

API Enhancements in Vault 2019

Paul Gunn

Vault Software Architect



About the speaker



Paul Gunn

Paul has been a member of the Vault team since 2003. Throughout that time, he has been involved in the development of many features of the product including security, search, and replication with a focus on server-side functionality. Paul currently serves as a software architect for Vault.

Learning Objectives

- Learn about new API functionality available in Vault 2019
- Discover how this functionality was used to implement Project Sync
- Understand how these capabilities can be used in a custom application
- See the code behind and understand how it works

What is Project Sync?

The background features a light blue and white color palette. A prominent element is a curved, grid-like pattern of thin blue lines that sweeps across the lower right portion of the frame. Below this curve, there are faint, larger-scale geometric shapes, possibly triangles or polygons, in a slightly darker shade of blue. The overall aesthetic is clean, modern, and technical.

Name:
New Definition

Vault folder:
\$

- Enable cloud drive mapping
 - Enable manual sync
 - Upload related files based on Release bias

Cloud Drive folder:
C:\Users\gunnp\Fusion\William Schwerin\Vault\Paul\trdt

Schedule Folder Sync

Sync Folder:



Folder Path	Scheduled	Action
\$/Designs	No	Upload to Cloud Drive
\$/Inventor	Yes	Download from Cloud Drive

General Schedule Filter

Synchronization settings

- Daily at: 12:00 AM
- Every: 8 hours
- None

OK Cancel Apply Help





How is sync configuration stored?

The background features a dynamic, abstract design. It consists of a series of curved, overlapping planes in various shades of blue, ranging from light to dark. These planes are set against a white background. A grid of thin, light blue lines is visible, particularly in the lower right quadrant, where it appears to be part of a larger, curved structure. The overall effect is one of depth and modern, technological aesthetics.

Entity attributes overview

- Entity Attributes allow data to be programmatically associated with any entity
- This data is not directly user-visible and can be applied to read-only entities
- These are not considered part of history so do not create new file versions
- Attributes are in a user-defined namespace so multiple usages don't collide

Entity attribute object (EntAttr)

	Name	Description
	<u>Attr</u>	User-defined name of the attribute.
	<u>Cloaked</u>	Is the entity cloaked for the current user.
	<u>EntityId</u>	Entity tagged with this attribute.
	<u>Val</u>	Value of the attribute.

Entity attributes API

- `void SetEntityAttribute(long entityId, string namespace, string attribute, string val)`
 - Sets a named attribute on a given entity. A null val will delete an existing attribute.
- `EntAttr[] GetEntityAttributes(long entityId, string namespace)`
 - Gets all entity attributes associated with a given entity.
- `EntAttr[] FindEntityAttributes(string namespace, string attribute);`
 - Finds all entity attributes with a given attribute name.
- `EntAttr[] FindAllEntityAttributes(string namespace)`
 - Finds all entity attributes in the given namespace.













How are sync jobs scheduled?



Scheduled jobs overview

- Legacy functionality supports adding job for one-time, immediate execution
- New functionality supports scheduling a recurring job to run at a given cadence
- At scheduled times, a traditional job is added to the queue for normal execution / consumption
- Scheduled jobs can be viewed with other background tasks in ADMS console

Scheduled job object (SchedJob)

 CreateDate	The date the job was created.
 CreateUserId	The ID of the user who created the job.
 CreateUserName	The name of the user who created the job.
 Descr	A description of the job.
 ExecDate	Date/Time at which the job is first scheduled (can be DateTime.Now).
 ExecFreq	Frequency in minutes at which to schedule the job (e.g. 1440 minutes = daily).
 Id	A unique identifier for the job.
 IsOnSite	In a multi-site environment, this property tells if the file is on the local site.
 ParamArray	An array of parameters which provide meta-data about the job.
 Priority	The priority of the job. A lower number means a higher priority. 1 is the lowest possible number.
 Typ	The job type.
 VaultId	The ID of the Vault that the job applies to.

