AUTODESK® UNIVERSITY

MFG468044
Hack the Vault Job Processor

Thomas Rambach

CAD Systems Administrator | @cadtoolbox

About the speaker

Thomas Rambach

Corning Optical Fiber and Cable

CAD Systems Administrator (2017-Present)

GE Hitachi Nuclear Energy

Mech. Designer/ Software Engineering Specialist (2006-2017)

Flow Sciences, Inc.

Sr. Designer / CAD Manager (2002-2006)

Corning

Equipment Engineering Tech. (1996-2002)

cadtoolbox.com | @cadtoolbox

Swift Prints

Design Notes

Batch Logic

What is this class about?

1. WHAT IS THE VAULT JOB PROCESSOR?

We'll first dig into what exactly is the Vault job processor for those of you not as familiar with it and explain the basics of how it works.

2. WHAT ARE THE LIMITATIONS?

The Vault job processor has some limitations if you use it out-of-the-box. We'll identify those gaps and explain how these may impact your Vault data.

3. WHAT CAN YOU SOLVE WITH THIS CLASS?

Understanding the limitations of Vault, will help you understand what you're trying to solve by watching this class. We'll walk through some code chunks that solve specific problems.

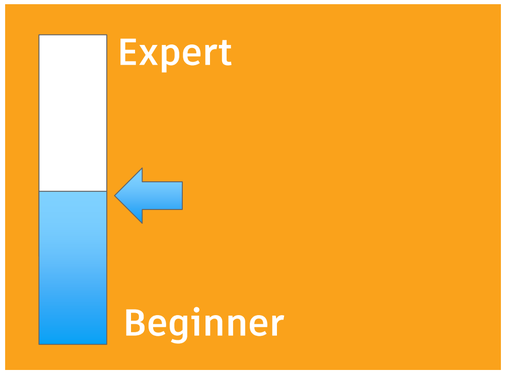
4. HOW DO YOU APPLY WHAT YOU LEARNED?

This is where we'll put it all together and show you what you can do if you spend a little time hacking the Vault job processor to make it do what you want it do do.

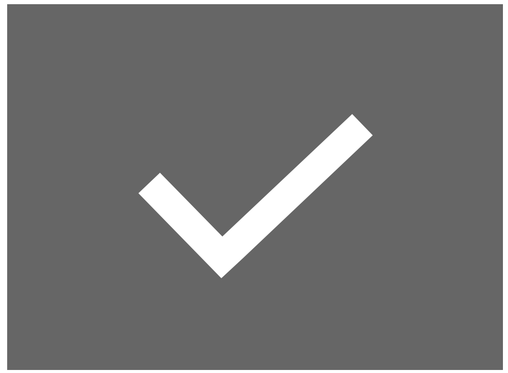
Requirements for this Class

Vault Administrator

Visual Studio



Experience



Desire to be more productive

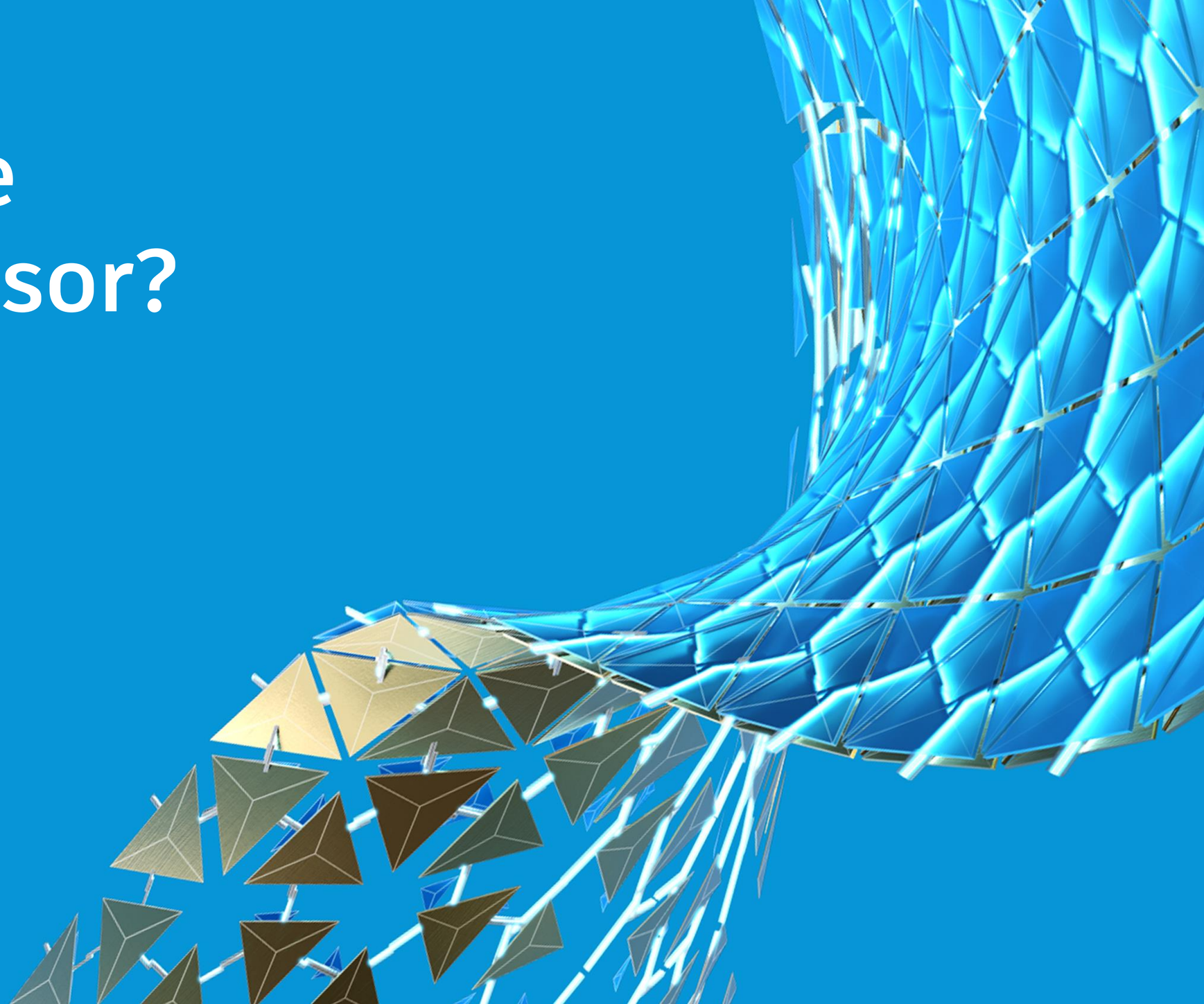
From this class, you will...

- Learn to allow the Job Processor to do more work.
- Be more productive.
- Learn to not be intimiated by coding.

