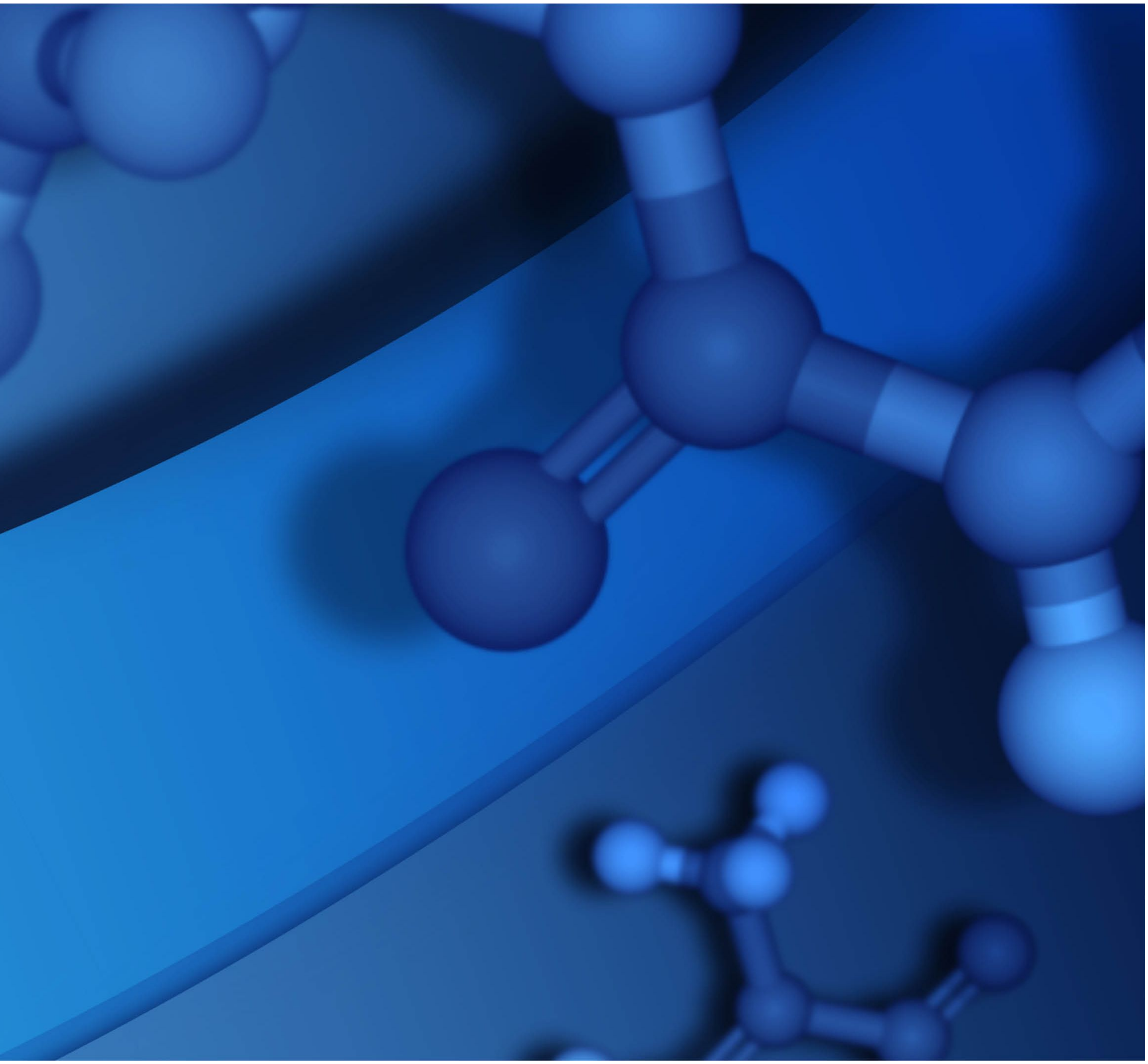
WORKBOOK ADMINISTRATION GUIDE

BIOVIA WORKBOOK 2021



Contents

Chapter 1: General Configuration Tasks

This chapter provides instructions and guidance to allow you to achieve common configuration tasks.

Creating a Login Domain List

You can create a list of connected machines and limit Workbook package downloads to the listed machines. To do this, you edit the login domain list XML file and then a run the commands on each client.

To create a list of connected computers and limit package downloads:

1. In a command window, navigate to the folder where `NotebookClientManager.exe` resides. By default, this is:

   ```
   C:\Program Files (x86)\BIOVIA\BIOVIA Workbook Client Manager\bin
   ```

2. Open a command line prompt and run the following command to see the list of arguments that you can specify when executing the Workbook client, `NotebookClientManager.exe`:

   ```
   NotebookClientManager.exe /?
   ```

3. Create an MRU xml file that identifies the user, the user's domains, and the user's servers by replacing `SomeUser`, `SomeDomain`, `SomeServer`, and `SomeOtherServer` in the following example with values appropriate for the user whose Workbook client machine you are configuring:

   ```
   <?xml version="1.0" encoding="utf-16"?>
     <MostRecentlyUsed
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <DisallowUserEnteredServer>true</DisallowUserEnteredServer>
       <IsLocked>true</IsLocked>
       <UserName>
         <string>SomeUser</string>
       </UserName>
       <Domains>
         <string>SomeDomain</string>
         <string>SomeOtherDomain</string>
   ```

```
    </Domains>
    <Servers>
      <string>SomeServer</string>
      <string>SomeOtherServer</string>
     </Servers>
  </MostRecentlyUsed>
```

4. Name and save the file to disk, for example:

   ```
   C:\Temp\MyMru.xml
   ```

5. Execute the Workbook client using the `/install-mrus` command line argument to install the new MRU file:

   ```
   NotebookClientManager.exe /install-mrus C:\Temp\MyMru.xml
   ```

6. Verify the setup by launching the Workbook client.

   The login dialog box should be pre-populated with the values you entered in the file.

7. Repeat this procedure on each user's client computer.

To remove the download limiting feature:

- Use the `/clear-mrus` command line parameter to clear the computer-level MRUs and user-level MRUs.

  The `/clear-mrus user` command only clears the MRU for the currently logged-on Windows user, rather than for all users.

Configuring a Proxy Service

To configure a proxy service for the Database Web Service:

1. Back up the `Symyx.Notebook.Application.exe.config` file making any changes to the file.
2. Open the `Symyx.Notebook.Application.exe.config` file in a text editor.
3. Add the `defaultProxy` enabled setting to the `Symyx.Notebook.Application.exe.config` file, for example:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <configuration>
   ...
     <system.net>
       <defaultProxy enabled="true" useDefaultCredentials="true">
       </defaultProxy>
     </system.net>
   </configuration>
   ```

   If the Lookup service fails to function, change the settings as follows:

   ```
   <system.net>
     <defaultProxy enabled="true" useDefaultCredentials="true">
       <proxy usesystemdefault=”True”/>
      </defaultProxy>
   </system.net>
   ```

4. If the Lookup service still fails to function, add proxy server exceptions for `*.discoverygate.com`.
5. Save and close the file.

Caching

BIOVIA uses two caching to improve performance and response time:

- Assembly cache

  The assembly cache is common to all users on a specific client. The assembly cache contains BIOVIA Workbook assemblies that were downloaded from Vault. The assemblies are Section assemblies, form Sections, Material Sections, File Sections, and might include custom assemblies used by scripts or other extensions. The assembly cache is located in the following directory:

  ```
  %SYSTEMDRIVE%\ProgramData\Symyx Technologies\AssemblyCache
  ```

- Object cache

  The object cache is isolated by user account on the client. The object cache contains VaultObjects that were accessed by the user. The object cache includes documents, sections, forms, signature policies, and vocabularies. The object cache helps improve response time by reducing the number of calls to the server. The object cache is located in the following directory:

  ```
  %SYSTEMDRIVE%\ProgramData\Symyx Technologies\ObjectCache
  ```

Maintaining Caches

It might necessary to clear the object or assembly caches periodically. The reasons include:

- Switching between environments that are based on a cloned database, that is. copying Production database to Test database.
- Object cache uses too much disk space.
- Clean up the object cache after disabling caching in 6.9x.

Object Cache PowerShell Example

The following PowerShell command deletes the `ObjectCache` hierarchy under in user folders in the Symyx Technologies folder without affecting the local storage folders.

```
Remove-Item "C:\ProgramData\Symyx Technologies\*\*\ObjectCache" -Recurse
```

Assembly Cache PowerShell Example

The following PowerShell command clears all items stored in the AssemblyCache. This has an effect on performance in subsequent BIOVIA Workbook sessions as the assembly cache is rebuilt.

