White Paper

EMC Documentum D2 4.1 Architecture

A primer

Abstract

This white paper describes the component technologies comprised by the Documentum D2 product. It covers the various layers included in the implementation and describes the public interfaces and extension points.

March 2013

EMC²

Part Number h11494

Table of Contents

Executive summary

EMC Documentum® D2 provides a configuration driven, modern UI on top of the Documentum content repository. It allows customers to customize client interactions with content in the repository primarily through changes in configuration.

This white paper presents a high level architecture for the D2 product, describing the various parts of the application stack, and how they interact with configuration. It also describes, in more detail, how the product can be extended when customization is required in order to meet customer use cases.

Audience

Part 1 of this white paper provides a general introduction to the product architecture, and may be useful to anyone wishing to gain a better understanding of the product, or planning an installation. Part 2 provides more detail on the extension points, and will be more useful when contemplating customizations to the existing functionality. It is recommended to gain a solid understanding of the existing capabilities in the D2 product before considering any customizations, since many use cases that formerly required customizations in other clients can be achieved using the out of the box configuration capabilities in D2. It is also recommended to have basic Java and/or JavaScript programming skills in order to implement any extensions or customizations.

Part 1 – D2 Client Architecture

In this section, we will cover the architecture of the D2 UI, the configuration UI, and installed server components on the Documentum Content Server.

The D2 application can be roughly characterized in 3 tiers, as seen in Figure 1.

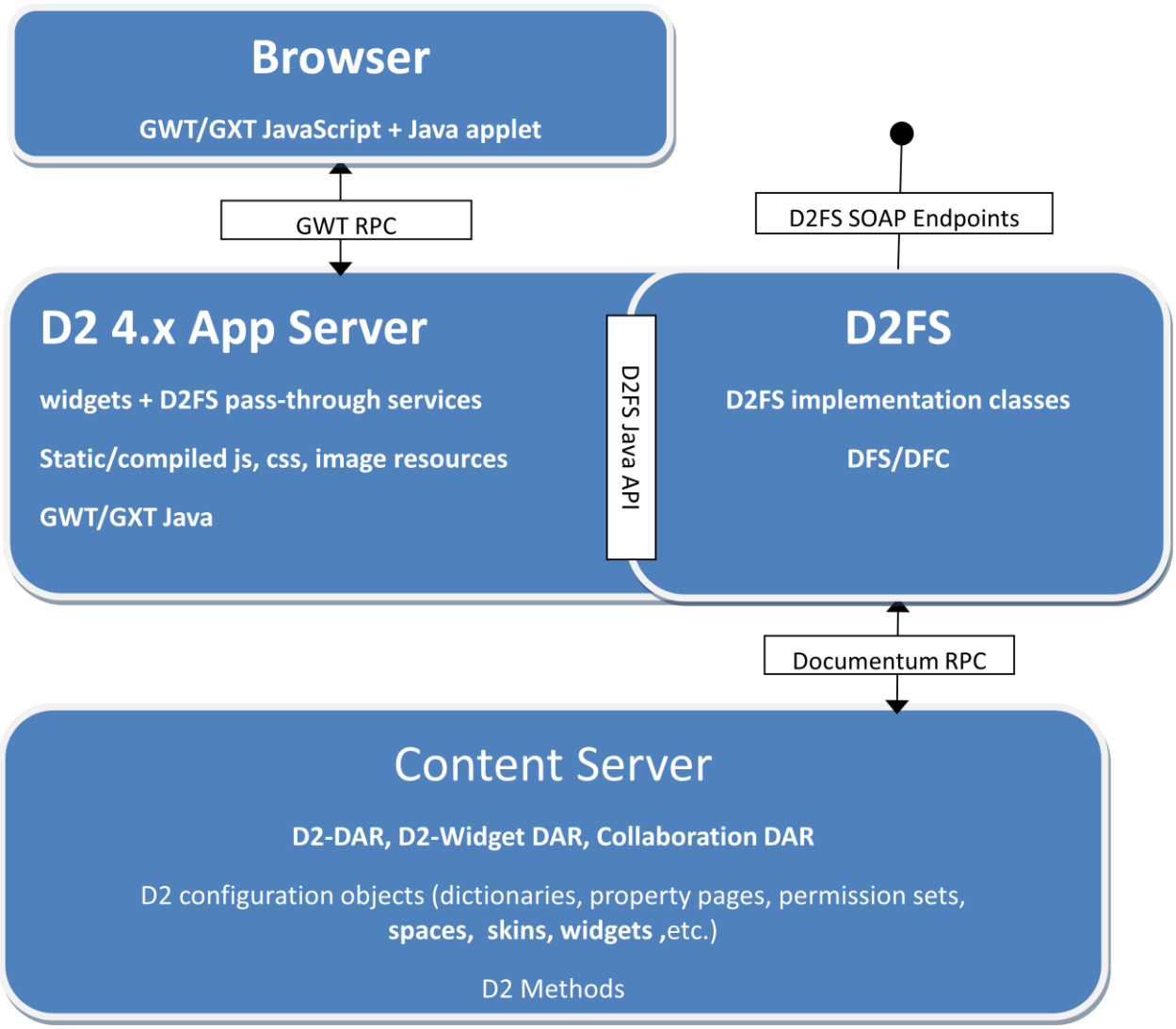

Figure 1 – D2 Client Architecture

Browser Layer

D2 is a web 2.0 browser based application; as with any other web based client, end users of D2 interact with the product by launching the browser of their choice, navigating to the D2 application URL, and logging into the product.

There are, however, a few distinctive characteristics of the D2 browser client logic worth noting:

1. **GWT/GXT based**: D2 uses Sencha GXT and the Google Web Toolkit (GWT) to manage client JavaScript. As a result, the vast majority of the client code is authored in Java, and compiled into a JavaScript "permutation" that is delivered to the browser in a single large file. As another result, the client JavaScript code is delivered obfuscated, and is not readily extensible or debuggable without the original source. (See "Part 2 – Extending D2" for

alternative techniques to make changes to the D2 client.) More details on GXT are available on the Sencha website.

2. **Single page**: As with many other web 2.0 clients, D2 is loaded onto a single HTML page, and we do not navigate away from that page until the application is closed. The principal reason is that each time the page is loaded, all of the client JavaScript must also be interpreted and loaded into browser memory. For a complex application such as D2, this takes a significant amount of processor time. Rather than switching HTML pages when the user navigates in the application, D2 manipulates the HTML DOM to remove parts of the UI and replace them with new parts. The end result is that we are able to maintain a richer application, because we do not need to re-render the entire UI on each navigation.
3. **GWT RPC communication**: All communication between the browser client and the D2 application is accomplished through standard HTTP requests, and some of these requests are simple requests for static resources (e.g. images, static script resources, CSS, etc.) The majority of the exchanges between the client and the application, however, are GWT RPC calls to fetch dynamic data, usually backed by the D2FS services layer. GWT RPC is a data marshalling layer provided by GWT to allow generated JavaScript client code to make remote procedure calls to Java code running on the server. Customers monitoring web traffic between the browser and the application server will notice that the data payload for these calls is neither XML nor JSON, but rather a wire protocol specific to GWT RPC that can prove somewhat difficult to parse. For more details on GWT RPC see this page on the Google developer site.