Scheduled job API

- SchedJob AddScheduledJob(string type, string desc, JobParam[] paramArray, int priority, System.DateTime execDate, int execFreqInMinutes)
 - Adds a scheduled job with given execution date and frequency
- void DeleteScheduledJob(long id)
 - Deletes the given scheduled job
- SchedJob GetScheduledJob(long id)
 - Gets information about the given scheduled job
- SchedJob[] GetScheduledJobs()
 - Gets information about all scheduled jobs

Vault Notification Sample



Demo

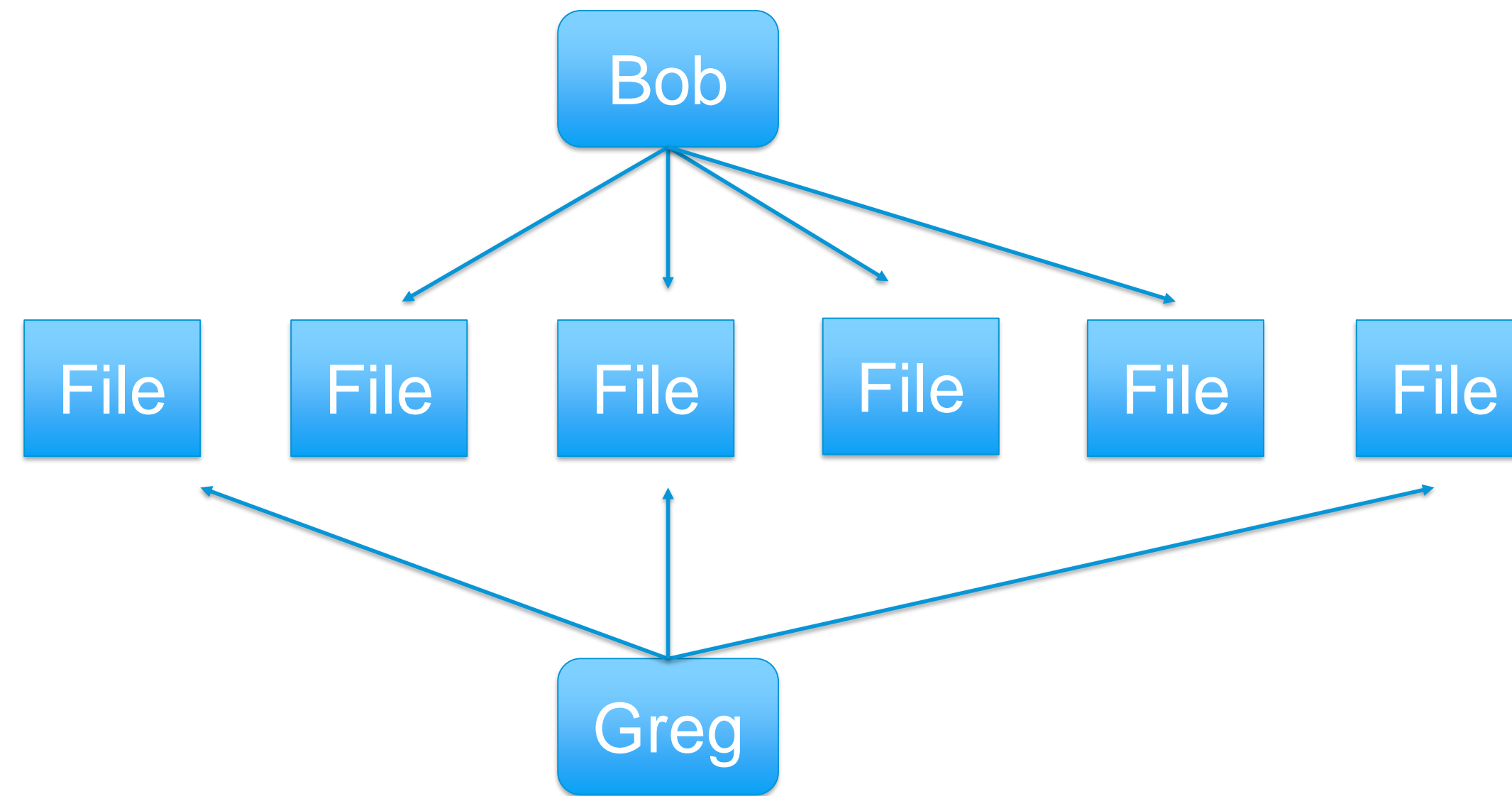
Limitations of the Sample Application

- Email
 - Assumes the email SMTP server is on 'localhost'
 - Assumes the email addresses for vault user have been correctly configured
- Job Scheduling
 - A different scheduled job is created for each user, which could be inefficient
 - The user must have permissions to create a scheduled job
- Fit and finish
 - There are no options for including file dependencies and drawings
 - Integration with Vault Explorer via command extension

How is the notification list stored?



