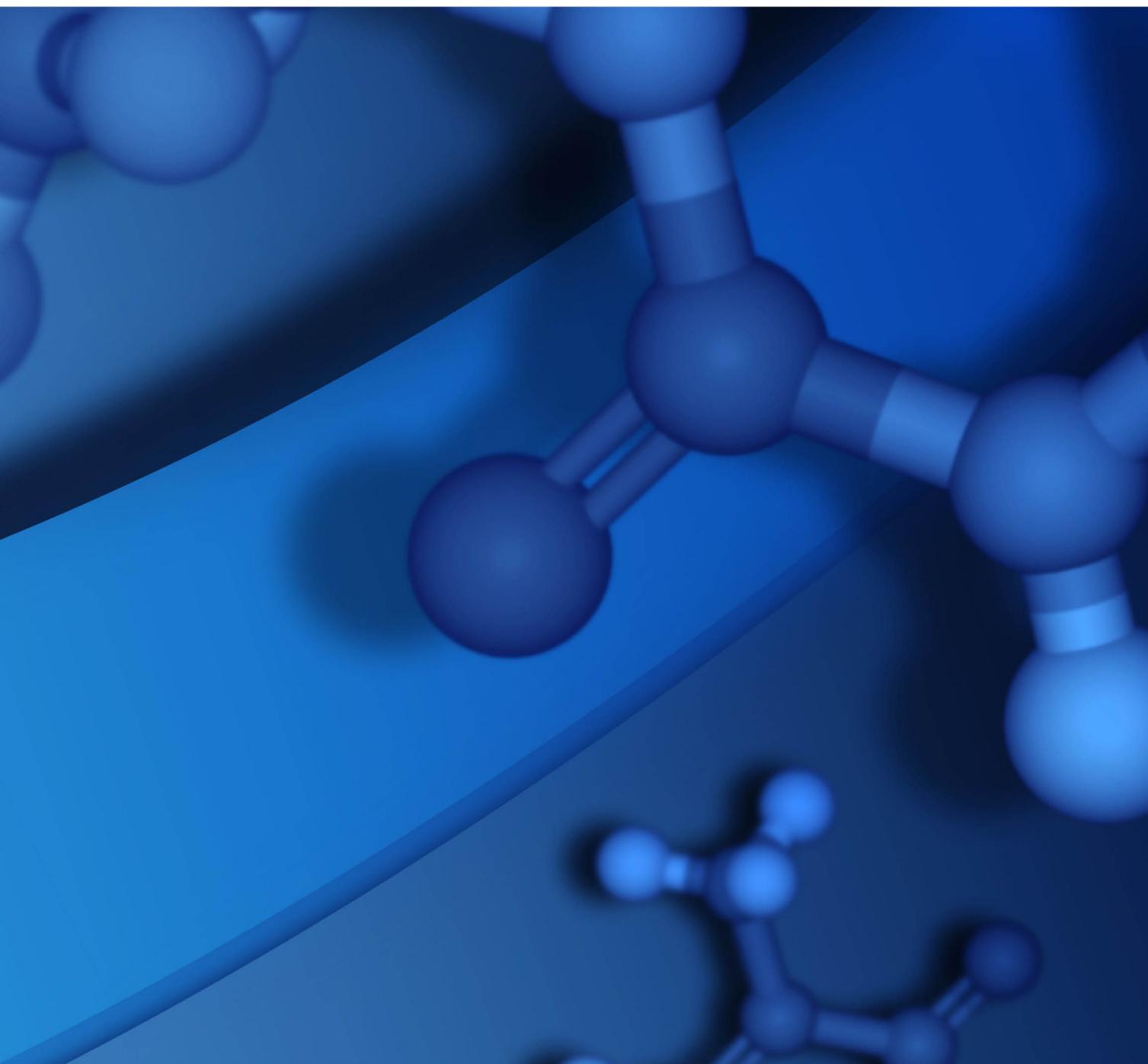


VAULT SERVER ADMINISTRATION TOOLS GUIDE

BIOVIA WORKBOOK 2021



Copyright Notice

©2020 Dassault Systèmes. All rights reserved. 3DEXPERIENCE, the Compass icon and the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3DVIA, 3DSWYM, BIOVIA, NETVIBES, IFWE and 3DEXCITE, are commercial trademarks or registered trademarks of Dassault Systèmes, a French "société européenne" (Versailles Commercial Register # B 322 306 440), or its subsidiaries in the U.S. and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

Acknowledgments and References

To print photographs or files of computational results (figures and/or data) obtained by using Dassault Systèmes software, acknowledge the source in an appropriate format. For example:

"Computational results were obtained by using Dassault Systèmes BIOVIA software programs. BIOVIA Workbook was used to perform the calculations and to generate the graphical results."

Dassault Systèmes may grant permission to republish or reprint its copyrighted materials. Requests should be submitted to Dassault Systèmes Customer Support, either by visiting <https://www.3ds.com/support/> and clicking **Call us** or **Submit a request**, or by writing to:

Dassault Systèmes Customer Support
10, Rue Marcel Dassault
78140 Vélizy-Villacoublay
FRANCE

Contents

Chapter 1: About This Guide	1
Vault Server Setup Checklist	1
Logging in to the Vault Administration Console	2
Managing Vault Server Objects	2
Chapter 2: Assigning Default Templates and Workflow Actor Roles	4
Assign a Default Template to a User	4
Assign Users and Groups to Workflow Actor Roles	4
Remove Workflow Actors from Users	5
Chapter 3: Administering Vault Server Repositories	6
Manage Repository Subscriptions	6
Subscribing Users and Groups to Repositories	7
Unsubscribe a User or Group from a Repository	7
Manage Access to Vault Objects	8
Summary of Vault Object Permissions	8
Add Repository Folders	10
Rename a Repository	10
Starting the Vault Services	11
Stopping the Vault Services	11
Chapter 4: Configuring Application Permissions	12
Summary of Application Permissions	12
Using the Import/Export Application Permission Utility	16
How the Utility Works	16
Running the Export Command	17
Editing and Tokenizing the AppPermissions.xml File (Optional)	17
Running the Import Command	18
Example Application Permission Files	19
Example AppPermissions.xml	19
Example permissionConfiguration Files	20
Sample Application	23
Implementing the DisableUndoMyCheckouts Permission	24
Requirements for Moving a Vault Folder or Object	25
Pipeline Pilot Configuration Keys	25
Configuring Pipeline Pilot RunProtocol Settings (New Installs Only)	26

Compound Registration	26
Modify Registration Service Properties	27
Registration Service Permission Parameters	27
Database Web Service and Material Property Lookup Service	28
Locking Recipe Sections at Specific Workflow Stages	28
Locking the Task Plan at Specific Workflow Stages	29
External Structures Conversion	30
Configure the External Data Conversion Service	30
Specify Document Conversion Template	31
Modify Section Settings in Document Conversion	32
Chapter 5: Developing Signature Policies	33
Signature Policy Events	33
Create a Signature Policy	33
Set Meanings or Reasons	34
Modify Signature Policies	35
Signature Policy Properties Reference	35
Document Template Management Tools Signature Policy	36
Chapter 6: Defining Workflows	37
Workbook Activities from Workflows	37
Workflow Design Best Practices	38
Workflow Examples	39
Workflow SDK	39
Archive Using a Workflow	40
Create Vault Workflow Actors	40
Add Workflow Definitions	41
Removing a Workflow Definition	41
Move Experiments or Objects Between Workflow Stages	42
Generate a Workflow Association	42
Generate Workflow Association Code Example	43
Get VaultID for Generating a Workflow Association	44
Enable Workflow Associations	45
Removing Workflow Associations	45
Disabling Workflow Associations	45
Placeholder Formats	46
Appendix A: Administering Vault using PowerShell Scripts	48

PowerShell Prerequisites	48
Initial PowerShell Commands	48
Vault Identifiers	50
Retrieve a Vault ID with Get-VaultId	50
Retrieve a Vault ID with Oracle SQL*Plus	51
PowerShell Scripts	51
Add-Association.ps1 Script	53
Change-WorkflowAssociationEnable-AddHistory.ps1 Script	55
Connect-Server.ps1 Script	56
Create-WorkflowActor.ps1 Script	57
Create-WorkflowAssociation.ps1 Script	58
Get-VaultId.ps1 Script	59
Get-VaultObject.ps1 Script	60
Load-Assemblies.ps1 Script	64
Publish-WorkflowActorAssociation.ps1 Script	64
Save-VaultObject.ps1 Script	66
Set-DefaultTemplate.ps1 Script	68
Set-Permissions.ps1 Script	69
Create Print Audit History Script	71
Create Signature Policy Examples	72
Standard signature policy	74
Countersign signature policy	74
Message Queuing Function Example	75
Manage Workbook Folders	76
Create a Folder	76
Retrieve a Folder Using the Folder Name	77
Retrieve Folders with the Same Name	77
Manage Workbook Permissions	77
Permissions	78
Retrieve Folder Permissions	78
Create Read Permissions	79
Create Write Permissions	79
Allow Read and Deny Permissions	79
Store Variables Using CSV files	80
Appendix B: Audit Trail Actions	81

Action Types 82

Chapter 1:

About This Guide

This guide focuses primarily on how to use the **Vault Administration** tool, which is also known as the **Vault Administration Console**. The Workbook Installer installs this tool after it runs the Vault Deployment Utility. This guide also describes several other scripts and tools for administering Vault repository objects. For additional information on administration tasks and tools, see the *Vault Server Administration Guide*.

You use the Vault Administration Console to perform the following tasks:

- Assign a default template to users.
- Assign workflow actor roles created using the Workflow Designer to appropriate users and groups.
- Manage repository folders and subscribe users and groups to repositories.
- Configure application permissions so that Workbook features are correctly enabled or disabled and so that Workbook can interface correctly with other applications and services, such as the look-up service, registration, and external data repositories, if you use them.
- Create signature policies for use in workflows, forms, and tables.
- Create associations between workflow definitions and users, groups, experiments, folders, and repositories.
- Transition experiments in a workflow between various stages of that workflow.

Vault Server Setup Checklist

The following checklist identifies tasks you must perform before you can perform the Vault Server administration and configuration tasks identified in this guide.

Task	Actions	Done
Verify installation of Vault Server components.	Verify that the BIOVIA Vault Server and the Vault Administration Console are installed. If you use it, also verify that the optional Workflow Designer are installed.	<input type="checkbox"/>
Identify projects, users, user groups, and access requirements.	List the projects and their users. Group users based on their access requirements for projects and for tasks, such as filling material substance tables, creating templates, creating property sets, and creating tables. Define permission levels accordingly.	<input type="checkbox"/>

Task	Actions	Done
Plan your repositories.	A default repository is created when you install Vault Server, but you can create additional repositories. Each repository requires a separate Oracle Database schema. BIOVIA recommends using separate repositories if you have discrete unrelated projects and you need to segregate the projects and their users. For more information, consult your DBA and see the <i>Vault Server Administration Guide</i> .	<input type="checkbox"/>
Plan the folder structure for each repository.	Decide whether separate folders are required for each user or for each user group. Take permission requirements into account: <ul style="list-style-type: none"> ■ If you create a read-only and give a user group write permissions to that folder, the user group permissions take precedence over the folder permissions. ■ If you create a read-only folder, but some folders lower in its hierarchy need to be read-write, apply the Traverse Folder permission to the folders above the read-write folder. 	<input type="checkbox"/>
Verify that a secure certificate is installed on the computer that users will log in to when they access Workbook.		<input type="checkbox"/>

Logging in to the Vault Administration Console

Membership in the Vault Global Administrators group is required to modify properties using the Vault Administration Console.

IMPORTANT! The Vault Administrator user *must* be a member of the Vault Global Administrators groups and the Repository Administrators groups. Do not remove the Vault Administrator user from these groups. Similarly, do not remove any of the default BIOVIA users from the Vault Global Administrators groups.

To log in to the Vault Administration Console:

1. From the Start menu, select **Vault Administration Console**.
2. In the Console Root\Vault Administration\Vault Server, expand **Vault Administration**.
3. Right-click **Vault Server** and select **Server > Login**.
4. In the **Log in to Vault**, type the Global Administration user name and password.
5. In the **Domain** field, select the domain for the server.
6. In the **Vault Server** field, select the specific Vault server to manage and click **OK**.

Managing Vault Server Objects

You can bulk-load objects to BIOVIA Vault Server by creating a comma-separated values (.csv) file containing the required data, and then executing a custom PowerShell script that calls the file.

BIOVIA Vault Server provides PowerShell scripts to assist with bulk-loads.

The objects that you manage in the Vault Administration Console, shown below, include:

- Repositories
- Signature Policies
- Workflows

You can only delete associations with groups. All other objects remain in the Vault database. BIOVIA recommends that you do not delete groups in BIOVIA Foundation Hub. Signing permission problems occur when groups are deleted. Contact Dassault Systèmes Customer Support for more information.

You can create custom PowerShell scripts for specific tasks at your site. BIOVIA recommends this procedure for bulk-loading vocabularies into the BIOVIA Vault Server database. For information on PowerShell scripts, see [Administering Vault using PowerShell Scripts](#) on page 48.

Note: The Vault Administration Console contains a **New Window from Here** link in the **Actions** pane. Do not use the link, because clicking the link can produce unexpected results.

Chapter 2:

Assigning Default Templates and Workflow Actor Roles

Assign a Default Template to a User

You can assign only one default template to a user at a time. You can assign a default template to users rather than to groups. By default, users are not assigned to a template. You can change the default template for a user at any time.

The Properties dialog's Profiles tab in the Vault Administration Console is the only location where some of the templates installed with BIOVIA Vault Server are visible. The templates in the Site repository are not searchable using Notebook Explorer. To make any of the templates in the Site repository available, you must designate a template as the user's default template, or create a copy of the template and store it in a versioned repository.

Use the `Set-DefaultTemplate` PowerShell script to set up the default template. For more information, see [Set-DefaultTemplate.ps1 Script](#).

To assign a default template to a user:

1. In the Vault Administration Console, expand the Vault server node, and select **Users**.
2. In the **Users** pane, select the user, and click **Properties**.
3. In the **Selected Item Properties** dialog for the user, click the Profile tab.
4. Click the (...) ellipsis button to open the **Default Document Template** window.
5. In the **Default Document Template**, click the check box next to the template that you want to assign to the user.
6. Click **OK**.

Assign Users and Groups to Workflow Actor Roles

This task is relevant only for Workflow Designer users. You use the Vault Administration tool to assign users and groups to the workflow actor roles that you defined in Workflow Designer.

The Workflow Designer enables you to identify the *stages* through which an object can be transitioned, the *workflow actor roles* that can edit the object at various stages in the workflow, and the workflow actor role required to transition the object from one stage to another. When an object transitions from a stage that allows editing by an actor role to a stage that does not allow editing by that same actor role, the object's permissions are automatically updated.

IMPORTANT! BIOVIA strongly recommends against assigning administrative users or user groups to workflow actor roles. Users such as the Vault Administrator, Global Administrator group, and members of the Global Administrative group require continuous permissions for all objects in all locations and workflow stages in order to perform their administrative duties. Workflow Designer roles can result in the loss of such permissions.

To assign a workflow actor role to a user or group:

1. In the Vault Administration Console expand the Vault Server node and then click **Users or Group**.
2. In the **Users or Group** pane, select the user or group to assign to workflow actor roles, and then click **Properties**.
3. Click the **Workflow Actors** tab.
4. In the **Select Actors** dialog, click **Add**.
5. Select the workflow actor roles to assign to the user or group.
6. Select the **Actor** (user or group) to assign to the **Type**.
7. Click **Apply** and **OK**.

Tip: Changes to workflow actor roles and associations can take up to five minutes to take effect.

Remove Workflow Actors from Users

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console.

To remove a Vault workflow actor from users:

1. In the Vault Administration Console, expand the Vault Server node, and select the **Users** node.
2. In the **Users** pane, select a user, and select **Properties**.
3. In **Properties** dialog, click the **Workflow Actors** tab.
4. In the Workflow Actors tab, right-click the workflow actor and select **Remove**.
5. Click **OK**.

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

Chapter 3:

Administering Vault Server Repositories

Vault Server uses information repositories to manage the data stored in the Oracle database. The data includes the objects and the data provided by users. Vault provides three default repositories added when you create the Vault database.

You can add and modify objects in the repositories. Objects added to the Vault repositories are saved in the associated Oracle tables on the database computer.

Note: Workflow Tools is a Vault schema but it is not a repository.

Repository Name	Description
Site Repository	Contains the published workflow objects, including: <ul style="list-style-type: none">■ BIOVIA Workbook Assemblies■ Balance server configuration information■ Framework application permissions■ Workbook application permissions■ Property Set Definitions■ BIOVIA Workbook templates <p>Note: You cannot delete objects in the site repositories.</p>
Home Repository	Contains the roaming repository for each user within the organization that can access the BIOVIA Vault Server. The Home repository stores objects, the user profile, and only the latest versions of user files. Users can delete, and synchronize files in the Home Repository. Auditing, workflow, and archiving are not allowed. The user cannot search the Home Repository because it is not indexed.
VersionedRepository	Contains the document management repository in BIOVIA Vault Server. The versioned repository tracks changes, preserves version history, stores versions, allows auditing, workflow, and archiving. The versioned repository does not allow delete or synchronization.

Run the RequeueVaultObjects utility on objects with the Processed status. The utility enables the Search function to work correctly and to display the requested data in Notebook Explorer.

Manage Repository Subscriptions

Users cannot access a Vault Server repository until an administrator subscribes them or groups to which they belong to the repository. This is done by using the Vault Administration Console.

Subscribing a user or group to a repository enables the user or group to *view* that repository. To enable the user or the group's members to actually perform Workbook *actions* within a repository, you must switch to Foundation Hub to give them the appropriate permissions for those actions. For details, see the *Foundation Hub Administration Guide*.

To go a level deeper and control who can act on specific *folders* or *objects* within a repository, you can override the inherited permissions on the folders, subfolders, and objects within a repository. For details, see [Modify the Permissions on Vault Objects](#).

Subscribing Users and Groups to Repositories

You must explicitly subscribe users and user groups to each Vault repository that they need. Users cannot view a repository or its objects until they or a group to which they belong has been subscribed to that repository.

Note: To manage subscriptions to repositories, you must be a member of the Vault Global Administrators group.

To subscribe users or groups to repositories:

1. Log on to the Vault Administration Console as a member of the Vault Global Administrators group and expand the Vault Server node.
2. Select the **Groups** or the **Users** node.
3. Select the specific group or user that you need to subscribe to a repository.
4. Select **Properties**, and click the **Repository Subscriptions** tab.
Each row on this tab represents a specific repository.
5. In the row that identifies the repository, click the **Allow** check box to select it.

Note: If the **Allow** check box is *already* selected, clicking it causes it to become unselected and causes the **Deny** check box to become selected. If neither checkbox is already selected or if only the **Deny** check box is already selected, clicking **Allow** causes the **Allow** check box to become selected and the **Deny** check box to become unselected. Clicking the **Deny** check box has no effect.

6. Click **Apply** and **OK**.

An audit history entry is created each time you subscribe or unsubscribe a user or a group. For more information, see [Audit Trail Actions](#).

Unsubscribe a User or Group from a Repository

If you need to remove a user or group's subscription to a repository so that they can no longer see or use it, you can unsubscribe them.

Note: You must be a member of the Vault Global Administrators group to manage repository subscriptions.

