Abstract

This white paper explains some common use cases for integrating D2 functionality into other clients, which types of D2 functionality will work out of the box, and which can be achieved with lightweight integration.

March 2013
# Table of Contents

**Summary** .................................................................................................................. 4  
**Audience** ...................................................................................................................... 4  
**Basic Interoperability** ................................................................................................. 4  
  *Reading D2 Objects from Other Clients* ......................................................................... 4  
    *What to watch for* ...................................................................................................... 5  
  *Sandboxing D2 from other Application Data* ................................................................. 5  
  *Asynchronous Behaviors of D2 that can be Used without Integration* ....................... 6  
**Read/Write Interoperability using Only D2CoreMethod** .............................................. 6  
**Extended Interoperability using D2FS** .......................................................................... 7  
**Interoperability with TBOs, SBOs, and other BOF Customizations** ......................... 12  
  *SBOs* .......................................................................................................................... 12  
  *TBOs and aspects* ...................................................................................................... 12  
**Conclusion** .................................................................................................................. 13  
**Code Samples** ............................................................................................................ 14
Summary
This paper provides guidelines to customers and partners wishing to integrate Documentum D2® into an existing or new installation, in conjunction with custom clients built using Documentum Foundation Services® (DFS) or Documentum Foundation Classes® (DFC). In it we will describe the use cases that should work out of the box, those requiring only minimal changes, and the features of D2 that would require extensive work to expose in other clients.

We will also cover some potential issues to be aware of during integration.

Audience
This white paper is intended for customers and partners who already have a basic understanding of the features of D2, DFC, and the Documentum Business Object Framework (BOF).

Basic Interoperability
For many basic use cases, D2 can coexist with other clients installed into the same repository without issues. D2 stores property values, content, renditions, and permissions as standard Documentum objects in the repository, so basic interoperability is assured on this basis.

Reading D2 Objects from Other Clients
In most cases, no special attention is required in order for other clients to consume or display documents, folders, or object metadata that has been created from D2. Again, D2 uses standard representations for data in the repository, so most of the logic in your external client should work as expected when browsing content created by D2. Expect standard Documentum behavior for all of the following:

- Folder hierarchy/object location
- ACLs
- Document content
- Versions
- Virtual documents
- Relationships
- Renditions
- Object properties
What to watch for

- If you use D2 dictionary configurations to map values into labels in the D2 UI, your client will get the raw dictionary value when querying objects through DFC, DFC, or DQL. Possible mitigation strategies include using D2 dictionaries with keys only (no labels), using calls into D2FS to find the correct display label, or configuring the dictionary as a DQL dictionary, and looking up the correct label through DQL.

- If you use C2 view, print, or export configurations to automatically apply watermarking to content during these operations, other clients will not show the watermarks configured in D2. Available workarounds include switching to rendition configurations (which store a watermarked rendition in the repository on object save) or calling into D2FS in order to get the URI for the content with the watermark applied as necessary.

Sandboxing D2 from other Application Data

If you need to install D2 into the same repository as another client, but you do not wish users of D2 to interact with the data for that client, D2 offers a variety of configurations to allow the administrator to hide data from the end user.

- Contents of the document list widget may be filtered using Filter configurations
- Search and query form configurations may be used to control the availability and scope of searches
- Checkout and property configurations may be used to control write access to content in cases where D2 read-only access is required. Create a new checkout configuration that you will use for your read-only content, uncheck the “allow checkout” box as shown in Figure 1 and then associate that configuration to a context describing the documents you want to suppress. Similarly, you can create properties configurations with edit controls and/or the OK button disabled to disallow changing of properties from within D2, as shown in Figure 2

![Figure 1. Unchecking "Allow checkout"](image)
Menu configurations may be used to constrain the availability of actions on content in contexts where it is desired to disable properties changes. (Not available in 4.0—only in 3.x and 4.1)

Asynchronous Behaviors of D2 that can be Used without Integration

D2 includes some features that are triggered by background processes, and do not require any direct interaction from client applications in order to take effect. As a result, these features can be used across clients with no integration work required, beyond initial set up of the D2 configuration:

- Mail list/mail server configurations may be set up to send notifications for new content, changed content, or significant changes in state to content (for example, when a request document reaches “approved” state).
- Lifecycle batch configurations may be used to automatically move objects in a D2 lifecycle into another lifecycle state, based on properties changes or other criteria.
- BOCS cache configurations may be used to selectively pre-populate content into BOCS caches for faster first time retrieval from remote locations.

As with all D2 configurations, care should be taken when designing the contexts and configurations with these options to ensure that they will not conflict with the business logic installed in other clients.

