AUTODESK UNIVERSITY 2014

Programming Item BOMs Through the Vault API

Doug Redmond – Autodesk Inc.

SD5329 Vault Professional 2015 software introduces many exciting new features for working with bill of materials (BOM) data. On top of that, Vault Professional 2015 R2 adds even more BOM features. This class will go over those features at an API (application programming interface) level, and we will also go over what has changed. The new features have made the old APIs incompatible. If you're upgrading your app from an earlier version of Vault software, this class will give you advice about needed code changes.

Learning Objectives

At the end of this class, you will be able to:

- Learn how to perform common BOM workflows
- Learn how to make use of the new item features in the Vault 2015 software API
- Learn how to make use of the new item features in the Vault 2015 software R2 API
- Learn how to update your item code from an earlier version of the Vault software development kit

About the Speaker

Doug Redmond has 9+ years of experience developing and using the Vault API. He has been at Autodesk almost 15 years working on various Data Management products including Streamline, Productstream Professional and PLM 360. He is currently working on a in the cloud... doing cloud stuff.

He is the author of Vault Mirror, Project Thunderdome and several other Vault apps.

Vault 2015

Vault 2015 is phase 1 of a major re-design of the Item and BOM feature in Vault. New features include better Item edit workflows, on\off BOM rows and a fix for edited out of turn issues. Click here for more new features in Vault 2015.

At the API level, changes include a new workflow for assigning Files to Items, new options for viewing BOM data and new APIs for dealing with "off" BOM rows. At a technical level, the SDK DLLs are now using .NET 4.5. Visual Studio 2012 and 2013 are the recommended development environments.

Since Items were re-designed, there is not full compatibility at the web services level. Vault 2014 and 2013 clients should not make calls to the ItemService or PackageService on a Vault 2015 server.

Assign Item

The workflow has changed for assigning a set of Files to a set of Items in a BOM structure. Previously PromotFiles() in the ItemService did most of the work. In Vault 2015, PromoteFiles() has been removed. The replacement is 4 new functions that must be called in a sequence. It's a bit more work on the client developer, but it is more efficient on the Vault server side. By breaking the operation into 4 calls, fewer operations are going on per server call. The calls are more efficient and the database locks are smaller.

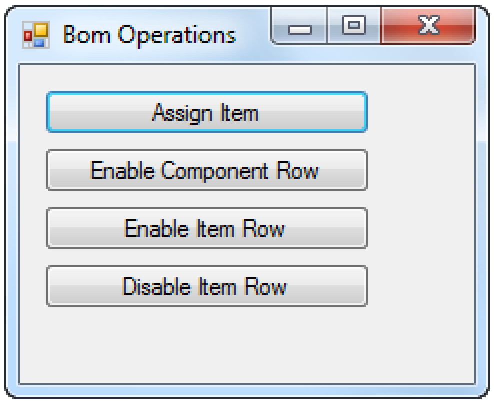
PromoteFiles() has been replaced with the following sequence of functions ItemService:

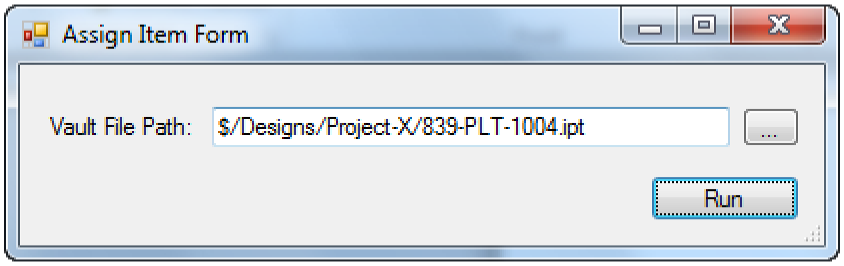
1. AddFilesToPromote()
2. GetPromoteComponentOrder()
3. PromoteComponents()
4. GetPromoteComponentsResults()

Once these calls are complete, the rest of the process is the same. The client has a set of uncommitted Items. Those Items can be edited if needed before calling UpdateAndCommitItems(), which finalizes the changes.