To unsubscribe a user or a group from a repository:

1. Log on to the Vault Administration Console as a member of the Vault Global Administrators group and expand the Vault Server node.
2. Select the **Groups** or the **Users** node.
3. Select the specific group or user that you need to subscribe to a repository.
4. Select **Properties**, and click the **Repository Subscriptions** tab.
Each row on this tab represents a specific repository.
5. In the row that identifies the repository, click **Allow** to unselect the Allow check box.

Note: When the **Allow** check box is *already* selected, clicking it causes it to become unselected and causes the **Deny** check box to become selected. If neither checkbox is already selected or if only the **Deny** check box is already selected, clicking **Allow** causes the **Allow** check box to become selected and the **Deny** check box to become unselected. Clicking the **Deny** check box has no effect.

6. Click **Apply** and **OK**.

An audit history entry is created as a result of unsubscribing groups to repositories. For more information, see [Audit Trail Actions](#).

Manage Access to Vault Objects

Note: If an object is subject to a workflow definition whose rules impact permissions, the workflow rules supersede any rules that are applied directly to the object.

The permissions for accessing Vault objects are specified and maintained by using the Notebook Explorer in the Workbook client application. Authorized administrators set and modify the permissions of any object at any level in Notebook Explorer to control what can be done with that object, and by whom (by which users and groups).

To access the **Permissions** menu option, a user must have the Vault/Administrator application permission. To view the current permissions for a selected object (folder, subfolder, or specific object), the user must also have `ReadPermissions` for that object. To change the object's permissions, the user must also have `UpdatePermissions`.

To determine a user's permissions for an object, the system pools the permissions granted directly to the user with those granted to the user's groups. Any explicitly *denied* permissions supersede *allowed* permissions. For example, if the pooled permissions include a `writeData` *allow* and a `writeData` *deny* permission for the same folder, the user cannot write data to that folder.

For a list of Vault object permissions, see [Summary of Vault Object Permissions](#) on page 8.

An audit history entry is created when permission settings are changed.

Summary of Vault Object Permissions

Workbook provides the following permissions for controlling access to objects (folders, subfolders, experiments, templates, and so on) in a Vault repository.

Object Permission	Tasks that Require this Permission
Checkout	Change the checkout state of an object.
CreateFolder	Create subfolders. This task also requires the following: <ul style="list-style-type: none"> ■ Membership in the Vault Global Administrators group. ■ The Vault/Folder.Administrator⁺ application permission⁺.
Delete	Delete objects. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Note: Objects that have been checked in to the repository <i>cannot</i> be deleted.</p> </div>
ReadData	Read object content. This task also requires the following:

Object Permission	Tasks that Require this Permission
	<ul style="list-style-type: none"> Read, Update, and Write permissions for the PropertySetDefinitions folder in the Site Repository. The Vault/PropertySetEditor application permission[†].
ReadPermissions	View the permissions granted to all users of an object, instead of only those granted to you.
ReadProperties	View all properties of an object <i>except</i> its permissions. <p>Note: By default, the Pipeline Pilot Query Service that gathers data from disparate data sources treats the ReadProperties permission the same as the ReadData permission—users with only ReadProperties are also allowed to ReadData. As of 2019 SP1, however, you can configure the Workbook IDS Security Plug-in to differentiate between these two permissions and to filter property and content data accordingly.</p>
Rollback	Roll an object back to an earlier version. This task also requires the following permission: <ul style="list-style-type: none"> workFlow Transition
TraverseFolder	See all folders within the repository. To see folder content requires additional permissions. If a user has ReadProperties, this permission is not needed.
UpdateFlags	Change the settings of the following flags on content objects: <ul style="list-style-type: none"> ReadOnly - prevent all updates to the object. Hidden and System - prevent the object from being displayed to end users. Archive - include the object in the next run of the archive process. Updating flags also requires: <ul style="list-style-type: none"> The Vault/PropertySetEditor application permission[†]. The following permissions for the Site Repository's PropertySetDefinitions folder: Read, Write, and Update. <p>Note: The user interface does not display the settings of update flags.</p>
UpdatePermissions	Change the permissions for Vault objects. This task also requires the following permission: <ul style="list-style-type: none"> The Symyx.Administration application permission called Administrator.[†]
UpdateProperties	Change the properties of an object.
WorkflowTransition	Transition an object from one workflow stage to another. <ul style="list-style-type: none"> To transition an object, a user must also be identified as an allowed actor in the workflow definition.

Object Permission	Tasks that Require this Permission
WriteData	<p>Change the content of an object.</p> <ul style="list-style-type: none">■ This task also requires Read, Update, and Write permissions for the PropertySetDefinitions folder in the Site Repository for the application System.PropertySetEditor.

†For information about application permissions, see [Summary of Application Permissions](#) on page 12.

Add Repository Folders

You can add folders to versioned Vault Server repositories.

To add a folder:

1. In the Workbook client application, expand the **Repositories** node of the Notebook Explorer **Browse** tab.
2. Right-click the versioned repository to which you want to add a subfolder and click **New Folder**.
3. In **New folder** dialog, type a name for the folder, and then click **Ok**.

Rename a Repository

If a repository is renamed, you must run the RequeueVaultObjects utility for all objects in the repository to display a user's search results correctly in Workbook.

When renaming a repository, use single-byte characters in the name. The following characters in the repository name can cause problems:

- Kanji single-byte and double-byte characters entered using the Input Method Framework.
- Extended characters.

Run the RequeueVaultObjects utility on objects with the Processed status. The utility enables the Search function to work correctly and to display the requested data in Notebook Explorer.

To rename a repository:

Note: Do not rename the Site or Home repositories.

1. From the Start menu, select Vault Administration Console.
2. In the Console Root\Vault Administration\Vault Server, expand the Vault server node, and [login](#).
3. Expand the **Repositories** node, right-click the repository to rename and select **Properties**.
4. In the **Properties** General tab, in **Name** type a new name for the repository, and click **Apply**.
5. Click **OK** to close repository properties.
6. Log out of the Vault Administration Console.
7. Log in to the Vault Administration Console to verify that the new name is displayed in the Versioned Repository. Then log out again.
8. [Stop Vault services](#); if load balanced, stop Vault services on each server.
9. Change the name of the Versioned Repository in the vaultvariables.nbx.config file to use in the future such as when you upgrade BIOVIA Workbook.

10. Requeue processed objects using the following command:
`RequeueVaultObjects.exe -repository <newRepositoryName> -status Processed`

Starting the Vault Services

Use the Windows Server Manager to start the Vault Server services.

Note: The Oracle instance that BIOVIA Vault Server connects to must be started before the Vault services. If the database is shut down for any reason, stop the Vault services, restart the Oracle instance, and then restart the Vault services.

To start the Vault services:

1. Open the Windows **Server Manager** and choose **Configuration > Services**.
2. Start the following services in this order:
 - a. World Wide Web Publishing Service
 - b. Vault Tomcat Server Service
 - c. Vault Workflow Service
 - d. Vault Message Processing Service
 - e. Vault Hub Synchronization Service
 - f. Vault Client Service

Stopping the Vault Services

IMPORTANT! Before you shut down the Oracle database or start an upgrade, ensure that all users are logged off, and then stop the Vault services.

To stop the Vault services:

1. Open the Windows **Server Manager** and choose **Configuration > Services**.
2. Stop the following services in this order:
 - a. Vault Client Service
 - b. Vault Hub Synchronization Service
 - c. Vault Message Processing Service
 - d. Vault Workflow Service
 - e. Vault Tomcat Server Service
 - f. World Wide Web Publishing Service

Chapter 4:

Configuring Application Permissions

When you install Workbook and run the Vault Deployment Utility, a core set of DLLs and application "permission" files provided by Symyx Framework Services is installed.

An application "permission" is a global setting that enables or disables a specific Workbook feature or an interface from Workbook to an external application or service such as BIOVIA Registration, Database Web Service, the External Data Conversion Service, or Pipeline Pilot.

You use Vault Administration Console to identify Vault Server application permissions and to provide the configuration details necessary to support each feature or interface that you require. The list of application permissions in Vault Administration Console varies based on the applications and services you use.

Note: To control who can actually *use* a feature that is enabled by an application permission, you edit the settings of the Vault Server entry in your **Foundation Hub > Admin & Settings > Applications**. You can use the Vault Administration Console to view, but not to modify, the permissions for users and groups.

Summary of Application Permissions

The following table identifies most Vault Server application permissions and the feature supported by each permission. Permissions flagged by an † character are permissions that are not installed with core installations; you will see them only if you have installed the application or service that requires them or, in some cases, only if you manually add them. You might also see permissions that are not listed in this table, if you have interfaces from Workbook to additional applications and services.

To control who can actually *use* a feature that is enabled by an application permission, you edit the settings of the Vault Server entry in your **Foundation Hub > Admin & Settings > Applications**. You can use the Vault Administration Console to configure an application permission and to view application permissions, but *not* to modify the permissions for users and groups.

Note: Ignore the **Application** or column in the list of permissions. In Foundation Hub, the application is always listed as **Vault**.

Vault Application Permission	Feature Supported by the Permission
External Data Conversion Service	Convert document data from Symyx Process Notebook and Accelrys Formulations Notebook so that the data can be used in Workbook. For information about configuring this service, see Configure the External Data Conversion Service .
MaterialInfoManager	Look up material or inventory using a material information resolver configured to interface with CISPro, OpenEye, CIMS, DiscoveryGate, and other external systems. For information about configuring lookup services, see the <i>Vault Server Installation Guide</i> .

Vault Application Permission	Feature Supported by the Permission
	<p>Note: To use a lookup service, users must have the Foundation Hub Assign the LookUp Service permission.</p>
RunProtocol	<p>Run Pipeline Pilot Client protocols.</p> <p>Note: The Analyze tab in Workbook experiments as well as the Remove Salts function in parallel chemistry Enumerated Products sections run Pipeline Pilot protocols. Consequently, users who require these functions must be in a group or role that has the RunProtocol permission.</p>
‡SearchExtensions	<p>Access non-native custom search functionality developed for your Workbook installation.</p> <p>For information about developing and configuring such extensions, see "Search Extensions" in the <i>Workbook SDK Developers Guide</i>.</p>
Administrator	<p>Access and use the Vault Administration Console, which is required not only to configure Workbook application permissions, but also to administer Vault Server repository subscriptions and folders, signature policies, and workflows.</p>
Folder.Administrator	<p>Add repository folders and move folders and their objects by using the Workbook client application's Notebook Explorer.</p> <p>Note: To <i>add</i> a folder, users must be a member of a group that has this permission <i>and</i> be a member of the Vault Global Administrators group. For requirements for <i>moving</i> vault folders and objects, see Requirements for Moving a Vault Folder or Object on page 25.</p>
GroupProfile.Administrator	<p>Administer group profiles by using the Workbook client application's Tools menu. For more information, see the Workbook client help.</p>
Forms.Editor	<p>Create and edit forms.</p> <p>Note: The create and edit permissions for <i>a specific</i> form or other Workbook object is controlled directly from the Workbook client application's Notebook Explorer. For more information, see the Workbook client help.</p>
Operation.Editor	<p>Create and edit operations to use in Structured Recipe sections. For more information, see the Workbook client help.</p>
Recipe.Editor	<p>Create, edit, and clone Structured Recipe sections.</p>

Vault Application Permission	Feature Supported by the Permission
	<p>Note: To edit recipes that are document <i>templates</i>, users also be a member of a group that has the <code>PropertySetEditor</code> and <code>TemplateEditor</code> permissions. (No special permissions are needed to execute recipes.)</p>
Report.Editor	Create, edit, and save report templates.
Template.Editor	<p>From Notebook Explorer, generate usage reports for a selection of templates in a managed source repository to identify which content objects in the repository use the selected templates.</p> <p>Note: To be identified in a template usage report, the content objects must be visible to the user who is running the report. Consequently,</p> <ul style="list-style-type: none"> ■ The objects must not be new, read-only, hidden, or system. ■ The user who runs the report must have either or both of the following permissions, as well as the <code>Template.Editor</code> permission: <code>Read Data</code>, <code>ReadProperties</code>.
TemplateManagementTools	<p>Generate template usage reports (see previous row) <i>and</i> add, remove, and replace template sections.</p> <ul style="list-style-type: none"> ■ To add, remove, and replace template sections identified in a usage report, the following additional permissions are required: <code>Checkout</code>, <code>UndoCheckout</code>, <code>ReadData</code>, <code>WriteData</code>, and <code>UpdateProperties</code>. ■ To update forms identified in a usage report, this additional permission is required: <code>Symyx.Notebook.Forms.Editor</code>. <p>The content identified in a template usage report varies based on the permissions of the user generating the report. Users with more permissions can get more comprehensive reports that users with fewer permissions.</p>
†TaskPlanSection	Create a Task Plan section and use Task Planner within Workbook; access the Foundation Hub landing page that shows your task plans.
SectionReportPipelinePilotProtocol Path	Execute the Pipeline Pilot protocol that creates a Section Report.
TransferTemplate	<p>Export and import templates by using Notebook Explorer menu options.</p> <ul style="list-style-type: none"> ■ For information about using Notebook Explorer to transfer templates, see the BIOVIA Workbook online help.

Vault Application Permission	Feature Supported by the Permission
	<ul style="list-style-type: none"> For information about using export and import functionality from the API, see the SDK API Reference > ImportExportVaultObjects interface in the Symyx.Notebook.ImportExport namespace.
SectionTemplate.Editor	Edit, move, save, save as, and browse section templates.
RunAnalysis	<p>Access the Analyze tab in the Workbook client application's Notebook Explorer and run Pipeline Pilot Client protocols.</p> <p>Note: To run a protocol, users must also be a member of a group that has the RunProtocol permission.</p>
‡DisableUndoMyCheckouts	<p>Prevent Workbook users from discarding their <i>own</i> checkouts. If you need to prevent such discards, you must manually add this permission to the system by using the Vault Administration Console and then apply it to appropriate groups by using Foundation Hub. To add the permission to the system, see Implementing the DisableUndoMyCheckouts Permission on page 24. To assign it to groups, see the <i>Foundation Hub Administration Guide</i>.</p> <p>Note: This permission has no impact on the separate UndoCheckout permission. Administrators are typically granted UndoCheckout so that they can undo the checkouts of <i>other</i> users' experiments, such as users who have left the company.</p>
UndoCheckout	<p>Undo <i>another</i> user's checkout. The Undo Checkout action results in the loss of any work that has not been checked in. Give this permission only to qualified administrators.</p>
Widget.Administrator	<p>Use the following Workbook client application features:</p> <ul style="list-style-type: none"> Manage Widgets button, which creates subfolders under the Site Repository Widgets folder. Publish icon, which supports publishing widgets and deleting published widgets. <p>For more information, see Form Editor Scripting for Validation, Updating, or Display in the <i>BIOVIA Workbook Administration Guide</i>.</p>
Workflow.Administrator	<p>Move checked in documents to <i>any</i> stage in the workflow process by using the Workbook client application's Administrative Move feature.</p> <p>Note: An administrative move can also be done using the Vault Administration Console, but this requires a separate administrator permission.</p>

Vault Application Permission	Feature Supported by the Permission
NotebookConfiguration	<p>Enables users who have the <code>writeData</code> permission but do <i>not</i> have the <code>workflowTransition</code> permission to edit and check in documents that are in a workflow.</p> <p>By default, users must have the <code>workflowTransition</code> permission to edit and check in workflow documents. To override this behavior, you can use the Vault Administration Console to change the NotebookConfiguration permission's <code>RequireTransitionPermissionToEditDocumentsInWorkflow</code> property from its default of <code>true</code> to <code>false</code>.</p>
PropertySetEditor	<p>Use the Property Set Editor in the Workbook client application to create or update property set definitions, which control the columns and fields that are displayed in table and form sections. Creating and updating property set definitions also requires the following:</p> <ul style="list-style-type: none"> ■ The <code>Vault/Administrator</code> application permission. ■ <code>Read</code>, <code>Update</code>, and <code>Write</code> permissions to the Site repository's <code>PropertySetDefinition</code> folder.
SymyxRegistration	Not used.
NotebookExplorerContextualViewerConfiguration	Internal use only. Do not use.

Using the Import/Export Application Permission Utility

You can use the Import/Export Application Permission Utility to review, edit, and copy application permissions between Vault Server environments. This command-line utility exports all editable application permission sets into a single XML file. For certain complex permission configurations, it generates a related text file that is referenced by the XML file. You can easily review and modify the XML and related text files as needed, and then import them back into the same or into a different Vault Server.

Reviewing and editing permissions in a single exported XML file and related text files is much easier than using the Vault Administration Console, which shows only one permission configuration at a time. You can also create scripted applications that use the utility for mass updates.

How the Utility Works

When you run the export command, the utility creates a file named `AppPermissions.xml` that identifies all relevant application permissions and their configurations.

If a permission's configuration is difficult to represent in the XML file (for example, if the configuration contains special characters or long data), the utility stores that configuration in a separate numbered text file such as `permissionConfiguration1.txt`, `permissionConfiguration2.txt`, and so on. The `AppPermissions.xml` file uses an `externalFile` property to reference such external text files. Representing complex configurations in separate text files simplifies the `AppPermissions.xml` file and editing of complex permission configurations. For an example `AppPermissions.xml` file and

some corresponding `permissionConfiguration` text files referenced by the xml file, see Example Exported Application Permission Files.

The utility is an add/update utility. It does not delete permissions or configurations. When you import the exported files into a Vault Server, the permission set for each application that has not yet been configured on that Vault Server are added, and the permission set for each application that has already been configured is overwritten to match the imported version.

Notes:

The utility (`ImportExportAppPermissionsUtility.exe`) and its configuration file (`ImportExportAppPermissionsUtility.exe.config`) are installed by default in `C:\Program Files (x86)\Accelrys\Vault\Utilities`.

The utility does not export application permissions that are not editable, such as global application permissions, but it does log them if you set `<level value="DEBUG" />` in the `<logger name="BIOVIA">` section of the utility's `.config` file. Other options are INFO, WARNING, and ERROR. DEBUG shows everything, INFO shows errors, warnings, and info messages, WARNING shows errors and warnings, and ERROR shows only errors. To avoid excessive logging, the logging level should normally be left at its default setting (**ERROR**).

Running the Export Command

To execute the Export command:

1. On the Vault Server whose applications permissions are to be exported, open a command-line prompt and change directory to the Vault Utilities directory, by default `C:\Program Files (x86)\Accelrys\Vault\Utilities`.

2. Enter the following command:

```
\ImportExportAppPermissionsUtility.exe <server> <domain>\<administrator-user> <password> export <exportdirectory>
```

Where:

`<server>` is the fully-qualified domain name of the BIOVIA Vault Server from which the permissions are to be exported.

`<domain>\<administrator-user> <password>` is the domain, username, and password of that Vault Server's Vault Global Administrator account.

`<exportdirectory>` is the directory to which to export the files, for example `C:\temp`.

Note: If you run the command with no parameters, with invalid parameters, without a complete set of parameters, the utility displays a message that explains the required parameters.

For an example `AppPermissions.xml` file and some `PermissionConfiguration` text files referenced by the XML file, see [Example Application Permission Files](#) on page 19

Editing and Tokenizing the AppPermissions.xml File (Optional)

After export and prior to import, you can edit the values in the exported XML file to suit the requirements of the destination Vault Server.