Application Server Layer (D2.war)

D2 is packaged in a war file, and can be deployed on most J2EE application servers (Apache Tomcat, Oracle Weblogic, etc.) Although it is packaged into a single web application, D2 is split into 2 major parts:

1. **The D2 web application** includes most of the presentation logic, including the GWT and GXT libraries, custom UI controls, static web resources, and pass-through services to allow the browser client to invoke D2FS services via GWT RPC. This layer does not link to the Documentum DFC or DFS client libraries—all communication between the D2 application and the Documentum repository passes through the D2FS interface.
2. **The D2FS library** includes all of the business logic for the D2 application. This includes all communication with the Documentum content server (through DFS and DFC), and all of the logic to interpret D2 configuration and apply it to service requests. D2FS is also exposed as a set of SOAP services from the D2 application. In the 4.0 release, D2FS was installed as a separate web application, and this set of SOAP services was used for communication between D2 and D2FS. In 4.1, the two applications have been merged, and the D2 layer now uses the D2FS Java API instead, but the SOAP interface remains available for external clients. Some examples for calling SOAP services from an external client are presented in "Part 2 – Extending D2." Plug-ins to D2 (for example, C2, O2, etc.) also sit in the D2FS layer, and can intercept existing calls into the D2FS interface to change or enhance the behaviors of D2.

Content Server Layer

The base D2 installation includes three DAR files:

1. **D2-DAR.dar** includes most of the Documentum types required to store configuration objects in the Documentum repository. All configuration types are subtypes of d2_module_config or d2_moduledoc_config, depending on whether the configuration is stored as object metadata, or as xml content. The latter type is primarily used to represent forms, dialogs, mail templates, or other structured content, where a set of repeating attributes is not sufficient to represent the configuration.
   The D2-DAR installation also includes a number of methods and jobs used by D2 to implement some of the D2 configurations. Some of these methods are invoked by a scheduled job in order to complete background tasks. Others are directly invoked by the D2 application in response to events such as object saves. For more details on the jobs installed by D2, please refer to EMC Documentum D2 Administrator Guide, Appendix B, D2 Jobs reference.
2. **D2Widget-DAR.dar** includes new configuration types related to the 4.x generation of the D2 UI. For example, configuration types for widgets, workspace layouts, and themes are all included in D2Widget-DAR.dar.
3. **Collaboration_Services.dar** provides types and BOF modules for Documentum collaboration capabilities. D2 requires this DAR file in order to enable commenting capabilities in the D2 client.

In addition to the types and objects installed into the repository by DAR files, D2 configuration itself is also stored in the Documentum repository. Each D2 application consists of a set of configurations that are stored in the repository as persistent objects. When application configuration is edited with the D2-Config client, the underlying configuration objects in the repository are added, deleted, or modified accordingly.

File Transfer - the D2 Java applet and D2-Bocs

Transfer of file content between the browser client and the D2 application is somewhat more complex than the ordinary connections we make to the application server, and thus it merits separate discussion here. For content transfer, D2 relies on a Java applet. This gives us several advantages over the standard HTML4 ‹fileinput› form control, including multiple file upload, upload of entire folder structures, drag-in of files from the desktop, and client side compression of content prior to upload.

In basic installations, where most users are in close geographic proximity, the D2 applet connects directly back to the D2 application using a standard HTTP connection. However, when a Documentum BOCS cache is needed in order to speed up transfer of large files to users in remote locations, D2 requires D2-BOCS to be installed onto the same server as the BOCS cache. (These are the components marked as “optional” in Figure 2 below.) D2-BOCS serves two main purposes:

1. It facilitates communication between the D2 applet and the BOCS cache
2. It affords D2 an opportunity to manipulate content on its way in and out of the repository. This is necessary, for example, when the C2 plug-in is used to place configuration based watermarks into content, or the O2 plug in is used

to set MS Office document properties based on attributes of the Documentum object.

