           |
|           |           \---action_icons
|           |                   action_sample_icon.svg
|           |
|           \---test
|                   Gruntfile.js
|                   karma.conf.js
|
\---target
```

Files and their purpose

- `/`
    - pom.xml - Maven build file. Used to build the plugin from source code to its distributable format.

- `src/main/java/` - Directory containing all Java source code
    - com/emc/D2PluginVersion.java - Declares identification information for the entire plugin using `D2SVTESTVersion` class.
    - com/opentext/d2/rest/context/jc/PluginRestConfig_D2SVTEST.java - Java configuration for spring components like Beans, Interceptor, Filter etc. This class also declares a component scanner for Spring runtime to automatically load REST Controller and related components.
    - com/opentext/d2/smartview/d2svtest/D2SVTESTPlugin.java - Declares a blanket plugin. Additional code could be put inside this class to implement any functional service plugin.
    - com/opentext/d2/smartview/d2svtest/api/D2SVTESTVersion.java - Holder for plugin identification information. Loads relevant data from `d2svtest-version.properties` file resource.
    - com/opentext/d2/smartview/d2svtest/rest/package-info.java - Declares package metadata for JDK and IDE.
- `src/main/resources/` - Directory containing all plugin related metadata and other static resources
    - smartview/SmartView.properties - Descriptor for D2SV UI runtime. Content of this file is read by D2SV UI runtime and appropriate UI artifacts from this plugin are discovered and linked to the UI.

      INFO

      This file won't be present for the plugins where Smartview UI support is not enabled.

      DANGER

      Changing content of this file will bring about runtime incompatibility and the UI artifacts will never get discovered by D2SV UI runtime.

    - D2Plugin.properties - Another descriptor for D2SV plugin system to identify this plugin separately from other plugins deployed in the same runtime. Content of this file is interpreted as a uniqueue namespace identifier.

      DANGER

> Changing content of this file will lose the namespace convention used throughout the source code and makes this plugin unmanageable by the SDK as well as D2SV runtime.

  - d2svtest-version.properties - Contains name, version and base package for Java classes defined by this plugin. This file is read by `D2SVTESTVersion` class and supplies the metadata information to D2SV plugin system.

    > **DANGER**
    >
    > Changing content of this file may render D2SV runtime to correctly load class files from the deployed plugin jar.

- `src/main/smartview/` - Home of the NPM project that represents the Javascript and related source code for D2SV UI.

  > **DANGER**
  >
  > Even though the source code here is layed out as an NPM project but you should never execute `npm install` command in this directory. Doing so will completely break the setup as the `node_modules` folder is a softlinked directory and managed by the `package.json` from SDK workspace root.

  > **TIP**
  >
  > If for some reason an additional NPM based dependency is required for a plugins UI code, the dependency should be added in the `package.json` from SDK workspace root, subsequently followed by `npm install` command execution in the workspace root directory itself. After that the plugin specific Javascript code can refer to it in usual manner.

  - `lib` - Shortcut to the `lib` directory from SDK workspace root. The directory hosts all D2SV UI dependency libraries, that are used while running the Javascript only portion through a NodeJS light-server.
  - `node_modules` - Shortcut to the `node_modules` directory from SDK workspace root. The directory hosts all NPM packages used to test, build the Javascript source code and pack those

into distributable format, besides serving them through NodeJS server for debugging/testing purposes.

- `out-debug` - The directory that will contain non-minified output from Javascript code post compilation.
- `out-release` - The directory that will contain minified output from Javascript code post compilation.
- config-editor.js - Temporary RequireJS configuration override file used while building the UI code.
- d2svtest.setup.js - NodeJS module to setup/re-instate the directory softlinks for `lib` and `node_modules`.
- Gruntfile.js - Master task definition file used by `Grunt` while testing/building the source code.
- package.json - NPM package manifest for the UI code.
- server.conf.js - Lets configure the remote URL to use as back-end while serving the application through NodeJS.
- .csslintrc - Linter rules for CSS files. Used to validate if all CSS source files meet D2SV standard.
- .eslintrc.yml - Linter rules for Javascript files. Used to validate if all JS source code meet D2SV standard.
- .eslintrc-html.yml - Linter rules for HTML & HBS(Handlebars HTML template) files. Used to validate if matching source files meet D2SV standard.
- .npmrc - Local NPM configuration, used by NodeJS engine before running any NPM/NodeJS scripts.

  > **INFO**
  >
  > All these HTML, JS, JSON, CSS etc files are not part of the actual plugin source code. Some of them facilitate build, test, packaging of the actual source code whereas, others enable serving the entire front-end of D2SV application including this plugin and connects it to a remote back-end for testing/debugging purposes. Any of these files should not be modified.

- src/bundles/d2svtest-bundle.js - Plugin UI bundle, it must contain direct/indirect reference to all AMD modules defined in this plugin so that they are correctly packaged during build.

> **INFO**
>