Read/Write Interoperability using Only D2CoreMethod

The D2CoreMethod is a Java method installed as part of the standard D2 installation. It runs on the Java Method Server (JMS), and it is invoked by D2 each time an object is saved. It is responsible for enforcing configuration rules for auto-linking, auto-naming, and security.
For DFC or DFS clients wishing to enforce D2 configuration rules on content without directly invoking D2FS or linking D2 libraries, it is possible to do this for the subset of configurations that are implemented within the D2 core method, by invoking the core method after the object is saved, just as D2 itself does.

The following configurations may be invoked using the D2 core method:

- **Auto Link**: Calculates locations for content based on metadata and creates folders as necessary to materialize those locations.
- **Auto Naming**: Creates or modifies object names based on metadata and/or numbering rules.
- **Security Template**: Calculates & creates ACLs for content based on object metadata.

For a sample illustrating direct invocation of the D2CoreMethod from DFC, please refer to Figure 3.

For additional usage information regarding the parameters and usage of the D2 core method, please refer to the EMC Documentum D2 Administrator Guide, Appendix B, D2 Jobs reference, under D2CoreJob.

### Extended Interoperability using D2FS

For DFC or DFS clients wishing to make use of D2 configuration that is not available through the D2CoreMethod, and is not invoked in the background through batch processing, it should be possible to invoke most of the features in D2 using the D2FS API. In some cases however, it would be necessary to also enhance the client UI, in order to represent the dialogs, controls, etc. ordinarily shown by D2 when the feature is used. The following table presents a list of the configuration modules included in D2, indicates whether or not they can be invoked through D2FS, and also indicates whether or not additional UI would be required, beyond what would ordinarily appear in, for example, a WDK application.