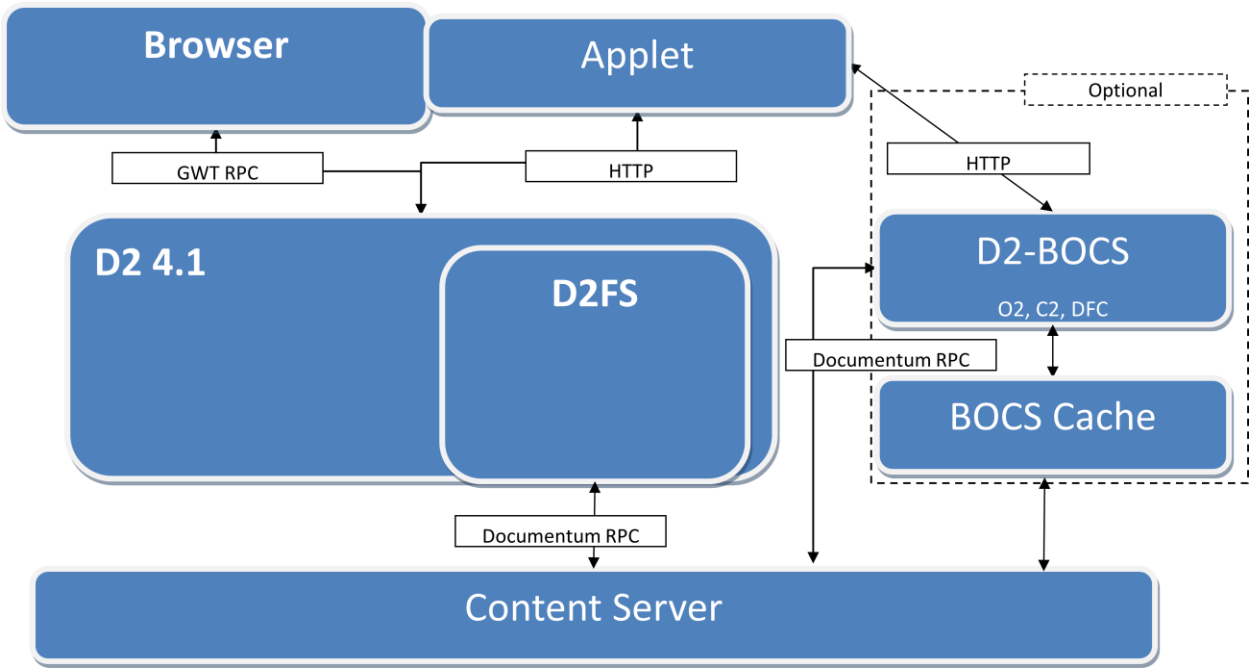

Figure 2 - Content Transfer Components

NOTE: Because D2-BOCS needs to apply configuration rules in order to complete C2 and O2 content operations, it must make a connection to the Documentum repository in order to fetch the application configuration objects as well as the metadata for the document itself. To do this, D2-BOCS opens a DFC session with the repository, and DFC makes Documentum RPC calls to the repository to fetch the required data. Since the connection between the BOCS cache and the Content Server is generally relatively high latency, and RPC connections can be chatty, this can cause some delay in content transfers. We have taken steps to limit the traffic between D2-BOCS and the Content Server, but nonetheless, in this type of deployment, it is advisable to try to limit the number of applicable O2 and C2 configurations, and to use D2 tuning parameters so that only large files are transferred through the BOCS cache. For smaller files it is often faster to connect directly to the D2 application.

D2-Config

The architecture for the D2-Config configuration client differs significantly from that of the D2 end-user client. While D2-Config is also a web 2.0 client running on a browser and application server, it is based on the older 3.x architecture for D2. Rather than using GXT to render the UI into HTML, UI is delivered to the browser in XML representation, and the D2-Config runtime uses XSL transformation to turn that XML into HTML/JavaScript UI. Another notable difference is that D2-Config, like the D2 3.1 client, relies on an ActiveX control on the browser client, and thus may only be launched from Internet Explorer browsers.

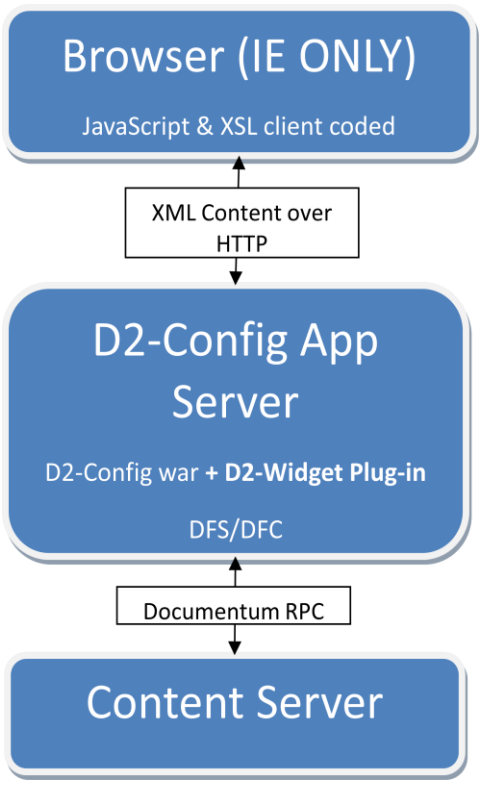

Figure 3 - D2-Config Architecture

Part 2 – Extending D2

In this section, we will discuss the various extension points available to customers and partners wishing to add additional UI or business logic not available natively in the D2 product. Prior to developing any extension, EMC recommends carefully reexamining the functionality available through D2 and its plug-ins in order to ensure that your use cases cannot be achieved through simple configuration. In almost every case, using the existing capabilities to achieve your use cases will provide a cheaper and more reliable solution than customization.
If, however, you have satisfied yourself that you really cannot meet your needs through configuration, a variety of customizations are presented below.

Invoking D2FS services Remotely

If you need to invoke D2 capabilities from another client, or background process, a good option is to use the SOAP interface for D2FS.
The D2FS API is exposed from the D2 web application as a set of SOAP services, and as a result may be called from any technology capable of consuming SOAP web services. In this section, we will look at examples for invoking D2FS from generated Java stub libraries, but it should be possible to generate stub classes using other technologies as well.
To create and use Java stub libraries for D2FS:

1. Deploy D2 to a local application server.
2. Use the **`wsimport`** command (included with your JDK) to generate a set of java source files for the stub libraries you will need to connect. Add the **`-keep`** option in order to

preserve the generated java files after the command has finished running. Depending on where you have deployed D2, the command will look something like:

```
wsimport -keep http://localhost:8080/D2/ws/d2fs.wsdl
```

3. Add code to your client to retrieve repository information, and initialize a Context object with client credentials. Your code will look something like the following:

```
static ModelPortService service = new ModelPortService();
…
ModelPort port = service.getModelPortSoap11();
GetRepositoryRequest repoReq = new GetRepositoryRequest();
repoReq.setId(repositoryId);
Repository repo = port.getRepository(repoReq).getRepository();
if (repo == null) {
    System.out.println("could not connect to repository: " +
         repositoryId);
}
Context context = new Context();
context.setRepository(repo);
context.setLogin(login);
context.setPassword(password);
context.setUid(uid);
context.setWebAppURL(webAppUrl); // e.g. "http://localhost/D2"
```

4. After you have built a Context object, you may use it to invoke services in the D2FS interface. Consult the D2FS JavaDocs for details of the various services available. For example, Figure 6 shows complete code for an example executable that calls the browser service to fetch various nodes in the browser tree.

External Widgets

External widgets provide customers and partners the ability to add new widgets to the D2 UI. External widgets are loaded into an ‹iframe› from another web application. In the 4.0 release, interactivity between external widgets and the rest of the D2 application was extremely limited; the URL for the external widget could be built from context parameters, and the widget could be reloaded on particular events, but no true interactivity was supported. D2 4.1 adds the ability to publish and subscribe to D2 application events from a simple JavaScript API. These events can be used to react to UI events, launch dialogs. As we will see later, combining D2 widgets with custom actions can even allow us to invoke D2FS services from an external UI.

For more details on adding external widgets to D2 and publishing and subscribing to events using the D2 external widget API, refer to the “EMC Documentum D2 Administrator Guide”, and “EMC Documentum D2 External Widgets” (white paper part #h11538)

D2FS Listener Plug-ins

D2FS listener plug-ins provide the ability to override existing methods in the D2FS interface. Any method in the D2FS interface may be overridden by a plug-in that subclasses the service class, and in the implementation of the overriding method,

you can choose to add logic that applies before, instead of, or after the original method. In most cases you will want to call into the parent class invoking the original logic with modified input parameters or manipulate the results before returning them, but doing so is not strictly necessary.

D2FS listener plug-ins are covered in more detail in section 3 of "EMC Documentum D2 Plugins" included with the D2 SDK. This document also provides details for packaging and building a plug-in so that it may be installed into a D2 application.