Sample Code:

For an example of this workflow see **AssignItemForm** in the **BOMOperations** example in the Additional Materials download.





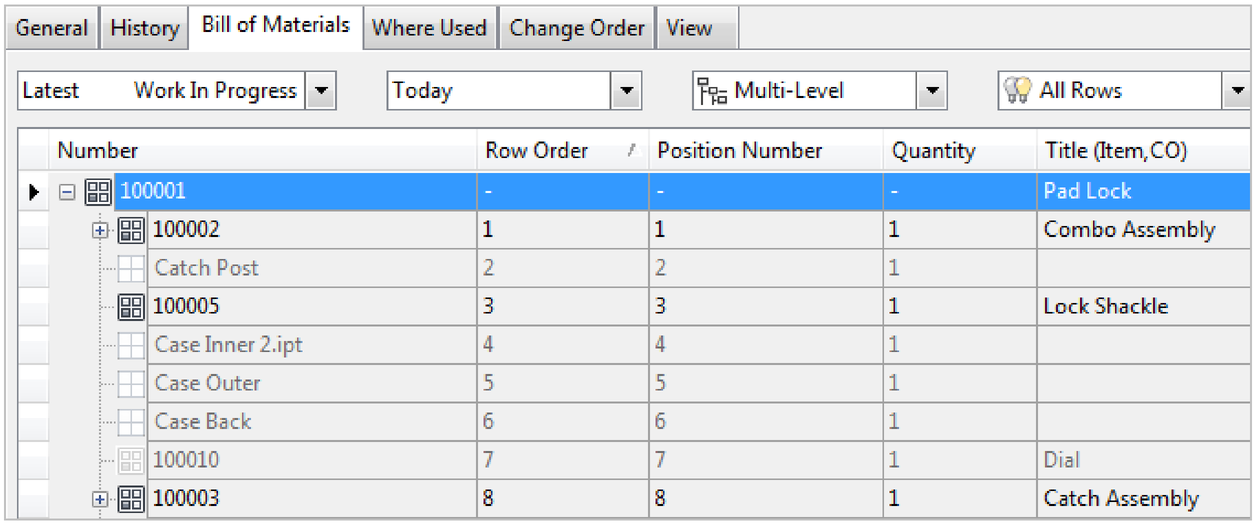
Vault File Path: $/Designs/Project-X/839-PLT-1004.ipt

On/Off BOM Rows

Another new feature is the ability to have an “off” BOM row. There are a few reasons that one may want to shut off a row. Many times the BOM is defined first and the row is a placeholder for the part that needs to be designed. When the design is done, it can be merged into the BOM and the row becomes enabled.

Another workflow is if a part is available cheaper from another vendor. The original part can be disabled in the BOM and the purchased part is added in. The original part could also be removed from the BOM, but having it as an “off” row makes it easier to switch back to it if needed.

There are actually two types of “off” rows: disabled Item rows and Component rows. In the Vault Explorer client, both of them are displayed as a grey row in the BOM. However, there are some differences if you look close. Disabled Item rows have an ▦ icon, and Component rows have an ⊞ icon. Also Item rows have more properties in the grid. Components only have Number, Row Order, Position Number and Quantity.