> AMD modules, that are referred by atleast one other AMD module through `define()`, are called statically referred modules. Dynamically referred AMD modules are never referred by any other module using `define()` rather they are referred only through `require()`. The UI bundle must contain direct reference to all dynamically referred modules. It's very important to know that for an entry, all other statically referred modules from that module are automatically added e.g. suppose, A statically refers to B and that statically refers to C but C dynamically refers to D, in this scenario, you should put an entry for A and an entry for C to completely refer all the modules A, B, C, D.

- src/test/extensions.spec.js - Sample unit test. Shows how to write unit tests for each JS module in this plugin.
- src/utils/theme/action_icons/action_sample_icon.svg - A sample icon resource, could be used to represent a menu action. To know more about action icons and how to use them in UI refer to Use icons in D2SV
- src/utils/theme/action.icons.js - Holder of all the action icons. This file is auto-generated everytime the UI code is being built.

  > **TIP**
  >
  > After adding a new svg file in `action_icons` directory, you need to build the UI code once to re-generate this holder file.

- src/utils/startup.js - Hooks to D2SV application startup phase. Generally used to run additional custom logic that has to happen during the startup i.e before any of the UI components starts to render on the page. Provides two hook points, namely `beforeStartup()` and `afterStartup()` their purpose is pretty much self explanatory.
- src/component.js - Declares the UI bundles in this plugin. Used by the UI framework while building the UI code as well as used while serving the code through NodeJS server. This file normally wouldn't require any change unless the UI source code declares a RequireJS plugin.

  > **CAUTION**

> At this time only one UI bundle per D2SV plugin is supported by the runtime. So you must not try to split the `d2svtest-bundle.js` file into multiple smaller bundles.

- src/config-build.js - RequireJS configuration used while building the UI code. This file is auto-generated every time the UI code is being built. So any changes done to this file gets automatically discarded.
- src/d2svtest-init.js - Used to supply additional RequireJS configuration to AMD modules in this plugin. Also can be used to re-configure any other AMD modules defined by the D2SV UI itself or any of its dependencies.

  > **INFO**
  >
  > All the plugins deployed on D2 Smartview do not have a mechanism to specify their loading order. So if multiple plugins try to configure the same AMD module then whichever plugin is loaded last, configuration from that plugin will apply.

- src/d2svtest-extensions.json - Single file to register all the D2SV UI API extensions defined by this plugin.
- src/Gruntfile.js - Task definition file used to build the source code.
- test/Gruntfile.js - Task definition file used to start karma server to run unit test on source code/build output.
- test/karma.conf.js - Karma configuration file used while running unit tests. It leverages the base configuration from `@dctm/dctm-web-core` NPM package which is a crucial part of the UI framework and comes pre-bundled within the SDK.

opentext™

Where to start?

Well, D2 Smartview UI has many UI constructs like command, shotcut tile, list tile, application scope perspective, rest-controller etc. Answer to the question, depends on what you're trying to accomplish.

A good starting point might be to look at the packaged samples. `D2-AdminGroups` sample specifically covers all of the above stated constructs.

After extracting the sample, go thorugh D2 Admin Groups Sample documentation to know about the key concepts and strctures implemented in the sample. Then -

1. Build it once
2. Deploy the compiled artifact in `WEB-INF\lib` folder of a running D2-Smartview
3. Follow How to debug to run the sample in debug mode and put break points(in internet browser's developer console) at at different javascript modules in the project.

Once familiarized, try exploring different workspace assistant options to add new components to the sample or create a fresh plugin project and add it there to see how it works.

TIP

Familiarize yourself with the rest of "How to" topics. The API documentation helps getting to know different parts of the front-end & back-end components.

Overview

The API includes classes, objects, methods & extension points that could be used to enhance/alter existing components of D2SV runtime or write brand new components for it.

The D2SV runtime is comprised of

FRONT-END

Runs on Internet browsers, written using Javascript, HTML, CSS.

Components of the application are loaded using RequireJS framwork that comes bundled with the runtime. A bunch of other open-source libraries are pre-packed as part of the runtime. Below is a list of libraries and their RequireJS module dependency path -

- BackboneJS (nuc/lib/backbone)
- MarionetteJS (nuc/lib/marionette)
- UnderscoreJS (nuc/lib/underscore)
- jQyery (nuc/lib/jquery)
- MomentJS (nuc/lib/moment)
- jQuery Fancy tree (d2/lib/fancytree/jquery.fancytree)
- D3 JS (csui/lib/d3)

  Available requirejs plugins -

  - i18n - Loads a localization module
  - hbs - Loads handlebar template files (*.hbs)
  - json - Loads JSON files (*.json)
  - css - Loads CSS files (*.css)

BACK-END

Runs on Web application container, Written in Java.

opentext™

Context

Gathers models, collections, or plain objects to be shared among multiple scenarios and fetch them together. Objects in context are managed by their *factories*.

This is a base class. `PageContext`, `PortalContext`, `BrowsingContext` or `PerspectiveContext` are classes to create instances of.