From this class, you will not…

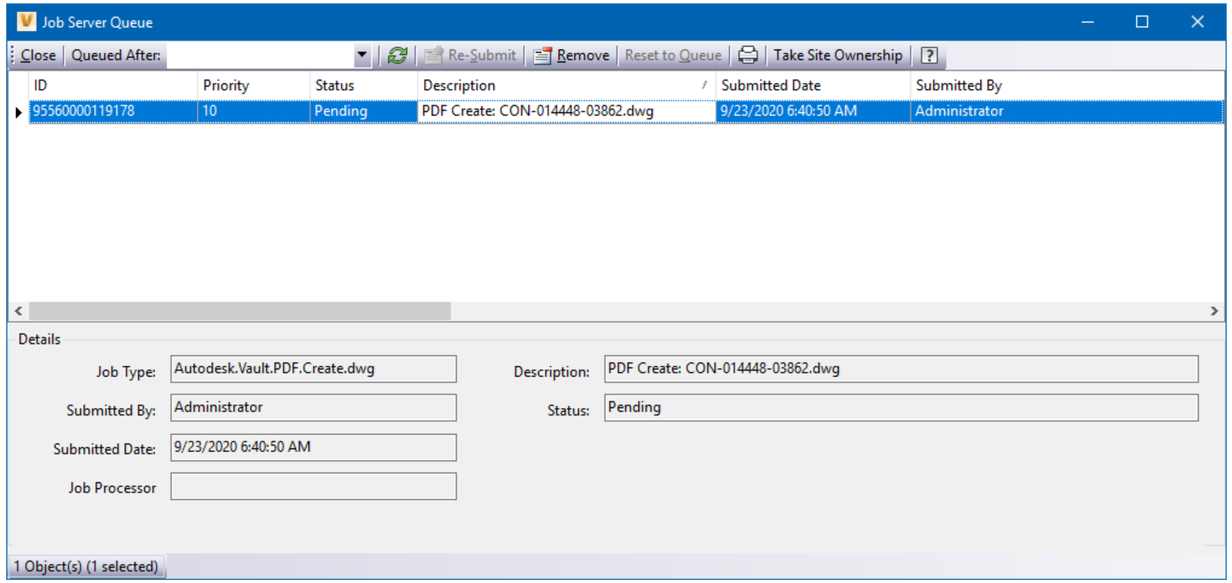
- Learn anything illegal or immoral.
- Be able to turn on and off traffic signal lights using your phone.
- Need a floppy disc of back door codes.
- Receive a cool nickname like…
  - DWG-Zero
  - IPT Plauge
  - Generative Blade
  - Joey
- Be required to skateboard around your office.

What is the Job Processor?

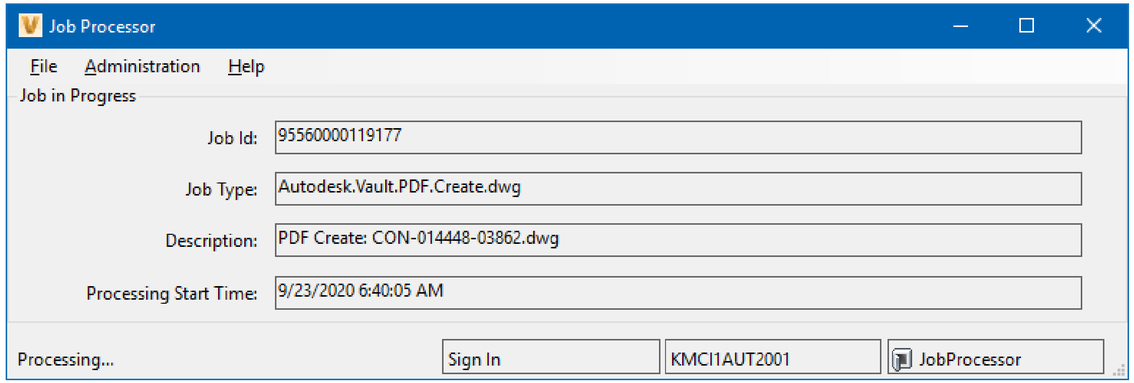


What is the Vault Job Processor?

"The Job Processor is a separate application that reserves queued jobs and pulls them from the job server to process them. Since the job processor is installed along with the Vault client, any workstation with the appropriate edition of Vault can be used to process jobs."

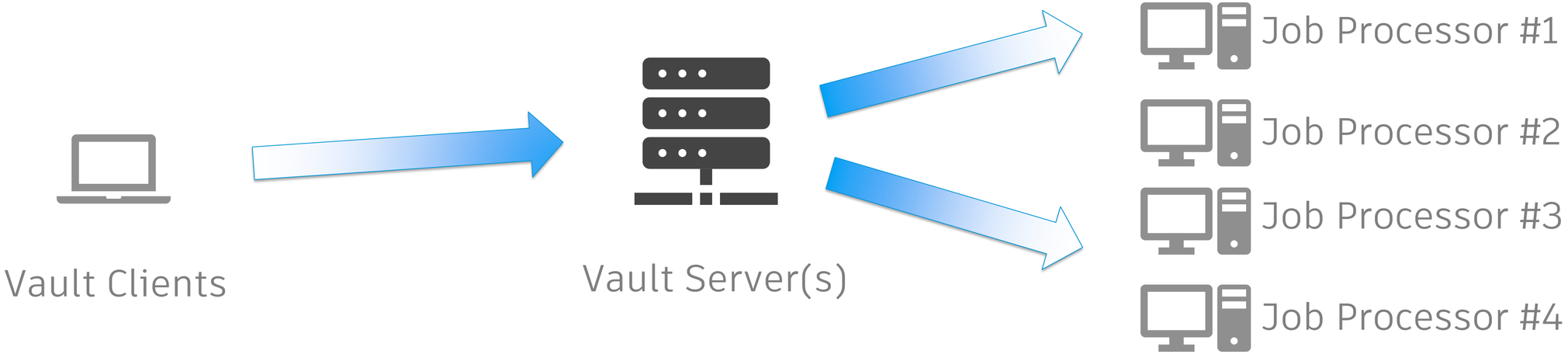


Vault Server: Job Processor Job Queue



Job Processor: Active Job

Job Processor Infrastructure



Vault users perform transaction that initiate jobs like checking in a file, performing a lifecycle state change.

Vault Server(s) store and manage the job queue.

Job Processors select the next job to process based on priority.

How do you feed jobs to the Job Processor?

ON FILE CHECK-IN

DWF Visualization files can be queued for creation on check-in of a file from within Inventor or AutoCAD.

ON LIFECYCLE TRANSITION

On lifecycle state change, actions can be performed using the job processor such as synchronizing properties, updating DWF, updating PDF.

MANUALLY

On launch of ‘Update Visualization Attachment’ a job will be created to update the DWF attachment or on manual selection of the PDF create command.

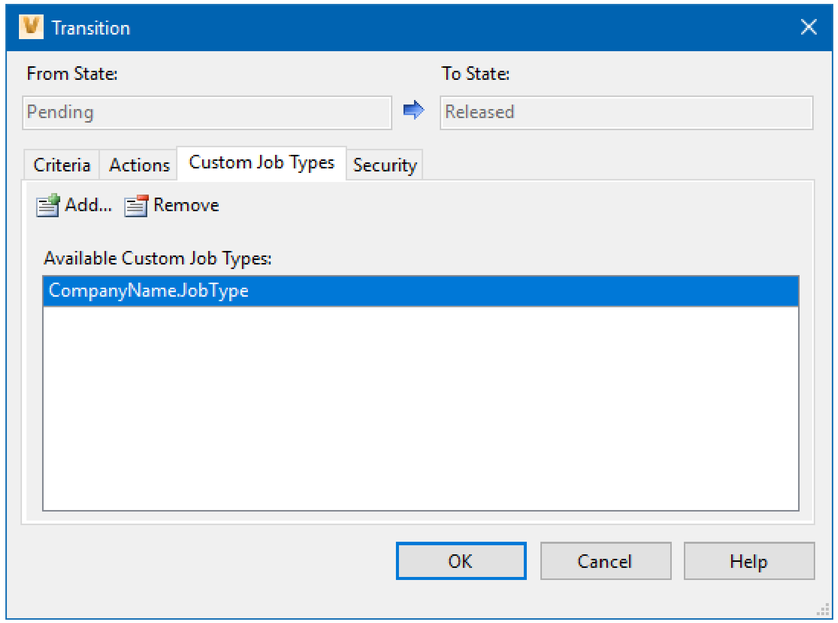
PRO TIP

Edit the file “JobProcessor.exe.config” located in C:\Program Files\Autodesk\Vault Client 2021\Explorer:

<add key="PeriodInMinutes" value=“10"/>

Change the value to a lower number to increase the waiting time between checks for new jobs.