If you need to import the exported application permissions into more than one Vault Server environment, such as a production server, development server, and QA server, you can tokenize the values of permission configurations such as URLs that vary between your destination servers. You can then create a separate tokens text file (or other readable type of file) that provides the appropriate

values for each server, and specify the appropriate tokens.txt file as an import parameter when you import the application permissions.

1. To tokenize a value in the AppPermissions.xml file or in a permissionConfiguration text file, enclose its name between tilde and curly brace characters. For example, to call a token **BASEURL**, enter ~{BASEURL}~.

Note: Token names must use only alphabetic characters. Numbers, spaces, and special characters are not supported.

2. Create a separate tokens.txt file (for example, devtokens.txt, qatokens.txt, and so) on that contains a complete set of token name/value pairs, such as the following:

```
BASEURL:http://base-server.testdomain
CIMSURL:http://cims-url.testdomain
DISCOVERYGATE:http://discovery-gate.testdomain
PIPELINEPILOTURL:http://pipeline-pilot.testdomain
```

You can name the token files whatever you like and store them in the same folder as the rest of the files or in another accessible folder.

Running the Import Command

When you run the Import command, the utility reads and validates the exported XML and any permissionConfiguration text files. If validation passes, the utility imports the values into the Vault server. If validation fails, the utility halts and reports the errors.

To import the permissions:

1. On the Vault Server into which the exported application permissions are to be imported, open a command-line prompt and change directory to the Vault Utilities directory, by default C:\Program Files (x86)\Accelrys\Vault\Utilities.
2. Execute the following command:

```
\ImportExportAppPermissionsUtility.exe <server> <domain>\<administrator-
user> <password> import <importdirectory> <C:\temp2\tokens.txt>
```

Where:

- <servername> is the fully-qualified domain name of the BIOVIA Vault Server into which the permissions are to be imported.
- <domain>\<administrator-user> <password> is the domain, username, and password of the Vault Global Administrator account for that Vault Server
- <importdirectory> is the directory in which the exported permission files reside, for example C:\temp
- <C:\temp2\tokens.txt> is the name of a tokens file, if used.

The utility connects to the source Vault Server and imports the permissions described in C:\temp\AppPermissions.xml, its permissionConfiguration text files (if any), and the specified tokens.txt file (if any).

Note: If a file contains an incomplete or erroneous permission set for any application, the import process halts and displays an error message. In this case, you must correct the files, and then re-run the import process. The utility does not commit added or updated permission sets from the file unless they are all complete and valid.

Example Application Permission Files

The content of the `AppPermissions.xml` file varies based on which applications you use with Vault Server. In the following example, some name/value pairs specify a configuration value, some use the "externalFile" property to identify a permissionConfiguration text file from which to retrieve the configuration, and others were tokenized after export so that an external environment-specific token file can be specified when the import command is executed. Example permissionConfiguration files and an example `tokens.txt` file follow the example `appPermissions.xml` file.

Example AppPermissions.xml

```
Permissions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" metadata="Exported from lp5-rds8-
dsa.dsone.3ds.com by scitegic user on 01/30/2018 14:30:34">
  <Permission Application="Compose" Name="RecipeUser">
    <config name="ComposeBaseAddress">~{BASEURL}~:9964/</config>
    <config name="CaptureBaseAddress">~{BASEURL}~:9964/</config>
    <config name="DesignBaseAddress">~{BASEURL}~:9964/</config>
    <config name="workbookToFoundationMap" externalFile="permissionConfiguration13.txt"
  />
    <config name="workflowStagesForLocking" />
    <config name="NewTestEntry" />
  </Permission>
  <Permission Application="ExternalDataConversionService" Name="External Data Conversion
Service">
    <config name="ExternalDataConversionService"
externalFile="permissionConfiguration4.txt" />
    <config name="LJ-AXP" externalFile="permissionConfiguration5.txt" />
    <config name="MX-AXP" externalFile="permissionConfiguration6.txt" />
    <config name="MX-AXF" externalFile="permissionConfiguration7.txt" />
  </Permission>
  <Permission Application="LookupService" Name="MaterialInfoManager">
    <config name="ResolverXML" externalFile="permissionConfiguration8.txt" />
    <config name="Symyx.Framework.MaterialInfoLookup.OpenEye.OpenEyeResolver"
externalFile="permissionConfiguration9.txt" />
    <config
name="Symyx.Notebook.DiscoveryGateMaterialInfoLookup.RefDataWebServices.DiscoveryGateWeb
ServiceResolverwithMolName" externalFile="permissionConfiguration10.txt" />
    <config name="Accelrys.Notebook.CimsMaterialInfoLookup.CimsMaterialInfoResolver"
externalFile="permissionConfiguration11.txt" />
    <config
name="Biovia.Notebook.CisProMaterialInfoLookup.CisProInventoryMaterialInfoResolver"
externalFile="permissionConfiguration12.txt" />
  </Permission>
  <Permission Application="NotebookExplorerContextualViewerConfiguration"
Name="NotebookExplorerContextualViewerConfiguration">
    <config name="ObjectTypesToViewerTypes" externalFile="permissionConfiguration1.txt"
  />
  </Permission>
  <Permission Application="PipelinePilot" Name="RunProtocol">
    <config name="Endpoint">~{PIPELINEPILOTURL}~:9943/</config>
    <config name="ProtocolRoot">Protocols/Web Services/workbook/Experiment</config>
    <config name="AnalysisProtocolRoot">Protocols/Web
Services/workbook/Analysis</config>
  </Permission>
  <Permission Application="RegistrationService" Name="SymyxRegistration">
    <config name="ServiceName">MDLRegistration</config>
    <config name="ApplicationName">ChemBioAE</config>
    <config name="StructurePath">Substance/Compound</config>
    <config name="PollingInterval" />
  </Permission>
</Permissions>
```

```
<config name="IsentrisUrl" />
  <config name="DefaultUser" />
  <config name="DefaultPassword" />
</Permission>
<Permission Application="SearchExtensionsService" Name="SearchExtensions">
  <config name="SearchExtensionTypes" externalFile="permissionConfiguration2.txt" />
  <config name="example" externalFile="permissionConfiguration3.txt" />
</Permission>
<Permission Application="Symyx.Notebook" Name="Folder.Administrator" />
<Permission Application="Symyx.Notebook" Name="GroupProfile.Administrator" />
<Permission Application="Symyx.Notebook" Name="Logon" />
<Permission Application="Symyx.Notebook" Name="Operation.Editor" />
<Permission Application="Symyx.Notebook" Name="Recipe.Editor" />
<Permission Application="Symyx.Notebook" Name="Report.Editor" />
<Permission Application="Symyx.Notebook" Name="RunAnalysis" />
<Permission Application="Symyx.Notebook" Name="TaskPlanSection">
  <config name="workflowStagesForLocking" />
</Permission>
<Permission Application="Symyx.Notebook" Name="TemplateManagementTools">
  <config name="TestAdd">Some value</config>
</Permission>
<Permission Application="Symyx.Notebook" Name="widget.Administrator" />
<Permission Application="Symyx.Notebook" Name="workflow.Administrator" />
<Permission Application="System" Name="NotebookConfiguration">
  <config name="RequireTransitionPermissionToEditDocumentsInWorkflow">false</config>
</Permission>
<Permission Application="TransferTemplatesService" Name="TransferTemplates">
  <config
name="ImportExportImplementation">Symyx.Notebook.ImportExport.ImportExportVaultObjects,
Symyx.Notebook.VaultObjectExport, Version=18.1.100.0, Culture=neutral,
PublicKeyToken=fb4b5791c48b7e8a</config>
  </Permission>
</Permissions>
```

Example permissionConfiguration Files

Example 1 – XML Only

```
<?xml version="1.0" encoding="utf-16"?>
<ExternalDataConversionService>
  <SourceDataType name="LJ-AXP">
    <aka>LJ-AXP v5.1</aka>
    <aka>LJ-AXP v5.2.3</aka>
  </SourceDataType>
  <SourceDataType name="MX-AXP">
    <aka>MX-AXP v5.1</aka>
    <aka>MX-AXP v5.2.3</aka>
  </SourceDataType>
  <SourceDataType name="MX-AXF">
    <aka>MX-AXF v5.1</aka>
    <aka>MX-AXF v5.2.3</aka>
  </SourceDataType>
</ExternalDataConversionService>
```

Example 2 – XML with MOL Script

```
<?xml version="1.0" encoding="utf-16"?>
<SourceDataType name="LJ-AXP">
  <Template vaultpath="Site\ExternalDocumentConversionTemplates67\LabJournal
Experiment" />
  <SectionMappings>
    <Converter
```

```

class="Symyx.Notebook.LegacyDocumentConversion2.LegacyFormConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Background
Information</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="False"
isDynamicSection="False">Background Information</Section>
    </OutputSections>
  </ConverterConfiguration>
</Converter>
</Converter>
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyTextConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Background
Description</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="False"
isDynamicSection="False">Background Description</Section>
    </OutputSections>
  </ConverterConfiguration>
</Converter>
</Converter>
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyTextConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Procedure</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="False" isDynamicSection="False">Record
Procedure</Section>
    </OutputSections>
  </ConverterConfiguration>
</Converter>
</Converter>
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyTextConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Results and
Conclusions</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="False" isDynamicSection="False">Results

```

```
and Conclusions</Section>
  </OutputSections>
</ConverterConfiguration>
</Converter>
<Converter
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyExternalFileConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Embedded Reports</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="True" isDynamicSection="False">Embedded
Reports</Section>
    </OutputSections>
  </ConverterConfiguration>
</Converter>
<Converter
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyExternalFileConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Material
Characterizations</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="True" isDynamicSection="False">Material
Characterizations</Section>
    </OutputSections>
  </ConverterConfiguration>
</Converter>
<Converter
class="Symyx.Notebook.LegacyDocumentConversion2.LegacyMaterialListConverter,
Symyx.Notebook.LegacyDocumentConversion2, Version=18.1.100.2931,
Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a">
  <ConverterConfiguration>
    <InputDataSections>
      <IntermediaryDataSection>Materials</IntermediaryDataSection>
    </InputDataSections>
    <OutputSections>
      <Section insertIfNotFound="True">Material Amounts</Section>
    </OutputSections>
    <SpecificToConverter>
      <Mappings>
        <Map column="MaterialName" property="Material.Name" />
        <Map column="MolecularFormula" property="Material.MF" />
        <Map column="MolFile" property="Material.Structure" />
        <Map column="Role" property="Material.Role" />
        <Map column="StoichiometricCoefficient"
property="ReactionMaterial.StoichiometricCoefficient" />
        <Map column="Purity"

```

```

property="ReactionMaterial.PurityConcentration" />
    <Map column="Density" property="Material.Density" />
    <Map column="Comments" property="Material.Comments" />
    <Map column="PlannedAmount" property="PlannedAmount.Amount" />
    <Map column="ActualAmount" property="ActualAmount.Amount" />
    <Map column="LimitingReagent"
property="ReactionMaterial.LimitingReagent" />
    <Map column="MolecularWeight" property="Material.MW" />
    <Map column="CAS" property="Material.CASNumber" />
    <Map column="IsActualAmountPure"
property="ReactionMaterial.AdjForPureLR_SYS" />
    </Mappings>
    <MolScripts>
        <InitialMolScript>SpecialHConverter.SetHPlusUnattachedType('H+');
// convert unattached H+'s to the 'H+' (uncharged) pseudoatom
// NOTE: requires H+ in ptable

SpecialHConverter.SetH2Type('H2'); // convert H-H fragments to the 'H2'
pseudoatom
// NOTE: requires H2 in ptable
</InitialMolScript>
        <CleanMolScript>
// fix errant (H0) query values
Find(A_QHCOUNT,A_QHCOUNT_ZERO).Set(A_QHCOUNT,A_QHCOUNT_OFF);

// convert any problematic H atoms
SpecialHConverter.Convert ( UNDEFINED ) ; // convert the Target structure
</CleanMolScript>
    </MolScripts>
</SpecificToConverter>
</ConverterConfiguration>
</Converter>
</SectionMappings>
</SourceDataType>

```

Sample Application

You can also write standalone applications that use the utility's assemblies. The most relevant assembly is "BIOVIA.ImportExportAppPermissions.dll." You can create separate applications for export and import, or one application that does both.

The general steps for writing such a program are:

1. Define any variables needed for processing, such as Vault server address, username, password, and so on.
2. Authenticate into Vault.
3. Create a method to handle the Export function.
 - a. Create an instance of the PermissionExporter class.
 - b. Call Export on the instance of the class.
 - c. Handle any errors that came back from the Export process.

Example:

Expand source

```
var exporter = PermissionExporter.GetInstance();
```

```
var statusExport = exporter.Export(directory, fileName);
if (!statusExport.Success)
{
    throw new Exception("Export was not successful: " +
        statusExport.StatusMessage, new Exception(statusExport.ExceptionText));
}
```

4. Create a method to handle the Import function:
 - a. Create an instance of the PermissionImporter class.
 - b. Call Import on the instance of the class.
 - c. Handle any errors that come back from the Import process.

Example:

Expand source

```
var importer = PermissionImporter.GetInstance();
var statusImport = importer.Import(exportFile, directory, tokenFile);
if (!statusImport.Success)
{
    throw new Exception(string.Format(CultureInfo.InvariantCulture,
        "Export was not successful: {0}", statusImport.StatusMessage));
}
```

5. Confirm the results by logging in to the Vault Administration Console and viewing the added and updated permissions.

If desired, you can chain functions together into a single application that performs the following main steps:

1. Authenticate into Vault.
2. Export from Vault.
3. Modify the exported files if needed. For example, update data, add token keys, or read values from token files.
4. Import into Vault.
5. Validate exit state and/or confirm changes.
6. To repeat for another Vault, authenticate on that Vault, and then repeat step 1 and steps 3-5.

Implementing the DisableUndoMyCheckouts Permission

Tip: To use a command-line utility to add or update permission configurations for services and applications, see [Using the Import/Export Application Permission Utility](#). The command-line utility enables you to easily export all editable application permissions, edit them using a text editor, tokenize them for use on more than one Vault Server if needed, and then re-import them into the same or a different Vault Server.

To implement the `DisableUndoMyCheckouts` permission, which prevents users from undoing their own checkouts, add the permission using the Vault Administration Console, and then use Foundation Hub to apply it to the user groups to whom it should apply:

1. Open the Vault Administration Console and log on to Vault Server as a member of the Vault Global Administrators group.
2. Expand the Vault Server node and select **Application Permissions**.
3. In the **Actions** pane, click **Add**.

4. In the **New Permission** dialog, in the **Application** field, type *Symyx.Notebook*, in the **Permission** field, type `DisableUndoMyCheckouts`, and then click **OK**.
5. Use Foundation Hub to apply the permission to groups whose members are not allowed to undo their own checkouts:
 - a. Log on to Foundation Hub and access **Admin and Settings > Security > Groups**.
 - b. Open the group whose checkouts you need to disable and select **Edit**.
 - c. From the **Permissions** list, select **DisableUndoMyCheckouts** and click **Assign**.

Note: You can delete this permission from both the Vault Administration Console and Foundation Hub by clicking the **Delete** button in the Actions pane of the Vault Administration Console.

Requirements for Moving a Vault Folder or Object

To move a folder:

- The folder must be checked in to a repository with version control.
- The user must belong to a group that has the `Folder.Administrator` permission.

To move an object:

- The object must be checked in.
- The following object permissions must be Allowed:
 - Read Data
 - Read Properties
 - Update Properties
 - Write data

Pipeline Pilot Configuration Keys

To run Pipeline Pilot Client, users must have the `PipelinePilot.RunProtocol` application permission. You must configure settings in to enable BIOVIA Workbook and Pipeline Pilot Client to work together. The configuration keys, you must set are:

- `Endpoint`
Endpoint is the fully qualified path to the Pipeline Pilot Server that BIOVIA Workbook should use the fully-qualified domain name, `https://<fully-qualified domain to the Pipeline Pilot Server>:9943`.
- `ProtocolRoot`
The `ProtocolRoot` key sets the Pipeline Pilot folder that displays in the Template Editor when setting up toolbar buttons that access Pipeline Pilot Client protocols. The default value is `Protocols/web_services/workbook/Experiment`.
- `AnalysisProtocolRoot`
The `AnalysisProtocolRoot` key sets the Pipeline Pilot Client folder that displays in the Notebook Explorer Analyze tab. The default value is `Protocols/web_services/workbook/Analysis`.

To configure the settings for users with permission to run Pipeline Pilot Client protocols, see [Configure Pipeline Pilot RunProtocol Settings](#).

Configuring Pipeline Pilot RunProtocol Settings (New Installs Only)

To enable Workbook users to run standard and customer-specific Pipeline Pilot protocols, you must configure the RunProtocol application permission.

Tip: To use a command-line utility to add or update permission configurations for services and applications, see [Using the Import/Export Application Permission Utility](#). The command-line utility enables you to easily export all editable application permissions, edit them using a text editor, tokenize them for use on more than one Vault Server if needed, and then re-import them into the same or a different Vault Server.

To use the Vault Administration Console to configure the Pipeline Pilot Client RunProtocol settings, perform the following steps:

1. Open the Vault Administration Console and log on to Vault Server as a member of the Vault Global Administrators group.
2. Expand the Vault Server node and select **Application Permissions**.
3. In **Application Permissions**, select the PipelinePilot.RunProtocol permission, and click **Properties**.
4. In the **PipelinePilot.RunProtocol** dialog, select the **Configuration** tab.
5. For the **Value** of the **Endpoint**, type the HTTPS URL for the fully-qualified domain name for the Pipeline Pilot server, for example: `https://<fully-qualified PipelinePilot_server_name>:9943`.

Compound Registration

BIOVIA Workbook provides user-initiated compound registration from the Synthetic Chemistry section. Users can register one or more compounds per request. The process returns the SubstanceID and BatchID for each compound and displays the values in the experiment's Synthetic Chemistry section.

Compound registration is implemented as a Symyx Framework service. BIOVIA Workbook does not require configuration to support Registration. To implement an alternative registration system, contact Dassault Systèmes Customer Support.

To use Registration features from Workbook:

- BIOVIA Isentris 4.x or higher must be installed and running on a server on your network.
- Registration service 1.5 SP4 or higher must be installed and running on the same server as Isentris.
- BIOVIA Isentris 4.x or higher application framework must be installed each Workbook client computer.

BIOVIA Workbook and BIOVIA Vault Server support the versions of BIOVIA Isentris and BIOVIA Registration identified in *Workbook 2021 Client System Requirements*. For information about installing and administering Isentris and Registration applications, see the *Isentris Installation and Configuration Guide*, the *BIOVIA Isentris Administration Guide*, and the *BIOVIA Registration Installation and Administration Guide*.

Modify Registration Service Properties

Tip: To use a command-line utility to add or update permission configurations for services and applications, see [Using the Import/Export Application Permission Utility](#). The command-line utility enables you to easily export all editable application permissions, edit them using a text editor, tokenize them for use on more than one Vault Server if needed, and then re-import them into the same or a different Vault Server.

To use the Vault Administration Console to modify Registration Service properties, perform the following steps:

1. Open the Vault Administration Console and log on to Vault Server as a member of the Vault Global Administrators group.
2. Expand the Vault Server node and select **Application Permissions**.
3. In the **Application Permissions** pane, and double-click **RegistrationService**.
4. In the **Registration Properties** dialog, select the **Configuration** tab.
5. Select the property to modify, enter your changes, and click **OK**.