Figure 8, Figure 9, and Figure 10 show a sample plug-in that overrides the getContent and getFilteredContent methods for the content service, calling into the parent class, and then manipulating the object name property before results are returned to the client. This causes all file and folder names to appear in upper case in the contents widget.

Custom Actions

Custom menus and custom actions provide the ability to invoke D2FS services or fire UI events. Menus may be customized in the UI of D2 in one of two ways. One can either use the D2 Config UI to change menu items, or, for menu customizations that are more fixed, and associated to plug-in functionality, one can create a menu delta file. Configuration through the D2 Config UI is covered briefly in the the "EMC Documentum D2 Administrator Guide." Adding menu items through menu delta XML files is covered in more depth in Section section 5 of "EMC Documentum D2 Plugins" included with the D2 SDK. This guide is worth reading even if you plan to modify menus through the D2 Config UI, however, as it covers the various options for custom menu actions in considerably more depth.

To help understand how custom menus may be used in conjunction with D2FS to extend D2, here are a couple of examples.

The first example shows a custom menu item that calls into an existing D2FS service using a custom action. This action will call into the saveProperty method of the D2PropertyService to set the object_name property to "Special," effectively renaming the object. As you can see, the parameters we need to set for this action in D2 Config closely follow the parameters detailed in section 5 of "EMC Documentum D2 Plugins." After the service returns, the action is configured to fire the D2_ACTION_REFRESH_DOCUMENT event, which causes the UI to reload the metadata for the document, reflecting the updated name.

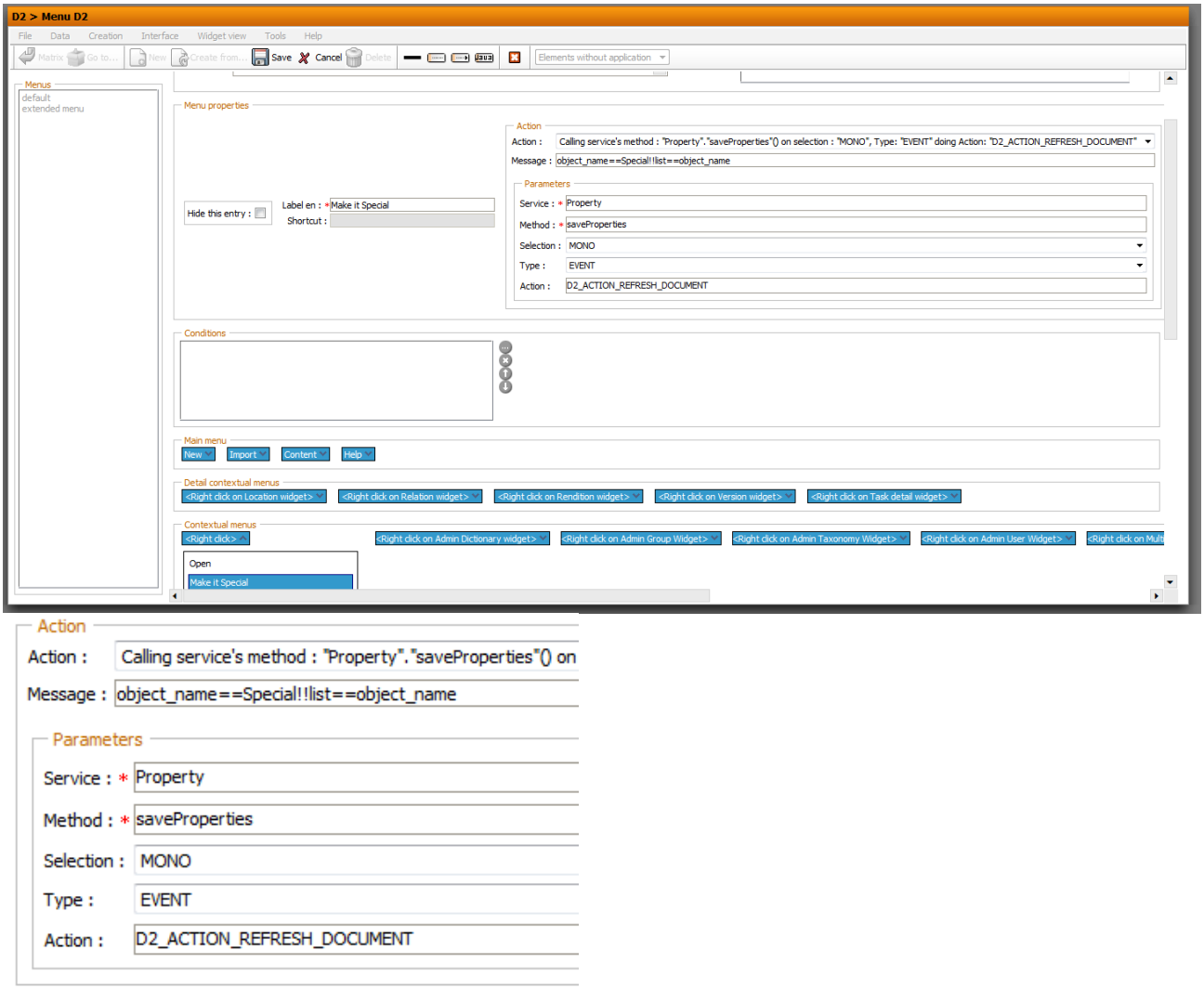

Figure 4 - Custom action settings

The second custom action example illustrates a custom action D2FS plugin, invoked from a custom menu configured through D2 Config. This sample plug-in is extremely simple, and only concatenates the input parameters that have been passed into it, and returns them in a return parameter "echoVal," but your plug-in may be far more complex, and can make multiple calls into the interfaces of D2FS to create, manipulate, or retrieve objects.

D2 Configuration for the action is shown below, and the associated code for the custom action plug-in is included in the listings section (Figure 11).