You can also add custom job types on lifecycle state transition



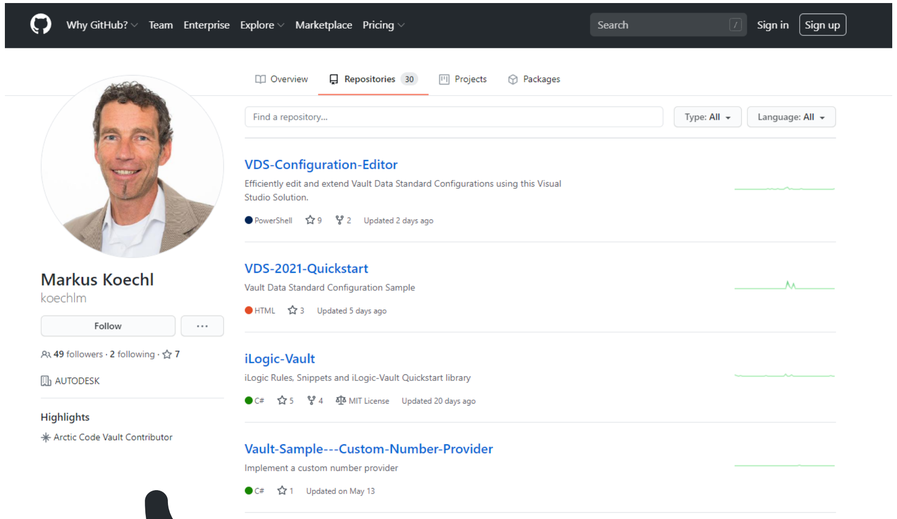
Custom jobs are added to the appropriate lifecycle transition as a custom job type. A developer creates the custom job type with matching name that the job processor runs.

Custom jobs are loaded from the Vault extension folder:

%allusersprofile%\Autodesk\Vault[year]\Extensions\

More Information: https://knowledge.autodesk.com/support/vault-products/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Vault-Admin/files/GUID-A298690E-A937-4317-89C9-B04C9950DF2D-htm.html

Creating your first custom Job Type



Thank you Marcus!

A great starting point with the basic framework necessary for creating your first custom job type is provided free by Marcus Koechl (Autodesk)

GitHub – Vault-Sample---InventorExportAnySampleJob

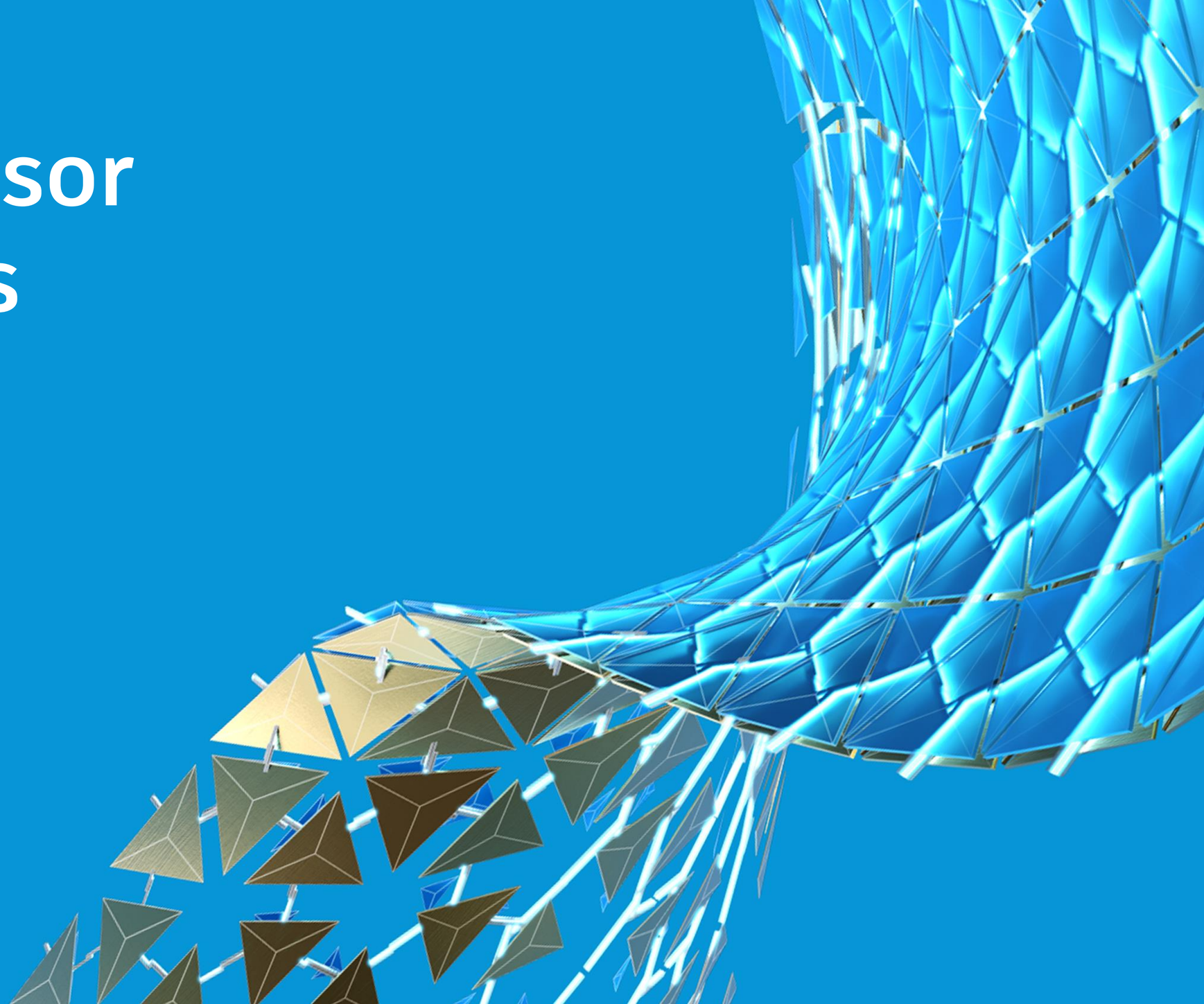
https://github.com/koechlm/Vault-Sample---InventorExportAnySampleJob/blob/master/Autodesk.VLTINVSRV.ExportSampleJob/JobExtension.cs

What you'll need:

1. Download the GitHub project and place folder in Extensions directory.
2. Follow requirements on GitHub readme to use as-is.
3. Edit using Visual Studio. I recommend using VS2019.
4. Republish to Extensions folder on Job Processor.