See also:

[Registration Service Permission Parameters](#)

Registration Service Permission Parameters

Only the ServiceName, ApplicationName, StructurePath, and PollingInterval configuration parameters are used in the current release. The other parameters are reserved for possible future use. For more information, see Configuration parameters of the RegistrationService permission.

Parameter	Description
ServiceName	Specifies the name of the Isentris registration service. You only need to change this if your Isentris administrator changes it from the default of MDLRegistration.
ApplicationName	Specifies the application name of the registration service. Previous releases of BIOVIA Registration offered two configurations: Default and ChemBioAE. Only ChemBioAE is supported with BIOVIA Registration 1.5 SP3; the version supported for use with BIOVIA Workbook 2021. Customers can create their own configurations. The ApplicationName parameter enables adapting the notebook implementation to user needs.

Parameter	Description
StructurePath	<p>The registration service uses a hierarchical object model to package registration requests. The root of the hierarchy is the Batch object. Depending on the service configuration, the location of the primary structure to register in the object hierarchy can vary. This parameter allows you to specify the mapping from the structure property in the materials section to the registration object tree.</p> <p>The structure path string implicitly begins at the Batch object and each element in the path is separated by a forward slash character (/).</p> <p>The ChemBioAE configuration requires the structure to appear as a property of the Batch/Substance/Compound object. Therefore, the default StructurePath configuration is Substance/Compound. Because the root object, Batch, is implied it does not appear in the configuration string.</p> <p>The older Default application profile required the structure to appear as a property of Batch/Substance/Fragment/Fragment_Molecule/RegMolecule. If you are using the older Default configuration, you need to set StructurePath to Batch/Substance/Fragment/Fragment_Molecule/RegMolecule.</p> <p>If you have a custom profile, then you must provide a custom structure path.</p>
PollingInterval	<p>The registration service does not send a notification when an operation completes. BIOVIA Workbook polls at regular intervals to check for a completed operation.</p> <p>This parameter is the interval in milliseconds between polls. If the property is empty, the default of 250 milliseconds is used.</p> <p>You can adjust this value to balance application responsiveness against the load on the server.</p>

Database Web Service and Material Property Lookup Service

You can auto-populate the property data for materials in the Synthetic Chemistry section in the Vault Administration Console. For example, entering a compound name or structure into the Synthetic Chemistry section can result in the density, CAS Registry Number, and other data for the material being displayed automatically. The material property lookup is implemented as a Framework service.

BIOVIA Workbook supports the OpenEye chemical name to structure converter, and supports the Database Web service that provides access to BIOVIA-hosted content sources such as the Available Chemicals Directory (ACD).

To add alternative or additional resolvers to the Material Lookup service, and for more about licensing and configuring the Database Web Service, contact Dassault Systèmes Customer Support.

For more information, see "Database Web Service" in the *BIOVIA Vault Server Installation Guide*.

Locking Recipe Sections at Specific Workflow Stages

You can lock Recipes sections in experiments to prevent them from being edited when they reach configurable, specific stages in a workflow definition. Identify which stages must be locked in the workflowStagesForLocking property.

To identify recipe stages to lock:

1. In the Vault Administration Console, open the **Recipe Users** application permission.
2. Set the value of the property whose name is `workFlowStagesForLocking` to a comma-separated list of stages. Example:

Approved, Abandoned, UnderReview

Note: If the `workFlowStagesForLocking` row is not listed, add it manually.

Locking the Task Plan at Specific Workflow Stages

You can use Workflow Designer to configure workflows that "lock" task plans when they reach certain stages (typically *released* or *abandoned*) to prevent further changes to task plans that have reached those stages. Foundation Hub marks a task plan that has reached such a stage as read-only. (Do not confuse this type of workflow-based "locking" with the locking that a user can apply to a Workbook section.)

To identify these read-only stages, you use the Vault Administration Console.

To identify task plan stages to lock:

1. In the Vault Administration Console, open the **Symyx.Notebook|TaskPlanSection** application permission.
2. Click the **Configuration** tab.
3. Set the value of `workFlowStagesForLocking` to a comma-separated list of stages. Example:

Approved, Abandoned, UnderReview

Note: If the `workFlowStagesForLocking` row is not listed, add it manually.

4. **(Important)** If your Vault Server and Foundation Hub are installed on *separate* computers, edit the following configuration files to circumvent a defect that prevents Foundation Hub from receiving Workbook messages that instruct it to treat a task plan as read-only.

- `windowsServices\Accelrys.Vault.IsolationChamber.exe.config`
- `windowsServices\Symyx.Vault.Message.Processing.Service.exe.config`

- a. Navigate to `<install_path>\BIOVIA\Vault\Utilities` and open the following file, which already has the correct authentication section, in a text editor:

`Symyx.Vault.Database.Utility.exe.config`

- b. Select and copy (**Ctrl+C**) the authentication-related lines to the clipboard. Example:

```
<section name "authentication"
type="accelrys.AEP.Authentication.Configuration, ... >
</configSections>
<authentication defaultAuthenticationProvideForSigning="AEP">
  <uri>http://your-server:9954</uri>
  <cache-time>00:00:30</cache-time>
</authentication>
```

- c. Navigate to `<install_path>\windowsServices\MessageHandlers\VaultQueue` and open the following file in a text editor:

`Accelrys.Vault.IsolationChamber.exe.config`

- d. Select the following line in the file and replace it (**Ctrl+V**) with clipboard content shown under

Step 4b, then save and close the file:

```
</configSections>
```

- e. Open the `Symyx.Vault.Message.Processing.Service.exe.config` and repeat Step 4d.

External Structures Conversion

BIOVIA Vault Server cannot index the documents for searching in structures and documents with hidden attributes such as those found in structures available in ChemSeek and in version 5 documents. When you run the External Conversion Service, all structures are automatically modified with a Cheshire script to mitigate the search problem. The structure cleaning is automatic. BIOVIA does not recommend modifying the Cheshire script available in the configuration, as shown below:

```
<MolScripts>

  <InitialMolScript>SpecialHConverter.SetHPlusUnattachedType('H+');

    // convert unattached Ha's to the 'H+' (uncharged) pseudoatom

    // requires H+ in ptable

  SpecialHConverter.SetH2Type('H2');
  // convert H-H fragments to the 'H2' pseudoatom

  // Requires H2 in ptable
</InitialMolScript>

  <CleanMolScript>

    // fix errant (H0) query values

  Find(A_QHCOUNT,A_QHCOUNT_ZERO).Set(A_QHCOUNT,A_QHCOUNT_OFF);

  // convert any problematic H atoms

  SpecialHConverter.Convert (UNDEFINED);

  // convert the Target structure

</CleanMolScript>
</MolScripts>
```

The `InitialMolScript` is run once against the Cheshire engine. The `CleanMolScript` is run for each molecule target.

See also

[Configure the External Data Conversion Service](#)

Configure the External Data Conversion Service

BIOVIA Vault Server installs a set of templates that enable the user to convert data from Symyx Process Notebook version 5.6 and Symyx Formulations Notebook version 5.6 into the templates. These templates are in the `ExternalDataConversionTemplates.voexp` file. BIOVIA recommends that

you publish the voexp file to a managed repository instead of to the Site Repository so that you can access the templates to make necessary edits in the BIOVIA Workbook client.

Tip: To use a command-line utility to add or update permission configurations for services and applications, see [Using the Import/Export Application Permission Utility](#). The command-line utility enables you to easily export all editable application permissions, edit them using a text editor, tokenize them for use on more than one Vault Server if needed, and then re-import them into the same or a different Vault Server.

To use the Vault Administration Console to configure the external data conversion service, perform the following steps:

1. Open the Vault Administration Console and log on to Vault Server as a member of the Vault Global Administrators group.
2. Expand the Vault Server node and select **Application Permissions**.
3. In **Application Permissions**, double-click **ExternalDataConversionService**.
4. In the **ExternalConversionService | External Conversion Service Properties** dialog, click the **Configuration** tab.

The following table lists the converter file names.

Name	Description
LJ-AXP	Lab Journal and Sketch converter
MX-AXP	Matrix (Process Notebook) converter
MX-AXF	Matrix (Formulations Notebook) converter

File names in Matrix v5 are the output section names for MX-AXP and MX-AXF converters. If files are not assigned names, the application uses the output section name assigned in Matrix Process Converter or the Matrix Formulations converter.

The Formulation matrix converter does not convert material amount information.

Specify Document Conversion Template

Modify the template vaultpath attribute of any converter to change the attribute on initial setup of the converters, after publishing the ExternalDataConversionTemplates.vozip file.

If you are converting two document templates with the same template vaultpath, the first template is used.

To specify the document conversion template:

1. Select the appropriate converter, for example, use LJ-AXP for Lab Journal documents.
2. Copy the configuration file into a text editor, and then click OK to close the Configuration panel.
3. In the text editor, enter the vaultpath to the template you want to use in the following section.

```
<?xml version="1.0" encoding="utf-16"?>
  <SourceDataType name="LJ-AXP">
    <Template
      vaultpath="Site\ExternalDocumentConversionTemplate65\LabJournal
      Experiment />
```

4. Save the file.

5. Copy the modified XML file.
6. In the Vault Administration Console
7. Click the down arrow at the end of the LJ-AXP row.
8. Select all of the text that displays.
9. Right-click and select Delete.
10. Right-click and select Paste to enter the file you copied in step 2.
11. Click OK and exit the Console.

Modify Section Settings in Document Conversion

You can change the sections that external data is converted into by modifying the converter XML. The following example applies to the Material Amounts section.

Replace the Materials Characterizations name with a name of your choosing in the following section. The variable is shown in italics.

```
</InputDataSections>
  <IntermediaryDataSection>Materials</IntermediaryDataSection>
</InputDataSections>
<OutputSections>
  <Section insertIfNotFound="True">Materials Amounts</Section>
</OutputSections>
```

You can change the `insertIfNotFound` attribute in the Materials section. When set to True and a section named *Materials Amounts* is not found in the document, a new Materials section is inserted into the document, renamed to Materials Amounts, and the data is inserted in the new section. When the `insertIfNotFound` attribute is set to False and it is not found, the conversion does not take place on that data section. However, other sections are converted.

For a Matrix document conversion, you can change the `isDynamic` attribute used when the source document might contain a number of sections of the same type. When the `isDynamic` attribute is set to True, additional sections of the same type in the source document prompt the creation of new sections in the new document. When the `isDynamic` attribute is set to False, additional sections of the same type overwrite the last converted section.

1. Save the file.
2. Copy the modified XML file.
3. In the Vault Administration Console return to the ExternalConversionService > External Conversion Service Properties > Configuration dialog:
4. Click the down arrow at the end of the LJ-AXP row.
5. Select all of the text that displays.
6. Right-click and select Delete.
7. Right-click and select Paste to enter the file you copied in step 2.
8. Click OK.

Chapter 5:

Developing Signature Policies

2021 provides administrators with the ability to design custom signing policies. The Vault Administrator creates custom signature policies in the Vault Administration Console. The policies are then referenced by name in the Workflow Designer, the Vault Administration Console console, and the notebook **Experiment Editor**. The available signature policies are listed in the application that is used to select the Signature Policy.

Design signature policies so that they are widely reused. You can create a signature policy for a specific type of signing action, but it is best practice to design them more generally. A signature policy is a set of rules for how to create a signature. For example, does your policy allow comments allowed, define how signatures are authenticated, or define how the logged in user provides the signature? With the exception of meanings and reasons, it is not possible to customize them extensively.

Signature Policy Events

Entering an incorrect password while attempting a Signature is recorded as a signature policy event. The user is prevented from performing any other actions when an incorrect password is used.

The following list contains events in which you might want to define in your signature policy:

- Rollback
- Administrative Move
- Check In
 - A check-in signature is in effect only when an experiment is not enrolled in workflow.
- Workflow transition
- Locking or unlocking sections
- Overriding a value in a section
- Entering or modifying values in a section
- Entering or confirming values of Property Data Set (PDS) properties
- Delete a section

In workflow transitions, signatures are associated with specific workflow actions, and the actor who performs the action must also provide the signature. The actor or user who performs check in, rollback, and administrative move events must also provide the required signatures.

Other client actions associated with signatures such as signing in forms, signing in tables, and signing for section locking or unlocking, have two properties. The two properties are the signature policy to use for the action, and an explicit list of users or groups that are allowed to perform the signature.

You cannot delete signature policies. You must enforce a signature on an administrative move by creating a signature policy with the name, *Administrative Move*, to make the administrative move option available in the client. Creating a signature policy with the rollback event is optional.

Create a Signature Policy

Before you create a signature policy, you should check the names of the existing policies so you only create a policy with a unique name.

If a signature policy does not exist, you cannot complete an administrative move, and the administrative move action fails without a warning. Ensure that a signature policy exists before proceeding with an administrative move.

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console.

To create a signature policy:

1. From the Start menu, select Vault Administration Console.
2. In the Console Root\Vault Administration\Vault Server, expand the Vault Server node, and [login](#).
3. Expand the **Signature Policies** node, and select a signature policy to update, and click **Properties**.
4. In the **Selected Item Properties** dialog, on the **Signature Policy Editor** tab, type a name for the signature policy in **Name**.
5. Define the reason or meanings for the signature policy. For more information, see [Set Meanings or Reasons](#) and [Signature Policy Properties Reference](#).
6. (Optional) In **Description**, type a description for the meaning or reason.
7. (Optional) In **Authentication Provider**, specify the code or tool used to validate the signature.
8. (Optional) In **Credential Input Control**, the user interface control used to obtain the user's credentials.
9. (Optional) In **Signature Provider**, specify code used to sign the request.
10. In **Allowed Signers**, select the allowed signators.
11. Click **Apply** and **OK**.

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

Set Meanings or Reasons

Signing meanings address the regulatory compliance requirements for experiments in some industries. An example of a signature meaning is *read and understood*, a common phrase for witnessing and action. You can set the signing meanings or reasons for change in a signature policy. A signature policy cannot have both signing meaning and reason.

Meanings

Signing meanings are used for document-level signatures such as those attached to workflow transitions. Reasons are used for electronic signatures such as those attached to form fields or table cells.

- Signing meanings are always a list from which the user makes a selection. The signer cannot create a signing meaning.
- If Signing meanings are present, they are required.

Reasons

Reasons address the digital signature, and represent the reason for the change such as to correct an error or re-measured a material. Reasons are selected from a list, or represent the user entered information as to the *reason* for the change.

If a reason list exists, the user must choose a reason from the list.

- When the reason list is empty, and reasons are allowed, the user can type in a reason.
- If Reasons are allowed, the **Required** check box determines whether the user must enter or select a reason.

To set meanings or reasons:

1. In the Vault Administration Console, expand the Vault server node.
2. Expand the **Signature Policies** node, and select a signature policy to update, and click **Properties**.
3. In the Selected Item Properties dialog, on the **Signature Policy Editor** tab, type a name for the signature policy in **Name**.
4. Click add (+) in the Meanings or Reason group, and type a meaning or reason to use with the signature such as an FDA policy number used by the team or other relevant phrase or code.
5. When defining reasons, select **Allowed** or **Required** as the property for the reason.
The default value is Allowed.
6. (Optional) Select the additional attributes to apply to the Meaning or Reason.
 - (Optional) In **Description**, type a description for the meaning or reason.
 - (Optional) In **Authentication Provider**, specify the code or tool used to validate the signature.
 - (Optional) In **Credential Input Control**, the user interface control used to obtain the user's credentials.
 - (Optional) In **Signature Provider**, specify code used to sign the request.
 For more information, see [Signature Policy Properties Reference](#).
7. In **Allowed Signers**, select the users allowed to add digital signatures.
8. Click **Apply** and **OK**.

Modify Signature Policies

Membership in the Vault Global Administrators group is required to modify properties using the Vault Administration Console.

To modify a signature policy:

1. From the Start menu, select Vault Administration Console.
2. In the Console Root\Vault Administration\Vault Server, expand the Vault Server node, and [login](#).
3. Expand the **Signature Policies** node, and select a signature policy to update, and click **Properties**.
4. In the **Selected Item Properties** dialog on the **Signature Policy Editor** tab, enter your changes and click **OK**.

Signature Policy Properties Reference

The following table describes the option that you can set while defining signature policies.

Option	Values	Description
Reasons	Allowed or Required	Default is Allowed. If Required, the user must provide a reason for the request.
Comments Allowed	Yes or No	Default is Yes. If checked, you can add comments to the signature policy request.

Option	Values	Description
Require Single Object Signing	Yes or No	Default is No. If Yes, the user must sign each document individually, no batch signing.
Require Open Object	Yes or No	Default is No. If Yes, the user must open the object to sign it.
Force User to Re-Authenticate	Yes or No	Default is No. If yes, the user must enter their Workbook account password at the time of signing.
Unrestricted	Yes or No	Default is Yes. Any valid Workbook user can sign. If No, select another Allow user options.
Require Current User	Yes or No	Default is No. If Yes, only the currently logged in user is allowed to sign.
Require Different User	Yes or No	Default is No. If Yes, the currently logged in user is not allowed to sign. Another valid Workbook user must sign. This different user must enter their user name and password on the computer of the user who triggered the signature policy.
Description	Optional	Allows entering a description of the signature policy request.
Authentication Provider	Default is Windows authentication	Specifies the code used to authenticate the signature request.
Credential Input Control	Default is username/password	Specifies the UI control that displays in the signature dialog to gather the credentials for the signature.
Signature Provider	Default is SHA-256	Specifies the code library used to sign the request.

Document Template Management Tools Signature Policy

You can define a signature policy for the Document Template Management tools that enable users with `TemplateManagementTools` permissions to update form, section and document templates. Name the custom signature policy `Document Template Update` to enable BIOVIA Workbook to find the policy when a user tries to update templates. You must also assign `TemplateManagementTools` permissions to individual users who need to update templates.

The `Document Template Update Signature Policy` is an optional signature policy, added in the Vault Administration Console console under the `Signature Policies` node. When a user selects `Update`, after running the `Usage Report`, they get a signing dialog to complete. The user's signature applies to the entire update but if the user cancels the signing dialog the update does not occur.

Set the following options for the `Document Template Update Signature Policy`:

- Comments, require single object signing, require open object, force user to re-authenticate
- Allowed signatory
- Other: reason - allowed, required

Chapter 6:

Defining Workflows

You can use BIOVIA Workflow Designer to design custom workflows for your experiments. These workflows identify workflow actor types and roles for the different stages of a workflow and activities to perform at the different stages. They can also identify signatory requirements for stage transitions that require digital signatures.

When you publish a workflow definition from Workflow Designer, it becomes available in your Vault Server. You use the Vault Administration Console in Vault Server to associate the actor roles you set up for workflow stages with actual Foundation Hub users and groups.

The following workflow activities require specific actor types:

- Add experiments to the Inbox of BIOVIA Workbook users
- Send email
- Allow transitions
- Set experiment-level permissions

Other activities that can be triggered by workflow definitions include the following:

- Removal of comments and Inbox entries
- Responses to experiment-level review data
- Responses to events based on a predefined time-out period

Workflow actors are the set of roles that you can assign to users and groups. For example, the administrator can assign the Witness role to a user or a group. Defining a workflow actor role creates a new group called, *Actor Role for User*. Add individual users to groups. In BIOVIA Workbook, workflow actor groups are not displayed.

A workflow actor type is the name designation assigned to the relationship between users that the workflow service can interact with to perform instructions in a workflow definition. The relationships are defined in the **Workflow Actors** tab of the property dialog. You can only define workflow actor types for users. For example, a workflow definition can include an instruction such as "give the supervisor writeData permissions". The definition determines the system group for the supervisor and applies the writeData permission to it. If the supervisor role has not been defined for the author, no system group exists, the definition does not perform that task.