Latest Work In Progress | Today | Multi-Level | All Rows

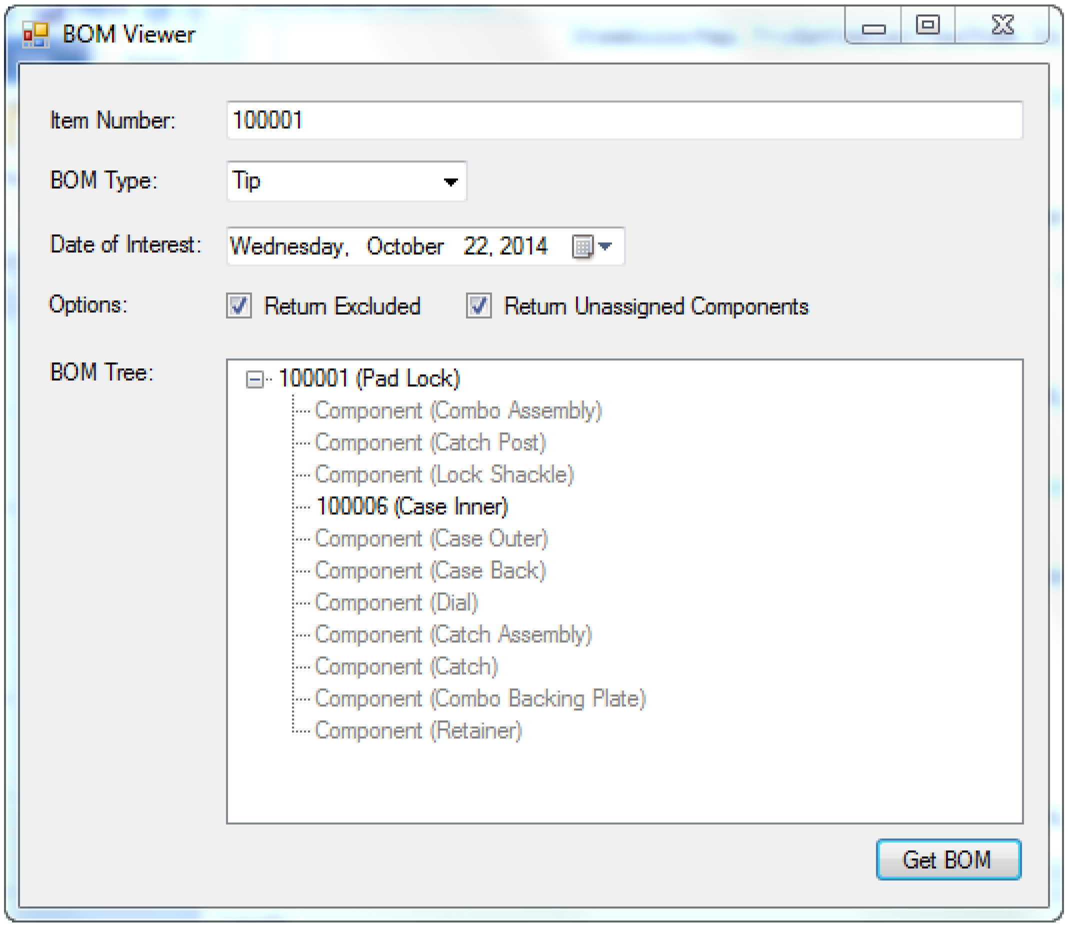
Number	Row Order	Position Number	Quantity	Title (Item,CO)
100001	-	-	-	Pad Lock
100002	1	1	1	Combo Assembly
Catch Post	2	2	1	
100005	3	3	1	Lock Shackle
Case Inner 2.ipt	4	4	1	
Case Outer	5	5	1	
Case Back	6	6	1	
100010	7	7	1	Dial
100003	8	8	1	Catch Assembly

(Bonus game: See if you can find which rows are component rows in the above image)

At the API level, Item rows and Component rows look more different. Each row is represented by an ItemAssoc object in the ItemBOM object in the API. An ItemAssoc consists of a parent Item and a child, which may be a Component or an Item. If the CldItemId is not 0, then the row is an Item row. The IsIncluded property tells if it's enabled or not. If BOMCompId is not 0, then the row is a Component row. Component rows are always disabled.