** First: Use on development database using test data only ***

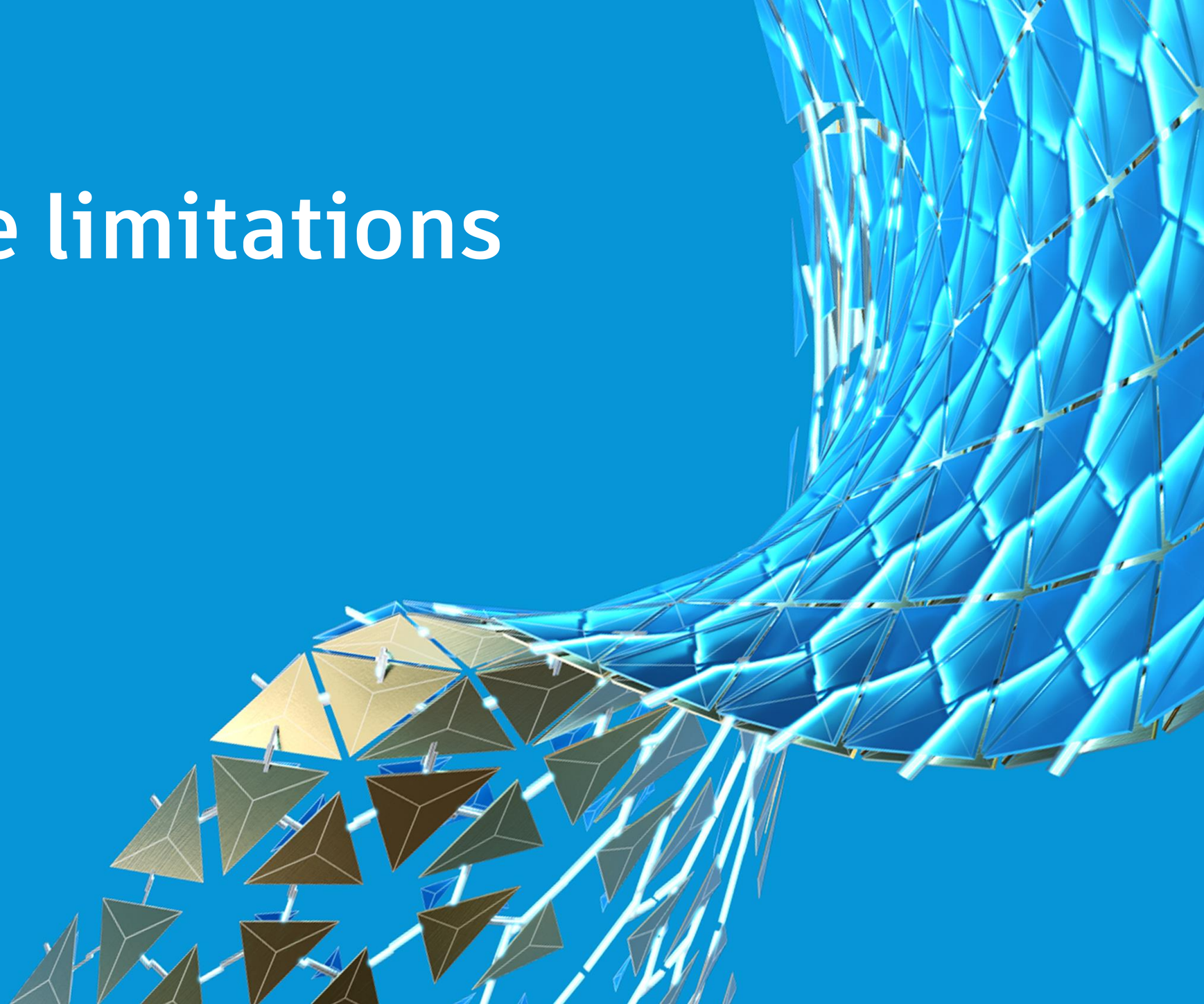
Job Processor Limitations



Vault has Limitations on How Jobs are Processed

- **Default Vault Actions:**
  - Limited actions available. (Sync Properties, Update DWF, Update PDF)
- **Custom Job Types:**
  - **Runs on lifecycle transition (Ex: Review → Released).**
  - No way to differentiate by file type.
    - Job will be created for all files that transition.
  - Need to create different job processor extensions to handle different lifecycle transitions.
  - Priority is always 100 and will run last after all other jobs are run in the queue.
    - Jobs are processed in ascending priority order. 1 first, 2, 3, etc. up to 100 or more.

Solving the limitations



* Disclaimer

The following slides are how I solved the limitations and may not be always be ideal and certainly there may be different ways or better ways. I don’t work for Autodesk. I’m not a reseller or professional developer.

I am a curious CAD Admin with specific needs to be more productive with limited budgets.

Let’s start by breaking down the STEP Export Sample Job

```
34  namespace Autodesk.STEP.ExportSampleJob
35  {
36      public class JobExtension : IJobHandler
37      {
38          private static string JOB_TYPE = "Autodesk.STEP.ExportSampleJob";
39          private static Settings mSettings = Settings.Load();
40          private static string mLogDir = JobExtension.mSettings.LogFileLocation;
41          private static string mLogFile = JOB_TYPE + ".log";
42          private TextWriterTraceListener mTrace = new TextWriterTraceListener(System.IO.Path.Combine(
43              mLogDir, mLogFile), "mJobTrace");
44
45          #region IJobHandler Implementation
46          public bool CanProcess(string jobType)
47          {
48              return jobType == JOB_TYPE;
49          }
50
```

Look at JobExtension.cs. The method CanProcess returns the job type and determines if the extension is appropriate for the job being processed

Next, execute the method that does the work…

```
51          public JobOutcome Execute(IJobProcessorServices context, IJob job)
52          {
53              try
54              {
55                  FileInfo mLogFileInfo = new FileInfo(System.IO.Path.Combine(
56                      mLogDir, mLogFile));
57                  if (mLogFileInfo.Exists) mLogFileInfo.Delete();
58                  mTrace.WriteLine("Starting Job...");
59
60                  //start step export
61                  mCreateExport(context, job);
62
63                  mTrace.IndentLevel = 0;
64                  mTrace.WriteLine("... successfully ending Job.");
65                  mTrace.Flush();
66                  mTrace.Close();
67
68                  return JobOutcome.Success;
69              }
70              catch (Exception ex)
71              {
72                  context.Log(ex, "Autodesk.STEP.ExportSampleJob failed: " + ex.ToString() + " ");
73
74                  mTrace.IndentLevel = 0;
75                  mTrace.WriteLine("... ending Job with failures.");
76                  mTrace.Flush();
77                  mTrace.Close();
78
79                  return JobOutcome.Failure;
80              }
81
82          }
83
```

Execute will run the job and return the JobOutcome, success or failure from the mCreateExport method.

Some job filters are built into the sample already…

``
// only run the job for files
if (mEntClsId != "FILE")
return;
``
	Only runs the job against files. Not inadvertently applied to other entity types such as Items, Folders, etc.

``
// only run the job for ipt and iam file types,
List<string> mFileExtensions = new List<string> { ".ipt", ".iam" };
ACW.File mFile = mWsMgr.DocumentService.GetFileById(mEntId);
if (!mFileExtensions.Any(n => mFile.Name.Contains(n)))
{
return;
}
``
	Filters the job to only run Part (.ipt) or Assembly (.iam) files.

``
// apply execution filters, e.g., exclude files of classification "substitute" etc.
List<string> mFileClassific = new List<string> { "ConfigurationFactory", "DesignSubstitute", "DesignDocumentation" };
if (mFileClassific.Any(n => mFile.FileClass.ToString().Contains(n)))
{
return;
}
``
	Further filters out any file classified as a ConfigurationFactory, DesignSubstitute or DesignDocumentation

Refer to the mCreateExport method. Allows to fine tune a custom job to only process files you want processed by a job that runs against all files being transitioned.

What other filters can be applied?