Workbook Activities from Workflows

A Workflow definition can trigger several types of Workbook activities:

- Initialization activities, which execute when an object *enters* a stage.
- Finalization activities, which execute when an object *exits* a stage.
- Transition-time and EventDriven activities, which when an object is transitioned from one stage to another.

Staged transitions

Staged transitions identify the actors types that are allowed to move an object from its current stage to another stage. Review and If Else activity allows the workflow definition to determine activities to execute based on an experiment's Review Results display.

For example, if the following activities do not apply such as UnableToReview, HasErrors, and HasWarnings, the workflow can automatically use the PassedReview activity.

If SafetyReview fails, the object transitions to InProgress, but if the SafetyReview activity succeeds, the object transitions to SafetyApproved.

Parallel activities

Parallel activities are activities that can be in progress at the same time. For example, PrintExperimentReport and AssignTaskToWitness can execute in parallel.

Security activities

Security activities are activities that require particular workflow actor roles and permissions. For example, a user with a witness role might not have the permission to open or check out an object, and an author might not have the permission to roll back an object.

Timed activities

Timed activities are activities that execute after a specified period has elapsed. For example, the Workflow Definition could trigger the sending of an email or apply permissions to an experiment if an experiment is not transitioned for a period of six months. Timed activities cannot trigger transitions.

Email activities

Email activities use SMTP to send a message to one or more actors. For example, the transition of an experiment into the SafetyApproved stage might cause an email to be sent to a quality assurance manager.

Task activities

Task activities appear in the Notebook Explorer Inbox to alert actors of an assignment such as a witness duty.

Remove Comment activities

Remove comment activities remove all messages, comments, and workflow-generated tasks from an experiment. Such activities usually occur right before an experiment is put into its archive or final state.

Remove Comments removes all Annotations, which includes Comments, Tasks, and Messages. Remove Task is a subset of Remove Comments, so it is not necessary to do both.

Workflow Design Best Practices

To design a Workflow Definition that is well-suited to the business rules of your organization, consider:

Document the business rules of your organization.

What is the sequence of stages? How do responsibilities change from person to person in the lifecycle of the experiment?

Who are the actors? What are the permissions such as read, write, update, rollback, delete of the various actors at the various stages?

What are the transitions, that is, all the possible routes between stages?

Can any two actors be consolidated into a single actor? For example, is it possible for the Reviewer and Approver to be the same person?

Can any person be in the role of more than one actor? For example, can a supervisor also be a witness?

Which activities such as sending an email need to occur when an object is transitioned to another state?

What is the signature policy? Which actors sign off at which stage? Are all signatures digital? How does a signing event affect the stage of the document?

To design a Workflow Association that is well-suited to the business rules of your organization, consider: What should trigger a document getting associated to a Workflow Definition?

It might be helpful to write a textual description of the workflow and also make a visual flow chart to diagram the various stages, actors, activities, and signature events. Careful planning to understand and communicate the organizational requirements increases the likelihood that the design is stable and successful. Consider asking all the stakeholders to review your documentation of the business rules, getting their feedback and approval while you are still in the planning phase.

- Build the actors and signature policies by using the Administration Console or PowerShell.
- Build the workflow definition by using the BIOVIA Workflow Designer graphical user interface.
- Publish the workflow definition.
- Compile and publish directly to Vault.
- Compile to a VOZIP package and publish by using the Administration Console.
- Activate the workflow by creating a workflow association that specifies the type of object to manage and its path. Design the association so only one workflow is applied to any given object.
- Test the workflow as a prototype and refine if necessary. The Administration Console can disable an association.

Workflow Examples

The Workflow SDK allows creating a custom activity to you can use to:

- Integrate a BIOVIA Workbook workflow with an external system.
- Enforce standard operating procedures at your organization that are not yet covered by the activities that BIOVIA provides.

Examples:

- Prevent an author from being the author's own witness
- Prevent any actor from modifying an archived experiment
- Send a PDF report to an external system

Building a custom activity requires the following tasks:

- Using `ExampleActivity` template that is installed with the BIOVIA Framework SDK, and using the Dependency properties to expose configuration parameters to the Workflow Designer.
- Implementing the custom activity, including logging for debugging to help administrators troubleshoot any problems later.
- Signing your assembly using Visual Studio Project Settings so that the Assembly Cache can fulfill the request for your assembly.

Workflow SDK

The Workflow SDK allows an SDK developer to create a custom activity, which can be useful for:

- Integrating a BIOVIA Workbook workflow with a system that is external to BIOVIA Workbook.
- Enforcing standard operating procedures at your organization that are not yet covered by the activities that provides.

- Prevent an author from being the author's own witness
- Prevent any actor from modifying an archived experiment
- Send a PDF report to an external system

To build a custom activity:

1. Start from the `ExampleActivity` template that is installed with the BIOVIA Framework SDK, and use `Dependency Properties` to expose configuration parameters to the Workflow Designer.
2. Implement the custom activity. Include logging for debugging and to help administrators troubleshoot any problems later.
3. Sign your assembly, using `Visual Studio Project Settings`, so that the Assembly Cache can fulfill the request for your assembly.

Archive Using a Workflow

BIOVIA Workbook integrates with Enterprise Content Management (ECM) systems such as Documentum and Iron Mountain for long-term record retention.

For example, a Vault Workflow can:

- Print a Completed experiment to PDF.
- Transfer the PDF and metadata to the ECM.
- Invalidate the ECM record if the experiment is reverted.
- Republish to ECM if the experiment is signed off again.

Create Vault Workflow Actors

Note: Do not assign the Workflow Actor role to the `Global.Administrators` group, and do not include the `Global.Administrators` group in any permission setting done by a Workflow. The `VaultAdministrator` user or a user in the `Global Administrator` group needs to retain the permissions necessary to perform administrator functions.

A workflow can modify the permissions for experiments for any one or group assigned to the specific actor role.

You should not rename (change the Title) the workflow Author actor. The name change prevents the Vault Administration Console from displaying any workflow actor and causes the BIOVIA Workbook client and Workflow Designer to experience an unexpected error when attempting to retrieve Vault objects.

Membership in the Vault Global Administrators group is required in order to modify properties using the Vault Administration Console.

To create workflow actors:

1. In the Vault Administration Console, expand the Vault Server node, and expand the **Workflows** node.
2. Right-click the **Workflow Actors** node, and select **Add**.
3. In the **New Workflow Actor** dialog, in **Description**, type a brief description of the role.
4. In **Title**, type the name for the workflow actor role, and click **OK**.

Note: Do not use a hyphen (-) within the name of the workflow actor role.

You can view the newly defined actor in the Workflow Actors pane.

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

Add Workflow Definitions

You create a workflow definition using the Workflow Designer that ships with BIOVIA Vault Server. After you create the workflow definition, you add it to the Vault database.

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console.

To add a workflow definition:

1. In the Workflow Designer, you must publish the workflow to a .vozip package.
The .vozip is a Vault Object .zip file.
2. Copy the .vozip file onto the computer on which the Vault Administration Console is installed.
3. In the Vault Administration Console, expand the **Repositories** node.
4. Right-click the **Site** folder and select **Publish to folder**.
A window opens that allows you to select a .vozip file.
5. Select the .vozip package that you created in Workflow Designer.
6. When publishing is complete, the new folder, workflowActivityAssemblies is listed under the Site folder.
7. Expand the Workflows node and select **Workflow Associations**. You need to generate a workflow association for the new workflow definition.

For more information, see [Create Vault Workflow Actors](#) and [Generate a Workflow Association](#).

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

Removing a Workflow Definition

You can remove workflow definitions. The workflow definition continues to exist in the Vault repository to support documents that are associated with the workflow definition. When a workflow definition is removed, it is marked as hidden using the `Flags` property.

Any workflow associations that references a removed workflow definition are deleted, and the workflow definition is renamed. None of the existing instances of the workflow definition are affected.

You can rollback to the removed workflow definition. However, you cannot activate new documents in the workflow because the workflow associations were deleted.

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console.

To remove a workflow definition

1. In the Vault Administration Console, expand the Vault Server, and expand the **Workflows** node, and select **Workflow Definitions**.
2. In the **Workflow Definitions** pane, right-click the workflow definition, and select **Remove**.
3. Click **OK**.

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

Move Experiments or Objects Between Workflow Stages

The BIOVIA Workbook workflowTransition permission allows an administrator to manage the Workflow process. If a repository workflow has been set up for experiments and objects, whenever a user checks in an experiment or object, a dialog appears that allows the user to select a workflow transition to execute. At times it might be necessary for an administrator to perform a transition that does not follow the normal workflow. This is called an administrative workflow transition. You can use the Vault Administration Console to perform an administrative workflow transition. For more information, see *Transition Workbook Items* in the BIOVIA Workbook online help.

To support administrative workflow transitions, you must have a signature policy with the title `Administrative Move`, or the option will not be available in the Workbook client application. For more information, see [Create a Signature Policy](#).

To move an experiment or object between workflow stages:

1. From the Start menu, select Vault Administration Console.
2. In the Console Root\Vault Administration\Vault Server, expand the Vault Server node and [log in](#) as a member of the Vault Global Administrator's group.
3. Expand the **Repositories** node.
4. Navigate to the folder that contains the experiment to be transitioned.
5. In the **Actions** panel, select **Workflow Instances**.
6. Select the appropriate experiment row from the middle pane, right-click, and select **Move to Stage**.
The ID in Notebook Explorer translates to the Object ID in this grid. You can find the ID of an object in Notebook Explorer by viewing its properties.
7. In **Move WorkflowInstance to Stage**, select a value from the **Stage** list, and select another stage from the list to transition the experiment to that stage.
8. Click **OK**.

Generate a Workflow Association

When a Vault object is checked in, the workflow engine evaluates all enabled workflow associations. If any are returned true, the Vault object is associated to the workflow definition to which the association belongs.

Use a unique workflow association name. Do not insert a period in the name. The association fails to compile if the period character is in the Name field.

Note: To determine the repository or folder for any workflow association, right-click the name of the association and select Properties. In the Properties dialog, click the Code parameter.

To generate a workflow association:

1. In the Vault Administration Console, expand the Vault server, and expand the **Workflows** node.
2. Right-click the **Workflow Associations** node and select **Generate**.
3. In the **Generate Workflow Association** dialog, type a unique name for the workflow association in the **Name** field.
4. Click Choose to select the workflow definition for the association.
5. (Optional) Click **Use parameters** to add parameters to the association.
6. (Optional) When using parameters, in the Parameters group, select the **VaultObjectType**.

7. (Optional) Type a string in `VaultPath` to prefix the association.
For example in the `VaultPath`, type `Analytical` for a workflow association with a repository named, `Analytical`.
8. (Optional) Click **Use source** to add source code to use during the association generation.
If the **Use source** radio button is selected, the **Use parameters** button is disabled, and the **Compile** button at the bottom of the dialog becomes active.
9. Click **Generate**. The source code displays in the **Source Code** area of the **Generate Workflow Association** dialog.
10. Click **Compile**. The compilation results display in the Compiler results window.
11. Click **Publish**.
The new workflow association should display in the **Workflow Associations** node after you click **Refresh**. If workflow associations does not display, you need to index new or modified Vault objects. For more information, see [Running the Vault Indexing Utility in the BIOVIA Vault Server Administration Guide](#).

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

See also:

[Get VaultID for Generating a Workflow Association](#)

[Generate Workflow Association Code Example](#)

Generate Workflow Association Code Example

You can use source code to assist in generating a workflow association. Your code must return *true* or *false*. When *true* is returned, the object is associated with the workflow that determines all future actions. When *false* is returned, the object is not associated with the workflow and the workflow steps do not apply to that object.

For example, if you want to run the workflow only when the object description is set to a specific value, you can use the following code:

```
// BEGIN Code section generated by
Symyx.Framework.workflow.workflowAssociationGenerator

// Need to check the description string is not null or empty, as well
// as check that the description contains the specific value
if ( !string.IsNullOrEmpty(obj.Description) &&
    obj.Description.ToUpperInvariant().Contains("Your description to check") )
{
    return true;
}
else
{
    return false;
}
// END Code section generated by
Symyx.Framework.workflow.workflowAssociationGenerator
```

The next code example returns true if the Unified Resource Identifier (URI) for the object document template matches the Vault ID of 381b40a8-8a1e-4106-a806-97c1359604ee, you change the Vault ID in the next step, the code returns false.

The workflow is run if the object has a specific template. Add the PropertyInfo block to your source code to use this example.

```
// BEGIN Code section generated by
Symyx.Framework.Workflow.WorkflowAssociationGenerator
if (obj.ObjectType == VaultObjectType.Document)
{
    PropertyInfo templateUriProperty =
        obj.GetType().GetProperty("TemplateUri");
    VaultUri templateUri = templateUriProperty.GetValue(obj, null)
        as VaultUri;
    // return true when the object's template URI Vault ID is equal to a
    // specific value
    return templateUri.VaultId.ToString().Equals(
        "DocumentTemplate.381b40a8-8a1e-4106-a806-97c1359604ee");
}
else
{
    return false;
}
// END Code section generated by
Symyx.Framework.Workflow.WorkflowAssociationGenerator
```

See also:

[Generate a Workflow Association](#)

[Get VaultID for Workflow Association](#)

Get VaultID for Generating a Workflow Association

In your source code used in the generate a workflow association, change the Vault ID of the template to an ID stored in the Vault repository.

To get a document template Vault ID:

1. Start Oracle SQL*Plus and log in as the owner of the Site schema.
2. Run the following commands in SQL*Plus:

```
column GUID format a45
column name format a25
```

Select the GUID name from the vaultobject where objecttype is similar to 'DocumentTemplate';

If you have templates in Vault, your output should look similar to the following example:

```
GUID NAME
-----
-----
                DocumentTemplate.103ddca5-c662-44bc-acb4-b7db Blank Experiment
                2f320973
```

3. In the source code window of the [Generate Workflow Association](#) dialog, set the document template Vault ID to the value in the GUID column returned by SQL*Plus.

For example, change DocumentTemplate.381b40a8-8a1e-4106-a806-97c1359604ee in the source code to DocumentTemplate.103ddca5-c662-44bc-acb4-b7db2f320973.

See also:[Generate a Workflow Association](#)[Generate Workflow Association Code Example](#)

Enable Workflow Associations

An Enable column and filter were added to the Workflow Associations grid display in BIOVIA Workbook. You must have Vault Administrator permission to modify a workflow association.

1. In the Vault Administration Console, expand the Vault Server node, and expand the **Workflows** node.
2. Select **Workflow Associations**.
3. In the **Workflow Associations** pane, select the workflow association, and click **Enabled**.

Yes displays in the workflow association row.

An audit history entry is created when you inactivate a user. For more information, see [Audit Trail Actions](#).

See also:[Generate a Workflow Association](#)

Removing Workflow Associations

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console. .

To remove a workflow association:

1. In the Vault Administration Console, expand the **Vault Server**, and select **Workflows**.
2. In the **Workflows** node, select **Workflow Associations**.
3. In the **Workflow Associations** pane, right-click the workflow association to remove, and then select **Remove**.
4. Click **OK**.

The workflow association is deleted from Vault Server.

An audit history entry is created for this action. For more information, see [Audit Trail Actions](#).

See also:[Generate a Workflow Association](#)

Disabling Workflow Associations

You must have membership in the Vault Global Administrators group to modify properties using the Vault Administration Console.

1. In the Vault Administration Console, expand the Vault server, and expand **Workflows**.
2. Select **Workflow Associations**, in the **Workflow Association** pane, right-click the workflow association and select **Disable**.

An audit history entry is created when you inactivate a user. For more information, see [Audit Trail Actions](#).

See also:

[Generate a Workflow Association](#)

Placeholder Formats

A Vault object has various properties. A placeholder allows you to get the value of a Vault object property. You also can get a Vault actor.

You can only use placeholders in a `SymyxSendMessageActivity` workflow activity.

Placeholders use the following format:

- `%actor%`

Actor is a Vault actor. For example, `%Supervisor%`, this returns the Supervisor actor.

- `%property%`

Property is a Vault object property. For example, `%VaultPath%`, this returns the full path to the current Vault document being processed by a workflow.

Always place the Vault actor or Vault object property between % characters.

Placeholder Examples

Use placeholders to include data in emails sent from a workflow. The following example shows how to use placeholders in an email's destination address, sent from address, and body:

To: `%Supervisor%;%primary.witness%`

From: `%System.Transitioner%`

Body: The document `%VaultPath%` is ready for your review.

The email's destination and sent from addresses contain actors (Supervisor, primary.witness, and System.Transitioner). The email body contains a Vault object property, `VaultPath`.

The actors used in the email example are:

Actor	Description
Supervisor	Represents a recipient of the email. The Supervisor actor is not a standard actor; you must create the Supervisor actor to use this value.
primary.witness	Represents another recipient of the email.
System.Transitioner	Represents the actor who started the workflow transition. This actor is the user checking a document into Vault. As part of the check in, a workflow transition is run and the user checking in the document is recorded in System.Transitioner.

There are limitations to where you can use Vault actors and Vault object properties:

- You can only use Vault actors in an email address.
- You can only use Vault object properties in an email subject and body.
- You can use `%System.DefaultFrom%` and `%System.DefaultTo%` for the email addresses, which are the values defined in the `WorkflowEmailSection` of the configuration file on the middle-tier server where BIOVIA Vault Server is installed, for example, `C:\Program Files (x86)\symyx\Symyx.Vault.PrivateService\web.config`.

- If you use `%System.Transitioner%` in an email address, but there is no email address defined for that actor, then the email address for `%System.DefaultFrom%` is used for the *From* address of the email, and `%System.DefaultTo%` is used for the *To* email address.

The following example shows the use of other placeholders.

To: `%Author%`

Subject: Your document `%Title%` has entered workflow

`%Author%` is the creator of the Vault document.

`%Title%` is the name of the Vault document.

You can use any Vault object property in the subject and body of an email. For example, `VaultPath` is the folder path to the Vault document.

For a list of core properties, see the data model in the *BIOVIA Vault Server Installation Guide*.

You can use any property in any property set assigned to a Vault object.

If there is a duplicate name in two different property sets for a Vault object, then there is no guarantee which one is retrieved.

Appendix A:

Administering Vault using PowerShell Scripts

PowerShell is a command-line shell and scripting language for task and configuration management. BIOVIA Vault Server is deployed with scripts that create functions that you can call from PowerShell. The Vault Server installer includes PowerShell 3.0.

The code in the PowerShell Vault scripts is subject to change.

If you copy sample code from this PDF document and paste it into a text editor, the text might change from the original. To ensure that the text is valid and usable, you might need to restore indents, remove extra spaces, and retype characters for single and double quotes.

PowerShell Prerequisites

- Install the required security certificate on the computer on which you will run PowerShell.
- To install Vault Administration Console, you must first run the Vault Deployment Utility, and download and run the `VaultAdministrationConsoleInstaller.exe`.
- Use the Vault Administration Console version that matches the version of the BIOVIA Vault Server. If the versions numbers are not the same, you need to upgrade the administrative tools console.