```
Remove-Item "C:\ProgramData\Symyx Technologies\AssemblyCache" -Recurse
```

Specifying ChemDraw as the Global Default Structure Editor

Workbook supports two structure editors: BIOVIADraw and Perkin Elmer's ChemDraw. BIOVIA Draw is the default structure editor.

The default structure editor can be changed two ways:

- An administrator can change the *global* default to ChemDraw. When this is done, the first time a user logs in to Workbook from a client system, ChemDraw instead of BIOVIA Draw is automatically set as that user's default structure editor for that system.
- A user can change his or her *individual* default editor setting on the system by using the Workbook Preferences screen.

Because preference settings are saved on a per-user, per-system basis, users who have different preferences can share the same system without losing their preferences, and the same user can select different preferences for different systems.

For instructions for setting individual user preferences, see the Workbook client's online help.

To specify ChemDraw as the global default structure editor:

1. Open a command prompt and navigate to the `<Vault Installation directory>\BIOVIA\Vault\Utilities` directory.
2. Use `VaultADMStoreManager` with the generate command to generate a working directory for the profile.
3. Open the `Symyx.Notebook.Application.exe.config` file. If Workbook is open, first close it.
4. In the `applicationSettings` section, locate the `defaultStructureEditor` element.
5. Change the `displayName` and `chemDrawKitType` attributes to the supported ChemDraw version that is installed on client computers, for example:

   ```
   <defaultStructureEditor displayName="ChemDraw <version_number>"
   chemDrawKitType="<fully_qualified_assembly_name>"/>
   ```

   Use the appropriate ChemDraw `<version_number>` that you have available and the `<fully_qualified_assembly_name>`.
6. Save and close the `Symyx.Notebook.Application.exe.config` file.

Enabling the Citrix Clipboard to Support Chemical Structure Data

If you use BIOVIA Draw (or other chemical structure editors) installations on a Citrix server and need to cut-and-paste structures, you must update the Citrix server registry.

Otherwise, when a user copies a structure, the image of the structure is copied, but the data used to render it is not copied, and the image cannot be edited.

Note: To copy chemical structures from a Workbook installation on a Citrix server to a locally installed version of a Microsoft Office application, you must have a locally installed version of BIOVIA Draw.

To add chemical structure clipboard formats to a Citrix server:

1. Log on to the Citrix server computer as an adminstrator and select **Start** > **Run**.
2. Type **regedit**, and then click **OK**.
3. In the Registry Editor, navigate to:

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Citrix\wfshell\
   Virtual Clipboard\Additional Formats
   ```
4. Add the **Additional Formats** key, and then add the following subkeys and values:

Subkey Name	Type	Value	Comment
ChemDraw Interchange Format	String	ChemDraw Interchange Format	Supports ChemDraw
Data Object	String	Data Object	
Embed Source	String	Embed Source	
Embedded Object	String	Embedded Object	

Subkey Name	Type	Value	Comment
MDLCT	String	MDLCT	Supports BIOVIA Draw
MDLNative	String	MDLNative	
MDLSK	String	MDLSK	Supports BIOVIA Draw
Object Descriptor	String	ObjectDescriptor	
OLE Private Data	String	Ole private Data	

5. Navigate to:
   `[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Server\Compatibility\Applications`
6. Add the **AccelrysDraw** key, and then add the following subkeys and values:

Subkey Name	Type	Value	Comment
Flags	DWORD (32-bit) Value	dword:0000000f	Supports pasting chemical structures and reactions
OpenClipboardDelayInMilliSecs	DWORD (32-bit) Value	dword:00000050	Supports pasting chemical structures and reactions
OpenClipboardRetries	DWORD (32-bit) Value	dword:00000020	Supports pasting chemical structures and reactions

7. Save your changes and and close the Registry Editor.

Specifying the Initial Frequency of Auto Save

BIOVIA Workbook can automatically save active documents during a session based on the *Auto save* frequency value.

You can specify the initial value of the auto save frequency in the `Symyx.Notebook.Application.exe.config` file. The initial value is zero (0) minutes, as a result, the active documents are not automatically saved.

Scientists can set user preferences in the BIOVIA Workbook Preferences that override the initial value that you specify in the `Symyx.Notebook.Application.exe.config` file.

To change the initial value for the Auto save frequency:

1. Open the `Symyx.Notebook.Application.exe.config` file in a text editor.
2. In the `applicationSettings` section, change the `InitialAutoSaveInterval` value to the time interval, in minutes, to automatically save active documents, for example:

```
<setting name="InitialAutoSaveInterval" serializeAs="String">
  <value>5</value>
</setting>
```

3. Save and close the `Symyx.Notebook.Application.exe.config` file.

Configuring the Default Open Behavior

By default, when a user with sufficient permissions double-clicks a document in the Explorer or on their Home page, Workbook checks out the document and opens it for *edit*. With this default behavior, users who want to open the document as *read-only* have to be aware of the right-click "Open (Read-only)" option.

Opening a document for edit and then discarding it is categorized as a data-loss risk, even if the experiment was opened in edit mode unnecessarily or by mistake. To reduce the number of such discards, you can change the default setting of the `DefaultOpenBehavior` parameter in the `Symyx.Notebook.Application.exe.config` file from `WriteableIfPossible` to a more restricted setting.

To change the default document open behavior:

1. Access the Working Directory of the ADM Profile used for Workbook Client.
2. Open the `Symyx.Notebook.Application.exe.config` file in a text editor.
3. Locate the `Symyx.Notebook.Application.Properties.Settings` section and change the DefaultOpenBehavior property's setting to one of the following:
   - **WritableIfPossible:** If the user has sufficient permissions, open the document for edit and, if necessary, check it out. If sufficient permissions do not exist, open it in read-only mode. Workbook uses this by default, if the `DefaultOpenBehavior` element is not in the app.config file.
   - **ReadOnly:** Always open in read-only mode.
   - **WritableIfNewOrCheckedOutToMe:** If the document exists in the user's private repository, open it for edit. Otherwise, open it as read-only.
   - **WritableIfAuthor:** If the document exists in the user's private repository, or if the user is an author (Creator or VersionCreator) of the document, open it for edit and, if necessary, check it out. Otherwise, open it as read-only.

   Example that sets ReadOnly

   ```
   <?xml version="1.0" encoding="utf-8" ?>
       <configuration>
        ...
       <applicationSettings>
           ...
           <Symyx.Notebook.Application.Properties.Settings>
            ...
              <DefaultOpenBehavior="ReadOnly">
           </Symyx.Notebook.Application.Properties.Settings>
       ...
   ```
4. Save and close the file.

Configuring Widgets

In Workbook you can use widgets to perform a number of operations.

Add a Folder for a Widget

You must have `Widget.Administrator` permission to manage widgets.

To add a folder for a widget:

1. In Notebook Explorer, click the **Home** tab.
2. On the **Home** tab, click the **Manage Widgets** Manage.
3. In **Manage Widgets**, click **Add Folder**.

4. In **Add New Folder**, in **Enter a new folder name**, type a name for the folder.
5. Click **OK**.

Publish a Widget

You must have `Widget.Administrator` permission to manage widgets.

You can publish to a Widgets folder or to a subfolder.

To publish a widget:

1. From the BIOVIA Workbook **Home** tab, select a widget, click **Widget Settings** .
2. In **<widget_name> Settings**, modify the configuration settings, and click **OK**.
3. Click **Publish** .

Running RequeueVaultObjects

After you add viewable columns in Notebook Explorer, run the `Requeuevaultobjects` utility to enable existing experiments to display the added columns. For more information, see "Customize Viewable Columns for Data" in the *BIOVIA Workbook Administration Guide*.

To run the RequeueVaultObjects utility:

1. Open a command window.
2. Navigate to the `RequeueVaultObjects.exe` utility, which resides in a folder similar to the following:

   ```
   C:\Program Files (x86)\BIOVIA\Vault\Utilities
   ```

3. Execute the following command once for each versioned repository and each relevant status:

   ```
   Requeuevaultobjects.exe -repository <RepositoryName> -queues
   propertyExtractionMessageHandler -status <Status>
   ```

   where:

   <RepositoryName> is the versioned repository name, enclosed in quotation marks if it contains spaces. Example: "Research and Development".

   <Status> is one of the following:

   - Processed. Always run the command for this status.
   - Unknown. Always run the command for this status.
   - Failed. Run the command if the repository contains objects that have this status and those objects contain data for the viewable columns that were added.

4. Open an experiment in Notebook Explorer and verify that the columns are displayed.

Changing the Experiment Editor Context Menu Options

You can control some of the menu options that scientists normally get when they right-click an experiment, template, or form in the Experiment Editor by changing the corresponding setting values in the `Symyx.Notebook.Application.exe.config` file.