- **Lifecycle state**
  - What if you want to handle the job differently if applied to multiple lifecycle state transitions? Instead of creating multiple extensions, you can create a lifecycle state filter.

Let's first lay some "context":

If you notice from the previous sample, the Execute method passes **context** and **job** to the mCreateExport method. Context is the Job Processor which houses the current connection. The job, is the job currently being processed:

```
//start step export
mCreateExport(context, job);
```

context	▪ Connection ▪ Errors ▪ InventorObject
job	▪ Description ▪ Id ▪ JobType ▪ Params ▪ Priority ▪ VaultName

The connection holds the keys to the kingdom

- **The connection allows access to all of the Vault services**
  - You can use the various services to interact with Vault and perform actions on the file during the processing of the job.

AdminService	Contains methods for manipulating users and groups.
AnalyticsService	Contains methods for Analytics within a vault
BehaviorService	Contains methods for manipulating behaviors.
CategoryService	Contains methods for manipulating categories.
ChangeOrderService	Contains methods for creating and manipulating change orders.
CustomEntityService	A collection of methods related to the Custom Entity entity type.
DocumentService	Contains methods for manipulating files and folders within a vault.
DocumentServiceExtensions	Contains more methods for manipulating files and folders within a vault.
ItemService	Contains methods for manipulating items.
JobService	Contains methods for manipulating the job queue.
LifeCycleService	Contains methods related to the lifecycle behavior.
NumberingService	Contains methods for retrieving and manipulating Numbering Schemes and configured Numbering Providers
PropertyService	Contains methods for manipulating properties on Entities.

Reference the Vault SDK Documentation for the complete list and the members of each service

Lifecycle State Filter

Get the file to process by picking up the jobs entity ID:

```
using ACW = Autodesk.Connectivity.WebServices;

Connection connection = context.Connection;
Autodesk.Connectivity.WebServicesTools.WebServiceManager mWsMgr = connection.WebServiceManager;
ACW.File mFile = mWsMgr.DocumentService.GetFileById(mEntId);
```

Now get the file lifecycle state name:

```
mFile.FileLfCyc.LfCycStateName;
```

Lifecycle State Filter

What can you do now that you have the lifecycle state name?

```
String sLifeCycleState = mFile.FileLfCyc.LfCycStateName;

switch (sLifeCycleState)
{
        Case “Released”:

        // Do something if a file is released

        break;

        Default:

        // Do something different if the file is not released

        break;
}
```

Category Filter

Get the file category and do something different based on the category name

```
String sFileCategory = mFile.Cat.CatName;

switch (sFileCategory)
{
        Case “Engineering”:

        // Do something if a file is an engineering category

        break;

        Case “Document”:

        // Do something different if the file is a document category

        break;
}
```

Property Filter

There are 2 ways to get properties of a file.

1. Vault Properties : Retrieves the properties of a file as they are stored within Vault.

2. File Properties : If the Job Processor downloads the file for processing, the file properties can be retrieved directly for processing regardless of if they are indexed in Vault.

We'll cover retrieving Vault Properties in this class. Because I only have an hour. ;)

Property Filter

First you have to define the property definition for the property you want to retrieve:

```
// Define the Files Properties to Retrieve
PropDef[] docPropDefs = mWsMgr.PropertyService.GetPropertyDefinitionsByEntityClassId("FILE");

PropDef DescriptionDef = docPropDefs.Single(n => n.DispName == "Description");
```

Get the files properties for the specific property definitions:

```
fileProperties = PropertyService.GetProperties("FILE", new long[] { fFileID }, new long[] {DescriptionDef.Id, AnotherProperty.Id});
```

Then get the property value if the property name matches the property name you want by cycling through them:

```
// Assign the Property Values
foreach (var key in fileProperties)
{
        string propDisplayName = context.Connection.PropertyManager.GetPropertyDefinitionById(key.PropDefId).DisplayName;

        if (propDisplayName == DescriptionDef.DispName)
        {
            fileDescription = key.Val?.ToString() ?? "";
        }

}
```

Sets the property value to blank if the property is not found

Filters are cool and all, but I want to update a property

Get the files master ID:

```
mMasterID = mFile.MasterId;
```

- Master ID = The ID# of the file regardless of version in the Vault.
- File ID = The ID# of the specific file version in the Vault.

Update the file properties by passing in the property definition ID and the new property value:

```
context.Connection.WebServiceManager.DocumentService.UpdateFileProperties(new long[] { mMasterID },
                    new PropInstParamArray[] { new PropInstParamArray() { Items = new PropInstParam[] { new
PropInstParam() { PropDefId = DescriptionDef.Id, Val = "New Description" } } } });
```

This is good information, but what about Hacking the Job Processor?

Stay With Me

The JobService Allows Direct Manipulation of the Job Processor

- **The JobService has several useful members that can be used to interact with the Job Queue:**

AddJob	Adds a new job to the queue
AddScheduledJob	Adds a scheduled job with given execution date and frequency.
DeleteJobByID	Deletes a job from the queue.
DeleteScheduledJob	Deletes the given scheduled job.
GetJobsByDate	Get all jobs from the queue queued on or after the specified start date.
GetJobQueueEnabled	Tells if the job queue is enabled.
GetScheduledJob	Gets information about the given scheduled job.
GetScheduledJobs	Gets information about all scheduled jobs.
ReserveNextJob	Reserve the next job in the queue
ResubmitJob	Resubmit a job of the specified Id to the queue.
SetJobQueueEnabled	Enables or disables the job queue.
UpdateJobFailure	Inform the job queue that the client was unable to complete the job.
UpdateJobSuccess	Inform the job queue that the job was successfully completed.

Add Job

You can add new jobs to the Job Processor with values you specify.

```
Public Function AddJob( _
  ByVal type As System.String, _
  ByVal desc As System.String, _
  ByVal paramArray() As JobParam, _
  ByVal priority As System.Integer _
) As Job
```

The Job Name. This sets what Job Processor extension will pickup the job.

Job description can be whatever you want to help identify it in the queue.

Job parameters are values passed into the job. This is powerful, you'll see.

Job priority can be anything you want. 1-100 (actually 1-999).

Add Job: Parameters

What are Job Parameters? MUST BE PASSED AS STRINGS

```
Job oJob = e.Context.Application.Connection.WebServiceManager.JobService.AddJob("CompanyName.JobType",
"Custom Job Name: " + mFile.Name, new JobParam[] { param1, param2, param3 }, 50);
```

Job priority is 50.

Define parameters to pass. The sky is the limit:

```
JobParam param1 = new JobParam()
JobParam param2 = new JobParam();
```

Create the parameters

```
param1.Name = "EntityId";
param1.Val = oLatestFileVer.Id.ToString();
```

Pass the file version ID as EntityID. This is so the job processor knows which file to run. We can convert to MasterID in the job if needed.

```
param3.Name = "EntityClassId";
Param3.Val = "FILE";
```

Pass the entity type. In this case, it's "FILE" because we're processing a file.

You can pass in anything you like. Category Name, Lifecycle State, Properties, Etc.

Add Job: Job Type

You can also pass in a job type parameter to allow your single job processor extension to process multiple job types:

```
JobParam param4 = new JobParam()
param4.Name = “JobType";
param4.Val = “UpdateProperties”
```

Get the Job Type Parameter

```
// Get the Job Type from the Parameters
        try
        {
            oJobType = job.Params[“JobType"];
        }
        catch (Exception ex)
        {
            oJobType = "Default";
        }
```

Process based on the Job Type value

```
// Update The File Properties Job
        if (oJobType == “UpdateProperties")
        {
            UpdateProperties(context, job);
        }

// Release File Job
        if (oJobType == "ReleaseFile")
        {
            ReleaseFile(context, job);
        }
```

Is the File Still Being Processed?