```
// Create a new context.
var context = new PageContext();
// Get the (main contextual) authenticated user
var currentUser = context.getModel(UserModelFactory);
```

Factory

Is the "overlord" of objects in the context. The parent class returned from 'nuc/contexts/factories/factory' is usually called by different names like `ObjectFactory`, `ModelFactory` or `CollectionFactory` to express what the descended factory will take care of.

- Creates an instance of the object, which will be returned to the caller.
- Assigns a unique prefix to the object, so that the same object can be obtained using the factory at different places.
- Can override how the model or collection is fetched.

A factory has to specify a unique `propertyPrefix` in the prototype and set the object managed by it to `this.property`:

```
var TestObjectFactory = ObjectFactory.extend({

  propertyPrefix: 'test',

  constructor: function TestObjectFactory(context, options) {
    ObjectFactory.prototype.constructor.apply(this, arguments);
    this.property = new TestObject();
  }

});

// Request an object with the default identifier
// (internally stored with prefix 'test')
var test = context.getObject(TestObjectFactory);
```

Objects are stored using `propertyPrefix` in the context. The `propertyPrefix` is used alone for globally unique objects, or as a base for multiple objects having the same factory, but different attributes:

```
// Request a separate object with a specific identifier
// (internally stored with prefix 'test-id-1')
var test = context.getObject(TestObjectFactory, {
  attributes: {id: 1}
});
```

Factory can be used just for the object creation, if you don't want to learn about its constructor parameters.

```
// Request a standalone object, not shareable by the context
var test = context.getObject(TestObjectFactory, {
  unique: true,
  temporary: true,
  detached: true
});
```

Fetchable Factory

Exposes `fetch` method, which should fetch its model. Whenever the context is fetched, this method will be called.

```
var FavoriteCollectionFactory = CollectionFactory.extend({

  propertyPrefix: 'favorites',
  constructor: function FavoritesCollectionFactory(context, options) {
    CollectionFactory.prototype.constructor.apply(this, arguments);

    var connector = context.getObject(ConnectorFactory, options);
    this.property = new FavoritesCollection(undefined, {
      connector: connector,
      autoreset: true
    });
  },

  fetch: function (options) {
    return this.property.fetch(options);
  }

});
```

The `isFetchable` method can be added to be able to check dynamically, if the object is fetchable or not.

```
var NodeModelFactory = ModelFactory.extend({

  propertyPrefix: 'node',

  constructor: function NodeModelFactory(context, options) {
    ModelFactory.prototype.constructor.apply(this, arguments);

    var connector = context.getObject(ConnectorFactory, options);
    this.property = new NodeModel(undefined, {connector: connector});
  },

  isFetchable: function () {
    return this.property.isFetchable();
  },

  fetch: function (options) {
    return this.property.fetch(options);
  }

});
```

opentext™

Configurable Factory

Factories are usually created once per object type, but they need to be able to create multiple object instances. With just the factory provided, the object will be constructed with default options:

```
// Get the (main contextual) node
var currentNode = context.getModel(NodeModelFactory);
```

With the second argument, additional options can be passed to control the object creation. The `attributes` will be used to uniquely stamp the new object, so future calls to `getObject` with the same attributes will return the same object. Also the `attributes` will be passed to the constructor of the object, if it is a `Backbone.Model`:

```
// Get original where the (main contextual) node points to, if it is
// a shortcut
var originalId = currentNode.get('original_id'),
    original = context.getModel(NodeModelFactory, {
      attributes: {id: originalId}
    });
```

Below the property called like the factory prefix you can pass additional options to the newly created object's constructor by the `options` property:

```
// Get original where the (main contextual) node points to, if it is
// a shortcut and make it fetchable by the connector
var originalId = currentNode.original.get('id'),
    original = context.getModel(NodeModelFactory, {
      attributes: {id: originalId},
      node: {
        options: {connector: currentNode.connector}
      }
    });
```

If the new object is a `Backbone.Model`, you can specify different attributes for the constructor, than the attributes, which control the unique stamp of the object. While the former should be as minimum as to compose the unique stamp, the latter could be more complete to pre-initialize the new object:

```
// Get original where the (main contextual) node points to, if it is
// a shortcut, make it fetchable by the connector, but pre-initialize
// it will all properties available so far
var originalId = node.original.get('id'),
    original = context.getModel(NodeModelFactory, {
      attributes: {id: originalId},
      node: {
        attributes: node.original.attributes,
        options: {connector: currentNode.connector}
      }
    });
```

Finally, if you already have the new object created and you only need the context to make it shareable, you can pass it to the property called like the factory as-is:

```
// Get original where the (main contextual) node points to, if it is
// a shortcut, and share the same object, which has been obtained
// with the contextual node
var originalId = node.original.get('id'),
    original = context.getModel(NodeModelFactory, {
      attributes: {id: originalId},
      node: node.original
    });
```

Detached Objects

Objects, which are added to the context after the context was fetched are needed to be fetched manually, if they need fetching at all. Also, as manually fetched objects, when the context is re-fetched, they are not re-fetched again. Their users decide, when they should be re-fetched.

```
// User information, which does not refresh automatically and will be
// discarded, when clear() is called on the context
var ownerId = node.get('owner_user_id'),
    owner = context.getModel(MemberModelFactory, {
      attributes: {id: ownerId},
      detached: true
    });
```

Detached objects should merge the `Fetchable` mixin, which allows fetching only once on demand by `ensureFetched`:

```
// Make sure, that the model was fetched once, before accessing
// its properties
owner
    .ensureFetched()
    .done(function () {
      console.log('Login:', owner.get('name'));
    });
```

Permanent Objects

Objects like the authenticated user need not be re-created during the application lifecycle. After being requested for the first time, they should remain in the context for all scenarios. (The only way how to re-create them is to reload the entire application - the application page.)