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Available in D2CoreMethod</th>
<th>Available in D2FS</th>
<th>Requires Specific UI (not in WDK)</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-Naming</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>creates or modifies object names based on metadata and/or numbering rules</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>calculates &amp; creates ACLs for content based on metadata</td>
<td></td>
</tr>
<tr>
<td>Auto-Link</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>calculates location(s) for content based on metadata and creates folders as necessary to materialize those locations</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Option 1</td>
<td>Option 2</td>
<td>Option 3</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Checkin</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls application behaviors on check in (silent versus dialoged, major versus minor version, mandatory description.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Check-in UI must be customized to respect configurations for version policy etc.</td>
<td></td>
</tr>
<tr>
<td>Linked Document</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>specifies secondary documents that can/must be linked to the primary document during object creation (using either relations or virtual documents)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Requires the D2 check-in UI in order to add the linked document.</td>
<td></td>
</tr>
<tr>
<td>Rendition Server</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>configures settings for which rendition server is used to request a rendition</td>
<td></td>
</tr>
<tr>
<td>Menu</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls visibility of menu actions in the D2 UI</td>
<td></td>
</tr>
<tr>
<td>Toolbar</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls contents of toolbars in the D2 UI</td>
<td></td>
</tr>
<tr>
<td>Properties dialog</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls display of properties UI in both the dialog and widget form.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The D2FS returns an XML representation of the dialog that must be rendered into UI.</td>
<td></td>
</tr>
<tr>
<td>Templates</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls which content templates are used as a starting point when creating new document content (as opposed to importing content)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The D2 creation process requires attributes to be set before the list of templates can be chosen, not after, as in WDK.</td>
<td></td>
</tr>
<tr>
<td>Creation Matrix</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls import dialog flow, including which default values to use, what property page to use, what type of properties inheritance to apply, and any lifecycles and workflows to apply</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The creation matrix controls the D2 import and creation process, which differs significantly from WebTop/WDK import.</td>
<td></td>
</tr>
<tr>
<td>Default Values Template</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls initial values for object metadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Templates are controlled via a creation matrix, requiring user input.</td>
<td></td>
</tr>
<tr>
<td>Display Filters</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>filters which objects appear in the document list</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>D2FS</td>
<td>D2 UI</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls global search properties, such as which engine should be used, whether full text is allowed, and the maximum number of results that can be returned. Search configuration requires D2 search forms.</td>
<td></td>
</tr>
<tr>
<td>Workspace</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls layout of widgets presented in the D2 UI.</td>
<td></td>
</tr>
<tr>
<td>Widget</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls binding of D2 UI components into a the workspace.</td>
<td></td>
</tr>
<tr>
<td>Theme</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls color scheme for the UI.</td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>controls which types and properties are visible to the end user in the UI. D2FS returns an XML representation of the dialog that must be rendered into UI.</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>controls which types and properties are visible to the end user in the UI. D2FS returns an XML representation of the dialog that must be rendered into UI.</td>
<td></td>
</tr>
<tr>
<td>Query Form</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>custom query form UI, similar to a properties dialog, stored in &quot;public searches&quot;.</td>
<td></td>
</tr>
<tr>
<td>Subscription</td>
<td>no</td>
<td>yes</td>
<td>see note</td>
<td>sends mails to subscribers when events occur on content. Subscription requires D2 UI, but D2FS will register events on CRUD operations. Method server job processes &amp; sends actual emails. So any object saved through D2FS will trigger sending of subscription emails.</td>
<td></td>
</tr>
<tr>
<td>Template</td>
<td>no</td>
<td>yes</td>
<td>see note</td>
<td>controls additional behaviors/constraints/UI on top of existing Documentum workflows.</td>
<td></td>
</tr>
<tr>
<td>Workflow</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls state transitions through a D2 lifecycle. D2 lifecycles are independent of Documentum lifecycles, so D2FS should be used to interact with them.</td>
<td></td>
</tr>
<tr>
<td>Lifecycle</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls state transitions through a D2 lifecycle. D2 lifecycles are independent of Documentum lifecycles, so D2FS should be used to interact with them.</td>
<td></td>
</tr>
<tr>
<td>Relations</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>controls which type of relations may be added to a document. Add relationship UI in WDK presents a list of relationship types.</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Default</td>
<td>Yes</td>
<td>No</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-----</td>
<td>----</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Uniqueness Check</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>enforces arbitrary configuration based constraints at object save/creation</td>
<td></td>
</tr>
<tr>
<td>Mail Server</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>configures outbound &amp; inbound mail servers (inbound servers are used for tasks interactions)</td>
<td>When saving content through D2FS, the mail server configurations will be used &amp; mail sent.</td>
</tr>
<tr>
<td>Mail Attachments</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>controls what happens to attachments on a message imported into the repository (whether it is left attached to the message in original format and/or stored as a rendition in the repository. If the latter, it can also be filtered on content type.)</td>
<td></td>
</tr>
<tr>
<td>Structure Import</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>controls folder types &amp; security when importing folder hierarchies</td>
<td></td>
</tr>
<tr>
<td>Checkout</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>controls application behaviors for checkout</td>
<td></td>
</tr>
<tr>
<td>Copy Not Allowed</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>stops content in some contexts from being copied</td>
<td></td>
</tr>
<tr>
<td>Properties Inheritance</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>default/metadata values are inherited from another document. This configuration controls which properties are picked up.</td>
<td>This option requires the client to choose a source object. Apart from that, default values could be presented in a standard UI.</td>
</tr>
<tr>
<td>Bulk/Mass update</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>controls UI for updating properties on multi-select</td>
<td></td>
</tr>
<tr>
<td>Plug In</td>
<td>no</td>
<td>yes</td>
<td>(indirectly) yes</td>
<td>controls which plug-in actions &amp; UI are available to the end user</td>
<td></td>
</tr>
<tr>
<td>Module</td>
<td>Activated</td>
<td>Authorized</td>
<td>Restricted</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Audit</td>
<td>no (see note)</td>
<td>yes</td>
<td>no</td>
<td>records object lifecycle events for CRUD operations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Core Documentum audit trail events remain unchanged by D2. Only D2 events require calling through D2FS to be triggered.</td>
<td></td>
</tr>
<tr>
<td>Mailing List</td>
<td>n/a--batch job</td>
<td>n/a</td>
<td>n/a</td>
<td>sends mails to distribution lists automatically for documents matching specific contexts</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Executed by a job on the method server, so no action is required to enable this configuration.</td>
<td></td>
</tr>
<tr>
<td>Lifecycle Batch</td>
<td>n/a--batch job</td>
<td>n/a</td>
<td>n/a</td>
<td>executes lifecycle transitions automatically based on document context.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Executed by a job on the method server, so no action is required to enable this configuration.</td>
<td></td>
</tr>
<tr>
<td>BOCS Cache</td>
<td>n/a--batch job</td>
<td>n/a</td>
<td>n/a</td>
<td>pre-populates content into BOCS cache so that it will not be slow to retrieve the content for the first user to fetch it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Executed by a job on the method server, so no action is required to enable this configuration.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Configuration Modules
Interoperability with TBOs, SBOs, and other BOF Customizations

In general, because D2 uses standard DFS and DFC API calls to manipulate content in the repository, BOF level customizations included with other applications should continue to work as they would with other Documentum clients. However, some types of TBOs and aspects may not operate as expected during D2 save operations. This is due to the fact that during a normal save of an object within D2, multiple configurations may be applied to the object, and each configuration module will itself perform a DFC save operation. As a result, TBOs or aspects that customize the save operation may encounter logic or performance issues.

SBOs

There are no known issues with installing SBOs into a D2 enabled repository. Since D2 itself does not install or invoke SBOs, any SBOs added to the repository should not affect the behavior of D2.