If you queue a job for a file of the same job name that's already in the queue, what happens?

An error will be returned when this happens. You can graciously handle the error and duplicate jobs would not be added.

```
try
{
Job oJob =
e.Context.Application.Connection.WebServiceManager.JobService.AddJob("CompanyName.JobType",
"Custom Job Name: " + mFile.Name, new JobParam[] { param1, param2, param3 }, 50);
}
catch (Exception ex)
{
  // It's a dupe. I don't care about no stinkin' error
}
```

Is the File Still Being Processed?

What if you want to really know if it's in the queue?

In this example, we get an array of 5000 of the jobs submitted in the last day (we know 5000 is a safe number that gets all jobs). We then see if the job has the same filename in the description.

```
Job[] oJobList = JobSrvc.GetJobsByDate(5000, DateTime.Today.AddDays(-1));
 if (oJobList != null)
        {
           foreach (Job pJob in oJobList.ToArray())
           {
              if (pJob.Descr.Replace("Custom Job Name: ", "") == mFileName)
              {
                   Log.WriteLog(fFileName + " job still in queue as Job#: " + pJob.Id.ToString() + " Status: " + pJob.StatusMsg);
              }

           }
        }
```

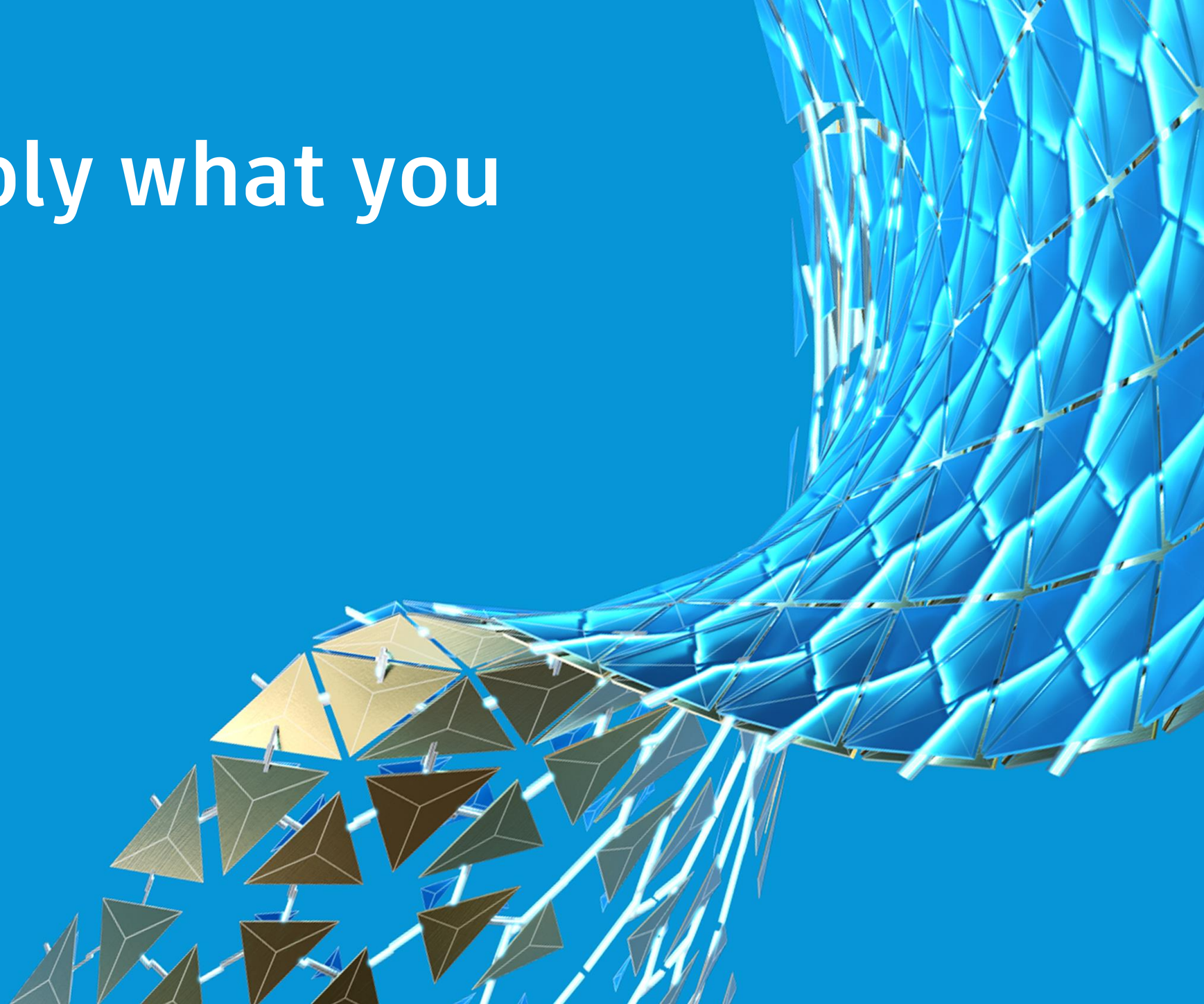
How do you really know a job completed?

If you're developing an application and want to know if the job is really done, you could check the job queue but that's not always reliable.

For custom Jobs, it's helpful if result is apparent:

- Property is updated
- File is moved
- State is changed
- Etc.