In the following example the **Close Tab**, **Close Tab and Save**, and **Close Tab And Save New Version** options have been hidden by changing their setting values from their default, `False`, to `True`:

```
<applicationSettings>
  <Symyx.Notebook.Properties.Settings>
```

```
    <setting name="HideCloseTabContextMenuItem"serializeAs="String">
       <value>True</value>
    </setting>
    <setting name="HideCloseTabAndCheckInContextMenuItem"
             serializeAs="String">
      <value>True</value>
    </setting>
    <setting name="HideCloseTabAndSaveContextMenuItem"
             serializeAs="String">
      <value>True</value>
    </setting>
    <setting name="HideSaveNewVersionContextMenuItem"
             serializeAs="String">
      <value>True</value>
    </setting>
    <setting name="HideSaveContextMenuItem" serializeAs="String">
      <value>True</value>
    </setting>
...
</Symyx.Notebook.Properties.Settings>
```

The following table identifies the elements in the file that control availability of menu options.

Element	Affected Options and Menus
Hide **CloseTab** ContextMenuItem	**Close Tab** option on the Browser style menu and on the traditional file menu
Hide **CloseTabAndCheckIn** ContextMenuItem	**Close Tab and Check In** option on the traditional file menu; **Close Tab** and **Save New Version** options in the Browser style menu
Hide **CloseTabAndSave** ContextMenuItem	**Close Tab** and **Save** options in the Browser style menu
Hide **SaveNewVersion** ContextMenuItem	**Save New Version** (**Check in** and **Checkout**) options on the traditional file menu
Hide **Save**ContextMenuItem	**Save** option on the traditional file menu
Disable **CloneOnCheckedOut** Documents	**Clone** and **QuickClone** options on the Browser menu are disabled for documents that are checked out if this element is set to True
CloneToLatestEnabled	**Clone** and **QuickClone** options on the Browser menu are *both* available if **CloneToLatestEnabled** is **True**; only **Clone** appears if it is **False**

Configuring Enumeration with Salt Stripping

BIOVIA Workbook does not automatically strip away the salts from enumerated products. You can create or configure a profile to the preference of the scientists using a reaction scheme section.

To create a profile that strips salts:

1. Use the Deployment Manager to execute the GENERATE command and create a working directory for the user profile that stripping salts as the default behavior.
2. In the working directory, create a backup copy of the `Symyx.Notebook.Application.exe.config` file.
3. Use SQL*Plus or a similar utility to log in to the Oracle Vault Site schema and run the following command:

   ```
   select name from vaultobject where name like '%Enumeration.Cheshire%';
   ```

4. Write down the exact version number that is similar to the following:

   ```
   Name \Accelrys.Notebook.Enumeration.CheshireEnumerator, Version=6.9,
   Culture=neutral, PublicKeyToken=fb4b5791c48b7e8a
   ```

5. In the `<setting>` element, edit the name attribute to the exact same version number such as:

   Before

   ```
   <setting name="EnumerationEngine" serializeAs="String">
     <value>Symyx.Notebook.Sections.ReactionScheme.CheshireEnumerator.
        CheshireEnumerationEngine, Symyx.Notebook.Sections.
            ReactionScheme.CheshireEnumerator</value>
   </setting>
   ```

   After

   ```
   <setting name="EnumerationEngine" serializeAs="String">
     <value>Accelrys.Notebook.Enumeration.CheshireEnumerator.
            CheshireEnumerationEngine, Accelrys.Notebook.Enumeration.
            CheshireEnumerator, Version=6.9, culture=neutral,
            publicKeyToken=fb4b5791c48b7e8a</value>
   </setting>
   ```

6. Run the `Symyx.Notebook.Application.exe` file in the working directory to test the modified configuration file.
7. Create a new profile using the ADM `ADD-PROFILE` command, or update a current profile using the `UPDATE-PROFILE` command.

   Note: Do not use a plus (+) character or other special characters in package or profile names. If you do so, when a user whose account uses that package or profile attempts to log on, the system displays an message that says it cannot log them on to Vault due to an error on the server.

8. (Optional) If you created a new profile, do the following:
   a. Use the ADM `DOWNLOAD-RULES` command to download the current Deployment Manager rules to a file.
   b. Edit the rules to assign the new profile to a user, group, or as the default.
   c. Use the `UPLOAD-RULES` command to update the current rules in BIOVIA Vault Server.
   d. Assign that profile as needed.
9. Deploy the new configuration.

   For more information on creating a release profile with this configuration, see *BIOVIA Deployment Manager*.

10. Provide information to users about the new configuration. Scientists using a reaction scheme section can remove the Strip salts from products option that is used is set to strip salts.

Experiment Template Scripting Restrictions

As an experiment template author, you can use the Experiment Editor to create Python scripts that respond to events in experiments created from the experiment template containing the scripts.

You can add toolbar buttons to experiments and create Python scripts that execute when a toolbar button on-click event occurs. For more information about scripting in the BIOVIA Workbook, see the *BIOVIA Workbook SDK Developers Guide*.

Do not use scripts in the following sections that are for internal use only:

- Grouped Materials
- Plate Layout
- Reaction List
- Spreadsheet

Changing the Default View Mode for Vault Server Documents

By default, when you open a document in **Editing Mode**, it opens in Section Layout. When you open a document in **Read-Only Mode**, it opens in Continuous Layout. You can change these defaults on each Vault Server Template.

To change the default View Mode in a Vault Server Template:

1. Open an existing Template or create a new Template.
2. Click on **Tools > Preferences > Document Tab**.
3. Under the **Views** section, you will see the following:
   - View mode when reading document: **Continuous**
   - View mode when editing document: **Section**
4. Use the pull-down menu to change the **View Mode** from **Continuous** to **Section**.

Now, when you open documents created by the above template, they display in **Section** layout. When you open documents not created by the above template, they display in **Continuous** layout.

The setting for the display mode stores with the document. The settings will remain the same for all users, unless changed by other users. Any documents created prior to changing this setting in the Template will use the original default preferences.

Chapter 2: Configuring Log Files

This chapter provides information about managing log files on Workbook client systems. By default, log-in operations are logged, but you can optionally change profiles or configuration files to also log other client operations.

Controlling Logging of Workbook Client Events

By default, log-in events are logged, but check-in, check-out-and-open, and script execution events are not logged.

If needed, you can change the logging defaults by using either of the following methods:

- To implement logging changes for many users, create a new deployment profile. For more information, see the *Deployment Manager Guide*.
- To implement temporary logging changes for specific users on specific client machines, Edit the logging settings in a **`Symyx.Notebook.Application.exe.config`** file, as described below.

To edit log settings in `Symyx.Notebook.Application.exe.config`:

1. Navigate to the package folder on the client system, for example:

   ```
   C:\ProgramData\Symyx Technologies\Deployments\Namespace\Package_Workbook_
   <release>
   ```

2. Find and copy the profile file of the user whose logging options you need to change, which is named using the following syntax:

   ```
   <windows domain>.<windows.username.<vault_domain>b.<vault_user>.config
   ```

3. In a text editor, open the copy of the user's profile config file and change the value of `OtherPerformanceCounters` to **True**.

   This setting makes the system generate log entries for check-in, check-out-and-open, and script execution events.

   Note: A separate setting, `LoginPerformanceCounters`, controls whether system generates log entries for log-in events.

4. If the folder already contains a `Symyx.Notebook.Application.exe.config` file, make a backup copy.

   IMPORTANT! This file provides settings for numerous functions, not just for logging. For more information, see the *Deployment Manager Guide*.

5. Rename your edited copy of the user profile file to exactly the following:

   ```
   Symyx.Notebook.Application.exe.config
   ```

6. Instruct the user to start the Workbook client by executing `Symyx.Notebook.Application.exe` from Windows Explorer, instead of by using the Windows Start > Programs option, which executes `NotebookClientManager.exe`.
   - When the user executes Symyx.Notebook.Application.exe, Workbook uses the settings in the Symyx.Notebook.Application.exe.config file.
   - When the user executes NotebookClientManager.exe, Workbook recreates the user's regular profile and uses its settings.

Location of Client Log Files

BIOVIA Workbook client computer log files are located in sub-directories under the `\\ProgramData\Symyx Technologies\LogFiles\Diagnostics`. The usage log files are as follows:

- CheckIn
- CheckOutAndOpen
- Login
- ScriptExecution

Login Performance Log File

The login performance log file captures each successful login response time for BIOVIA Vault Server. This feature allows the administrator to track client login time to determine if performance problems occur over time.

Log in time starts from the moment the user clicks the OK button on the login screen to the moment Notebook Explorer becomes visible. The login time is not collected for the following cases:

- The BIOVIA Vault Server login fails.
- The user logs into BIOVIA Vault Server in offline mode.
- The user logs into the BIOVIA Vault Server.

The login performance log file is named YYYY-clientName-userid-Login.log, and contains the following information:

Column Heading	Description
Date	Specifies the date of the login, using the Coordinated Universal Time (UTC) format.
Start Time (GMT)	Specifies the start time of login, using the UTC format.
UTC Offset	Specifies the time offset from UTC. It is generally given as hour and minute.
User	Specifies the domain\username for the user logs in.
Server	Specifies the server name for the login.
Client	Specifies the computer name of the client.
Version	Specifies the BIOVIA Vault Server version number.
Response Time (seconds)	Specifies the elapsed time for the login, in seconds.