The PowerShell scripts are installed at:

```
<installation_directory>\Program Files (x86)\BIOVIA\Vault  
Administration\PowerShell
```

- The Vault Server installer includes PowerShell 3.0.
- You must modify the system environment variable PATH and change the setting for PowerShell so that it will start the 32-bit version not the 64-bit version. Change:
`%SYSTEMROOT%\system32\windowsPowerShell\v1.0` to
`%SYSTEMROOT%\syswow64\windowsPowerShell\v1.0`

Initial PowerShell Commands

Note: Foundation Hub manages vocabularies, so there are no related Powershell commands. Refer to Foundation Hub 2021 Administration Guide for more information.

You can adapt the PowerShell example code for your own use.

If you copy sample code and paste it into a text editor, the text might change from the original. To ensure that the text is valid and usable, you might need to restore indents, remove extra spaces, and retype characters for single and double quotes.

Before running any PowerShell commands, perform the following steps in PowerShell:

1. Choose **Start > Programs > Windows PowerShell 1.0 > Windows PowerShell**.
2. Set the execution policy to unrestricted:
`set-executionpolicy unrestricted`
3. On a client computer in the directory `<installation_directory>\Program Files\BIOVIA\Vault Administration\PowerShell` add the directory to your path.

```

if($env:path -notlike "*Administrative Tools*")
{
    # Only set the path if it doesn't already exist
    $global:savePath = $env:path;
    $env:path = $env:path + ";
    "installation_directory\Program Files\BIOVIA\Vault
Administration\PowerShell";
}

```

4. Change directories to C:\Program Files\BIOVIA\Vault Administration\PowerShell
`cd "C:\Program Files\BIOVIA\Vault Administration\PowerShell"`
5. Run the PowerShell scripts to load the assemblies and create the required functions:
 - . .\Load-Assemblies.ps1
 - . .\Add-Association.ps1
 - . .\Add-User.ps1
 - . .\Connect-Server.ps1
 - . .\Create-workflowActor.ps1
 - . .\Create-workflowAssociation.ps1
 - . .\Get-VaultId.ps1
 - . .\Get-VaultObject.ps1
 - . .\Publish-workflowActorAssociation.ps1
 - . .\Save-VaultObject.ps1
 - . .\Set-DefaultTemplate.ps1
 - . .\Set-Permissions.ps1
6. Connect to Vault from PowerShell; set the Server, UserName, and Password parameters in the following line and run it:


```
> $ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -Password "XXXX";
```

The command calls Connect-Server and stores the returned workspace in an object called ws.
7. Verify that you are authenticated:


```
> $ws.IsAuthenticated
True
```

If you are authenticated, the command returns True. If False is returned, verify that you set the correct Server, User Name, and Password parameters in the previous step.

You can use `psbase` to display all the properties for an object. The following example uses `ws.psbase` to display all of the `ws` properties:

```
$ws.psbase

ActiveVaultServer : Symyx.Framework.Vault.VaultServer
ActiveServer      : Symyx.Framework.Vault.VaultServer
HasActiveServer   : True
SiteRepository    : {Repository.ab732e9a-a793-d437-5b04-2165cf8214c6}
UserRepository    : {}
VaultRepositories : {Repository.d8eb6c52-2f10-243e-e906-5ea1a00b7d74}
CurrentUser       : SYMYX-IC\vault.admin
HomeRepository    : {}
Repositories      : {Repository.d8eb6c52-2f10-243e-e906-5ea1a00b7d74}
IsAuthenticated   : True
AuthenticationState : Yes
IsOnline          : True
Comparer          : System.Collections.Generic.GenericEqualityComparer`1
[System.String]
Count            : 1
Keys             : {Server1}
Values           : {Symyx.Framework.Vault.VaultServer}
```

Vault Identifiers

Vault identifiers (IDs) are strings that uniquely reference objects in the BIOVIA Vault Server database. For example, `User.628eca9c-c1d8-84de-0876-47e3924171d1` references a Vault user object.

Many of the scripts contain example usage lines that illustrate how to call the function that the script creates. Some of the usage lines contain Vault IDs that reference fictitious objects. For example, the `Add-Association.ps1` contains the following line in the usage section that shows how to get a user object from Vault:

```
$User = Get-VaultObject -VaultId "User.628eca9c-c1d8-84de-0876-47e3924171d1"
-workspace $ws -Repository $ws.SiteRepository;
```

The `Get-VaultObject` function returns the Vault object with the ID specified in the `VaultId` parameter. `Get-VaultObject` returns the specified user and stores it in the `User` object.

Before running the example code in your system, you must retrieve the Vault ID for a user object that actually exists in your Vault installation.

To get an Vault ID, you can use the `Get-VaultId` function, created by the `Get-VaultId.ps1` script, or Oracle SQL*Plus.

Retrieve a Vault ID with Get-VaultId

To retrieve a Vault ID using the `Get-VaultId` function, use the following steps:

1. Open PowerShell, set the `Name` parameter in the following line to a user that exists in your Vault installation and then run the line:

```
$id = Get-VaultId -Name "Supervisor" -Type "User" -workspace $ws;
```

The Name and Type parameters specify the name and type of the object whose ID you want to retrieve from Vault. The workspace parameter specifies the workspace object you created when calling the Connect-Server function to connect to Vault.

2. To display the id:

```
$id
```

```
Prefix Guid
```

```
User f51f77e4-5a71-4d52-a058-39a40ee5a7e5
```

The Guid column in the output contains the Vault ID you need when calling Get-VaultObject. You can then call Get-VaultObject to retrieve the actual object from BIOVIA Vault Server:

```
$User = Get-VaultObject -VaultId "User.f51f77e4-5a71-4d52-a058-39a40ee5a7e5" -workspace $ws -Repository $ws.SiteRepository;
```

Retrieve a Vault ID with Oracle SQL*Plus

To retrieve a Vault ID using SQL*Plus:

1. Start SQL*Plus and log in to Oracle as the owner of the Site schema.
2. Run the following commands in SQL*Plus:

```
column GUID format a45
```

```
column name format a25
```

```
select GUID, name from vaultobject where name like '%Supervisor%';
```

If the object exists, you see output similar to the following example:

```
GUID NAME
```

```
User.0a6598c0-0162-7449-9a7c-5103346a30fc symyx-ic\Supervisor
```

The GUID column contains the Vault ID you need to use when calling Get-VaultObject.

3. From PowerShell, you can call Get-VaultObject to get the object from Vault, using the following:

```
$User = Get-VaultObject -VaultId "User.0a6598c0-0162-7449-9a7c-5103346a30fc" -workspace $ws -Repository $ws.SiteRepository;
```

4. To see all the users and their Vault IDs, run the following query in SQL*Plus:

```
select GUID, name from vaultobject where objecttype='User';
```

5. To see all the groups and their Vault IDs, run the following query in SQL*Plus:

```
select GUID, name from vaultobject where objecttype='Group';
```

6. To see all the object types, run the following query in SQL*Plus:

```
select unique objecttype from vaultobject;
```

PowerShell Scripts

The BIOVIA Vault Server PowerShell scripts are located in C:\Program Files (x86)\BIOVIA\Vault Administration\PowerShell on the computer where you install the Vault Administration Console. The following table lists the available scripts.

Script filename	Function created by the script allows you to
add-Association.ps1	Adds a workflow association.
Add-BalanceIntegration-Dictionary.ps1	Sets up balance integration.
add-User.ps1	Adds a user.
BalanceIntegration.ps1	Sets up balance integration.
change-workflowAssociationEnabled-AddHistory.ps1	Enables, disables, and removes Workflow Associations.
connect-Server.ps1	Connects to Vault.
Create-Section.ps1	Creates a document with a text section.
Create-SectionTemplate.ps1	Creates a document section template based in the text section.
Create-Template.ps1	Creates a new document template.
create-workflowActor.ps1	Creates a workflow actor.
create-workflowAssociation.ps1	Creates the Workflow Association for a specified Workflow Definition.
Get-AllByType.ps1	Gets all objects from the Site repository of a specific type
get-VaultId.ps1	Gets the ID of a Vault object.
get-VaultObject.ps1	Gets a Vault object.
Import-Forms.ps1	Imports all form files from a specified local folder
Index-VaultObject.ps1	Sends a specified object to the message processing queue for indexing.
LegacyConverter-Setup.ps1	Sets up legacy conversion
load-Assemblies.ps1	Loads the Framework assemblies.
MaterialInfoManager-Setup.ps1	Sets up material info
publish-workflowActorAssociation.ps1	Publishes an association between a workflow and an actor.

Script filename	Function created by the script allows you to
save-vaultObject.ps1	Saves a Vault object.
set-DefaultTemplate.ps1	Sets the default experiment template.
set-Permissions.ps1	Sets the permissions for an object.
Update-BalanceIntegration-Dictionary.ps1	Changes the balance server URL
VerifyFolderTitleChangeInSystemHistory.ps1	Example used internally for system verification. Verifies that renaming folder titles will be recorded in Audit history.
VerifySectionTemplateTitleChangeInSystemHistory.ps1	Example used internally for system verification. Verifies that renaming folder titles will be recorded in Audit history.
VerifySectionTitleChangeInSystemHistory.ps1	Example used internally for system verification. Verifies that renaming section titles will be recorded in Audit history.
VerifyTemplateTitleChangeInSystemHistory.ps1	Example used internally for system verification. Verifies that renaming template titles is recorded in Audit history.

Add-Association.ps1 Script

The Add-Association.ps1 script contains a function called Add-Association that adds a Workflow actor association to a user record in BIOVIA Vault Server. The script is the PowerShell version of the operation to add an actor role and a group that assigns role to a user record on the Workflow tab in the User properties of the Vault Administration Console. The script contains code similar to the following listing:

```
$usage = @'
. .\Load-Assemblies.ps1
. .\Connect-Server.ps1
. .\Create-workflowActor.ps1
. .\Add-Association.ps1
. .\Save-vaultObject.ps1
. .\Get-vaultObject.ps1
. .\Get-vaultId.ps1
$ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -
Password "XXXX";
$actor = Create-workflowActor -Name "Primarywitness";
Save-vaultObject -vaultObject $actor -workspace $ws;
# Alternatively if it already exists:
```

```
# $actorId = Get-VaultId -Name "PrimaryWitness" -Type
"WorkflowActor" -Workspace $ws;
# $actor = Get-VaultObject -VaultId $actorId -Workspace $ws -
Repository $ws.SiteRepository;
$user = Get-VaultObject -VaultId "User.628ecafc-c1d8-84de-0876-
47e3924171d1" -workspace $ws -Repository $ws.SiteRepository;
$target = Get-VaultObject -VaultId "Group.c5f785d4-4c6e-f445-e87d-
c6ee4853e423" -workspace $ws -Repository $ws.SiteRepository;
$association = New-Object
Symyx.Framework.workflow.workflowActorAssociation $target, $actor;
Add-Association -User $user -Association $association;
Save-VaultObject -VaultObject $user -Workspace $ws;
'@;

function Add-Association {
    param (
        [Symyx.Framework.Vault.User]$User=$(throw "must specify -
User"),
        [Symyx.Framework.Vault.Association]$Association=$(throw "must
specify -Association")
    )
    begin {
        $User.Associations.Add($Association);
    }
    process {
        # No process of pipeline
    }
    end {
        # No end activity
    }
}
```

The usage section of the script shows sample PowerShell commands. For example:

To create a workflow actor, call `Create-WorkflowActor`:

```
$actor = Create-workflowActor -Name "PrimaryWitness";
```

To save the new actor, call `Save-VaultObject`:

```
Save-VaultObject -VaultObject $actor -workspace $ws;
```

If you already have an actor you want to use, set the actor name and type in the following line and execute:

```
$actorId = Get-VaultId -Name "PrimaryWitness" -Type "WorkflowActor" -
workspace $ws;
```

```
$actor = Get-VaultObject -VaultId $actorId -Workspace $ws -Repository
$ws.SiteRepository;
```

To set the user and target group, modify the Vault IDs in the following lines and execute:

```
$user = Get-VaultObject -VaultId "User.628ecafc-c1d8-84de-0876-47e3924171d1"
-workspace $ws -Repository $ws.SiteRepository;
```

```
$target = Get-VaultObject -VaultId "Group.c5f785d4-4c6e-f445-e87d-c6ee4853e423" -workspace $ws -Repository $ws.SiteRepository;
```

To create an association and then save it, execute:

```
$association = New-Object Symyx.Framework.Workflow.WorkflowActorAssociation
$target, $actor;
```

```
Add-Association -User $user -Association $association;
```

```
Save-VaultObject -VaultObject $user -workspace $ws;
```

Change-WorkflowAssociationEnable-AddHistory.ps1 Script

With BIOVIA Vault Server, administrators can enable, disable, and remove workflow associations using the Vault Administration Console or API calls. When using API calls, BIOVIA recommends using the following script, so the audit data gets generated for these events. You can use a single API call to make any of these changes, critical system data changes occur without generating audit data. The Vault Administration Console generates the same audit data.

Run the Change-WorkflowAssociationEnabled-AddHistory.ps1 PowerShell script to enable, disable, and remove workflow associations and generate audit data.

1. Open the Microsoft PowerShell application:

Start > All Programs > Accessories > Windows PowerShell > Windows PowerShell (x86)

2. Navigate to the PowerShell folder, for example, on a 32-bit client computer:

```
cd C:\Program Files (x86)\BIOVIA\Vault Administration\Powershell or on a 64-bit client computer
```

```
cd C:\Program Files (x86)\BIOVIA\Vault Administration\Powershell
```

3. Run the Load-Assemblies.ps1 and Connect-Server.ps1 scripts.

For example:

```
. .\set-execution
unrestricted; . .\Load-Assemblies.ps1; . .\Connect-Server.ps1;
$server = server_name; $ws = Connect-Server
-Server $server - Username DOMAIN\user_name -Password password
```

The *server_name* is the fully qualified server name of your Vault server.

DOMAIN\user_name is a Vault administrator user.

4. Run the Change-workflowAssociationEnabled-AddHistory.ps1 script, type `. .\Change-workflowAssociationEnabled-AddHistory.ps1`; or `. .\Change-workflowAssociationEnabled-AddHistory.ps1`;

Query the Audit History table in the database to access the audit data generated by the Change-workflowAssociationEnabled-AddHistory.ps1.

Enable or disable workflow association and audit history

To enable or disable a workflow association and generate an audit history, type the following:

- `. .\ Change-workflowAssociationEnabled-AddHistory -workspace $ws -Name "WTA-Title" -EnableState $value`; or `. .\Change-workflowAssociationEnabled-AddHistory -workspace $ws -Name "WTA-Title" -EnableState $value`;

The variables in the command are as follows:

"`. .\Change-workflowAssociationEnabled-AddHistory.ps1`" loads the PowerShell script.

```
Change-workflowAssociationEnabled-AddHistory -workspace $ws -Name "WTA-Title" -EnableState $true;
Change-workflowAssociationEnabled-AddHistory -workspace $ws -Name "WTA-Title" -EnableState $false;
```

calls the `Change-workflowAssociationEnabled-AddHistory` method or the `Delete-workflowAssociationEnabled-AddHistory` method from the PowerShell script with `workspace`, `Name`, and `EnableState` as arguments.

The `EnableState` argument values are:

- *true* to enable workflow associations
- *false* to disable workflow associations

Remove workflow association

To remove a workflow association and generate an audit history to call the `Delete-workflowAssociation-AddHistory` method from the PowerShell script with `workspace` and `Name` as arguments, use the following:

```
. .\ Delete-workflowAssociation-AddHistory -workspace $ws -Name "WTA-Title";
or
. .\Delete-workflowAssociation-AddHistory -workspace $ws -Name "WTA-Title";
```

Connect-Server.ps1 Script

The `Connect-Server.ps1` script contains a function called `Connect-Server` that connects to an instance of Vault Server. The script contains code similar to the following listing:

```
function LoadAssemblies {
    $a = @{
        "Symyx.Framework" = [System.Reflection.Assembly]::LoadwithPartialName
        ("Symyx.Framework");
    }
    return (,$a);
}

function Connect-Server {
    param (
        [System.String]$Server=$(throw "must specify -Server"),
        [System.String]$UserName=$(throw "must specify -UserName"),
        [System.String]$Password=$null
    )

    begin {
        if(-not $Password) {
            $Password = read-host -Prompt Password;
        }
        $NSVaultworkspace = [Symyx.Framework.Vault.Vaultworkspace];
        $NSworkspaceLoginManager = [Symyx.Framework.Vault.workspaceLoginManager];
        $ws = new-object $NSVaultworkspace $Server;
        $wslm = new-object $NSworkspaceLoginManager $ws;
        $wslm.Login($Server, $Username, $Password, $false);
        $ws.Clear();
    }
}
```

```

    $ws.Im.AddServerToWorkspace();
    $ws.MakeCurrentWorkspace();
    return (,$ws);
}

process {
# Don't process anything from the pipeline
}

end {
# Nothing to do at end
}
}

>null = LoadAssemblies;

```

To execute in PowerShell, you set the Server, User Name, and Password parameters in the following line and then execute it:

```
$ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -Password "XXXX";
```

Create-WorkflowActor.ps1 Script

The Create-workflowActor.ps1 script contains a function called, Create-workflowActor, that creates a Workflow actor. The script contains code similar to the following listing:

```

$usage = @'
. .\Load-Assemblies.ps1
. .\Connect-Server.ps1
. .\Create-workflowActor.ps1
. .\Save-VaultObject.ps1
$ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -Password
"XXXX";
    $wa = Create-workflowActor -Name "Supervisor";
    Save-VaultObject -VaultObject $wa -Workspace $ws;
'@;

function Create-workflowActor {
param (
    [System.String]$Name=$(throw "must specify -Name")
)
begin {
    $actor = New-Object Symyx.Framework.Workflow.WorkflowActor $Name;
}

process {
    # No process of pipeline
}

end {
    return (,$actor);
}
}

```

The following example creates two workflow actors and saves them to Vault:

```
$wa1 = Create-workflowActor -Name "Supervisor";
```

```
Save-VaultObject -VaultObject $wa1 -workspace $ws;  
$wa2 = Create-workflowActor -Name "Primary Witness";  
Save-VaultObject -VaultObject $wa2 -workspace $ws;
```

Create-WorkflowAssociation.ps1 Script

The `Create-workflowAssociation.ps1` script contains a function called, `Create-workflowAssociation` that creates a Workflow association. The script contains code similar to the following listing:

```
# PS path> . .\Load-Assemblies  
# PS path> . .\Connect-Server  
# PS path> . .\Get-VaultObject  
# PS path> . .\Create-workflowAssociation  
# PS path> . .\Save-VaultObject  
# PS path> $ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -  
Password "XXXX";  
# PS path> $wd = Get-VaultObject -VaultId "workflowDef.bf2c0d51-d9b5-4024-  
ad17-c857817b6573" -workspace $ws;  
# PS path> $documentType =  
[Symyx.Framework.Vault.VaultObjectType]::Document;  
# PS path> $wa = Create-workflowAssociation -workspace $ws -Name  
"MyAssociation" -VaultObjectType $documentType -Definition $wd;  
# PS path> Save-VaultObject -VaultObject $wa -workspace $ws;  
  
function Create-workflowAssociation {  
    param (  
        [Symyx.Framework.Vault.Vaultworkspace] $workspace=$(throw "must  
specify -workspace"),  
        [System.String] $Name=$(throw "must specify -Name"),  
        [Symyx.Framework.workflow.workflowDefinition] $Definition=$(throw  
"must specify -Definition"),  
        [Symyx.Framework.Vault.VaultObjectType] $VaultObjectType=  
[Symyx.Framework.Vault.VaultObjectType]::All,  
        [System.String] $Path  
    )  
  
    begin {  
        $wag = [Symyx.Framework.workflow.workflowAssociationGenerator]::Create  
($Name, $VaultObjectType, $Path);  
        $assembly = $wag.Assembly;  
        $null = [Symyx.Framework.Extensibility.AssemblyCache]::Publish  
($assembly);  
        $wao = New-Object Symyx.workflow.Custom.$Name;  
        $wao.Title = $Name;  
        $wa = New-Object Symyx.Framework.Vault.Association  
([Symyx.Framework.Vault.AssociationTypes]::Dependency, $Definition, $false,  
"workflow definition");  
        $wao.Associations.Add($wa);  
        return (,$wao);  
    }  
  
    process {
```

```

    }

end {
}
}