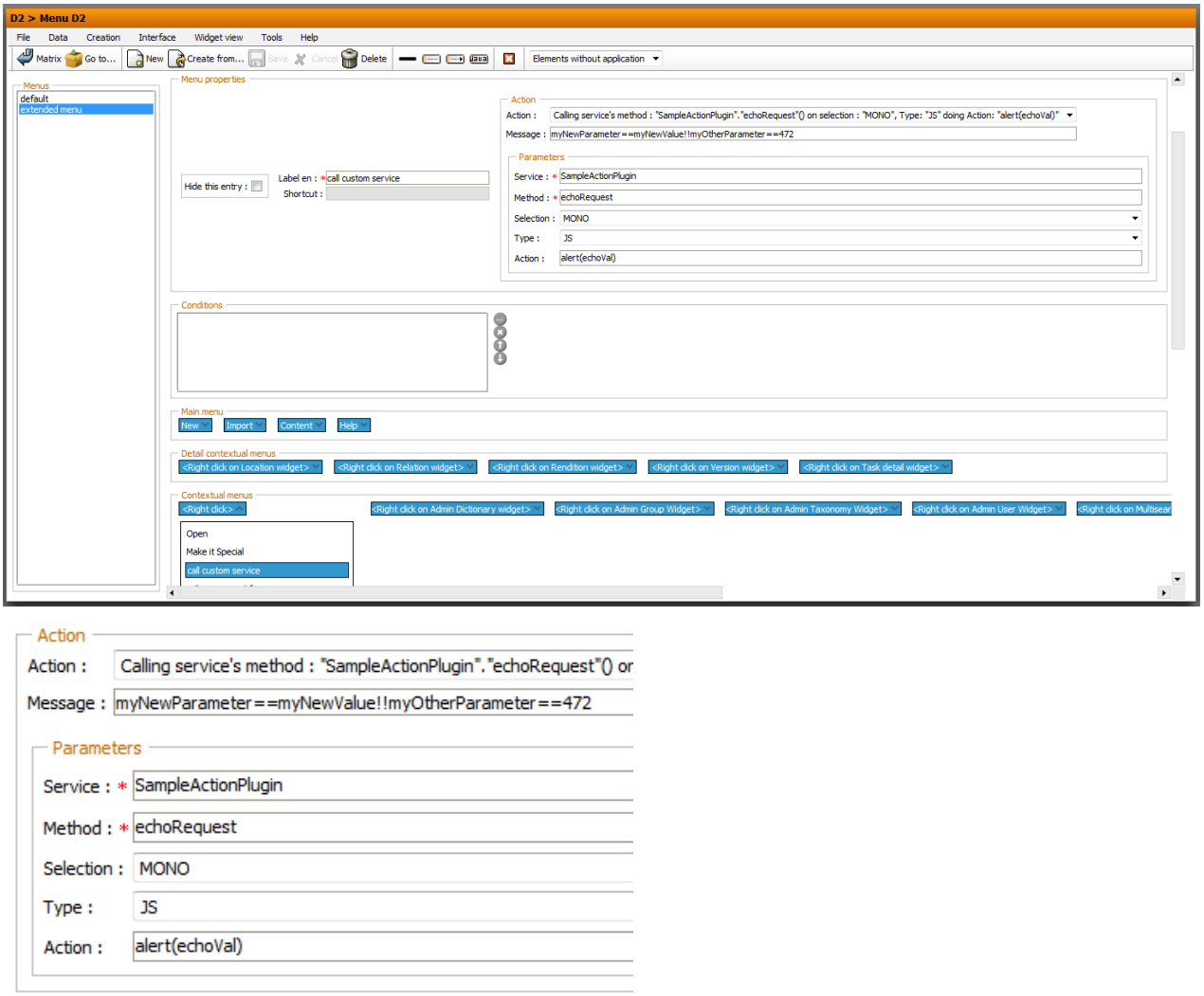

Figure 5 - Settings for calling an action plugin

Tying it all together—calling D2FS from a custom action in an external widget

The previous examples showed how to configure D2 to call existing and custom D2FS methods from a menu in the D2 UI. However, because each action is invoked by firing a D2 event, and as we have seen, D2 events may be fired by an external widget, it is also possible to use custom actions as a way to call into D2FS from an external widget. Using this approach has some limitations, but it gives us the ability to make D2FS requests without the added overhead of wiring up D2FS requests through the web application that is serving up the external widget.

This example calls into the same SampleActionPlugin we saw in the previous example. In order to be able to intercept the response of the service, we will subscribe to the D2_EVENT_WIDGET_INITIALIZED event. The widget then fires the same event that a custom menu item would fire—the D2_ACTION_EXECUTE event—and passes it the same set of parameters that we would typically set in D2 Config,

including instructions to fire the D2_EVENT_WIDGET_INITIALIZED event when the service completes:

```
function sendServiceRequest() {
    var msg = new OpenAjaxMessage();
    msg.put("myOtherParameter", 472);
    msg.put("myNewParameter", "myNewValue");
    msg.put("eService", "SampleActionPlugin");
    msg.put("CHANNEL_EVENT", "D2_ACTION_EXECUTE");
    msg.put("eMethod", "echoRequest");
    msg.put("eMode", "MONO");
    msg.put("rAction", "D2_EVENT_WIDGET_INITIALIZED");
    msg.put("rType", "EVENT");
    msg.put("parentId", "Repository");
    msg.put("oam_id", "Repository");
    d2OpenAjaxHub.sendMessage("D2_ACTION_EXECUTE", msg);
}
```

This causes the D2_EVENT_WIDGET_INITIALIZED event to be called, and the event payload to be set to the returned values from the service method. In the case of this example, we look for the "echoVal" value in the service payload, because that's the value we put there in our custom method.

Complete source for this external widget example is included in Figure 12.

NOTE: In this example, we use the D2_EVENT_WIDGET_INITIALIZED event in order to handle the service response, even though that is not the intended purpose for that event. The reason is that in the 4.1 release, it is not possible to define and use custom events from within custom actions or external widgets. In future builds, we will enhance D2 to allow you to create your own events, or provide a "D2_EVENT_CUSTOM" event that is specifically intended to carry custom event data. If you are using a later version of D2 than the 4.1 release, please check for these extensions, as re-using existing D2 events for this purpose is not ideal.

Conclusion

This white paper has presented a high level view of the D2 architecture, and some examples and explanations for common types of extension to D2. By now, you should have an idea of what types of extension are possible. For more details, please make sure to refer to the "EMC Documentum D2 Administrator Guide", "EMC Documentum D2 Plugins," and "EMC Documentum D2 External Widgets" (white paper part #h11538) as well as the D2FS Javadocs included with the D2 SDK.