Checkout and Open Performance Log File

The checkout and open performance log file captures the time it takes to open and check-out Vault objects. You can track operation performance and determine if performance problems occur over time. The following operations are logged:

- Creating a new document
- Opening an existing document
- Checking out an existing document
- Creating and opening other types of Vault objects such as operation, form, report template, report header/footer template

The checkout and open performance log file is named YYYY-clientName-userid-CheckOutAndOpen.log, and contains the following:

Column Heading	Description
User	Specifies the domain and username (domain\username) for the currently logged in user.
Mode	Indicates if the document that is opening is editable (READ AND WRITE) or in read-only (READ) mode.
Date	Specifies the start date of operation, using the Coordinated Universal Time (UTC) format.
Start Time (GMT)	Specifies the start time of operation, using the UTC format.
UTC Offset	Specifies the time offset from UTC. It is generally given as hour and minute, for example, (UTC-08:00).
Client	Specifies the client computer name.
Operation	Specifies one of the following operations performed by the user: ■ Checkout ■ SectionLoad ■ RunScript ■ Open
Object Name	Specifies the name or title of the Vault object.
Object Type	Specifies the type of the Vault object. For example, Document, Form, Operation.
Object Source	Specifies the source of the Vault object. For example, Blank Experiment.
Size	Specifies the total content size of the document being opened.
Operation Time (seconds)	Specifies the number of seconds taken to complete the operation.
Elapsed Time (seconds)	Specifies the number of seconds since the user started the activity.

Checkout and Open Performance Log Operations

The check-out and open performance log file captures the time it takes to open and check-out Vault objects. You can track operation performance and determine if performance problems occur over time. The operations are:

Operation	Description
CheckOut	The CheckOut operation is the first operation that happens when the document is being opened in editable mode. The time taken to check-out a document for editing is logged as a sub-operation within an Open operation.
SectionLoad	When the document the user has selected to open has continuous layout, the SectionLoad operation happens before the completion of open operation. If the document has a section layout, only the section that has focus is entered in the log. The log entry is made after the completion of the Open operation.
RunScript	The RunScript operation happens when scripts run at the document or section level during the Open operation. Scripts that run after the open operation is completed are not logged.
Open	The Open operation is the operation measured starting from the user initiating the action to open a document until it is displayed in the editor. The Open operation includes the CheckOut, SectionLoad and RunScript operations.

Check-in Performance Log File

The check-in performance log file captures the time it takes to check-in and transition Vault objects. You can track the performance of check-in operations and determine if performance problems are occurring. The following operations are logged:

Check-in operations:

- A user checks in a new document.
- A user checks in an existing document.
- A user checks in a document and keeps it checked out.
- Creating and opening other types of Vault objects such as operation, form, report template, report header/footer template

Transition operations:

- A user right-clicks a document and selects a new transition stage.
- A user initiates a transition when checking in a document

The check-in performance log file is named YYYY-clientName-userid-CheckIn.log, and contains the following:

Column Heading	Description
User	Specifies the domain and username (domain\username) for the currently logged in user.
Mode	Indicates if the document that is opening is editable (READ AND WRITE) or in read-only (READ) mode.

Column Heading	Description
Date	Specifies the start date of operation, using the Coordinated Universal Time (UTC) format.
Start Time (GMT)	Specifies the start time of operation, using the UTC format.
UTC Offset	Specifies the time offset from UTC. It is generally given as hour and minute, for example, (UTC-08:00).
Client	Specifies the client computer name.
Operation	Specifies one of the following operations performed by the user: ■ CheckInStatusDialog ■ TransitionStatusDialog ■ Add ■ SaveExisting ■ CheckIn ■ Transition
Object Name	Specifies the name or title of the Vault object.
Object Type	Specifies the type of the Vault object. For example, Document, Form, Operation.
Object Source	Specifies the source of the Vault object. For example, Blank Experiment.
Object Count	Provides the number of Vault objects processed in a batch.
Operation Time (seconds)	Specifies the number of seconds taken to complete the operation.
Elapsed Time (seconds)	Specifies the number of seconds since the user started the activity.
Transitioned from	Specifies the previous transition stage of a transition operation.
Transitioned to	Specifies the new transition stage of a transition operation.

Check-in Operations

The check-in performance log file captures the time it takes to check-in Vault objects. You can track operation performance and determine if performance problems occur over time. The operations are:

Operation	Description
CheckInStatusDialog	Specifies the pre-checkin operation during which the user opens the selection dialog for Repository Selection, Transition, and/or Signing. The `CheckInStatusDialog` operation does not include the time the user spent in making a selection for a Repository, Transition, or Signing.
TransitionStatusDialog	Specifies the pre-transition operation during which the user opens the selection dialog for Transition Stages Selection or Signing. The `TransitionStatusDialog` operation does not include the time the user

Operation	Description
	spent in making a selection for a Repository, Transition, or Signing.
Add	Specifies an operation that check-in a document that is new to the repository.
SaveExisting	Specifies an operation that updates a managed Vault object by adding the changes to the repository.
CheckIn	Specifies an operation that checks in the document to the repository.
Transition	Specifies the operation during which a document is transitioned from one stage to another.

Script Execution Performance Log File

The script execution performance log file tracks the run times of scripts that are defined in the experiment's event scripting or section properties, and as events in forms.

The script execution performance log file is named YYYY-clientName-userid-ScriptExecution.log. The following table describes contents in the script execution performance log.

Column Heading	Description
User	Specifies the domain and username (domain\username) for the currently logged in user.
Date	Specifies the start date of operation, using the Coordinated Universal Time (UTC) format.
Start Time (GMT)	Specifies the start time of operation, using the UTC format.
UTC Offset	Specifies the time offset from UTC. It is generally given as hour and minute, for example, UTC-08:00.
Client	Specifies the client computer name.
Experiment/Experiment Template name	Specifies the name of the experiment or experiment template with the script.
Object Name	Specifies the name or title of the Vault object.
Event	Specifies the name of the event that triggered the script execution, for example, `OnApplicationLoad`.
Object Type	Specifies the type of the Vault object, for example, Document, Form, Operation.
Object Source	Specifies the Vault object where the script is defined: ■ If the script is defined in a Form, Source is the Form name. ■ If the script is defined in an experiment, Source is the experiment name. ■ If the script is defined in a section, Source is the section name.
Execution Time (seconds)	Specifies the number of seconds taken to execute the script.

Chapter 3:
Configuring Experiment and Form Cloning Options

This chapter provides information about controlling what happens when a scientist clones an experiment.

Methods and Settings for Experiment Cloning

By default, Workbook provides two ways to clone an experiment:

- **Quick Clone:** Instantly creates a copy of the experiment.
- **Clone:** Displays a dialog box that lists the experiment's sections so that scientists can choose what to clone.

In both cases, the default Workbook behavior is to find the most recent version of the relevant template, create a new experiment based on that template, and then merge the content of the source experiment into the new experiment.

Global Setting for Cloning to Latest

The `CloneToLatestEnabled` setting controls whether Workbook looks for the most recent template:

- If it is **True** (the default), the latest template version is always used and both cloning methods, **Clone** and **Quick Clone**, are available.
- If it is changed to **False**, cloned experiments always use the same template version as the experiment being cloned, and scientists cannot choose which sections and data to clone.

In both cases, a template administrator can set section-level cloning properties to impose limits on exactly what sections and section data can be cloned.

Changes to the setting must be implemented through deployment profiles. For more information, see Disabling and Enabling Clone-to-latest.

Section-level Cloning Properties

Template administrators can use the following properties to control whether and how each section in a template can be cloned when experiments based on that template are cloned:

- **Allow Section Delete**: Controls whether scientists can exclude the section from the clone.

  If the section must always be cloned, set this property to `False`.
- **No Clone**: Controls whether scientists can copy the section's data from their source experiment to the clone.

  If the section data must never be copied, set this property to `True`.
- **Allow Clone Without Data**: If **No Clone** is `False`, controls whether a section can be copied without its data also being copied.

Any experiment sections that are not in the template can be omitted, cloned with data, or cloned without data.

Notes:
In templates from **pre-6.8** releases of Workbook, `No Clone` was `false` for all sections. If such a template section is updated by changing its `No Clone` section property to *true*, the data for later clones of experiments based on the updated template get will get the section's data from the previous version of the template, instead of from the experiment.

If a scientist renames a section inherited from a template or manually inserts a section, the cloned version might not include that section. When Workbook clones the experiment template using the behavior of versions released prior to 6.8, the section version and the properties values are cloned without changes. .

Result of Cloning To Latest

When an experiment based on an old template is cloned, the new experiment uses the settings from the newest version of that template and also inherits any sections and forms that are in the original experiment, but not in the template to which it is linked.

For example, if a template developer updates a script and a user clones an experiment that was created prior to the script update, the new experiment gets the new script. If a template developer adds a section, the new experiment gets the new section. If a template developer changes the name of a section, the new experiment gets *both* the old section and the new section.