Design of the notification list



```
private string AttributeNamespace
{
    get { return "Sample.VaultNotification." + UserName; }
}

private const string MostRecentAttribute = "MostRecentIterationId";
```

Loading the notification list

```
public IEnumerable<IEntity> Load()
{
    var attributes = Connection.WebServiceManager.PropertyService.FindEntityAttributes
        (AttributeNameSpace, MostRecentAttribute) ?? Enumerable.Empty<ACW.EntAttr>();

    attributes = attributes.Where(a => !a.Cloaked); // ignore files we no longer have access to

    if ( !attributes.Any() )
        return Enumerable.Empty<IEntity>();

    var resolvedIds = Connection.PersistableIdManager.ResolvePersistableIds(attributes.Select(a => a.Val));
    return resolvedIds.Select( rid => rid.Value );
}
```

Modifying the notification list

```
public void Add(IEnumerable<IEntity> entities)
{
    var persistentIds = Connection.PersistableIdManager.GetPersistableIds(entities, getLatest: false);

    foreach (var current in persistentIds)
    {
        Connection.WebServiceManager.PropertyService.SetEntityAttribute
            (current.Key.EntityMasterId, AttributeNamespace, MostRecentAttribute, current.Value);
    }
}

public void Remove(IEnumerable<IEntity> entities)
{
    foreach (var ent in entities)
    {
        Connection.WebServiceManager.PropertyService.SetEntityAttribute
            (ent.EntityMasterId, AttributeNamespace, MostRecentAttribute, null);
    }
}
```

Updating the notification list

```
public IEnumerable<IEntity> Update(IEnumerable<IEntity> entities)
{
    var modified = GetModified(entities);
    if( !modified.Any() )
        return Enumerable.Empty<IEntity>();

    var updatedFiles = Connection.FileManager.GetLatestFilesByIterationIds
        (modified.Select( e => e.EntityIterationId ).Values);

    Add(updatedFiles);
    return updatedFiles;
}
```

How is does the notification job work?

The background features a complex geometric design. A prominent element is a curved, blue-tinted surface that resembles a grid or a series of parallel lines, curving upwards from the bottom left towards the top right. Below this curve, there are several overlapping triangular and polygonal shapes in shades of light blue and white, creating a sense of depth and perspective. The overall aesthetic is clean, modern, and technical.

Scheduling the notification job

```
public void Create()
{
    if( m_job != null )
        return;

    // Schedule job recurring every 24 hours at midnight.
    //
    var frequency = TimeSpan.FromDays(1);
    var timeofday = DateTime.ParseExact("00:00", "HH:mm", System.Globalization.CultureInfo.InvariantCulture);
    var param = new ACW.JobParam() { Name = NotificationJobUser, Val = m_conn.UserName };

    m_job = m_conn.WebServiceManager.JobService.AddScheduledJob
        (NotificationJobType, "Watch list notifications for " + m_conn.UserName, new [] {param},
        50, timeofday, (int)frequency.TotalMinutes);
}
```

Executing the notification job

```
public JobOutcome Execute(IJobProcessorServices context, IJob job)
{
    var userName = job.Params[NotificationJob.NotificationJobUser];
    if( userName == null )
    {
        context.Log("User name parameter was not specified on the job", MessageType.eError);
        return JobOutcome.Failure;
    }

    var notificationList = new NotificationList(context.Connection, userName);
    var notificationReport = new NotificationReport(context.Connection, notificationList);
    notificationReport.SendReport();
    notificationList.Update(notificationReport.Modifications);

    return JobOutcome.Success;
}
```


Conclusion



Call to action

- Project Sync was built on top of these API foundations
- Other future features are also being built on this functionality
- Vault has no secret server APIs: what we can do, you can do.
- Amaze us with what you can do!



AUTODESK®

Make anything™

Autodesk and the Autodesk logo are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document.

© 2018 Autodesk. All rights reserved.