Code Samples

```
// these classes can be generated from the D2FS wsdl using
// wsimport -keep
import com.emc.d2fs.models.context.Context;
import com.emc.d2fs.models.node.Node;
import com.emc.d2fs.models.repository.Repository;
import com.emc.d2fs.schemas.models.ModelPort;
import com.emc.d2fs.schemas.models.ModelPortService;
import com.emc.d2fs.services.browser_service.*;
import com.emc.d2fs.services.repository_service.*;

// see associated listing for D2fsConstants
import com.emc.d2fs.service.D2fsConstants;

public class D2Sample {
    static ModelPortService service = new ModelPortService();

    public static void main(String[] args) {

        try {
            doWsSample("xcp", "dmadmin", "password", "some unique
                 session identifier", "http://localhost:8080/D2");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void doWsSample(String repositoryId, String login,
        String password, String uid, String webAppUrl) {
        try {
            System.out.println("Retrieving the port from the following
                service: " + service);
            ModelPort port = service.getModelPortSoap11();
            System.out.println("Invoking operation on the port.");

            // Get the repository information from the docbroker
            GetRepositoryRequest repoReq = new GetRepositoryRequest();
            repoReq.setId(repositoryId);
            Repository repo =
                port.getRepository(repoReq).getRepository();
            if (repo == null)
                System.out.println("could not connect to repository: " +
                    repositoryId);

            Context context = new Context();
            context.setRepository(repo);
            context.setLogin(login);
            context.setPassword(password);
            context.setUid(uid);
            context.setWebAppURL(webAppUrl);

            // Validate user credentials
            CheckLoginRequest checkLoginRequest = new
                CheckLoginRequest();
            checkLoginRequest.setContext(context);
            CheckLoginResponse checkLoginResponse =
                port.checkLogin(checkLoginRequest);
            if (!checkLoginResponse.isResult())
                System.out.println("login failed");

            // Get root nodes for the browser tree
```

```
            GetBrowserContentRequest getBrowserContentRequest = new
                GetBrowserContentRequest();
            getBrowserContentRequest.setContext(context);

            getBrowserContentRequest.setContentTypeName(D2fsConstants.ROOT);
            GetBrowserContentResponse response =
                port.getBrowserContent(getBrowserContentRequest);
            System.out.println("root nodes:");
            for (Node node: response.getNode().getNodes()) {
                System.out.printf("    id:%s type:%s label:%s\n", node.getId(),
                    node.getType(), node.getLabel());
            }

            // Get cabinets in the repository
            getBrowserContentRequest.setContext(context);
            getBrowserContentRequest.setId(repositoryId);

            getBrowserContentRequest.setContentTypeName(
                D2fsConstants.REPOSITORY);
            response = port.getBrowserContent(getBrowserContentRequest);
            System.out.println("cabinets in repository " + repositoryId + ":");
            for (Node node: response.getNode().getNodes()) {
                System.out.printf("    id:%s type:%s label:%s\n", node.getId(),
                    node.getType(), node.getLabel());
            }

            // Get contents of the user home cabinet
            String homeId=null;
            for (Node node: response.getNode().getNodes()) {
                if (node.getLabel().equals(login)) {
                    homeId = node.getId();
                    break;
                }
            }
            if (homeId != null) {
                getBrowserContentRequest.setContext(context);
                getBrowserContentRequest.setId(homeId);
                getBrowserContentRequest.setContentTypeName(
                    D2fsConstants.FOLDER);
                response = port.getBrowserContent(getBrowserContentRequest);
                System.out.println("home cabinet contents:");
                for (Node node: response.getNode().getNodes()) {
                    System.out.printf("    id:%s type:%s label:%s\n",
                        node.getId(), node.getType(), node.getLabel());
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 6 - Calling D2FS remotely: D2Sample.java

```
package com.emc.d2fs.service;

public interface D2fsConstants {

    //ContentTypeName
    public final static String REPOSITORY = "Repository";
    public final static String FOLDER = "Folder";
    public final static String VD = "vd";
    public final static String NODE_FAVORITES = "node_favorites";
    public final static String ROOT = "Root";
    public final static String SPACE_CABINET = "space_cabinet";
    public final static String DMC_ROOM = "dmc_room";
    public final static String DISTRIBUTION_REPORT = "distribution_report";

    public final static String NODE_SEARCH = "node_search";
    public final static String NODE_CHECKOUT = "node_checkout";
    public final static String NODE_LAST_SEARCH = "node_last_search";
    public final static String NODE_SAVED_SEARCHES = "node_saved_searches";
    public final static String NODE_SAVED_PUBLIC_SEARCHES =
        "node_saved_public_searches";
    public final static String NODE_FACET_VALUE = "d2_facet_value";
    public final static String NODE_FACET_DEFINITION = "d2_facet_definition";
    public final static String NODE_ASYNC_SEARCHES = "node_async_searches";
    public final static String NODE_INBOX_TASKS = "node_inbox_tasks";
    public final static String NODE_INBOX_NOTIFICATIONS =
        "node_inbox_notifications";
    public final static String NODE_ADMIN_USERS = "node_admin_users";
    public final static String NODE_ADMIN_GROUPS = "node_admin_groups";
    public final static String NODE_ADMIN_DICTIONARIES =
         "node_admin_dictionaries";
    public final static String NODE_ADMIN_TAXONOMIES = "node_admin_taxonomies";
    public final static String D2C_QUERY = "d2c_query";
    public final static String D2C_QUERY_WIZARD = "d2c_query_wizard";
    public final static String D2C_QUERY_FORM = "d2c_query_form";
    public final static String D2_QUERY_FORM_CONFIG = "d2_queryform_config";
    public final static String D2C_QUERY_CATEGORY = "d2c_query_category";
    public final static String DMC_RPS_RETENTION_POLICY =
        "dmc_rps_retention_policy";
    public final static String DMC_RPS_RETENTION_MARKUP =
        "dmc_rps_retention_markup";
    public static final String NODE_DISTRIBUTION = "node_distribution";

    //ViewMode
    public final static String VIEW_MODE_DETAILS = "details";
    public final static String VIEW_MODE_THUMBNAILS = "thumbnails";

    //CheckChildren
    public final static String CHECK_GROUP = "group";
    public final static String CHECK_FOLDER = "folder";
    public final static String CHECK_FOLDER_DOCUMENT = "folderOrDocument";
    public final static String CHECK_FOLDER_VD = "folderOrVD";

    //DetailTypeName
    public final static String AUDITS = "Audits";
    public final static String LOCATIONS = "Locations";
    public final static String RELATIONS = "Relations";
    public final static String RENDITIONS = "Renditions";
    public final static String SNAPSHOTS = "Snapshots";
    public final static String VERSIONS = "Versions";
    public final static String RETENTIONS = "Retentions";
    public final static String MARKUPS = "Markups";
    public final static String WORKFLOW_DETAILS = "WorkflowDetails";
    public final static String WORKFLOW_NOTES = "WorkflowNotes";
```

```
public final static String WORKFLOW_PERFORMERS = "WorkflowPerformers";