```

The start of the script shows sample PowerShell commands. For example:

Modify the Vault ID in the following line and call `Get-VaultObject` to retrieve an existing workflow definition:

```
$wd = Get-VaultObject -VaultId "workflowDef.bf2c0d51-d9b5-4024-ad17-
c857817b6573" -workspace $ws;
```

Set the document type:

```
$documentType = [Symyx.Framework.Vault.VaultObjectType]::Document;
```

Call `Create-WorkflowAssociation` to create the workflow association and then save it using `Save-VaultObject`:

```
$wa = Create-workflowAssociation -workspace $ws -Name "MyAssociation" -vaultObjectType
$documentType -Definition $wd;
Save-VaultObject -VaultObject $wa -workspace $ws;
```

Get-VaultId.ps1 Script

The `Get-VaultId.ps1` script contains a function called, `Get-VaultId` that returns the ID of a Vault object. The script contains code similar to the following listing:

```

$usage = '@'
. .\Load-Assemblies
. .\Connect-Server
. .\Get-VaultId
. .\Get-VaultObject
$ws = Connect-Server -Server XXXX -UserName "vault.admin" -Password XXXX;
$id = Get-VaultId -Name "Supervisor" -Type "WorkflowActor" -Workspace $ws;
$supervisor = Get-VaultObject -VaultId $id -Workspace $ws;
'@;
function Get-VaultId (
[System.String] $Name,
[System.String] $Type,
[Symyx.Framework.Vault.VaultWorkspace] $Workspace
){
begin {
$NSCoreProperty = [Symyx.Framework.Properties.CoreProperty];
$NSCondition = [Symyx.Framework.CorePropertyQueryCondition];
$NSQuery = [Symyx.Framework.Query];
$titleProperty = $NSCoreProperty::Title;
$typeProperty = $NSCoreProperty::Type;

```

```
$equalTo = [Symyx.Framework.QueryComparisonOperator+QueryComparisonOperators]::EqualTo;
$titleCondition = New-Object $NSCondition $titleProperty, $equalTo, $Name;
$typeCondition = New-Object $NSCondition $typeProperty, $equalTo, $Type;
$condition = [Symyx.Framework.CorePropertyQueryCondition]::op_BitwiseAnd($titleCondition,
$typeCondition);
$query = New-Object $NSQuery $condition;
$list = $Workspace.FindVaultIds($query);
return (,$list)[0];
}
process {
# No processing of pipeline
}
end {
}
}
```

The usage section of the script shows sample PowerShell commands. For example:

Call `Get-VaultId` to get the ID of an existing workflow actor named Supervisor:

```
> $id = Get-VaultId -Name "Supervisor" -Type "WorkflowActor" -Workspace $ws;
```

Call `Get-VaultObject` to get the actual Supervisor object:

```
> $supervisor = Get-VaultObject -VaultId $id -Workspace $ws;
```

Get-VaultObject.ps1 Script

The `Get-VaultObject.ps1` script contains a function called `Get-VaultObject` that returns a Vault object. The script contains code similar to the following listing:

```
$usage = '@'

. .\Load-Assemblies
. .\Connect-Server
. .\Get-VaultObject

$ws = Connect-Server -Server XXXX -UserName vault.admin -Password XXXX;

$folder = Get-VaultObject -VaultId "Folder.1e7cc73a-0da4-4826-a827-
625e7625c8ea" -workspace $ws -Repository $ws.HomeRepository;

$folder.Description;

'@;

function get (
    [System.String] $VaultId,
```

```

        [Symyx.Framework.Vault.VaultWorkspace] $Workspace,

        [Symyx.Framework.Vault.VaultRepository] $Repository =
$Workspace.SiteRepository

    ) {

        [Symyx.Framework.Vault.VaultId] $id = New-Object
Symyx.Framework.Vault.VaultId $VaultId;

        [Symyx.Framework.Vault.DataScope] $scope =
[Symyx.Framework.Vault.DataScope]::All;

        $signature = "TVaultObject Get\[TVaultObject\]\
(Symyx.Framework.Vault.VaultId, Symyx.Framework.Vault.DataScope)";

        $m = @($Repository.GetType().GetMethods() | ?{"%{$_}" -match
"$signature"})[-1];

        [type[]] $types = @([Symyx.Framework.Vault.VaultObject]);
        $generic = $m.MakeGenericMethod($types);
        $object = $generic.Invoke($Repository, @($id, $scope));
        return (,$object);
    }

function Get-VaultObject (

    [System.String] $VaultId,

    [Symyx.Framework.Vault.VaultWorkspace] $Workspace,

    [Symyx.Framework.Vault.VaultRepository] $Repository =
$Workspace.SiteRepository

    ) {

    begin {

    if($VaultId -ne $null) {

        $object = get $VaultId $Workspace $Repository;

        return (,$object);

    }
    }

```

```
    }

    process {

        if($_ -ne $null) {

            $object = get $_ $workspace $Repository;

            write-output (,$object);

        }

    }

end {

}

}
```

The usage section of the script shows sample PowerShell commands. For example:

Modify the Vault ID in the following line and then call `Get-VaultObject` to retrieve an existing folder:

```
$folder = Get-VaultObject -VaultId "Folder.1e7cc73a-0da4-4826-a827-625e7625c8ea" -workspace $ws -Repository $ws.HomeRepository;
```

To display the folder description:

```
$folder.Description;
```

The `Publish-WorkflowActorAssociation.ps1` script contains a function called `Publish-WorkflowActorAssociation` that publishes a workflow actor association. The script contains code similar to the following listing:

```
$usage = @'
```

```
.. \Load-Assemblies.ps1
```

```
.. \Connect-Server.ps1
```

```
.. \Create-WorkflowActor.ps1
```

```
.. \Add-Association.ps1
```

```
.. \Save-VaultObject.ps1
```

```
.. \Get-VaultObject.ps1
```

```
.. \Get-VaultId.ps1
```

```
.. \Publish-WorkflowActorAssociation.ps1
```

```
$ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -Password "XXXX";
```

```
$granteeId = Get-VaultId -Name "symyx-ic\vault.admin" -Type "User" -Workspace $ws;
```

```
$grantee = Get-VaultObject -VaultId $granteeId -Workspace $ws -Repository $ws.SiteRepository;
```

```
$targetId = Get-VaultId -Name "Global Administrators" -Type "Group" -Workspace $ws;
```

```
$target = Get-VaultObject -VaultId $targetId -Workspace $ws -Repository $ws.SiteRepository;
```

```
Publish-WorkflowActorAssociation -ServerWorkspace $ws -UserGrantee $grantee -Actor "Supervisor" -
GroupTarget $target;
'@;
function Publish-WorkflowActorAssociation {
param (
[Symyx.Framework.Vault.VaultWorkspace] $ServerWorkspace=$(throw "must specify -
ServerWorkspace"),
[Symyx.Framework.Vault.User] $UserGrantee=$(throw "must specify -User"),
[System.String] $Actor=$(throw "must specify -Actor"),
[Symyx.Framework.Vault.Group] $GroupTarget=$(throw "must specify -GroupTarget")
)
begin {
$actorId = Get-VaultId -Name $Actor -Type "WorkflowActor" -Workspace $ServerWorkspace;
$actor = Get-VaultObject -VaultId $actorId -Workspace $ServerWorkspace -Repository
$ws.SiteRepository;
$association = New-Object Symyx.Framework.Workflow.WorkflowActorAssociation $GroupTarget,
$actor;
Add-Association -User $UserGrantee -Association $association;
Save-VaultObject -VaultObject $UserGrantee -Workspace $ServerWorkspace;
}
process {
# No process of pipeline
}
end {
# No end activity
}
}
```

The usage section of the script shows sample PowerShell commands. For example:

Get the grantee Vault ID:

```
> $granteeId = Get-VaultId -Name "vault.admin" -Type "User" -Workspace $ws;
```

Get the grantee:

```
> $grantee = Get-VaultObject -VaultId $granteeId -Workspace $ws -Repository $ws.SiteRepository;
```

Get the target ID:

```
> $targetId = Get-VaultId -Name "Global Administrators" -Type "Group" -Workspace $ws;
```

Get the target:

```
> $target = Get-VaultObject -VaultId $targetId -Workspace $ws -Repository $ws.SiteRepository;
```

Publish the workflow actor association:

```
> Publish-WorkflowActorAssociation -ServerWorkspace $ws -UserGrantee $grantee -Actor "Supervisor" -
GroupTarget $target;
```

Load-Assemblies.ps1 Script

The Load-Assemblies.ps1 script loads the Framework assemblies. The script is similar to the following:

```
$script:framework = [System.Reflection.Assembly]::LoadWithPartialName  
("Symyx.Framework");
```

To execute in PowerShell:

```
. .\Load-Assemblies.ps1
```

Publish-WorkflowActorAssociation.ps1 Script

The Publish-WorkflowActorAssociation.ps1 script contains a function called, Publish-WorkflowActorAssociation that publishes a workflow actor association. The script contains code similar to the following listing:

```
$usage = '@'  
  
. .\Load-Assemblies.ps1  
  
. .\Connect-Server.ps1  
  
. .\Create-workflowActor.ps1  
  
. .\Add-Association.ps1  
  
. .\Save-VaultObject.ps1  
  
. .\Get-VaultObject.ps1  
  
. .\Get-VaultId.ps1  
  
. .\Publish-workflowActorAssociation.ps1  
  
$ws = Connect-Server -Server "XXXX" -UserName "vault.admin" -Password  
"XXXX";  
  
$granteeId = Get-VaultId -Name "symyx-ic\vault.admin" -Type "User" -  
Workspace $ws;  
  
$grantee = Get-VaultObject -VaultId $granteeId -Workspace $ws -Repository  
$ws.SiteRepository;  
  
$targetId = Get-VaultId -Name "Global Administrators" -Type "Group" -  
Workspace $ws;  
  
$target = Get-VaultObject -VaultId $targetId -Workspace $ws -Repository  
$ws.SiteRepository;  
  
Publish-workflowActorAssociation -ServerWorkspace $ws -UserGrantee $grantee  
-Actor "Supervisor" -GroupTarget $target;  
  
'@;
```

```
function Publish-WorkflowActorAssociation {
    param (
        [Symyx.Framework.Vault.Vaultworkspace] $Serverworkspace=$(throw "must specify -Serverworkspace"),
        [Symyx.Framework.Vault.User] $UserGrantee=$(throw "must specify -User"),
        [System.String] $Actor=$(throw "must specify -Actor"),
        [Symyx.Framework.Vault.Group] $GroupTarget=$(throw "must specify -GroupTarget")
    )

    begin {
        $actorId = Get-VaultId -Name $Actor -Type "WorkflowActor" -workspace $Serverworkspace;

        $actor = Get-VaultObject -VaultId $actorId -workspace $Serverworkspace -Repository $ws.SiteRepository;

        $association = New-Object Symyx.Framework.workflow.workflowActorAssociation $GroupTarget, $actor;

        Add-Association -User $UserGrantee -Association $association;

        Save-VaultObject -VaultObject $UserGrantee -workspace $Serverworkspace;
    }

    process {
        # No process of pipeline
    }

    end {
        # No end activity
    }
}
```

The usage section of the script shows sample PowerShell commands. For example:

To get the grantee Vault ID:

```
$granteeId = Get-VaultId -Name "vault.admin" -Type "User" -workspace $ws;
```

To get the grantee:

```
$grantee = Get-VaultObject -VaultId $granteeId -workspace $ws -Repository $ws.SiteRepository;
```

To get the target ID:

```
$targetId = Get-VaultId -Name "Global Administrators" -Type "Group" -workspace $ws;
```

To get the target:

```
$target = Get-VaultObject -VaultId $targetId -workspace $ws -Repository $ws.SiteRepository;
```

To publish the workflow actor association:

```
Publish-workflowActorAssociation -Serverworkspace $ws -UserGrantee $grantee -Actor "Supervisor" -GroupTarget $target;
```

Save-VaultObject.ps1 Script

The Save-VaultObject.ps1 script contains a function called Save-VaultObject that saves an object to BIOVIA Vault Server. The script contains code similar to the following listing:

```
function LoadAssemblies {  
  
    $a = @{"Symyx.Framework" = [System.Reflection.Assembly]::LoadWithPartialName  
        ("Symyx.Framework");  
  
    }  
  
    return (,$a);  
  
}  
  
function save ([Symyx.Framework.Vault.VaultObject] $object,  
[Symyx.Framework.Vault.Folder] $folder){  
  
    if($object -ne $null) {  
  
        # we have an object specified  
        $repository = $object.SourceRepository -as  
[Symyx.Framework.Vault.Repository];  
  
        #if($repository -ne $null) {  
  
            if($object.IsManaged) {  
  
                # Existing object, so update  
  
                $repository.Update($object);  
  
            } else {  
  
                # New object so add  
  
                $repository = $folder.SourceRepository -as  
[Symyx.Framework.Vault.Repository];
```

```

    $repository.Add($object, $folder);
}
}
}

function Save-VaultObject ([Symyx.Framework.Vault.Vaultworkspace]
$workspace,
    [Symyx.Framework.Vault.VaultObject] $VaultObject,
    [Symyx.Framework.Vault.Folder] $Folder) {
    begin {
        #write-Host "VaultObject is" $VaultObject.GetType().
FullName;

        if($Folder -eq $null) {
            #Folder is not supplied so use site repository root from
$workspace

            $Folder = $workspace.SiteRepository -as [Symyx.Framework.Vault.Folder];
        }
        save $VaultObject $Folder;
    }
    process {
        if($_ -ne $null) {
            save $_ $Folder;
        }
    }
    end {
        return;
    }
}

$a = LoadAssemblies;

```

For example, you call Create-Vocabulary and then save the vocabulary using Save-VaultObject:

```
> $myVocabulary = Create-Vocabulary -Name "MyVocabulary";  
> Save-VaultObject -VaultObject $myVocabulary -workspace $ws;
```

Set-DefaultTemplate.ps1 Script

The Set-DefaultTemplate.ps1 script contains a function that assigns the default experiment template for a specific user. The script contains code similar to the following listing:

```
$usage = @'  
  
    . .\Load-Assemblies  
  
    . .\Connect-Server  
  
    . .\Get-VaultObject  
  
    . .\Set-DefaultTemplate  
  
    $ws = Connect-Server -Server XXXX -UserName vault.admin -Password XXXX;  
  
    $user = Get-VaultObject -VaultId "User.4849a892-2919-c4d8-c835-  
108dfde8b5a1" -workspace $ws;  
  
    $template = Get-VaultObject -VaultId "DocumentTemplate.f2beb97a-1e08-4b63-  
9e01-08144653843f" -workspace $ws;  
  
    $templateUri = $template.VaultUri.ToString();  
  
    Set-DefaultTemplate $templateUri $user;  
  
'@;  
  
function Set-DefaultTemplate (  
    [System.String] $TemplateVaultUri,  
    [Symyx.Framework.Vault.User] $User  
) {  
    begin {  
        $key = "DefaultExperimentTemplate";  
        $User.Profile.Load();  
        $profile = $User.Profile;  
        $signature = "void SetValue\[T\](System.String, T)";  
        $dictionary = New-Object Symyx.Framework.Vault.VaultDictionary;
```

```

    $dictionary.Title = $key;

    $m = @($dictionary.GetType().GetMethods() | ?{"%{$_}" -match
"$signature"})[-1];

    [type[]] $types = @([System.String]);

    $generic = $m.MakeGenericMethod($types);

    $generic.Invoke($dictionary, @($key, $TemplateVaultUri));

    $profile.Add($dictionary);

    $profile.Save($key);

    return ($profile);

}

process {

}

end {

}

}

```

The usage section of the script shows sample PowerShell commands. For example:

Modify the Vault ID in the following line and then call `Get-VaultObject` to get a user:

```
$user = Get-VaultObject -vaultId "User.4849a892-2919-c4d8-c835-108dfde8b5a1"
-workspace $ws;
```

Modify the Vault ID in the following line and then call `Get-VaultObject` to get a template:

```
$template = Get-VaultObject -vaultId "DocumentTemplate.f2beb97a-1e08-4b63-
9e01-08144653843f" -workspace $ws;
```

Get the template URI:

```
$templateUri = $template.VaultUri.ToString();
```

Set the default template:

```
Set-DefaultTemplate $templateUri $user;
```

Set-Permissions.ps1 Script

The `Set-Permissions.ps1` script assigns the permissions for an object. The script contains code similar to the following listing:

```
function Set-Permissions {

    param (
```

```
[Symyx.Framework.Vault.VaultServer]$Server=$(throw "must specify -
Server"),
[System.String]$User=$(throw "must specify VaultId string for -User"),
[System.String]$Object=$(throw "must specify VaultId string for -Object"),
[System.Int32]$Allow=0,
[System.Int32]$Deny=0
)

begin {

# Set namespaces
$NSPermissions = [Symyx.Framework.Vault.Security.Permissions];
$NSVaultId = [Symyx.Framework.Vault.VaultId];

# VaultId of to user or group you want to give the permissions to
$factorId = New-Object -TypeName $NSVaultId -ArgumentList $User;

# VaultId of the object to set the permissions for
$subjectId = New-Object -TypeName $NSVaultId -ArgumentList $Object;

# I would like to see if these are really still needed;
# it does not seem to me that anyone would want to keep track of them
# $Server.RemoveExplicitPermissions($factorId, $subjectId,
"Administration");
# $Server.RemoveExplicitPermissions($factorId, $subjectId, "bootstrap");

$Server.SetExplicitPermissions($factorId, $subjectId, "Administration",
$allow, $deny);

# Print info
"Allow:";
```

```
$allow;  
  
"Deny:";  
$deny;  
  
"User:";  
$actorIdString;  
  
"Object:";  
$subjectIdString;  
  
}  
  
process {  
# Don't process anything from the pipeline  
}  
  
end {  
# Nothing to do at end  
}  
}
```

Create Print Audit History Script

You can use following example to create a Print-History.ps1 script and adapt the Print History function for your own use.

```
#$ws = Connect-Server $server $user $password  
  
$usage = @'  
. .\Load-Assemblies  
. .\Connect-Server  
. .\Print-History.ps1  
  
$ws = Connect-Server -Server XXXX -UserName vault.admin -Password XXXX;
```

```
$ws "User" "domain\username"

or

$ws (New-Object Symyx.Framework.Vault.VaultId $VaultIdString)

$null =[System.Reflection.Assembly]::LoadWithPartialName
("Symyx.Framework");

function Print-History

(

[Symyx.Framework.Vault.VaultWorkspace] $Workspace,

[String] $Type,

[String] $Title,

[Symyx.Framework.Vault.VaultId] $VaultId =
[Symyx.Framework.Vault.VaultId]::Empty

)

{

if ($VaultId -eq [Symyx.Framework.Vault.VaultId]::Empty){

$list = $Workspace.Get([Symyx.Framework.Vault.VaultObjectTypes]::$Type,
[Symyx.Framework.Vault.DataScope]::Minimal)

if (-not ($list.Contains($Title))){ throw "No object of type $Type found
with Title $Title" }

else {$VaultId = $list.FindByTitle($Title).VaultId}

}

write-Host "Getting history for $VaultId"

$history = $Workspace.GetHistory($VaultId)

Format-Table -Property "HistoryType", "UserName", "Description",
"Context", "ServerTimeStamp", "Version", "ObjectId", "ParticipantId" -
InputObject $history.AllHistory -AutoSize

}

}
```

Create Signature Policy Examples

You can create signature policies using the examples shown below.