```
// User information, which does not refresh automatically and will not be
// discarded, when clear() or fetch() is called on the context
var ownerId = node.get('owner_user_id'),
    owner = context.getModel(MemberModelFactory, {
    attributes: {id: ownerId},
    permanent: true,
    detached: true
  });
```

Permanent objects are usually detached too, unless they should be re-fetched with every context re-fetch.

Temporary Objects

Objects like the original node need to be shared across function scopes and object boundaries, but should not be re-created and re-fetched multiple times. When the lifecycle of the current (main contextual) node ends, they should be discarded from the context, so that they would not get re-fetched with the new context content.

```
// Shareable original node information, which will be discarded, as soon
// as clear() or fetch() is called on the context
var originalId = shortcut.original.get('id'),
    original = context.getModel(NodeModelFactory, {
      attributes: {id: originalId},
      temporary: true
    });
```

Factory Life-Cycle

The context is a single-instance object that lives as long as the web page lives. (There may be multiple contexts, if parts of the page were supposed to work separately, but that would be a rare case.) The web page serves different purposes during its life. Having just single context instance means that the content of the context has to be able to be exchanged to reflect the current page content.

The context supports two changes of the page content:

- refresh - the page (views) will be reused, only the data will be reloaded
- exchange - the page will be rebuilt (current views will be destroyed and new ones will be created) and new data wil be loaded

These changes can be induced by the following methods of the context: `clear` and `fetch`. The `clear` removes the factories and thus their data from the context. The `fetch` reloads (or loads, initially) the data by letting the factories fetch.

```
// render a new page    <----------------------------------+
context.getObject(...) // get objects from the context     |
context.fetch()        // fetch collected factories  <--+  |
// work with the page                                   |  |
// open another object on the same page  ---------------+  |
context.clear()        // prepare for the next page        |
// navigate to other page  -------------------------------+
```

If the page has to show a different scenario (exchange), the `clear` will be called, then the page will be rebuilt and eventually the `fetch` will be called to load the data. If the page should show the same scenario with different data (refresh), just `fetch` will be called.

Factories together with the objects that they maintain can be removed from the context when `fetch` and `clear` are called to allow some objects to stay forever and the other objects temporarily only after new data are to be loaded. When factories are used to request objects from context, they can be passed options, or these options can be set to `this.options` in the factory's constructor: `permanent` and `temporary` take care of the life-cycle, `detached` and `unique` have other purposes.

How factories are removed from the context when `clear` and `fetch` are called:

operation / flag	refresh (fetch)	exchange (clear + fetch)
permanent	stay	stay
normal	stay	drop
temporary	drop	drop

The `detached` flag does not affect the factory's life. It prevents the factory ever getting fetched. The `unique` flag appends a unique number to the factory prefix, so that one factory can be put to the context multiple times to maintain different objects.

Declarative options control what factories are allowed to fetch when `fetch` is called. In addition to the static rules below, the actual fetchability is checked by the `isFetchable` method of the context:

method / flag	fetch
permanent	allowed

method / flag	fetch
normal	allowed
temporary	N/A (*)
detached	forbidden

(*) Temporary factories are removed from the context when the `fetch` method starts executing. It does not make sense to discuss their fetchability.

Methods

getObject(factory, options): object

Returns an object maintained by the specified factory. If the object has not existed yet, it will be created, otherwise the previously created instance will be returned.

The object existence is made unique by the property prefix defined by the factory. The full unique property stamp consists of this prefix and of the context `attributes`, which can be passed in the second argument.

If the object is to be created, the second argument can carry parameters for its constructor under the property named by the factory's property prefix; usually `attributes` and `options` for a model or `models` and `options` for a collection. Instead of constructor parameters, this property can point to an already created object, so that the factory just stores it as-is.

The second argument can contain boolean flags to control how the context will handle the object: `detached`, `permanent`, `temporary` and `unique`.