Sample Code:

If you want to read BOM data from Vault, then GetItemBOMByItemIdAndDate() in the ItemService is the function that you want to call. For an example of how to use this function and how to read out Item/Component rows, see the BOMViewer app in Additional Materials.


BOM Viewer

Item Number: 100001

BOM Type: Tip

Date of Interest: Wednesday, October 22, 2014

Options: ☑ Return Excluded ☑ Return Unassigned Components

BOM Tree:

```
⊟ 100001 (Pad Lock)
    Component (Combo Assembly)
    Component (Catch Post)
    Component (Lock Shackle)
    100006 (Case Inner)
    Component (Case Outer)
    Component (Case Back)
    Component (Dial)
    Component (Catch Assembly)
    Component (Catch)
    Component (Combo Backing Plate)
    Component (Retainer)
```

Components

In order to understand Component rows in the BOM, one has to understand Components. There are several aspects to Components, and any one can serve as the definition:

- A Component is an object in a CAD file that we want to track.
- A Component is the glue between a Vault File and Item.
- A Component is a prototype of a Vault Item.

It's this last definition that relates best to the current discussion. When assigning an assembly File to a BOM, Vault 2015 has the option of just creating the root Item of the BOM. The rest is left as Components. These Components are not full Items yet, but they have the potential to become Items when needed. The engineer can then enable the rows as needed. The enabling process promotes the Component into a full Vault Item object.

Enabling and Disabling Rows

The easiest case to star with is the enabling and disabling of Item rows. The main function here is UpdateItemBOMAssociations in the Item services. As stated before, a BOM row is an ItemAssoc object. So enabling a row involves updating the IsIncluded value on that object.

Here is the full sequence of steps:

1. EditItems on the parent Item
2. GetItemBOMByItemIdAndDate to find the BOM association.
3. UpdateItemBOMAssociations to set the isIncluded value
4. UpdateAndCommitItems on the parent Item

Enabling a Component row is harder. First, the Component has to be promoted into an Item. Next, the BOM association needs to be updated to enable the row. It's basically the promote and the row-enable workflows combined into one. Once the operation is done, the result is an enabled Item row.

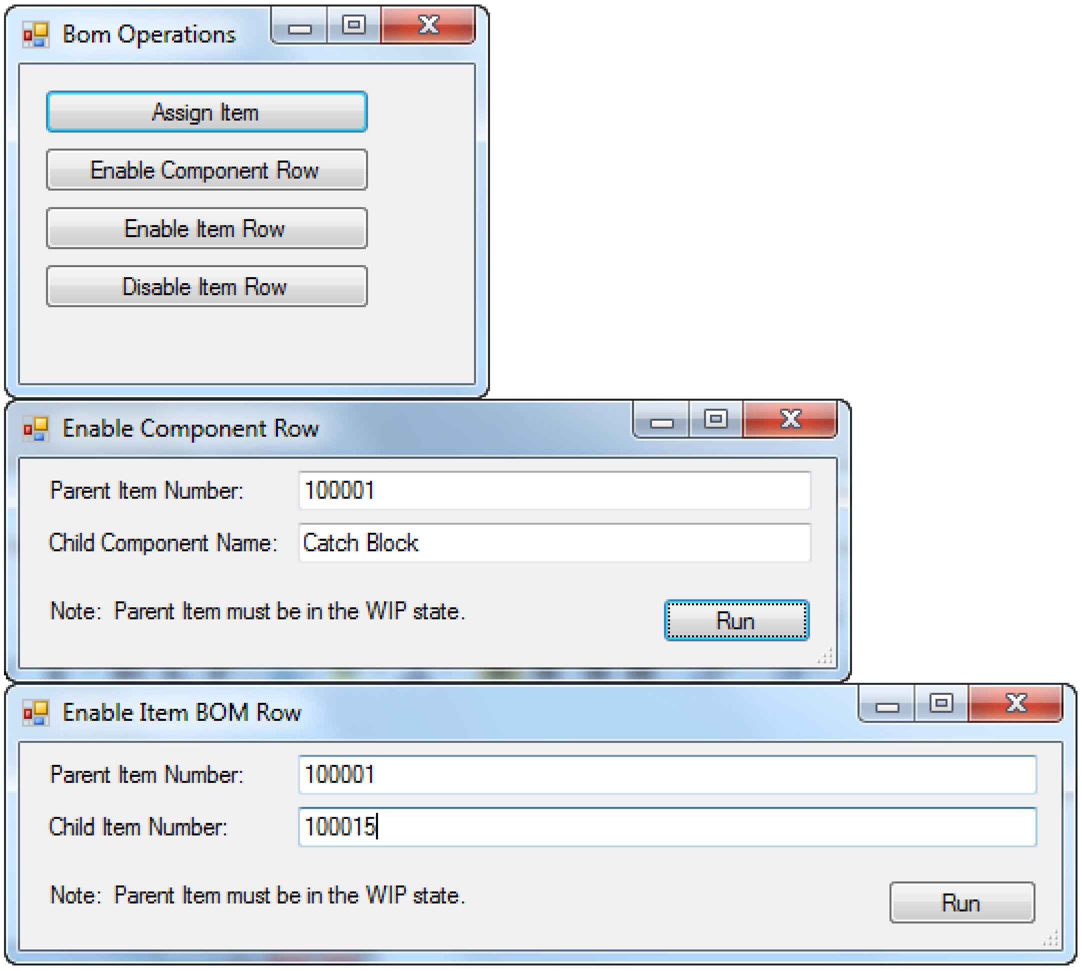
Here is the full sequence of steps:

1. EditItems on the parent Item.
2. GetItemBOMByItemIdAndDate to find the BOM association.
3. AddComponentsToPromote on the child component
4. GetPromoteComponentOrder.
5. PromoteComponents
6. GetPromoteComponentsResults

7. UpdateItemBOMAssociations on the association. Set childItemId to the Item Id from the promote call. Set isIncluded to true.
8. UpdateAndCommitItems on the parent Item and the Item from the promote call.

Sample Code:

Both workflows are illustrated in the BOM Operations app in Additional Materials.



Parent Item Number: 100001

Child Component Name: Catch Block

Note: Parent Item must be in the WIP state.

Parent Item Number: 100001

Child Item Number: 100015

Note: Parent Item must be in the WIP state.

Vault 2015 R2

Vault 2015 R2 is the second and final phase of the Item and BOM re-design. New features include customizable Item lifecycles, BOM row grouping, and BOM reports. The 2015 R2 release also includes a completely re-build of the Copy Design feature. Click here for more new features in Vault 2015 R2.

At a technical level, 2015 R2 is a completely separate release from 2015. In other words it is not a service pack. 2015 R2 has its own version number and SDK. Older Vault clients are compatible with 2015 R2 server as long as the client does not perform any Item operations. The Item and Package services will not work for older clients.

Item Lifecycles

Items now share the same lifecycle engine as Files, Folders and Custom Objects. That engine has been in place for many years now. So if you know how to manage File lifecycles, you also know how to manage Item lifecycles. If you don't know File lifecycles, here is a quick rundown on how to manage Item lifecycles through the API.

Here are the relevant areas of the API:

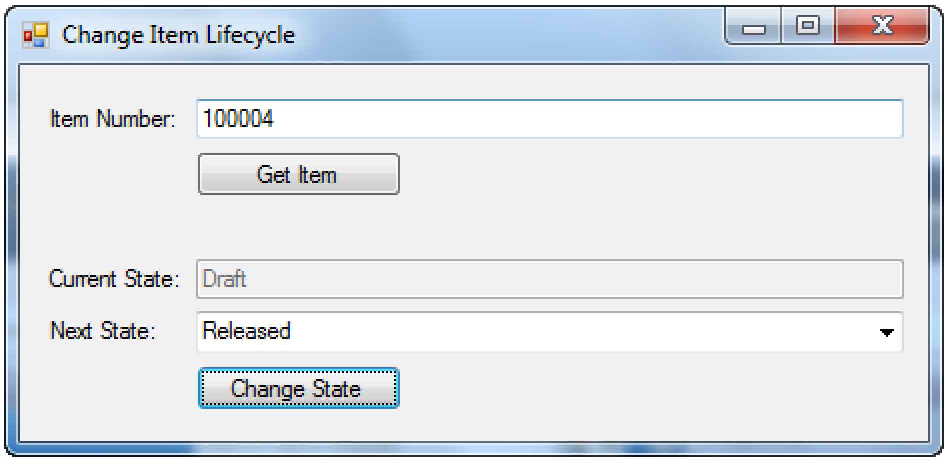
LifeCycleService – This service manages the lifecycle engine, but is not geared to any specific entity type. You can use this service for things like seeing the states and transitions on a lifecycle. You can also customize the state settings and behaviors through this service. This service does not perform actions on specific Entities (such as Files or Items). Those operations are done in other services.

ItemService.UpdateItemLifeCycleStates – This function allows a set of Items to transition to another State in the Lifecycle. All state changes must be legal. There must be a Transition from the to and from State, and the user must have the appropriate permissions.
The Item should not be in edit mode when calling this function, and there is no need to commit after the function completes. It's one of the few ItemService functions that performs a complete operation in one call.

ItemService.UpdateItemLifeCycleDefinitions – This function allows a set of items to transition to another Lifecycle. In this case, the caller needs to specify the destination Lifecycle and the destination state within that lifecycle. There are no Transitions across different Lifecycles, so Transition rules don't apply. However the user must have the appropriate permissions and the new Lifecycle must be allowed in the Item's Category.
Again, this operation is a single call. The item should not be in edit mode, and no commit call is needed.

Sample Code:

See the Change Item Lifecycle app in Additional Materials for example code illustrating a state change on an Item.



Change Item Lifecycle

Item Number: 100004

Current State: Draft

Next State: Released

Copy Design

Vault 2015 R2 comes with a new Copy Design application. It's an optional component that is a separate EXE from Vault Explorer. If you run the "copy design" command inside Vault Explorer, you will get the old feature.

The new application is a complete re-design of the copy design feature. The UI is new, the workflows are new, and there are new APIs on the Vault server. These new APIs allow the server to perform the "heavy lifting" operations, such as copying the files and setting properties on the files. These APIs are undocumented in the SDK, but the Vault team said they would support them if a third party wanted to use them. So here is my documentation of the undocumented API functions.

CopyFile

```
byte[] FilestoreService.CopyFile (
  byte[] downloadTicket,
  bool allowSync,
  PropWriteReq[] writeReqs,
  out PropWriteResults writeResults)
```

<u>Description:</u>
This function copies a file in the file store. Previously, a copy involved a download and an upload.

<u>Parameters:</u>
downloadTicket – The access code to the file you want to copy. Call *DocumentService.GetDownloadTicketsByFileIds* to get a download ticket.
allowSync – If true and the file is not on the local file store, the file is copied as part of the operation. If false, the operation fails if the file is not in the local file store.
writeReqs – A set of properties to update within the file. For example, you can update Inventor iProperties in the copied file.
writeResults – The result of the property updates.

<u>Returns:</u>
The upload ticket for the new file.

GetContentSourcePropertyDefinitions

```
CtntSrcPropDef[] FilestoreService.GetContentSourcePropertyDefinitions (
  byte[] downloadTicket,
  bool allowSync)
```

Description:
This function lets you see what properties are available to edit in the file. This is not referring to Vault properties, but properties inside the file itself.