How to apply what you learned

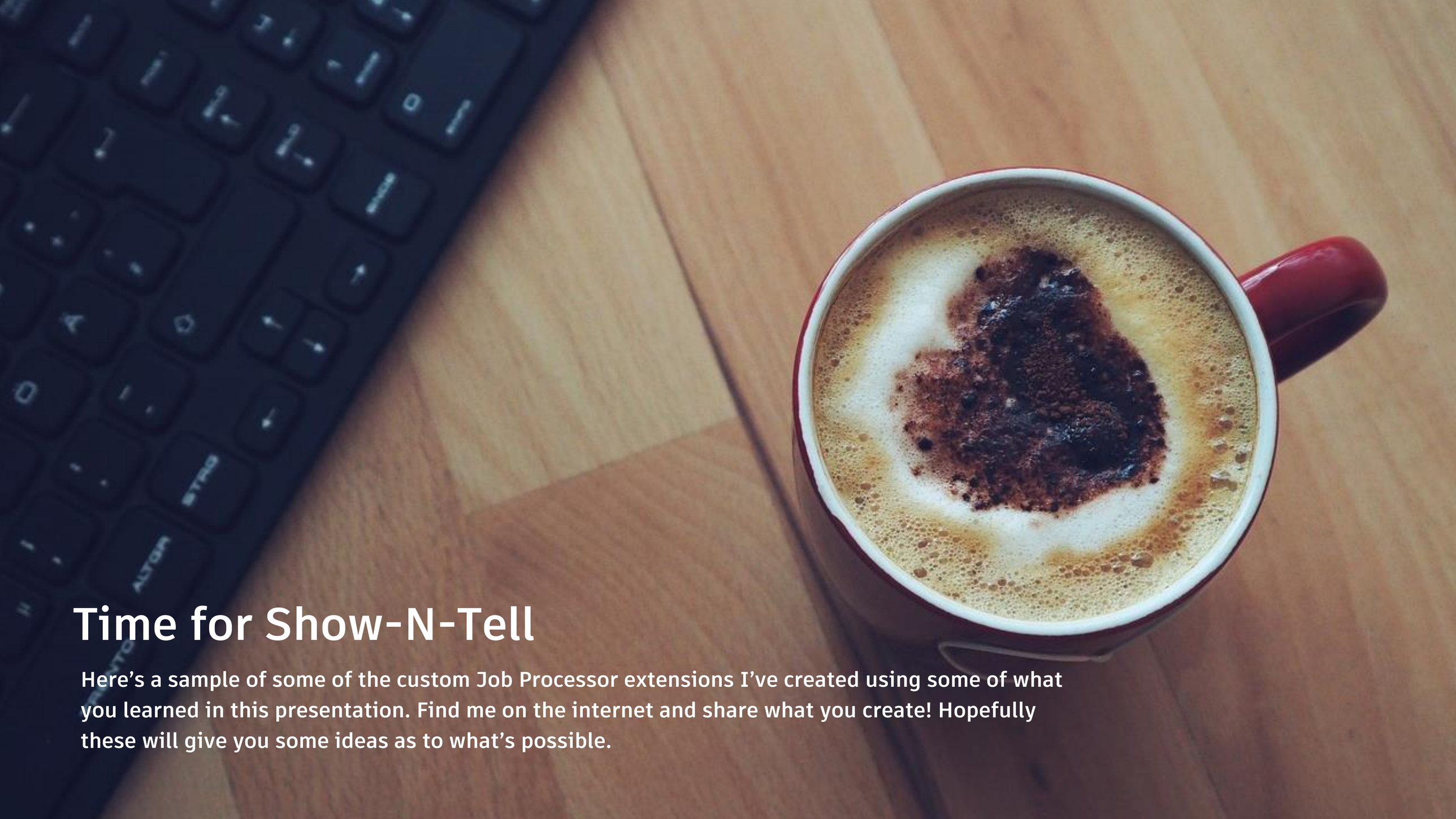


Download my GitHub Template

Right-Click to Add to Job Queue and Process Extension

- Creates a menu command when selecting files to add them to the job queue.
- Contains a Job Processor extension to run the jobs.
- Use it as a starting point.
- Be creative.
- Don’t screw up your data!

https://github.com/cadtoolbox/MFG468044

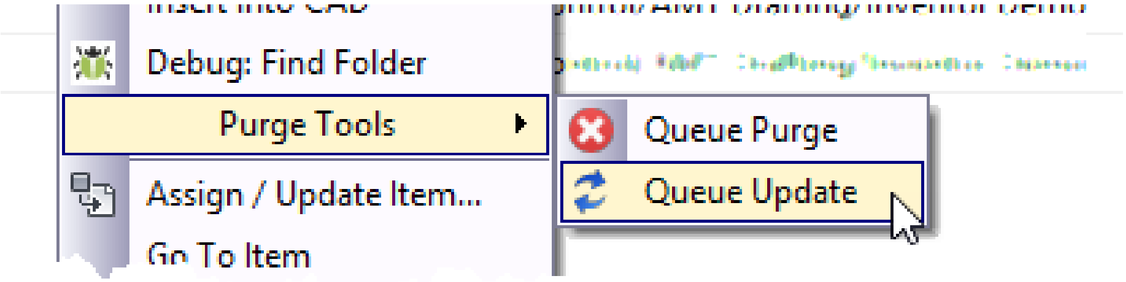


Time for Show-N-Tell

Here’s a sample of some of the custom Job Processor extensions I’ve created using some of what you learned in this presentation. Find me on the internet and share what you create! Hopefully these will give you some ideas as to what’s possible.

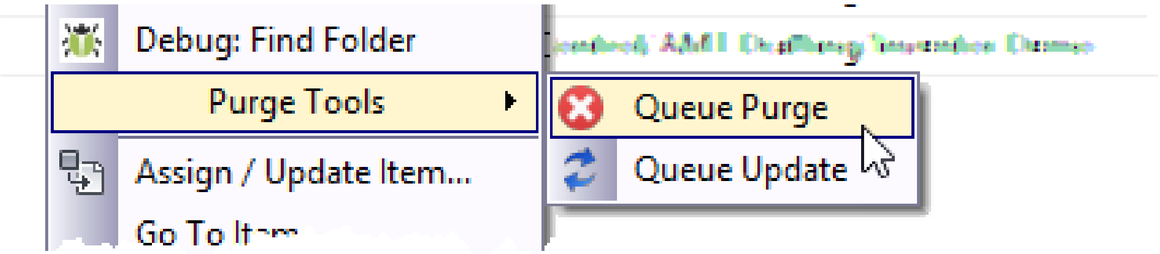
Job - **UpdateDesign**

- Downloads the file and updates the design in Inventor.
  - Migrates to latest release
  - Checks for sick dimensions, constraints, etc.
  - Updates properties
  - Turns off work features
  - Allows for purge of child file versions to be more effective.



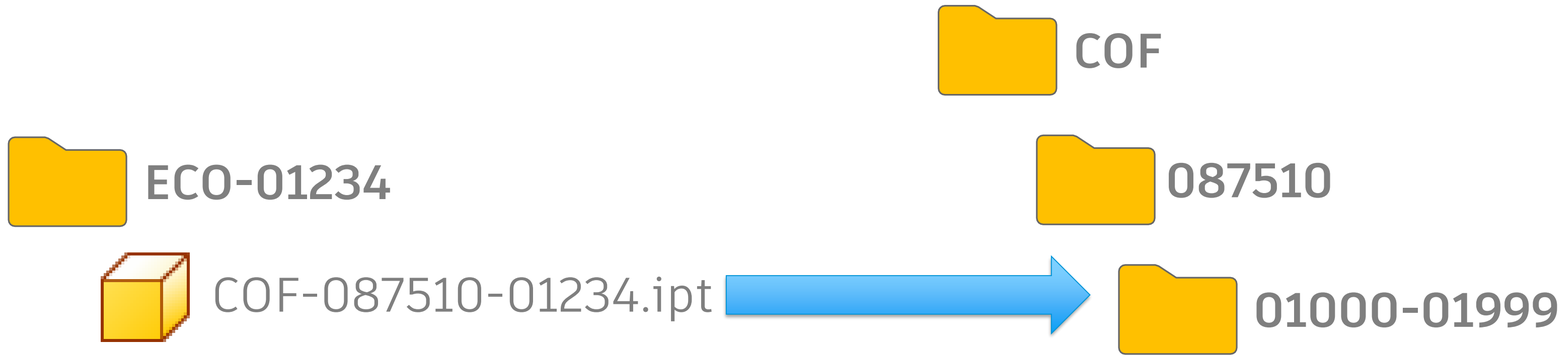
Job - **Purge**

- Purges file versions unconditionally based on rules.
  - Reads lists of comments to purge from a text file and purges matching versions.