The clone-to-latest behavior matches each section in the experiment to the corresponding section in the template based on section name and type, and picks up the latest properties for each matched section. If an experiment has sections that are not available in the latest template, they are copied as-is into the cloned experiment.

Features that are cloned include the following:

- Data, including reactions
- Signature policy, if any
- Formatting and data for planned and actual amounts
- Units for material sections
- Grid configuration for table sections, including pinned columns, columns selected for display, column order, sorting, and calculation units
- Row order, column order, and settings of Enable Reorder versus Enable Sort
- Quick text properties and options

Notes:

- Material sub-lots are *not* cloned.
- Linked sections are treated as one item, which is named using a forward-slash character (/) between the linked sections. Examples:
  - Reaction Scheme/Parallel Chemistry
  - Reaction Scheme/Synthetic Chemistry.

Example

Suppose your organization has a parallel chemistry template and several in-progress experiments based on that template.

If you decide that such experiments should require a signature policy and a dynamic toolbar from now on, you can add the policy and toolbar to your template. Scientists can then use QuickClone to create an updated version of their experiments that implements the new policy and includes the new toolbar.

Form Handling in Cloned Experiments

When an experiment includes a form, the version of the form used by clones depends on whether the template includes a corresponding form section:

- If the template that does *not* include a form section `SN.form`, clones get the latest available version of the form.
- If the template *does* include a form section, clones use the version of the form that is included in the template.

The source of the data for forms in cloned experiments depends on the setting of the `Cloneable` property of the form widget and of the form section in the experiment template:

- If `Cloneable` is *false* in either the widget or the template, the form's data is obtained from the template.
- If the `Cloneable` property is *true* in both the widget and the template, the form's data is cloned from the experiment.

Disabling and Enabling Clone-to-latest

Note: To use template management tools or template mapping options, you must have the `Template Editor` permission.

By default, the clone-to-latest feature is always enabled because it ensures that cloned experiments inherit the latest properties of the template on which they are based.

To disable or enable clone to latest:

1. Open a command prompt and navigate to the `<`*`Vault Installation directory`*`>\BIOVIA\Vault\Utilities` directory.
2. Use `VaultADMStoreManager` with the generate command to generate a working directory for the profile.
3. In the working directory, edit the `Symyx.Notebook.Application.exe.config` file to change the setting of the `cloneToLatestEnabled` parameter:
```
</configSections>
<CloneConfiguration cloneToLatestEnabled="true">
...
```
4. Save the file.
5. Test the change by running `Symyx.Notebook.Application.exe` from the working directory and verifying that the appropriate cloning options are available when you right-click an experiment in the Experiment Editor:
   - When CloneToLatestEnabled is true, you should have a QuickClone option and a Clone option, and the Clone option should display the Clone dialog box.

   Note: Another setting, **DisableCloneOnCheckedOutDocuments**, controls whether you can clone a document that is currently checked out. If that setting is enabled, test your CloneToLatest setting on a document that is not checked out.

- If CloneToLatestEnabled is false, you will have only a Clone option, and no dialog box is displayed.

6. Use `VaultADMStoreManager` with the update-profile command to update the profile.

Mapping Templates to a New Template for Cloning

You can map legacy templates to a new experiment template so that clones of experiments based on the legacy templates automatically use the new template.

Such mapping is easier than individually updating each legacy template.

To map a new template to your older templates:

1. Ensure that the updated template and the templates it supersedes have matching section names and types.
2. In Notebook Explorer, open the new template.
3. From Tools, click **Map Templates for Cloning**.
4. In **Map Templates for Cloning**, select the check box of each old template that you want to map to the newer template.
5. Click **OK**.

Updating a Template's Sections for Cloning

To support cloning with updated template sections, you can remove old versions of sections and then insert the new sections, making sure that their names and types match the names and types of the sections that you removed.

This approach ensures that clones of experiments based on the older sections will get the features of the replacement sections.

To update template sections to support cloning:

1. In Notebook Explorer, open the legacy template.
2. Note the name of each section that you need to map for replacement with a later version.
3. Right-click the section, and select **Delete** to remove the older sections.
4. From **View**, select **Properties**.
5. In **Properties**, on the **Experiment** tab, navigate to the **Template** group.
6. Select the **Insertable Sections** property row, and click the ellipses.
7. In **Choose Insertable Sections and Locations**, click the check box next to each section that users are allowed to add to the template, and click **OK**.
8. From **Insert**, select **Section**, and select each section to add to the template.
9. Add the toolbar buttons and scripts required for each specific section.
10. (Optional) Right-click the section, and select **Rename**, if the older section uses a custom name, rename the new section to use the same name.
11. Click **Save**.

References in Cloned Experiments

Cloned experiments can get references from either the source experiment or the template.

When the `Cloneable` property is set to:

- `Allowed` the data is cloned.
- `NotAllowed` does not clone the data from the source experiment.
- `AllowedNotData` behaves the same as Allowed but used for metadata.
- `Allow Nulls` property setting value controls how the references allow null values. When `Allow Nulls` is set to:
  - `CannotBeNull` does not allow null values. If the user tries to delete data and check in, the original data displays in the cell upon checkout.
  - `ShouldNotBeNull` renders a red x in the experiment until scientist enters a value.
- `Allow Updates` property setting value controls how the references are updated. When `Allow Updates` is set to:
  - `Always` updates are always allowed.
  - `Never` does not allows updates.
  - `Once` allows the scientist to clone an experiment one time only when the experiment contains properties with this setting. Data updates are not allowed.
  - `Until Saved` allows updates until the experiment is saved.
  - `Until Managed` allows updates until the experiment is checked in from a private repository to a managed repository.

If you have customized scripts, issues might arise because the custom code could run against a different section assembly.

Customizing Cloning Behavior

You can link a script to the clone action so that the script executes when an experiment is cloned. The following example forces the cloning action to duplicate the name of document it reproduces.

To customize the cloning action:

1. Upgrade the templates or set up the template mappings.
2. Create a backup copy of the `Symyx.Notebook.Application.exe.config` file.
3. Edit the `Symyx.Notebook.Application.exe.config` file by removing comment tags (shown here in red) from the XML element, `CustomPostProcessor`:
   ```
   <!-- You can add CustomPostProcessors here, that execute at the end of
   cloning if clone to latest is enabled. You must  define the
   CustomPostProcessers in a .NET Assembly available to BIOVIA Workbook,
   inherit from Accelrys.Notebook.Clone.ClonePostProcessor, and have a
   default public constructor, For example, this is a working post
   processor that ensures the clone has the exact same name as the original
   document,  "original", not "original_1.
   Remove the comment from the example to test the code.-->
   <!-- <CustomPostProcessor>
   Accelrys.Notebook.Clone.CloneShouldHaveTheSameNameAsTheOriginal,
   Accelrys.Notebook.Clone
   </CustomPostProcessor> -->
   ...
   </CloneConfiguration>
   ```
4. Create the custom post processor.

To create a custom post processor, define the custom post process in a .NET Assembly that is available to BIOVIA Workbook, inherit from `Accelrys.Notebook.Clone.ClonePostProcessor`, and use a default public constructor.

Example:

```
public class CloneShouldHaveTheSameNameAsTheOriginal :
ClonePostProcessor
  {
  public override void FinalizeClone(Document original, Document clone)
       {
         clone.Title = original.Title;
       }
  }
```

5. Test by logging in to Workbook, cloning an experiment, and verifying the name of the clone.
6. Create a release profile that deploys your cloning action configuration, as described in the document *BIOVIA Workbook Deployment Manager*.

Chapter 4: Configuring Reports

This chapter provides instructions and guidance to allow you to configure and manage reports.

Reporting Tools

BIOVIA Workbook provides tools that manage report definition files, report templates, report header/footer templates, and their associations with Vault objects. The reporting tools are located in `Program Data\Symyx Technologies\ Deployments\<ReleasePackage6.9x>` folder.

The `ReleasePackage` folder contains the following files:

- `ReportDefinitionPublisher.exe`

  Extracts, creates, and modifies a report definition. For more information, see "Using the Report Definition Publisher".
- `ReportDefinitionPublisher.exe.config`

  Configuration file for the Report Definition Publisher. By default, logging is enabled in this file. If you need to disable debugging, comment out the following lines

  ```
  <logger name="Symyx.Notebook.Tools.ReportDefinitionManagement"
          additivity="False">
    <level value="INFO" />
    <appender-ref ref="LogFileAppender" />
  </logger>
  ```
- `ReportAssociationManager.exe`

  Associates a report definition or template to a document or document template, and associates a header/footer template to a report definition or report template. For more information, see Associate Report Definitions with Documents or Templates.
- `ReportAssociationManager.exe.config`

  Configuration file for the Report Association Manager. By default, logging is enabled in this file. If you need to disable debugging, comment out the following lines:

  ```
  <logger name="Symyx.Notebook.Tools.ReportDefinitionManagement"
          additivity="False">
    <level value="INFO" />
    <appender-ref ref="LogFileAppender" />
  </logger>
  ```
- `ReportDefinitionManagement.dll`

  The class library used by the reporting tools. Verify that this file is in the same directory as `ReportDefinitionPublisher.exe` and `ReportAssociationManager.exe`.