TBOs and aspects

As mentioned above, it is primarily TBOs and aspects customizing object save that could face issues when installed together with D2. However, whether or not a customization will face issues depends heavily on the nature of the customization:

- If the operation logic in the TBO or aspect is idempotent (meaning that calling it multiple times has the same effect as calling it once) and it does not take very long to run, then it may operate smoothly alongside D2.
- Custom save operations that consume a lot of CPU, call outside services, or make multiple calls to the content server may cause object saving in D2 to become slow, due to the fact that the customization will be invoked multiple times.
- Customizations that perform consistency checking during save operations may cause the operation to fail, if the consistency check depends on the results of D2 configurations. Since the object may be saved multiple times before configuration is fully applied, some of the intermediate saves may fail consistency checking, due to the fact that the operation is not yet complete.

Possible mitigations for conflicts between BOF level customizations and D2 runtime logic include the following.

- In many cases, logic that formerly required a TBO can be replaced by D2 configuration. If the application needing the TBO is to be replaced with D2, this is usually the best approach. This approach may also be possible for non D2 clients in cases where the TBO logic can be emulated using configurations that are available in the D2CoreMethod.
- Reducing the number of configurations that apply to object types having the TBO will reduce the total number of object saves, and therefore reduce the number of times the customized save operation is invoked. It is easy to create a D2 context specific to an object type for this purpose.
• If you do not need the custom save logic to be invoked when an object is saved from D2, you can add a check to your code to detect whether or not the code is called from a D2 class, and skip the custom logic accordingly. See Figure 4 and Figure 5 for listings of a sample TBO that checks whether or not it is called from D2.

Conclusion

As we have seen in this paper, basic interoperability between D2 and other clients should be achievable without modifying other clients to call into D2FS. When more extensive interoperability is required, or it is desired to make use of more types of configuration from outside of D2, D2FS should make that possible in many cases.
public static void runD2CoreMethod (String id, IDfSession session) throws DfException {
    String docBase = session.getDocbaseName();
    String userName = session.getLoginUserName();

    IDfList parameterNames = new DfList();
    IDfList parameterTypes = new DfList();
    IDfList parameterValues = new DfList();

    parameterNames.appendString("METHOD");
    parameterTypes.appendString("S");
    parameterValues.appendString("D2CoreMethod");

    parameterNames.appendString("SAVE_RESULTS");
    parameterTypes.appendString("B");
    parameterValues.appendString(DfUtil.toString(true));

    parameterNames.appendString("LAUNCH_ASYNC");
    parameterTypes.appendString("B");
    parameterValues.appendString(DfUtil.toString(false));

    parameterNames.appendString("ARGUMENTS");
    parameterTypes.appendString("S");
    String args = String.format("-docbase_name %s -user_name %s -id %s -naming true -security true -autolink true -locale en", docBase, userName, id);
    parameterValues.appendString(args);

    IDfCollection results = null;
    try {
        results = session.apply(null, "DO_METHOD", parameterNames, parameterTypes, parameterValues);
    } finally {
        if (results != null)
            results.close();
    }
}

**Figure 3 Invocation of the D2CoreMethod from DFC**
package com.emc.sample;

import com.documentum.fc.common.DfException;
import com.documentum.fc.client.IDfBusinessObject;

public interface ISampleTBO extends IDfBusinessObject {
    public void save() throws DfException;
}

**Figure 4  Sample TBO Interface**
package com.emc.sample;

import com.documentum.fc.client.DfDocument;
import com.documentum.fc.common.DfException;
import com.documentum.fc.common.DfLogger;
import com.documentum.fc.common.IDfDynamicInheritance;

/**
 * Overrides doSave method
 */
public class SampleTBO extends DfDocument implements ISampleTBO, IDfDynamicInheritance {
    private static final String COPYRIGHT = "Copyright (c) EMC Corporation, 2012";

    protected void doSave(boolean isInSaveLock, String strInVersionLabel, Object[] inExtendedArgs) throws DfException {
        try {
            super.doSave(isInSaveLock, strInVersionLabel, inExtendedArgs);
            // Determine if this object save originates from a D2 method
            boolean calledByD2 = false;
            StackTraceElement[] stack = new Exception().getStackTrace();
            for (StackTraceElement s : stack) {
                if (s.getClassName().contains(".d2.") || s.getClassName().contains(".d2fs.")) {
                    calledByD2 = true;
                    break;
                }
            }
            if (calledByD2) {
                System.out.println("Skipping TBO logic...");
            } else {
                System.out.println("Applying TBO...");
            }
        } catch (Exception e) {
            DfLogger.error(this, "Exception in Sample TBO", null, e);
            throw new DfException(e.toString());
        }
    }

    // IDfBusinessObject Implementation
    public String getVersion() {
        return "1.0";
    }

    public String getVendorString() {
        return COPYRIGHT;
    }

    public boolean isCompatible(String arg0) {
        return true;
    }

    public boolean supportsFeature(String arg0) {
        return true;
    }
}

Figure 5  Sample TBO Implementation