In-progress Signature Policy Example

```
# Create a signature policy and set its attributes
$policy1 = new-object Symyx.Framework.Vault.Signing.SignaturePolicy;
$policy1.Title = "In Progress signature";
$policy1.Description = "No authentication required";
$policy1.AreCommentsAllowed = $FALSE;
$policy1.ForceUserToReauthenticate = $FALSE;
$policy1.IsObjectOpenRequired = $FALSE;
#$policy1.MeaningChoices.Add("")
#$policy1.DefaultMeaning
#$policy1.ReasonChoices.Add("")
#$policy1.DefaultReason
$policy1.IsReasonAllowed = $FALSE;
$policy1.IsReasonRequired = $FALSE;
$policy1.RequiresSingleObjectSigning = $FALSE;

# Add the signature policy to Vault

write-Host "Adding signature policy $($policy1.Title) to Vault";
$ws.Add($policy1,$ws.SiteRepository);
```

Completed Signature Policy Example

```
# Create a signature policy and set its attributes

$policy2 = new-object Symyx.Framework.Vault.Signing.SignaturePolicy;
$policy2.Title = "Completed signature";
$policy2.Description = "Used to ";
$policy2.AreCommentsAllowed = $FALSE;
$policy2.ForceUserToReauthenticate = $TRUE;
$policy2.IsObjectOpenRequired = $FALSE;
$policy2.MeaningChoices.Add("Completed; Ready for counter-signature")
$policy2.DefaultMeaning
#$policy2.ReasonChoices.Add("")
#$policy2.DefaultReason
$policy2.IsReasonAllowed = $FALSE;
$policy2.IsReasonRequired = $FALSE;
$policy2.RequiresSingleObjectSigning = $FALSE;

# Add the signature policy to Vault

write-Host "Adding signature policy $($policy2.Title) to Vault";
$ws.Add($policy2,$ws.SiteRepository);
```

Returned signature policy

```
# Create a signature policy and set its attributes
$policy3 = new-object Symyx.Framework.Vault.Signing.SignaturePolicy;
$policy3.Title = "Return to author signature";
$policy3.Description = "Used for sending documents back to authors for
further editing";
$policy3.AreCommentsAllowed = $FALSE;
$policy3.ForceUserToReauthenticate = $TRUE;
$policy3.IsObjectOpenRequired = $FALSE;
$policy3.MeaningChoices.Add("Edits required by author")
```

```
$policy3.DefaultMeaning
#$policy3.ReasonChoices.Add("")
#$policy3.DefaultReason
$policy3.IsReasonAllowed = $FALSE;
$policy3.IsReasonRequired = $FALSE;
$policy3.RequiresSingleObjectSigning = $FALSE;

# Add the signature policy to Vault

write-Host "Adding signature policy $($policy3.Title) to Vault";
    $ws.Add($policy3,$ws.SiteRepository);
```

Standard signature policy

The following snippet shows how to create a “standard” signature policy:

```
# Create a signature policy and set its attributes
$policy4 = new-object Symyx.Framework.Vault.Signing.SignaturePolicy;
$policy4.Title = "Standard signature";
$policy4.Description = "Comments and reasons are allowed";
$policy4.AreCommentsAllowed = $FALSE;
$policy4.ForceUserToReauthenticate = $TRUE;
$policy4.IsObjectOpenRequired = $FALSE;
#$policy4.MeaningChoices.Add("")
#$policy4.DefaultMeaning
#$policy4.ReasonChoices.Add("")
#$policy4.DefaultReason
$policy4.IsReasonAllowed = $TRUE;
$policy4.IsReasonRequired = $FALSE;
$policy4.RequiresSingleObjectSigning = $FALSE;

# Add the signature policy to Vault

write-Host "Adding signature policy $($policy4.Title) to Vault";
    $ws.Add($policy4,$ws.SiteRepository);
write-Host "Adding signature policy $($policy4.Title) to Vault";
    $ws.Add($policy4,$ws.SiteRepository);
```

Countersign signature policy

```
# Create a signature policy and set its attributes
$policy5 = new-object Symyx.Framework.Vault.Signing.SignaturePolicy;
$policy5.Title = "Countersign";
$policy5.Description = "Used to countersign an experiment";
$policy5.AreCommentsAllowed = $FALSE;
$policy5.ForceUserToReauthenticate = $TRUE;
$policy5.IsObjectOpenRequired = $FALSE;
$policy5.MeaningChoices.Add("Read and Understood");
$policy5.MeaningChoices.Add("Observed");
#$policy5.DefaultMeaning;
#$policy5.ReasonChoices.Add("");
#$policy5.DefaultReason;
$policy5.IsReasonAllowed = $FALSE;
$policy5.IsReasonRequired = $FALSE;
```

```
$policy5.RequiresSingleObjectSigning = $TRUE;

# Add the signature policy to Vault

write-Host "Adding signature policy $($policy5.Title) to Vault";
$ws.Add($policy5,$ws.SiteRepository);
```

Message Queuing Function Example

The QueueMessage function example illustrates how to add a message to a queue. You can adapt this function for your own use.

The function accepts three parameters:

- Workspace
- Target Vault ID
- Queue name, the default is vaultqueue.

```
function QueueMessage {
    param (
        [Symyx.Framework.Vault.Vaultworkspace] $workspace = $(throw "must specify
-workspace"),
        [Symyx.Framework.Vault.VaultId] $targetVaultId = $(throw "must specify -
targetVaultId"),
        [System.String] $queueName = "vaultqueue"
    )

    begin
    {
        #Create a message sender on the queue
        $messageSender = [Symyx.Framwork.Vault.Messaging.MessageSender]::Create
($queueName);

        # If the message sender was successfully created (i.e. not null), do...
        if ($messageSender -ne $null)
        {
            # Create a properties collection
            $properties = new-object Symyx.Framework.Properties.PropertyCollection;

            # Set the signature
            $signature = "Void SetValue\[T\](System.Enum, T)";

            # Get the methods from the properties collection
            $m1 = $properties.GetType().GetMethods();

            # Get the signature method from the properties collection
            $m = @($properties.GetType().GetMethods() | ?{"%{$_}" -match
"$signature"})[-1];

            # Get the types
            [type[]] $types = @([Symyx.Framework.Vault.VaultId]);

            # Make a generic method
            $generic = $m.MakeGenericMethod($types);
```

```
# Invoke the generic method and get the object back
$object = $generic.Invoke($properties, @
([Symyx.Framework.Properties.CoreProperty]::Identifier, $targetVaultId));

# Set the message subject
$subject =
[Symyx.Framework.Vault.Messaging.MessageSubject]::Vault.ToString();

# Set the message action
$action =
[Symyx.Framework.Vault.Messaging.VaultAction]::ObjectSaved.ToString();

# Send the message
$messageSender.Send($subject, $action, $workspace.CurrentUser.VaultId,
$properties);
}

return $null;
}

process {}
end {}

}
```

Manage Workbook Folders

You can manage Workbook folders using PowerShell. You must log in as an administrator and create a workspace named \$ws. For more information, see [Initial PowerShell Commands](#).

Create a Folder

To create a folder, the user account must have the CreateFolder permission on the parent folder. A repository has a default root folder that stores all top-level folders.

The following example creates a folder called, *My New Folder*, in the default root folder of an example repository called BIOVIA. You replace the repository with a repository that exists in your implementation:

```
#Get the parent folder
$parentId = Get-VaultID -Name "My New Folder" -Type "Folder" -workspace
$ws;
$parentFolder = Get-VaultObject -VaultId $parentId.toString() -workspace
$ws;

# Get the repository
$repository = $ws.GetRepository("");

# Create the sub-folder
$newFolder = New-Object Symyx.Framework.Vault.Folder;
$newFolder.Title = "My Sub Folder";

# Add new folder to the repository
$repository.Add($newFolder, $parentFolder);
```

To examine the folder using the Vault Administration Console, expand the tree **Console Root > Vault Administration > Vault Server > Repositories**.

The next example creates a sub-folder titled My Sub Folder within My New Folder:

```
# Get the parent folder
```

Vault does not enforce folder name uniqueness. Before adding objects to a folder, verify that you are using the correct folder. For more information, see [Retrieve Folders with the Same Name](#).

Retrieve a Folder Using the Folder Name

The PowerShell code example finds the ID of a folder with the name, My New Folder:

```
$folderId = Get-VaultId -Name "My New Folder" -Type "Folder" -workspace $ws;
```

If multiple folders have the same name, a list of folder IDs is returned. You need to iterate through the list, retrieve each folder, and check the folder VaultPath. For more information, see [Retrieve Folders with the Same Name](#).

The following example retrieves the folder whose Vault ID was retrieved in the previous example:

```
$folder = Get-VaultObject -VaultId $folderId.toString() -workspace $ws;
```

To ensure the folder was retrieved correctly, check the folder's VaultPath attribute:

```
$folder.VaultPath;
```

```
MyCompany\My New Folder
```

VaultPath contains the repository and the full path to the folder.

Retrieve Folders with the Same Name

Vault does not enforce uniqueness on folder names. If multiple folders share the same name, you must check the VaultPath for each folder. For example, assume there are two folders with the title My New Folder, the code below shows how to get the first folder and the folder VaultPath:

```
$folder = Get-VaultObject -VaultId $id[0].toString() -workspace $ws;
```

```
$folder.VaultPath;
```

The following code shows how to get the second folder and the folder VaultPath.

```
$folder = Get-VaultObject -VaultId $id[1].toString() -workspace $ws;
```

```
$folder.VaultPath;
```

Before adding objects to a folder, verify that the VaultPath matches your expected repository and folder locations. The following code checks that VaultPath is MyCompany\My New Folder:

```
if ($folder.VaultPath -eq "MyCompany\My New Folder")
{
    ... code to add objects to the folder
}
```

Manage Workbook Permissions

Objects have permissions that allow a user or group, the grantee, to perform specific actions on that object such as allow JoeUser to read an object. Each object has a set of permissions for each grantee that allows or denies an action.

To view the permissions for another user, you must have ReadPermissions. To modify the permissions for another user, the administrator must have UpdatePermissions. You must log in as the administrator and create a workspace named \$ws. For more information, see [Initial PowerShell Commands](#).

Permissions

Permissions are stored in the C# enumeration `Symyx.Framework.Vault.Security.Permissions`. For more information about the `Symyx.Framework.Vault.Security` namespace, see the Framework API documentation.

To reference the permissions in PowerShell, load the permissions enumeration:

```
$NSPermissions = [Symyx.Framework.Vault.Security.Permissions];
```

The following example displays `$NSPermissions`:

```
$NSPermissions
IsPublic IsSerial Name BaseType

True True Permissions System.Enum
```

The following example displays the Checkout permission, the double colon indicates Checkout is a static property.

```
$NSPermissions::Checkout
Checkout
```

Retrieve Folder Permissions

You can use the `GetObjectExplicitPermissions` method to retrieve an object's permissions. The following example retrieves My New Folder and the folder permissions:

```
$folderId = Get-VaultID -Name "My New Folder" -Type "Folder" -workspace $ws;
$permissions = $ws.ActiveVaultServer.GetObjectExplicitPermissions($folderId);
```

The table shows the members of the `ExplicitObjectPermissions` class:

Attribute	Description
AllowPermissions	List of permissions that are explicitly allowed on the object does not include inherited permissions.
ContextName	Indicates whether the permission was set by Workflow or Administration. You should not remove Workflow permissions. They are managed by the workflow definition and removing them could impact users' ability to work with the object.
DenyPermissions	Lists the permissions that are explicitly denied from the object.
GranteeGuid	Indicates the recipient, user or group, of the permission grant.
VaultId	Indicates the ID of the object.

The following example shows the contents of the permissions object:

```
$permissions
GranteeGuid: User.2bf33dfa-2c02-4a9b-9e81-774fa3f260fb
VaultId: Folder.21f48830-6d06-4562-8f93-f9b5ccf320e4
ContextName: Administration
AllowPermissions: CreateFolder, Delete, Checkout
DenyPermissions: NoPermission
```

In the example above, the original folder that the permissions object references was explicitly granted `CreateFolder`, `Delete`, and `Checkout` permissions using the Vault Administration Console.

To examine an object's permissions by using the Vault Administration Console:

- Right-click the object in the tree, select **Properties**, and click **Permissions**.

Create Read Permissions

In PowerShell, setting permissions on an object is done by:

- Creating a permissions object that stores the set of permissions that are allowed or denied.
- Applying the permissions to the grantee and the target object. This allows or denies the grantee the ability to perform the specific actions on the target object.

The following example loads the permissions and creates a permissions object named `readPermissions`, that contains the `ReadData` and `ReadProperties` permissions:

```
$NSPermissions = [Symyx.Framework.Vault.Security.Permissions];
$readPermissions = $NSPermissions::ReadData -bor
$NSPermissions::ReadProperties;
```

PowerShell binary operators work differently than C# binary operators. The binary OR operator (`-bor`) sets the value. The binary XOR operator (`-bxor`) toggles the current value:

0 is set to 1, 1 is set to 0; true is set to false, false is set to true

Create Write Permissions

The following example creates a permissions object named `writePermissions` that contains the `UpdateProperties` and

`writeData` permissions:

```
> $writePermissions = $NSPermissions::UpdateProperties -bor
$NSPermissions::WriteData;
```

Allow Read and Deny Permissions

Permissions are granted using the `SetExplicitPermissions` method of the `ActiveVaultServer` object. The following example shows how to allow a user to read from a folder, but deny the user from writing to the folder:

```
# Set the "allow" permission to read
$newAllowPermission = $readPermissions;
# Set the "deny" permission to write
$newDenyPermission = $writePermissions;
# Set the permission context to "Administration" (this allows changes to be
made using the Vault Administration Console)
$contextName = "Administration";
# Set the Vault ID of the grantee (the user or group to get the permissions)
$GranteeVaultId = "User.2bf33dfa-2c02-4a9b-9e81-774fa3f260fb";
# Retrieve the folder on which the permissions are set
$folderId = Get-VaultID -Name "My New Folder" -Type "Folder" -workspace
$ws;
$folder = Get-VaultObject -VaultId $folderId.toString() -workspace $ws;
# Set the folder permissions (this allows the user read access to the
folder, but denies write access)
$ws.ActiveVaultServer.SetExplicitPermissions($GranteeVaultId,
$folder.VaultId,
$folder.SourceRepositoryId, $contextName, $newAllowPermission,
$newDenyPermission);
```

Store Variables Using CSV files

You can import comma separated values (CSV) files in PowerShell and create a CSV file with the list of variables folders, users, groups, and permissions that you want to manage. You can import the CSV file and reference the variables in your PowerShell scripts. With the CSV files as the source, you can write generic PowerShell scripts that do not contain hard-coded values. The following table illustrates a possible CSV layout to manage folders and permissions:

Column	Description	Example
FolderTitle	Title of the folder to be processed.	123456-2009
ParentFolderTitle	Title of the parent folder.	MyParentFolder
RepositoryTitle	Title of the repository for the folder.	MyFolder
ParentFolderVaultPath	Full path of the parent folder.	Mycompany\Mycompanyfolder\123456-2009
SetPermission	Flag to indicate whether permissions are to be set on the folder (Yes/No).	Yes
AllowPermissionToSet	List of allowed permissions.	read
DenyPermissionToSet	List of denied permissions.	write
UserName	User ID of grantee in the form domain\user name.	vm-vault65\S.Smith
GroupName	Group title of grantee.	MyGroup

You can import a CSV file using the `Import-Csv` command. For more information on importing CSV files, in the PowerShell window, enter `get-help Import-Csv`.

Appendix B:

Audit Trail Actions

The following tables lists the actions that result in the creation of an audit trail.

See Audit trail actions in the Site repository lists of actions taken within the Vault Administration Console. These actions do not display in an Audit History report created within BIOVIA Workbook.

Type of Action	Actions
Site repository	System repository that manages your intellectual property
Users	<ul style="list-style-type: none"> ■ Add Vault users ■ Make a user inactive ■ Add a permission to a user ■ Remove a permission from a user ■ Deny permission to a user ■ Add a user to a group ■ Remove a user from a group ■ Assign a Workflow actor to a user ■ Remove a Workflow actor from a user ■ Subscribe a user to a repository ■ Unsubscribe a user from a repository ■ Assign repository permissions to a user
Groups	<ul style="list-style-type: none"> ■ Add permissions to a group ■ Remove permissions from a group ■ Deny permissions to a group ■ Add a group to another group ■ Remove a group from another group ■ Add members to a group ■ Remove members from a group ■ Subscribe a group to a repository ■ Unsubscribe a group from a repository
Objects	<ul style="list-style-type: none"> ■ Modify permissions on a Vault object ■ Assign repository folder permissions to a user or group
Applications	Assign application permissions to a user or group
Workflow	<ul style="list-style-type: none"> ■ Create a Workflow actor ■ Add a new Workflow definition to Vault ■ Remove a Workflow definition

Type of Action	Actions
	<ul style="list-style-type: none"> ■ Generate a Workflow association ■ Disable a Workflow association ■ Remove a Workflow association

Action Types

The following table contains a list of action types. For each action type, there are multiple auditable actions, for example, for a Content type the actions include add, change, delete, and save.

Type of Action	Actions
Versioned Repository-related	Versioned repositories contain your intellectual property
Document	Document-related actions that create an audit history entry
Experiment Title	Data:name
Experiment Properties	<ul style="list-style-type: none"> ■ Notebook ■ Association Target ■ Associations ■ Auto Name ■ References ■ Check-In Signature Policies ■ Availability ■ Class (Initial) ■ Class (Current) ■ Creation Date ■ Size ■ Content Writer ■ Contributors ■ Scope ■ Created By ■ Descriptions ■ Flags ■ ID ■ Language ■ Permissions ■ Preview ■ Published By ■ Related To ■ Rights ■ Initial Check-In Date

Type of Action	Actions
	<ul style="list-style-type: none"> ■ Source ■ Subject ■ Synchronization Revision ■ Name ■ Total Content Size ■ Type ■ Vault Path ■ Version Comment ■ Version Creation Date ■ Version Created By ■ Current Version ■ Workflow Stages ■ Scripts
Container Membership	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description
History -History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description

Type of Action	Actions
Signature History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description ■ Meaning ■ Reason ■ Comment ■ Signature Source ■ Transition
Version History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ CheckOut ■ UndoCheckout ■ Description
Workflow History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description ■ Stage
Combined History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description ■ Entry Type

Type of Action	Actions
System History	<ul style="list-style-type: none"> ■ Scope ■ Version ■ Modified On ■ Modified By ■ Content ■ Description
References	<ul style="list-style-type: none"> ■ Reference ■ Description ■ Last Modified By ■ Last Modified On ■ Type ■ Applies To ■ Vault Path ■ Autaname