//ColTypeName
public final static String NODE_CABINETS = "node_cabinets";
public final static String DETAIL_AUDITS = "detail_audits";
public final static String DETAIL_AUDITS_EXTENDED =
    "detail_audits_extended";
public final static String DETAIL_LOCATIONS = "detail_locations";
public final static String DETAIL_RELATIONS = "detail_relations";
public final static String DETAIL_RENDITIONS = "detail_renditions";
public final static String DETAIL_RETENTIONS = "detail_retentions";
public final static String DETAIL_MARKUPS = "detail_markups";
public final static String SEARCH = "search";
public final static String DEFAULT = "default";
public final static String DETAIL_VERSIONS = "detail_versions";
public final static String DETAIL_WORKFLOWS = "detail_workflows";
public final static String DETAIL_WORKFLOW_NOTES = "detail_workflow_notes";
public final static String DETAIL_WORKFLOW_PERFORMERS =
    "detail_workflow_performers";
public final static String DETAIL_WORKFLOW_RELATIONS =
    "detail_workflow_relations";
public static final String DETAIL_WORKFLOW_OVERVIEW =
    "detail_workflow_overview";
public final static String DETAIL_WORKFLOW_ATTACHMENTS =
    "detail_workflow_attachments";

//DialogType
public final static String DIALOG_TYPE_DIALOG = "dialog";
public final static String DIALOG_TYPE_MESSAGE = "message";
public final static String DIALOG_TYPE_RESULT = "result";
public final static String DIALOG_TYPE_SILENT = "silent";

//MenuTypeName
public final static String MENU_PORTAL = "MenuPortal";

public final static String MENU_CTX_MULTISEARCH = "MenuContextMultiSearch";

public final static String MENU_CTX = "MenuContext";
public final static String MENU_CTX_VD = "MenuContextVD";

public final static String MENU_CTX_DETAIL_LOCATIONS =
    "MenuContextDetailLocations";
public final static String MENU_CTX_DETAIL_RELATIONS =
    "MenuContextDetailRelations";
public final static String MENU_CTX_DETAIL_RENDITIONS =
    "MenuContextDetailRenditions";
public final static String MENU_CTX_DETAIL_VERSIONS =
    "MenuContextDetailVersions";
public final static String MENU_CTX_DETAIL_WORKFLOWS =
    "MenuContextDetailWorkflows";
public final static String MENU_CTX_DETAIL_WORKFLOWS_OVERVIEW =
    "MenuContextWorkflowOverview";

public final static String MENU_CTX_TASKS_LIST = "MenuContextTasksList";
public final static String MENU_CTX_DETAIL_WORKFLOW_DETAILS =
    "MenuContextDetailWorkflowDetails";

public final static String MENU_CTX_ADMIN_USERS = "MenuContextAdminUsers";
public final static String MENU_CTX_ADMIN_GROUPS = "MenuContextAdminGroups";
public final static String MENU_CTX_ADMIN_DICTIONARIES =
    "MenuContextAdminDictionaries";
public final static String MENU_CTX_ADMIN_TAXONOMIES =
    "MenuContextAdminTaxonomies";
```

```
    public final static String MENU_CTX_TREE_NODE_TASK =
        "MenuContextTreeNodeInboxTasks";
    public final static String MENU_CTX_TREE_NODE_SEARCH =
        "MenuContextTreeNodeSearch";
    public final static String MENU_CTX_SAVE_SEARCH_DIALOG =
        "MenuContextSaveSearchDialog";


    public final static String PARAM_WIDGET_TYPE = "widgetType";
    public final static String PARAM_CONTENT_PATH_ID = "contentPathId";

    //ExportType
    public final static String EXPORT_EXCEL = "Excel";

//  // DfType / Attribute Typer
//    public final static final int DF_BOOLEAN = 0;
//    public final static final int DF_INTEGER = 1;
//    public final static final int DF_STRING = 2;
//    public final static final int DF_ID = 3;
//    public final static final int DF_TIME = 4;
//    public final static final int DF_DOUBLE = 5;
//    public final static final int DF_UNDEFINED = 6;

    //DialogName
    public final static String IMPORT = "ImportDialog";
    public final static String MULTI_IMPORT = "MultiImportDialog";
    public final static String IMPORT_RENDITION = "ImportRenditionDialog";
    public final static String CREATE = "CreateDialog";
    public final static String IMPORT_ATTACHMENT = "ImportAttachmentDialog";
    public final static String PROPERTIES = "PropertiesDialog";
    public final static String D2WORKFLOW_LAUNCH = "D2WorkflowLaunchDialog";
    public final static String D2WORKFLOW_NOTE = "D2WorkflowNoteDialog";
    public final static String DISTRIBUTION_DIALOG = "DistributionDialog";
    public final static String TASK = "TaskDialog";
    public final static String TASK_PROPERTY = "TaskPropertyPageDialog";
    public final static String TASK_DELEGATE = "TaskDelegateDialog";
    public final static String SIGN_OFF = "SignOffDialog";
    public final static String QUERY_FORM = "QueryFormDialog";
    public final static String PREFERENCES = "PreferencesDialog";
    public final static String CHOOSE_TYPE = "ChooseTypeDialog";
    public final static String DESTROY = "DestroyDialog";
    public final static String CHANGE_STATE = "ChangeStateDialog";
    public final static String CHECKIN = "CheckinDialog";
    public final static String PERMISSIONS = "PermissionsDialog";
    public final static String CHANGE_PERMISSIONS = "ChangePermissionsDialog";
    public final static String PRINT = "PrintDialog";
    public final static String COMMENT = "CommentDialog";
    public final static String ANNOTATION_CONTROL = "AnnotationControlDialog";
    public final static String ADVANCED_SEARCH="AdvancedSearchDialog";
    public final static String SAVE_SEARCH = "SaveSearchDialog";
    public static final String DM_USER = "DmUserDialog";
    public static final String DM_GROUP = "DmGroupDialog";
    public final static String ADMIN_DICTIONARY = "AdminDictionaryDialog";
    public final static String EXPORT_DICTIONARY = "ExportDictionaryDialog";
    public final static String IMPORT_DICTIONARY = "ImportDictionaryDialog";
    public final static String SAVE_VERSION_DICTIONARY =
        "SaveVersionDictionaryDialog";
    public final static String LOAD_VERSION_DICTIONARY =
        "LoadVersionDictionaryDialog";
    public final static String SYNCPLICITY = "SyncplicityDialog";
    public final static String EXPORT_TAXONOMY = "ExportTaxonomyDialog";
    public final static String IMPORT_TAXONOMY = "ImportTaxonomyDialog";
```

```
    public final static String SAVE_VERSION_TAXONOMY =
        "SaveVersionTaxonomyDialog";
    public final static String LOAD_VERSION_TAXONOMY =
        "LoadVersionTaxonomyDialog";
    public final static String RELATION_CREATE = "RelationCreateDialog";
    public final static String MASS_UPDATE_DIALOG = "MassUpdateDialog";
    public final static String MASS_UPDATE_MODE_DIALOG = "MassUpdateModeDialog";
    public final static String SET_CHILD_BINDING_VERSIONS =
        "SetVDChildBindingVersionDialog";
    public final static String VD_INHERITED_TEMPLATES = "VdTemplateDialog";
    public final static String EXPORT_VALUES = "ExportValuesDialog";
    public final static String IMPORT_VALUES = "ImportValuesDialog";
    public final static String QUERY_CATEGORY = "QueryCategoryDialog";