Job - **MoveFile**

- Determines rules based on part number where the file needs to be moved it.
  - Determines new folder destination
  - Creates folder if it’s missing
  - Moves the file

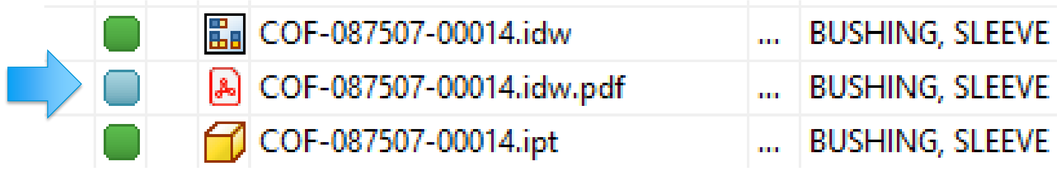


Job - **ReleaseFile**

- Determines if the file is Pending Release or Pending Obsolete
  - Changes the Lifecycle state to the appropriate final state

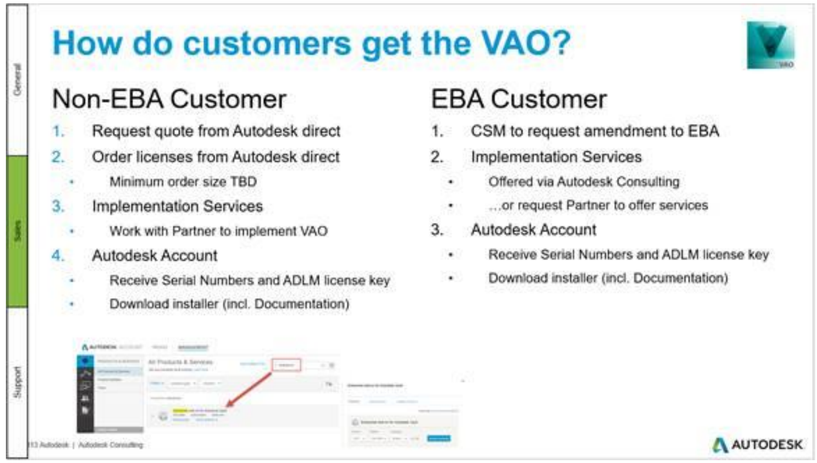
Job – **CreatePDF (in Vault)**

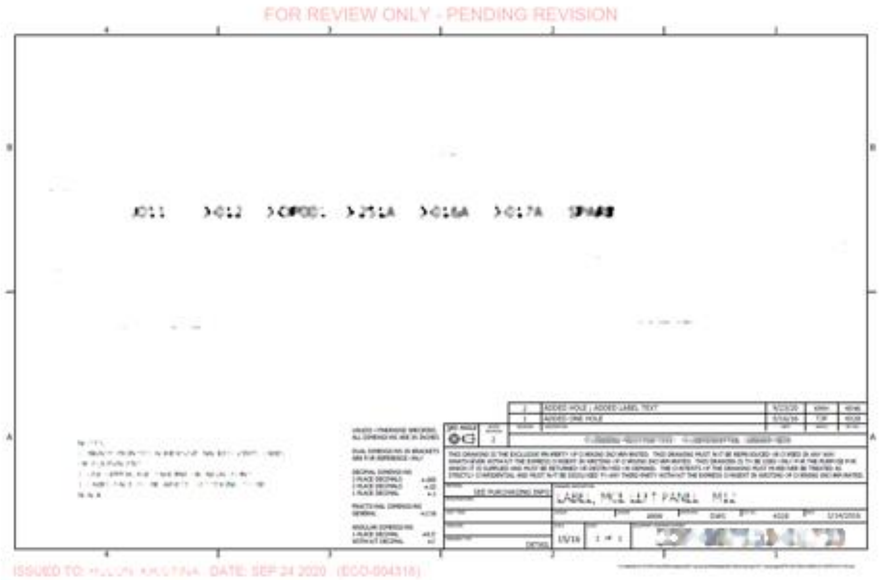
- Uses the out-of-the-box CreatePDF job but allows it to be created on demand while the ribbon Create PDF is turned off for all users.
  - Used to create only PDFs on release.



Job – **CreatePDF (PDF Package)**

- Uses the Vault Enterprise Tools PDF creation job extensions.
  - Passes parameters to create the PDF for Inventor, AutoCAD, Office files.
  - Creates external to Vault and copies to a shared drive.
  - When all PDFs have been created, they are merged and watermarked with appropriate properties from Vault





Job – **Batch Logic**

Batch Logic allows you to batch process iLogic rules for Autodesk Inventor files.

- Batch Logic is a stand-alone desktop application that processes Inventor files by selected folder. By selecting the folder to process, and selecting a rule to process, Batch Logic will run the iLogic rule against every Inventor file in the folder.

- Batch Logic+ runs within the Vault Job Processor and can be applied to Vault lifecycle state changes. Batch Logic will run the designated iLogic rule against the file being updated.

https://www.youtube.com/watch?v=7VQVXY8f1AQ

http://www.cadtoolbox.com/cad-tools/batch-logic/

Pulling it All Together Using Fusion Lifecycle

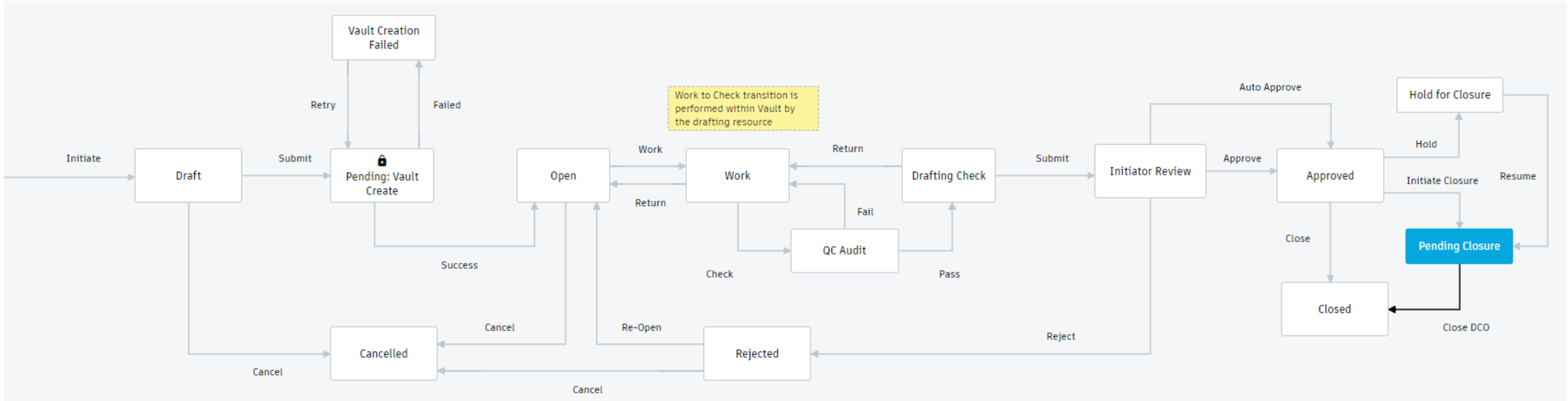
FUSION LIFECYCLE

AUTODESK UNIVERSITY

See You at AU2021

I'm using Fusion Lifecycle to drive the Change Order transitions in Vault. Each progression of the Change Order state updates data in FLC and spawns jobs in the Vault Job Processor to update the Vault file data or Change Order information. Vault change order states sync with FLC.

The custom FLC→Vault integration is the head that drives the entire process.



From a manual process to 17,000 lines of code driving engineering automation in less than 2 years

Key Takeaways

- The Job Processor can be utilized to make your teams more productive.
- Plan on infrastructure for the physical Job Processors.
- Lots of examples and templates available to help you get started.
- Don’t be scared.
- Don’t screw up your data.
- You can do this.
- Marcus Koechl is the man. Check out his AU class on iLogic and the Vault Job Processor!
- Stop wasting time. Doing things manually is so AU1997.
- You can now skateboard through your office.
- These aren’t key takeaways, just a list of things I wanted to leave with you.
- This is my first AU class…. ever! Please fill out the class survey.
- When do we get our AUGI beer mugs?