```
// Create a favorite node collection pre-initialized with some nodes
// until it gets fetched with the context
var favorites = context.getCollection(FavoriteCollectionFactory, {
  favorites: {
    models: [{type: 141}, {type: 142}]
  }
});
```

getCollection(factory, options): object

Behaves just like `getObject`, but looks more intuitive, if the expected result is Backbone.Collection.

getModel(factory, options): object

Behaves just like `getObject`, but looks more intuitive, if the expected result is Backbone.Model.

hasObject(factory, options): boolean

Returns if there is an object maintained by the specified factory.

hasCollection(factory, options): boolean

Behaves just like `hasObject`, but looks more intuitive, if the expected object is Backbone.Collection.

hasModel(factory, options): boolean

Behaves just like `hasObject`, but looks more intuitive, if the expected object is Backbone.Model.

clear(options): void

Discards all objects from the context, which are not permanent. When `options.all` is set to `true`, all objects will be discarded.

fetch(options): Promise

Fetches all objects in the context, which are not detached. Discards all temporary objects before that. The options will be passed to the `fetch` methods in factories that take care of the fetchable objects.

opentext™

suppressFetch(): boolean

Aborts fetching started by the `fetch` method. You can interrupt a running `fetch`in order to start another one, because the earlier result has become irrelevant. (Because a navigation got interrupted by yet another navigation, for example.)

Error event on the context will be never triggered and the returned promise will be never resolved. Sync event will be triggered immediately as the `suppressFetch` method is called to balance the earlier triggered `request` event. Events on the models and and collection will be triggered eventually, as their AJAX calls will finish.

This method does not abort the operation. It only allows another call to `fetch` be made and replace the one in progress.

Properties

fetching: Promise

The promise returned by `fetch` during fetching or `null` if no fetching is in progress.

fetched: boolean

`true` if the most recent `fetch` succeeded, `false` if the context has not been fetched yet, or fetching is in progress, or it failed.

error: Error

`null` if the most recent `fetch` succeeded, or fetching is in progress, or the context has not been fetched yet, an instance of `Error` if the most recent `fetch` failed.

Events

'before:clear', context

The context is going to be cleared.

'clear', context

The context has been cleared.

'request', context

The context is going to be fetched.

'sync', context

Fetching the context succeeded.

'error', error, context

Fetching the context failed.

'add:factory', context, propertyName, factory

A new factory has been added to the context.

'remove:factory', context, propertyName, factory

A factory has been destroyed and will be removed from the context.

Context Fragment

The context fragment can be used to fetch data for a dynamically added widget, instead of fetching the whole context, which would re-fetch data for widget created earlier.

```
// Subscribe a context fragment to the context, before
// a new widget is constructed and rendered.
contextFragment = new ContextFragment(context);
// Create the widget and render it to get the new models
// added to the context and to the context fragment too.
...
// Fetch only the new models. The new widget will update
// the displayed information as it is needed.
contextFragment.fetch();
// Unsubscribe the context fragment from the context,
// when it is not needed any more.
contextFragment.destroy() // Unsubscribe the fragment.
```

Details

The context supports two scenarios for changing the page content:

- refresh - the page (views) will be reused, only the data will be reloaded
- exchange - the page will be rebuilt (current views will be destroyed and new ones will be created) and new data wil be loaded

There is one more scenario, which you may see on the page:

- grow - new content (views) will be added to the page, which needs to load a new data, but the old data do not need to be reloaded

New views usually load the new data by `ensureFetched` and the context does not need to be involved in fetching the data. However, shared components might be used to add the new content, which depend in the context to load their data. Because only the owning view knows what part of the context will have to be fetched, it is responsible for collecting a fragment of factories for fetching:

```
// render a new page
context.getObject(...) // get objects from the context
context.fetch()        // fetch collected factories
// work with the page   <-------------------------------+
// introduce new content to the page                    |
new ContextFragment(context) // remember new objects    |
context.getObject(...)       // add other objects       |
contextFragment.fetch()      // load new data           |
contextFragment.destroy()    // stop context watching  --+
```

No factories are removed, when a context fragment is fetched and destroyed:

operation / flag	refresh (fetch)	exchange (clear + fetch)	grow (fragment fetch + destroy)
permanent	stay	stay	stay
normal	stay	drop	stay
temporary	drop	drop	stay

The fetchability of factories follows the rules which the context declared. In addition to the static rules below, the actual fetchability is checked by the `isFetchable` method of the context:

method / flag	fetch	fragment fetch
permanent	allowed	allowed
normal	allowed	allowed
temporary	N/A (*)	forbidden
detached	forbidden	forbidden

opentext™

(*) Temporary factories are removed from the context when the `fetch` method starts executing. It does not make sense to discuss their fetchability.

Methods

constructor(context)

Start watching the original context for new factories.

fetch(options): Promise

Fetches all objects in the context fragment, which are fetchable by their originating context. The options will be passed to the `fetch` methods in factories that take care of the fetchable objects.

clear(): void

Discards all objects from the context fragment. The context fragment remains subscribed to the context.

destroy(): void

Stops watching the original context for new factories. The context fragment will not be usable any more.

Properties

fetching: Promise?

The promise returned by `fetch` during fetching or `null` if no fetching is in progress.

fetched: boolean

`true` if the most recent `fetch` succeeded, `false` if the context has not been fetched yet, or fetching is in progress, or it failed.

error: Error

`null` if the most recent `fetch` succeeded, or fetching is in progress, or the context has not been fetched yet, an instance of `Error` if the most recent `fetch` failed.

Events

'request', context

The context fragment is going to be fetched. This event is triggered on the original context too. The `fetching`, `fetched` and `error` properties on the original context are not modified.

'sync', context

Fetching the context fragment succeeded. This event is triggered on the original context too. The `fetching`, `fetched` and `error` properties on the original context are not modified.

'error', error, context

Fetching the context fragment failed. This event is triggered on the original context too. The `fetching`, `fetched` and `error` properties on the original context are not modified.

'add:factory', context, propertyName, factory

A new factory has been added to the context fragment.

'before:clear', context

The context fragment is going to be cleared.

'clear', context

The context fragment has been cleared.

'destroy', context

The context fragment has been destroyed.

opentext™

PageContext

The simplest context, which can include and fetch models and collections, but does not provide any other functionality. If you use it with widgets, which expect changes based on their context-changing models, you will have to handle these changes yourself.