    // Documentum type
    public final static String DM_FOLDER = "dm_folder";

    public final static String FORMATS_PSEUDO_DIALOG = "FormatsPseudoDialog";
}
```

Figure 7 - Calling D2FS remotely: D2fsConstants.java

```
package com.emc.sample.api.services.plugins;

import com.emc.common.java.utils.IVersion;

import com.emc.d2fs.dctm.web.services.ID2fsPlugin;
import com.emc.d2fs.dctm.web.services.content.D2ContentService;
import com.emc.d2fs.models.attribute.Attribute;
import com.emc.d2fs.models.context.Context;
import com.emc.d2fs.models.item.DocItems;
import com.emc.d2fs.models.item.Item;

import com.emc.sample.api.SampleVersion;

public class D2ContentServicePlugin extends D2ContentService implements
        ID2fsPlugin {

    private static final IVersion VERSION = new SampleVersion();

    @Override
    public DocItems getContent(Context context, String contentId,
            String contentTypeName, String viewMode, String checkChildren)
            throws Exception {
        DocItems result = super.getContent(context, contentId, contentTypeName,
                viewMode, checkChildren);

        for (Item item : result.getItems()) {
            for (Attribute attr : item.getAttributes()) {
                if (attr.getName().equals("object_name")) {
                    attr.setValue(attr.getValue().toUpperCase());
                    break;
                }
            }
        }

        return result;
    }

    @Override
    public DocItems getFilteredContent(Context context, String contentId,
            String contentTypeName, String viewMode, String checkChildren,
            String filter, String filterPropName, String locateId)
            throws Exception {
        DocItems result = super.getFilteredContent(context, contentId,
                contentTypeName, viewMode, checkChildren, filter,
                filterPropName, locateId);

        for (Item item : result.getItems()) {
            for (Attribute attr : item.getAttributes()) {
                if (attr.getName().equals("object_name")) {
                    attr.setValue(attr.getValue().toUpperCase());
                    break;
                }
            }
        }

        return result;
    }

    @Override
    public String getFullName() {
        return VERSION.getFullName();
    }
```

```
    @Override
    public String getProductName() {
        return VERSION.getProductName();
    }

}
```

Figure 8 - D2FS listener plugin: D2ContentServicePlugin.java

```
package com.emc.sample.api;

import com.emc.common.java.utils.AbstractVersion;
import com.emc.common.java.utils.IVersion;

public class SampleVersion extends AbstractVersion {
    public static void main(String[] args) {
        IVersion version = new SampleVersion();
        System.out.println(version.getFullName());
    }
}
```

Figure 9 - D2FS listener plugin: SampleVersion.java

```
project.name=Sample
build.number=001
project.package=com.emc.sample
project.version=2.0.0.0
build.number.docappextension=b
compagny.name=EMC
build.number.extension=(beta)
```

Figure 10 - D2FS listener plugin: SampleVersion.properties

```
package com.emc.sample.api.services.plugins;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import com.documentum.fc.common.DfException;

import com.emc.d2fs.dctm.plugin.IPluginAction;
import com.emc.d2fs.dctm.utils.ParameterParser;
import com.emc.d2fs.dctm.web.services.D2fsContext;
import com.emc.d2fs.exceptions.D2fsException;
import com.emc.d2fs.models.attribute.Attribute;
import com.emc.d2fs.utils.AttributeUtils;

public class SampleActionPlugin implements IPluginAction {
    public List<Attribute> echoRequest(D2fsContext context)
            throws D2fsException, DfException {
        DateFormat dateFormat =
            new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        StringBuilder sb = new StringBuilder();
        sb.append("current time=").
            append(dateFormat.format(new Date()));

        ParameterParser parameters = context.getParameterParser();
        if (parameters != null) {
            for (Attribute a : parameters.getParameters()) {
                sb.append("   ").append(a.getName()).append('=')
                        .append(a.getValue());
            }
        }

        Attribute attr = AttributeUtils.createAttribute("echoVal",
                sb.toString());
        ArrayList<Attribute> ret = new ArrayList<Attribute>();
        ret.add(attr);
        return ret;
    }
}
```

Figure 11 - D2FS action plug-in: SampleActionPlugin.java

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title></title>

<script language='javascript' src="OpenAjaxManagedHub-all.js"></script>
<script language='javascript' src="D2-OAH.js"></script>

<script type="text/javascript">
    var d2OpenAjaxHub = new D2OpenAjaxHub();

    /* Callback that is invoked upon successful
       connection to the Managed Hub */
    function connectCompleted(hubClient, success, error) {
        if (!success)
            alert("Hub client NOT connected");
    }

    /* Callback that is invoked at widget init */
    function widgetInit(oMessage) {
        subscribeToResponseChannel()
    }

    function loadEventHandler() {
        //connecting hubclient
        d2OpenAjaxHub.connectHub(connectCompleted,
                widgetInit, onActiveWidget);
    }

    /* Callback that is invoked on receiving data*/
    function onServiceResponse(event, oMessage) {
        var responseValue = oMessage.get("echoVal");
        if (responseValue)
            alert("service response received:" + responseValue);
    }

    /* Callback that is invoked on widget activation*/
    function onActiveWidget(bActiveFlag) {
    }

    function subscribeToResponseChannel() {
        d2OpenAjaxHub.subscribeToChannel(
                "D2_EVENT_WIDGET_INITIALIZED",
                onServiceResponse);
    }
    function sendServiceRequest() {
        var msg = new OpenAjaxMessage();
        msg.put("myOtherParameter", 472);
        msg.put("myNewParameter", "myNewValue");
        msg.put("eService", "SampleActionPlugin");
        msg.put("CHANNEL_EVENT", "D2_ACTION_EXECUTE");
        msg.put("eMethod", "echoRequest");
        msg.put("eMode", "MONO");
        msg.put("rAction", "D2_EVENT_WIDGET_INITIALIZED");
        msg.put("rType", "EVENT");
        msg.put("parentId", "Repository");
        msg.put("oam_id", "Repository");
```

```
        d2OpenAjaxHub.sendMessage("D2_ACTION_EXECUTE", msg);
    }
</script>
</head>
<body onload="loadEventHandler();">
    <center>
        <input type="button" value="Send service request"
            onclick="sendServiceRequest()" />
    </center>
</body>
</html>
```

Figure 12 - Calling D2FS from an external widget: index.html