Parameters:
downloadTicket – the access code to the file you want to copy. Call DocumentService.GetDownloadTicketsByFileIds to get a download ticket.
allowSync – If true and the file is not on the local filestore, the file is copied as part of the operation. If false, the operation fails if the file is not in the local filestore.

Returns:
The properties that can be read from or written to the file.

Copy Design – API Workflows

So what can we do with these new APIs. Here are a couple of ideas..

Perform your own copy design. Here are the basic steps:

1. DocumentService.GetDownloadTicketsByFileIds
2. FilestoreService.CopyFile
3. DocumentService.AddUploadedFile

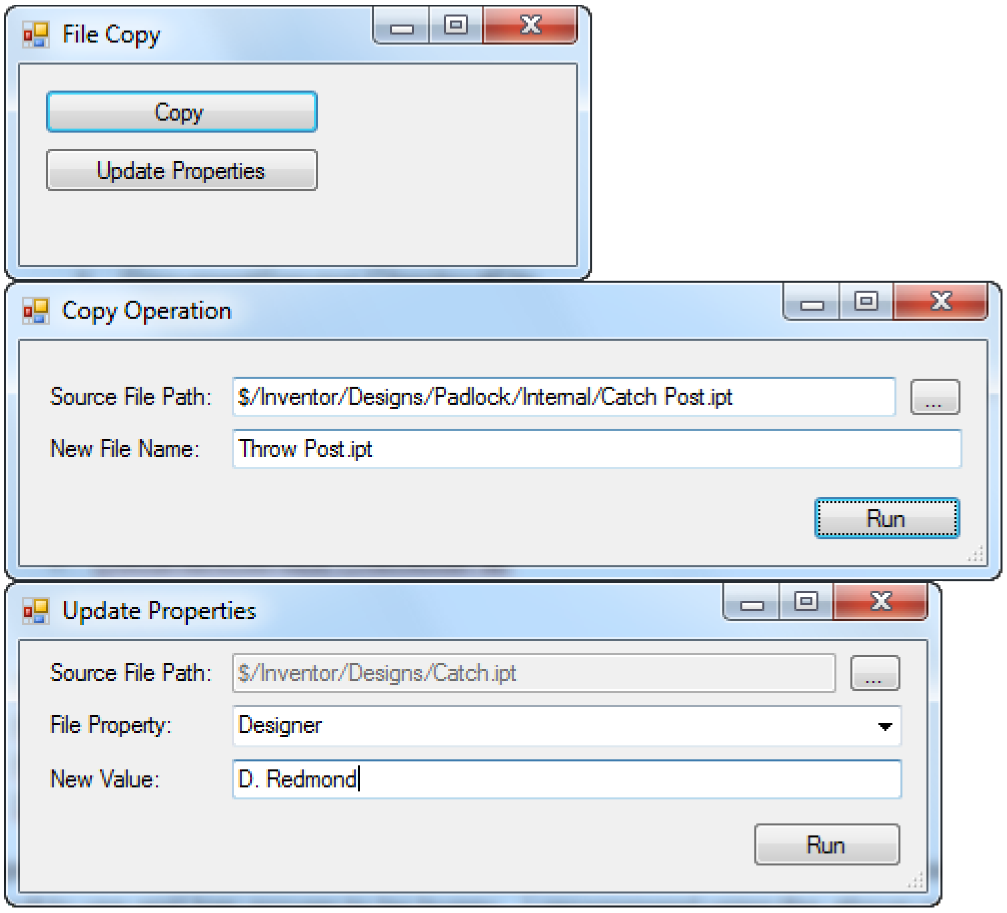
Another thing you can do is edit a file property without having to download the file or deal with a CAD API. Here are the basic steps:

1. DocumentService.CheckoutFile
2. FilestoreService.GetContentSourcePropertyDefinitions
3. FilestoreService.CopyFile
4. DocumentService.CheckinUploadedFile

This operation is also available through the IExplorerUtil class in the SDK. However that class is difficult to use and has proven to be buggy. I recommend using the above workflow for property updates instead of IExplorerUtil.