```
csui.require([
  'nuc/widgets/shortcut/shortcut.view',
  'nuc/contexts/page/page.context',
  'nuc/contexts/factories/next.node',
  'nuc/lib/marionette'
], function (ShortcutView, PageContext, NextNodeModelFactory, Marionette) {
  'use strict';

  var context = new PageContext(),
      nextNode = context.getModel(NextNodeModelFactory),

      region = new Marionette.Region({
        el: '#content'
      }),
      view = new ShortcutView({
        context: context,
        data: {
          type: 141
        }
      });

  // Perform some action if the widget triggered contextual node change
  nextNode.on('change:id', function () {
    alert('Node ID:' + nextNode.get('id'));
  });

  region.show(view);
  context.fetch();

});
```

Plugins

Plugins descended from `ContextPlugin` (nuc/contexts/context.plugin) can be registered. They will be constructed and stored with the context instance. They can override the constructor and the method `isFetchable(factory)`.

opentext™

CSS

Bundles and loads CSS stylesheets referred from JavaScript module dependendencies.

TODO: Write the documentation.

Load CSS bundle

Stylesheet bundles are loaded by a call to `styleLoad` in bundle indexes, for example:

```
define([
  ...
], {});

require(['require', 'css'], function (require, css) {
  css.styleLoad(require, 'csui/bundles/csui-browse');
});
```

styleLoad(require, bundleName, separateRTLCSS?)

- `require` - the `require` function required for the module, where the `styleLoad` is going to be called
- `bundleName` - the name of the module bundle, for which the stylesheet will be loaded
- `separateRTLCSS` - if the stylesheet for the RTL text-writing direction is in the same or in a separate file (default is `true`)

If `separateRTLCSS` is `true` or not specified and the selected UI language requires the RTL text-writing direction, the stylesheet name will include the suffix `-rtl`. The RTL stylesheet will be used instead of the default one:

Bundle name	LTR text-writing direction	RTL text-writing direction

Bundle name	LTR text-writing direction	RTL text-writing direction
`csui/bundles/csui-browse`	`csui/bundles/csui-browse.css`	`csui/bundles/csui-browse-rtl.css`

If `separateRTLCSS` is `false` and the selected UI language requires the RTL text-writing direction, the default stylesheet name will be used. The default stylesheet is supposed to contain styles supporting both LTR and RTL text-writing direction:

Bundle name	LTR text-writing direction	RTL text-writing direction
`csui/bundles/csui-browse`	`csui/bundles/csui-browse.css`	`csui/bundles/csui-browse.css`

opentext™

I18n

Carries language settings and loads language modules for the selected locale.

TODO: Write the documentation.

Accept-Language in AJAX Calls

Smart UI has always set the chosen UI language to the `Accept-Language` header, when making AJAX calls via `Connector`. It ensures a consistent language of static assets (language pack) and the data (REST API responses). The UI language is chosen by the `locale` setting:

```
require(['i18n'], function (i18n) {
  console.log('locale:', i18n.settings.locale);
});
```

If you want to use a different locale for the data, you can set the property `acceptLanguage`, which will be sent to the server instead of `locale`:

```
require.config({
  config: {
    i18n: {
      locale: 'en-US',
      acceptLanguage: 'en-AU'
    }
  }
});
```

The values of both `locale` and `acceptLanguage` have to comply with BCP 47. They are case-insensitive and Smart UI will normalize them to lower-case before using them for loading static assets or sending in the `Accept-Language` header.

CS uses the property `i18n.settings.userMetadataLanguage` to support multi-lingual UI. Unfortunately, this property does not follow BCP 47. If `userMetadataLanguage` is set and `acceptLanguage` is unset, Smart UI will convert the value of `userMetadataLanguage` to `acceptLanguage`. If both `acceptLanguage` and `userMetadataLanguage` are unset, Smart UI will set the value of `locale` to `acceptLanguage` to stay compatible with previous versions.

Setting `acceptLanguage` to `null` will stop Smart UI from setting the `Accept-Language` header in AJAX calls:

```
require.config({
  config: {
    i18n: {
      locale: 'en',
      acceptLanguage: null
    }
  }
});
```

opentext™

RequireJS

This document describes changes to the original RequireJS. See the RequireJS website for the original documentation.

Changes

1. Make the `pkgs` configuration object mergeable.
2. Add an attribute `data-csui-required` to to every element added to document head.
3. Recognise `rename` in the configuration to implement module name mapping in addition to the `starMap` for an additional module compatibility layer.
4. Return module configuration merged from the mapped original names and new ones.
5. Introduce a method `moduleConfig(id)` on the local require function.

`pkgs` mergeable

Allows calling `require.config({ pkgs: ... })` multiple times to configure packages step-by-step. Kind of misused by the original version of the mobile app to remap modules.

Attribute `data-csui-required`

Allows detecting all scripts and links inserted by RequireJs and the `css` plugin to `document.head` to be able to remove them later. Used to wipe out all modules loaded from one server, before another version can be loaded from a different server.

`rename` map

The RequireJS `starMap` can be used for remapping modules to be able to load different functionality on different pages. It can be used to implement product-specific features, if a RequireJS library is reused in multiple products.

Another need for module remapping comes from refactoring, which moves a module to a different library, with or without deprecating the original module name. If a module needs to be remapped for compatibility, which was earlier remapped for product adaptation, the two map entries will conflict.

A direct conflict means that either the compatibility mapping, or the product adaptation will not work, depending on the order of the configuration statements:

```
csui/original -> nuc/moved      // keep compatibility with a moved module
csui/original -> custom/adapted // adapt a module for a new product
```

An direct conflict means that dependencies on the original module will not be adapted, if the adaptation maps only the new module name, because the `starMap` is not processed recursively:

```
csui/original -> nuc/moved      // keep compatibility with a moved module
nuc/moved     -> custom/adapted // adapt a module for a new product
```

The `rename` map is separate from `starMap` and solves the direct conflict. Modules remapped for compatibility are called "renamed" and have to be added to the `rename` map. The `starMap` continues to support product adapting. The module name normalisation makes use of both maps. If the `rename` map is configured alone, it will work like `starMap` alone.

When this configuration is used:

```
rename:  csui/original -> nuc/moved
```

The module names will be normalised like this:

```
csui/original -> nuc/moved // using rename
nuc/moved                  // just loaded
```

When this configuration is used:

```
rename:  csui/original -> nuc/moved
starMap: csui/original -> custom/adapted
```

The module names will be normalised like this:

```
custom/adapted                  // just loaded
csui/original -> custom/adapted // using starMap
nuc/moved     -> custom/adapted // using rename backwards and starMap
```

When this configuration is used:

```
rename:  csui/original -> nuc/moved
starMap: nuc/original  -> custom/adapted
```

The module names will be normalised like this:

```
custom/adapted                  // just loaded
csui/original -> custom/adapted // using rename backwards and starMap
nuc/moved     -> custom/adapted // using starMap
```

Merged module configuration

This is a feature supporting module renaming and remapping as discussed in the previous chapter.

An example of a situation:

1. The original module csui/original supported configuration.
2. The original module was adapted in a new product and the new module might need additional configuration.
3. The original module was renamed in the library and parts of the configuration started to be set using the new name.

An example of RequireJS configuration:

```
rename:  csui/original -> nuc/moved
starMap: csui/original -> custom/adapted
```

The result of `module.config()` called in custom/adapted will contain an object merged from configurations set for all three module names. Forward and backward `rename` map and `starMap` are used to discover the other module names.

`moduleConfig` method

The configuration of a RequireJS module may be needed in another module. It can be used to keep compatibility after refactoring the module tree. The functionality of `require.moduleConfig` is similar to `module.config`, the difference is that you have to supply the module name:

```
define(['require', 'module'], function (require, module) {
  // merge the old and new module configurations
  var config = _.extend({},
    require.moduleConfig('other-module'), // configuration of other module
    module.config()                       // configuration of this module
  );
});
```

opentext™

Documentum D2 Smartview SDK - 23.4.0

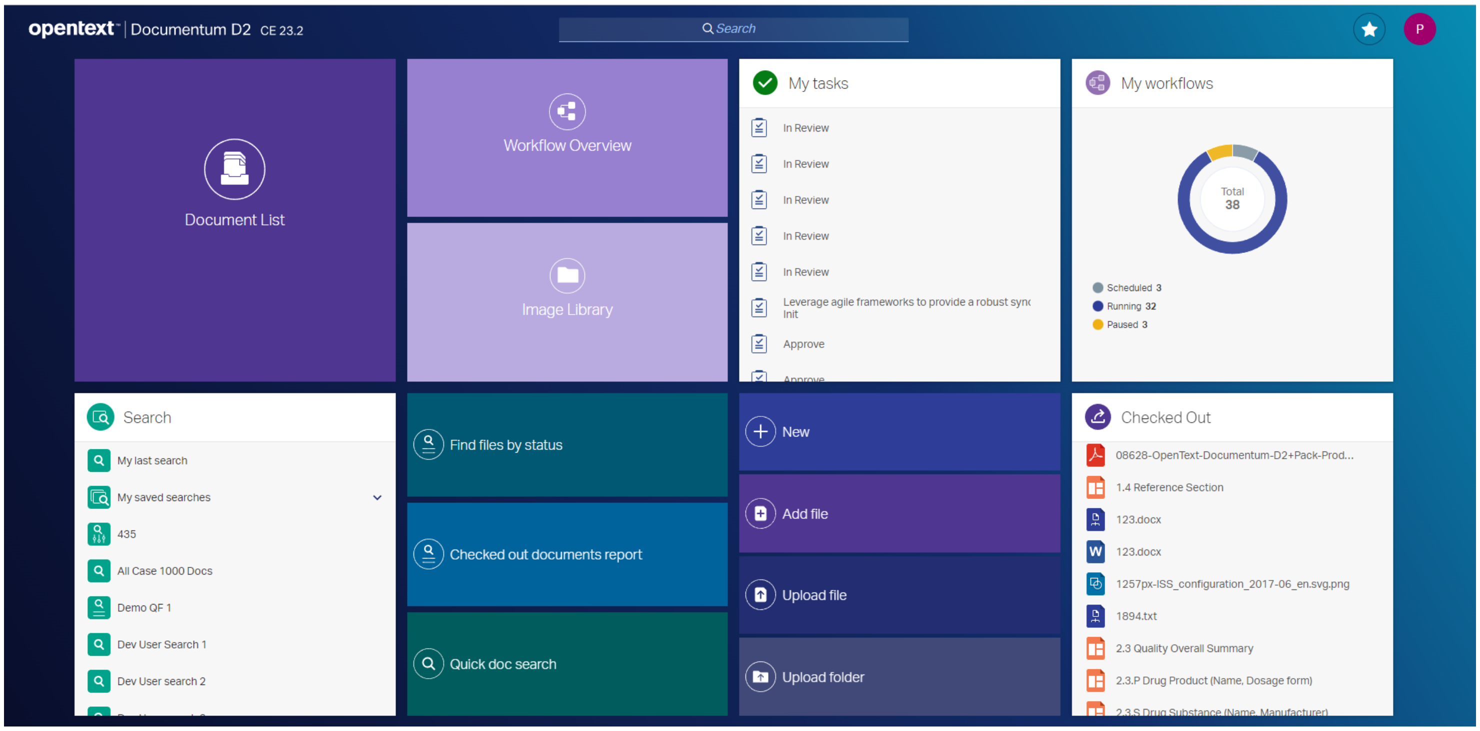
The D2 Smart View SDK consists of sources, binaries, documentation, and samples for -

- D2 Smartview UI extension enviornment.
- D2-REST services extension enviornment.
- D2 plugin development enviornment.

It also includes a few tools to create and maintain a development workspace.

With the D2 Smart View SDK you can build enterprise-ready software components for Documentum D2 Smartview runtime to cater custom business needs.

Out of the box, D2 Smart View landing page looks like:



How to prepare and start with the development environment

1. Download developer tools
2. Install developer tools

3. Create the development workspace
4. Get familiar with SDK tools
5. Create a plugin project
6. Start coding

1. Download the developer tools for your OS:

```
JDK         - JDK is required to compile Java code present within a development
workspace.
              Use JDK 17 or later.
              See https://openjdk.java.net
Maven       - Apache maven is the secondary build tool used in this SDK development
workspace.
              Recommended version is 3.8.2. A different version may not be fully
compatible.
              See https://maven.apache.org
NodeJS      - JavaScript VM to execute the SDK tools, build tools and to run the
development web server for UI code.
              Recommended version is 16 LTS. A different version may not be fully
compatible.
              See http://nodejs.org.
Grunt       - JavaScript task runner for building and testing UI code.
              See http://gruntjs.com. Nothing to be downloaded from this URL though.
```

2. Install the developer tools for your OS

```
JDK         - Run installer. Set the JAVA_HOME path variable to point to the JDK root
directory.
Maven       - Unzip & extract to a directory. Set MAVEN_HOME environment variable
pointing to the directory.
              Update PATH variable accordingly so that Maven commands can be executed
from command-line/terminal.
NodeJS      - Install the package for your OS. Set NPM_HOME path variable pointing to
the NodeJS
              installation directory. Update PATH variable so that Node & NPM commands
can be executed from
              command-line.
              It is recommended to avoid installing NodeJS under 'Program Files' as
```

opentext™

```
doing that has been known to create
              problem some times.
NPM         - Update the NPM module management tool to the latest version:
                npm install -g npm@latest

Grunt       - Install the command line task runner client as a global NPM module
                npm install -g grunt-cli
```

3. Create the development workspace:

```
# 1. Extract the SDK
# 2. Open a command prompt at the extracted folder

# Execute batch script ws-init.bat
>ws-init.bat

# It will take a while to fully initialize the workspace.
# Once initialization completes successfully, the workspace assistant starts
automatically. Select "Check out documentation" option to open documentation in default
browser.
#
# The directory where SDK was extracted becomes the root of the development workspace.
# It doesn't require to run ws-init.bat inside the initialized workspace again, unless
some other instructions specifically says to do so.
# If you want to run the workspace assistant anytime later, open a command
prompt/terminal at workspace root directory and run
>npm start

# Select "Nothing", to terminate the workspace assitant, if wanted.

# To access the documentation without the workspace assistant, you can run the following
command in a command prompt/terminal at the workspace root.
>npm run documentation
```

4. Get familiar with Workspace assistant

Check out the Workspace assistant. It's a good idea to familiarize yourself with the general aspects of the SDK, this can be done later though.

opentext™

5. Create a plugin project:

```
# Open command prompt at workspace root and run
>npm start

# Select "Create a new plugin project" from the workspace assistant options.
# Follow on-screen instruction and anser questions to create your first plugin project.
# Once done, type and run-
>npm run build

# Or, alternatively run the workspace assistant again and select "Build all plugins in
this workspace" option.
# This will build all projects in the workspace
```

6. Getting started with SDK development

If you are a new SDK developer, you can check out this documentation to get started.

7. Start coding

Check out the API documentation and start coding as per business requirement.