Report Builder and Report Template Designer

Creating a report definition is a command line process. The Report Builder is used with report definitions, and enables the user to configure the report before producing a report document or printed report. The Report Builder shows a preview of the current configurations made by the user. For example, if the use changes a font or color, or chooses to include or exclude a section or table, the preview is updated. The Report Builder is available to any user who is permitted to generate reports.

The Report Template Designer, on the other hand, is used to visually design and store a fixed report template, that is to choose the specific data, layout and style for report documents generated from that fixed report template. The Report Template Designer is available only to users with the appropriate permission, and would be used infrequently to design and store fixed report templates, which could then be used by many users to generate report documents in the fixed style.

A user who is permitted to use the Report Template Designer can work inside the Report Builder to configure a report from, say, the Experiment report definition, and then capture and save that configuration as a fixed report template. You might want to do most of the configuration in the Report Builder, and then use the Report Template Designer for smaller changes.

Configure Reports Using the Report Builder

BIOVIA Workbook provides two report definitions, Experiment and Audit Trail. If the user opens the Experiment report definition in the Report Builder, and decides to color the experiment title in red and exclude the chemical structure in the Materials section, every report generated using the Experiment report definition compiles to the same layout and content definition. Another scientist can change the report definition. Create report definitions to allow a scientist to maintain different report output configurations for different types of experiments.

Associating a report definition and a fixed report template with an experiment template provides access to view the Reports in the Experiment Explorer, the associated report definition or fixed report template appears in the list of available reports.

Forcibly Display User Hidden Data

You can override any user settings that hides data from PDF Reports created on the Vault Server or by a Workflow.

To forcibly display user hidden data:

1. Use `VaultADMStoreManager.exe` to create a working directory containing the `Symyx.ReportPrinter.exe.config` file.

   For more information, see the *BIOVIA Deployment Manager*.

2. Edit the file to add to the `applicationSettings` element.

```
<applicationSettings>
...
  <Symyx.Notebook.Properties.Settings>
    <setting name="ForciblyDisplayUserHiddenData"
             serializeAs="String">
    <value>True</value>S
    </setting>
  </Symyx.Notebook.Properties.Settings>
  <Symyx.Notebook.ApplicationShell.Properties.Settings>
...
</applicationSettings>
```

For more information, see Report Definition Publisher.

Display Signatures in Custom Headers and Footers

You must configure a custom header or footer to display signature in the Format property of the Transition Signature field. Specify the signature XML elements for the approver name, comment from the SendToApprove workflow event, the Finished by name, and date from the SendToFinish workflow event. For more information, see Signature XML Element.

Use the Report Header and Footer Designer to:

- Design custom headers and footers that display workflow signatures.
- Add the Transition Signature field of the experiment to the header or footer.

  In the Transition Signature field's Format property, specify the information display based on the specific workflow events.
- Define the display information in the signature XML elements.

To add signature information to a custom header or footer:

1. Open BIOVIA Workbook.
2. In Notebook Explorer, select **Create** > **New Report Header/Footer Template**.
3. In **Enter Report Header/Footer Template Title**, type a name for the template, and click **OK**.
4. In **Select Report Data**, select an experiment or experiment template containing data, and click **Open**.
5. In **Header and Footer Options**, from the **Object Palette**, select **Data Fields** > **Experiments** > **Transition Signature**.
6. Drag the **Transition Signature** field into one of the Designer canvas header or footer fields.
7. Select the **Transition Signature** field in the Designer canvas, adjust the **Height** and **Width** properties to fit the report page.
8. In **Properties of Data: Transition Signature**, select the **Format** property, type in the signature XML data.
9. Click **File** > **Save**.

Signature Element

The signature element is used to specify information related to the signature for a specific workflow event. If there are multiple signatures associated with the specified workflow transition event, the most recent signature is used.

XML is case-sensitive. Use the letter case found in the signature element's XML.

The following is the signature element's XML format:

```
<signature event="workflowEventName" property="signaturePropertyName"
signaturePropertyAttribute="signaturePropertyAttributeValue"/>
```

The `event` attribute specifies the workflow transition event for which the signature is associated.

The `property` attribute specifies a property name that indicates the type of signature-related information to display in the header or footer. The following table list the valid property names.

Signature property name	Description
comment	The comment associated with a signature
reason	The reason associated with a signature

Signature property name	Description
meaning	The meaning associated with a signature
user-name	The user name of the person who signed
full-name	The full name of the person who signed
client-time-stamp	The client time-stamp for the signature.
server-time-stamp	The server time-stamp for the signature.

The `signaturePropertyAttribute` can use the values listed in the table that follows. The attributes are related to the specified property name. You can specify more than one `signaturePropertyAttribute`.

Signature property attribute	Description
date-format	Controls the format of a System.DateTime. Use this with the client-time-stamp and server-time-stamp properties only. For example: `<signature event=“SubmitForReview“` `property=“server-time-stamp“` `date-format=“dd-MMM-yyyy“ />`
prefix	Inserts the specified prefix string before the property value. For example: `<signature event="SubmitForReview"` `property="user-name"` `prefix="Submitted by: " />` To add a carriage return in the displayed data, use the escape sequence \n. To add a tab, use the escape sequence \t. For example: `<signature event="SubmitForReview"` `property="user-name"` `prefix="\n\tSubmitted by: " />`
suffix	Appends the specified suffix string to the property value. For example: `<signature event="SubmitForReview"` `property="user-name"` `prefix="(Submitted by: "` `suffix=")" />` To add a carriage return in the displayed data, use the escape sequence \n. To add a tab, use the escape sequence \t. For example: `<signature event="SubmitForReview"` `property="user-name"` `prefix="\t(Submitted by: "` `suffix=")\n" />`
stages	Limits the display of signature information to workflow stages specified in a comma-delimited list. Using the stages property prevents the display of signatures that might no longer be valid because a document has been sent back to a previous stage in the

Signature property attribute	Description
	workflow. If specified, the document must be in one of the specified stages in order for something to appear in the header or footer. If not present, signature information might appear for any stage. stages="none" can be specified to indicate the stage where the document is no longer enrolled in a workflow.

Report Template and Report Definition

The *report template* specifies the information to include and the layout of that information.

When a report is generated using a report template, the BIOVIA Workbook scientist cannot change the information contained in the report or the layout elements.

The scientist can choose the experiment used as the source of the information, and the destination of the generated report, such as PDF or printer.

A *report definition* defines the elements of the report rather than its specific content and layout. The report definition provides sections it finds in the experiments it reports. A report definition defines a guideline of the content of the report the property data for experiments or sections and defines the information that is excluded such as history data. When a report is generated using a report definition, the report definition general guidelines are used to construct a report in the Report Builder.

The BIOVIA Workbook scientist user can configure and design the report and create a report definition that does not use the report definition provided by BIOVIA Workbook.

- Use a report template when the content, layout or style of the report must remain constant to meet a requirement such as with a Certificate of Analysis type report, where the structure of the report is fixed for filing or recording purposes.
- Use a report definition when the content of the report decided on by the report's creator.

The BIOVIA Workbook scientist user can generate a report using a report template that does not have a report definition file. A report template is a Vault object used to generate a specific report layout from an experiment or experiments and generate a report using a report definition file without a report template.

Report Definition Publisher

The Report Definition Publisher is a utility that extracts, creates, and modifies a report definition. A user with `Report.Editor` application permission can create custom report definition files, and upload the report definition files to Vault. The user must have the `Report.Editor` application permission to run the Report Definition Publisher utility.

To run the Report Definition Publisher:

1. Open a command window.
2. Change to the directory the utilities installation location.
3. Run the utility.

The command syntax is:

```
ReportDefinitionPublisher servername domain\username password
localPath: vaultPath: /x|/a|/u
```

The path variables are:

- `servername` represents the name of the Vault server.
- `domain\username` specifies the name of the Vault user with the Report.Editor permission. Use the domain-qualified username, for example, `symyx\vault.editor1`.
- `password` represents the password for the specified user.
- `localPath:"file path"` specifies the local path to a report definition file. For more information, see Report Definition File.
- `vaultPath:"Vault path"` the Vault path to the report definition file. Use the back slash ("\") as the Vault path separator. The back slash has the following options:
  - `/x` extracts the report definition specified by `vaultPath` and saves it to the `localPath`.
  - /a adds the report definition specified by `localPath` to the report definition name and location specified by `vaultPath`.
  - /u updates the report definition specified by `vaultPath` with the contents stored in `localPath`.