Sample Code:

Both of these workflows are illustrated in the File Copy app in Additional Materials.



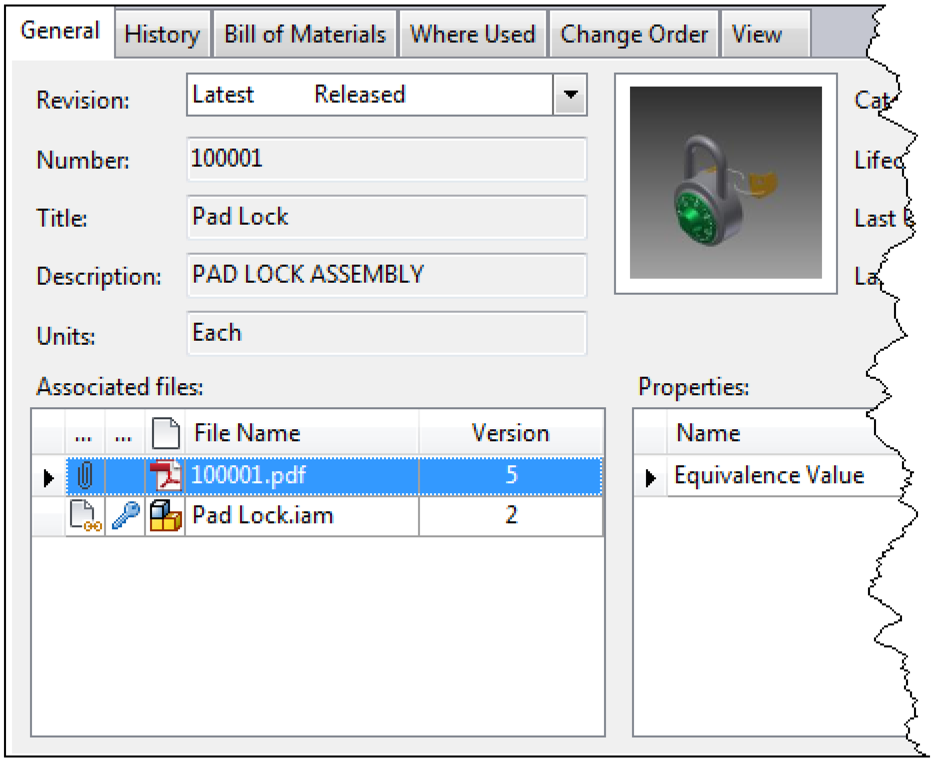
Source File Path: $/Inventor/Designs/Padlock/Internal/Catch Post.ipt
New File Name: Throw Post.ipt

Source File Path: $/Inventor/Designs/Catch.ipt
File Property: Designer
New Value: D. Redmond

Bonus Code Sample – BOM Report Job

Vault 2015 R2 introduces the BOM Report feature. In the Vault Explorer client, a user can generate a report of the BOM being viewed. The report can be printed or saved as and Excel or PDF file. It's a great feature, but it's only available through the UI. If you want to programmatically generate a report PDF, there are a lot of steps involved. You need to read the BOM data from the Vault server, format the results in data tables, send the data to the Microsoft Report Viewer and render the report.

Or you could just copy the code from the BOM Report Job sample in the Additional Materials. This example is more complex than the others because it is a Job Processor extension. It needs to be configured and deployed. The end result is automatic report generation in response to Item lifecycle state changes.



Revision: Latest Released

Number: 100001

Title: Pad Lock

Description: PAD LOCK ASSEMBLY

Units: Each

Associated files:

...	...	File Name	Version
		100001.pdf	5
		Pad Lock.iam	2

Properties:

Name
Equivalence Value

For more details, see my blog post (which also contains the source code): http://justonesandzeros.typepad.com/blog/2014/09/bom-report-job.html