Create Report Definitions

Clone an existing report definition to create a report definition. Use the default report definitions from the Experiment or Audit Trail report definition.

To create a report definition:

1. Extract one of the default report definitions from the Experiment or Audit Trail report definition.
2. Open a command window and change the directory to the location of the `ReportDefinition.exe`.
3. Run `ReportDefinitionPublisher.exe` using the /x option.
4. Modify the XML file to specify the new report definition.
5. Run `ReportDefinitionPublisher` using the /a option to upload the report definition XML file into Vault using a new report definition name.

   For example:

   ```
   ReportDefinitionPublisher TestVaultServer test\vault.admin
   vaultpassword localPath:"c:\data\NewReportDefinition.xml"
   vaultPath:"RepositoryA\FolderA\NewExperiment" /a
   ```

To see the report definition in the list of available reports for a document, associate the new report definition to a document or document template by using the Report Association Manager. For more information, see Associate Report Definitions with Documents or Templates.

Update Report Definitions

To update a report definition:

1. Run `ReportDefinitionPublisher` using the /x option, extract the report definition.

   For example:

   ```
   ReportDefinitionPublisher TestVaultServer test\vault.admin
   vaultpassword localPath:"c:\data\UpdateReportDefinition.xml"
   vaultPath:"RepositoryA\FolderA\NewExperiment" /x
   ```

2. Modify the downloaded XML file to update the report definition.

3. Modify the file `\\data\UpdateReportDefinition.xml` that contains the existing `NewExperiment` definition.
4. Run `ReportDefinitionPublisher` using the /u option to upload the modified XML file to Vault using the existing report definition name.
5. Update the `NewExperiment` definition with contents from `\\data\UpdateReportDefinition.xml`.

Associate Report Definitions with Documents or Templates

The Report Associations Manager is a utility that administers report associations. You must have `Report.Editor` application permission to use the Report Associations Manager. You can perform the following tasks:

- Associate report definitions or report templates to documents and document templates
- Associate report header and footer templates to report definitions and report templates

After a new report definition file is added to Vault, associate the new report definition to a document or document template. You can globally associate documents in the system to a specific report definition or template.

To associate a report template with a report definition:

1. Open a command window to run the Report Associations Manager.
2. Change to the directory the location of the utility.
3. Type the following command to run the utility:

```
ReportAssociationManager.exe servername domain\username password
targetPath: assocPath: /a|/u|/d/l /g /c
```

The variables represent:

- `servername` specifies the name of the Vault server
- `domain\username` specifies the name of the Vault user with the `Report.Editor` permission. Type the domain-qualified username, for example, `biovia\vault.editor1`.
- `password` specifies the password for the specified user
- `targetPath:"Vault path"` specifies the Vault path to the object that receives the association updates. Use the back slash (\) as the Vault path separator.

  You can use the following options in the target:

Use case	targetPath	assocPath
Associates a report definition or report template to a document or document template.	Specifies the Vault path to a document or document template. Verifies that the object is checked into Vault.	Specifies the Vault path to a report definition or template .
Associates a header and footer template to a report definition or report template.	Specifies the Vault path to a report definition or template.	Specifies the Vault path to a report header or footer template.

- `assocPath:"Vault path"` specifies the Vault path to association object. Use the back slash as the Vault path separator.

- /a indicates add the report association to the target Vault object
- /u indicates update the report association on the target Vault object
- /d indicates delete the report association from the target Vault object
- /l indicates list the report associations on the target. The `assocPath` parameter is ignored.
- /g indicates a global switch. When the global flag is set, the `targetPath` parameter is ignored and the command is applied to all valid objects in the system.
- /c indicates a cascade switch. If the target object in `targetPath` is a document template, then the command is applied to all documents created from the template.

Examples

Run the `ReportAssociationManager` tool using the /a option associate a document or document template to a report definition or template.

- The following example associates the *NewExperiment* report definition to a document template named *NewExperimentTemplate*:

  ```
  ReportAssociationManager TestVaultServer test\vault.admin vaultpassword
  targetPath:"RepositoryA\FolderA\NewExperimentTemplate"
  assocPath:"RepositoryB\FolderB\NewExperiment" /u
  ```

- The following example associates the *NewReportTemplate* report template to a document named *TestExperiment*:

  ```
  ReportAssociationManager TestVaultServer test\vault.admin vaultpassword
  targetPath:"RepositoryA\FolderA\TestExperiment"
  assocPath:"RepositoryB\FolderB\ReportTemplate" /a
  ```

Add Report Association to a Header and Footer Template

Run the `ReportAssociationManager` utility using the /a option. To associate a report definition or template to a header and footer template. The following example associates the `New Header and Footer` template to the `NewExperiment` report definition:

```
ReportAssociationManager TestVaultServer test\vault.admin vaultpassword
targetPath:"RepositoryA\FolderA\NewExperiment"
assocPath:"RepositoryB\FolderB\New Header and Footer" /a
```

Open a new command prompt window, before executing the `ReportAssociationManager` command.

Update Report Document and Template Associations

To update a document or document template to use the latest version of a report definition or template, run the `ReportAssociationManager` tool using the /u option.

- The following example updates the association of `NewDocumentTemplate` to a new report definition named `NewExperiment`:

  ```
  ReportAssociationManager TestVaultServer test\vault.admin
  vaultpassword targetPath:"RepositoryA\FolderA\NewDocumentTemplate"
  assocPath:"RepositoryB\FolderB\NewExperiment" /u
  ```

- The following example updates the association of `NewDocumentTemplate` and all documents created from it to the `NewExperiment`:

```
ReportAssociationManager TestVaultServer test\vault.admin
vaultpassword targetPath:"RepositoryA\FolderA\NewDocumentTemplate"
assocPath:"RepositoryB\FolderB\NewExperiment" /u /c
```

- The following example globally updates associations of all document templates to the `NewDocumentTemplate`:

  ```
  ReportAssociationManager TestVaultServer test\vault.admin
  vaultpassword assocPath:"RepositoryA\FolderA\NewExperiment" /u /g
  ```

Delete Report Associations

Run the `ReportAssociationManager` tool using the `/d` option to delete an association between a document or document template and a report definition or template.

- The following example deletes the association of `NewDocumentTemplate` from the `NewExperiment`:

  ```
  ReportAssociationManager TestVaultServer test\vault.admin
  vaultpassword targetPath:"RepositoryA\FolderA\NewDocumentTemplate"
  assocPath:"RepositoryB\FolderB\NewExperiment" /d
  ```

List Report Associations

Run the `ReportAssociationManager` utility using the `/l` option to list the report associations of a specific target. For example:

```
ReportAssociationManager.exe vault-server domain\user password
targetPath:"Repository A\Folder B\Experiment" /l
```

Run the `ReportAssociationManager` utility using the `/l /g` options to list the report associations of all valid objects in the system.

For example:

```
ReportAssociationManager.exe vault-server domain\user password /l /g
```

Report Definition File

A report definition is a collection of elements that control the presentation of document and section data in a report. When a report uses a report definition, users can interact with the report definition to reconfigure the report after generating it in the Report Builder.

BIOVIA Workbook includes an Experiment report definition and an Audit Trail report definition. The report definitions are installed in the Site repository, and are automatically associated with the document templates that are also installed with BIOVIA Workbook.

You can download a report definition from the Site repository into an XML file to modify the report definition.

- Use the ReportDefinitionPublisher utility to download an existing report definition.
- Modify the report definition XML file.
- Add the modified report definition file into Vault.

Report Definition XML Format

The following shows the hierarchy of a report definition XML file:

```
<ReportDefinition>
  <Document>
    <Snippet>
```

```
        <Groups>
          <Group Identifier="Group_Identifier"
                 Display="Display_Option" />
        </Groups>
      </Snippet>
      <Sections>
        <Section>
          <Snippet>
            <Groups>
              <Group Identifier="Group_Identifier"
                     Display="Display_Option" />
            </Groups>
          </Snippet>
       </Section>
      </Sections>
    </Document>
  </ReportDefinition>
```

Report Definition File Elements

The following table shows the XML elements in a report definition file, their descriptions, attributes, and child elements:

Element	Description	Attributes	Child Elements
ReportDefinition	The root element	None	Document
Document	The collection of document snippets and section with snippets	None	Snippet Sections
Snippet	Contains groups of data that specify the presentation of document and section data	None	Groups
Groups	Identifies a group of data items that are included or excluded by default in the report	None	Group

Element	Description	Attributes	Child Elements
Group	Report data items to be included in or excluded from the report	identifier distinguishes the report data from other data. id specifies the GUID equivalent of identifier; for internal use only. **Note:** The id attribute is included when you download a report definition file. The id attribute is ignored if it is included in an uploaded report definition file. display indicates whether the group is on or off in the report, and whether the scientist configuring the report in the Report Builder window is able to override the setting.	
Sections	Contains section definitions.	None	Section
Section	Contains snippets of groups of data items that are included or excluded by default in the report	Class specifies the class type name and assembly name, separated by a comma, of the section to which it applies. If Class is blank or not specified, the settings apply to all classes. Title specifies the title of the section. If Title is blank or not specified, the settings apply to all sections of the given class.	Snippet

Group: identifier

The identifier attribute identifies the report data that is included in a snippet. In the Snippet element under Document, the valid Group identifier values are:

- DOCUMENT_TITLE_GROUP specifies the document report includes its title.
- DOCUMENT_PROPERTIES_GROUP specifies the document report includes its properties.
- DOCUMENT_HISTORY_GROUP specifies the document report includes its history.
- DOCUMENT_REFERENCES_GROUP specifies the document report includes its references.

In the Snippet element under a Section, the valid Group identifier values are:

- SECTION_TITLE_GROUP specifies the section data report includes its title.
- SECTION_PROPERTIES_GROUP specifies the section data report includes its properties.
- SECTION_CONTENT_GROUP specifies the section data report includes its content.

Group: display

The display attribute indicates whether the group is on or off by default in the report, and whether the scientist configuring the report in the Report Builder window is able to override the setting. The following are the valid Group display values:

- Optional specifies the group is not included by default, but the user may switch it on. Optional is the default display value.

- `Always` specifies the group is always included, and the user cannot switch it off.
- `Default` specifies the group is included by default, but the user may switch it off.
- `Hidden` specifies the group is not included, so the user does not see the group.

Section: class

The `section` class attribute specifies the class type name and assembly name, separated by a comma, of the section to which it applies. If the class value is blank, the settings apply to all classes.

`Snippets` that apply to sections can indicate the section to which they apply in the `Class` attribute, and specifically titled sections, for example:

```
<Section
Class="Symyx.Notebook.Sections.ReactionScheme.ReactionSchemeSection,
Symyx.Notebook.Sections.ReactionScheme" Title="Results">
```

Experiment Report Definition File

The following shows the elements of the Experiment report definition file:

```
<?xml version="1.0"?>
 <ReportDefinition>
   <Document>
     <Snippet>
       <Groups>
         <Group Identifier="DOCUMENT_TITLE_GROUP"
                Display="Default" />
         <Group Identifier="DOCUMENT_PROPERTIES_GROUP"
                Display="Default" />
       </Groups>
     </Snippet>
     <Sections>
       <Section>
         <Snippet>
           <Groups>
             <Group Identifier="SECTION_TITLE_GROUP"
                    Display="Default" />
             <Group Identifier="SECTION_CONTENT_GROUP"
                    Display="Default" />
           </Groups>
         </Snippet>
       </Section>
      </Sections>
    </Document>
 </ReportDefinition>
```

Audit Trail Report Definition File

The following shows the elements of the Audit Trail report definition file:

```
<?xml version="1.0"?>
  <ReportDefinition>
    <Document>
      <Snippet>
        <Groups>
          <Group Identifier="DOCUMENT_TITLE_GROUP"
                 Display="Default" />
           <Group Identifier="DOCUMENT_PROPERTIES_GROUP"
                  Display="Default" />
           <Group Identifier="DOCUMENT_HISTORY_GROUP"
                  Display="Default" />
         </Groups>
       </Snippet>
     </Document>
   </ReportDefinition>
```

Custom Reporting

The Reporting API contains the `Symyx.Notebook.dll` library that includes the following namespaces enabling applications to interact with Workbook reporting capabilities. The library includes the following:

- `Symyx.Notebook.Reporting.Scrivo` contains the `NotebookReporter` class that encapsulates all reporting functionality.
- `Symyx.Framework.Reporting.Scrivo` contains helper classes and enumerations used with the `NotebookReporter` class.
- `Symyx.Framework.Reporting.Scrivo.IO` contains classes that represent report definitions and templates in the BIOVIA Vault Server repositories.
- `Symyx.Framework.Reporting.Scrivo.DataProvider` contains classes that support reporting for custom data types and properties.

Interacting with NotebookReporter

The `Symyx.Notebook.Reporting.Scrivo.NotebookReporter` class is the main class used for interacting with reporting. `NotebookReporter` encapsulates all reporting functionality. The following example shows creating a `NotebookReporter` in an application:

```
using Symyx.Notebook.Reporting.Scrivo;
 private NotebookReporter notebookReporter; private void CreateReporter()
  {
    notebookReporter = new NotebookReporter();
  }
```

Supplying data to NotebookReporter

The `NotebookReporter` class acquires data for reporting from the selected experiment or experiments. The experiments are provided to the `NotebookReporter` by the following `NotebookReporter.DataRequired` event:

```
notebookReporter.DataRequired += notebookReporter_DataRequired;
```

To supply the experiments for reporting, you must handle the `DataRequired` event. The following example demonstrates using the `DataRequired` event.

The `DataRequiredEventArgs` is in the `Symyx.Framework.Reporting.Scrivo` namespace in `Symyx.Framework.Reporting.dll`.

The `DataRequired` event expects a `Dictionary<VaultId, VaultObject>` for the `DataDescriptor.DataObject` event argument.

DataRequired Event Example

```
private void notebookReporter_DataRequired(object sender,
DataRequiredEventArgs e)
  {
    Dictionary<VaultId, VaultObject> selectedExperiments = new
Dictionary<VaultId, VaultObject>();
// Get all the selected experiments from.
// For example, a workspace browser:
   using (IEnumerator<VaultObject> enumerator =
browser.GetSelectedItems<VaultObject>().GetEnumerator())
    {
      while (enumerator.MoveNext())
      {
        if (enumerator.Current.ObjectType == VaultObjectType.Document)
        {
        selectedExperiments[enumerator.Current.VaultId] = null;
        }
      }
    }
// Set the data object to the selected experiments
e.DataDescriptor.DataObject = selectedExperiments;
  }
```

If the experiment you are working with is already open, and fully retrieved, then you can pass that experiment in the dictionary, by changing the line which adds the object to the dictionary:

```
selectedExperiments[enumerator.Current.VaultId] = myDocument;
```

Using this technique means that the NotebookReporter can use the open experiment, rather than having to instantiate it for the purposes of reporting.

IDisposable interface

The `NotebookReporter` class implements the `IDisposable` interface. When no longer needed, it must be disposed in order to free memory and resources. Before disposing of the NotebookReporter, you should also remove the `DataRequired` event handler:

```
notebookReporter.DataRequired -= notebookReporter_DataRequired;
notebookReporter.Dispose();
```

Open the Report Designer

To open the Report Designer and create a new fixed report, pass the enumeration value `DesignObjectType.ReportTemplate` to the `NotebookReporter.Design` method:

```
notebookReporter.Design(DesignObjectType.ReportTemplate);
```

To open the Report Designer for an existing report, pass a `Symyx.Framework.Reporting.Scrivo.IO.VaultReport` object to the `NotebookReporter.Design` method. For example:

```
public void DesignReportTemplate(VaultReport vaultReport)
 {
   notebookReporter.Design(vaultReport);
 }
```

You can pass the VaultId of a VaultReport object using the following:

```
public void DesignReportTemplate(VaultId vaultId)
  {
    notebookReporter.Design(vaultId);
  }
```

The `NotebookReporter.Design` method accepts a `System.Windows.Forms.Form` object parameter. This is optional, if the Form object is passed as an argument to the Design method, the Form is used as the owner of any dialogs created during the running of the method.

Opening the Report Builder

To open the Report Builder to display a configurable report, pass the enumeration value `DesignObjectType.ReportDefinition` to the `NotebookReporter.Design` method:

```
notebookReporter.Design(DesignObjectType.ReportDefinition);
```

To open the Report Builder for an existing report, pass a `Symyx.Framework.Reporting.Scrivo.IO.VaultReportDefinition` object to the `NotebookReporter.Design` method. For example:

```
public void DesignReportTemplate(VaultReportDefinition
vaultReportDefinition)
  {
    notebookReporter.Design(vaultReportDefinition);
  }
```

You can pass the `VaultId` of a `VaultReportDefinition` object to open the Report Builder. For example:

```
public void DesignReportTemplate(VaultId vaultId)
  {
    notebookReporter.Design(vaultId);
  }
```

The `NotebookReporter.Design` method accepts a `System.Windows.Forms.Form` object parameter. This is optional, if the Form object is passed as an argument to the Design method, the Form is used as the owner of any dialogs created during the running of the method.

Display a List of Reports

The `NotebookReporter.SelectAndRun` method enables selecting a report to generate, for example

```
notebookReporter.SelectAndRun();
```

The `NotebookReporter.SelectAndRun` method can pass a `System.Windows.Forms.Form object`. Passing the object is optional. If the Form object is passed as an argument to the method, the Form is used as the owner of any dialogs created while running the method.

Design Header and Footer Templates

To open the header/footer template designer, use the `NotebookReporter.DesignHeaderAndFooterTemplate` method:

```
notebookReporter.DesignHeaderAndFooterTemplate();
```

The default header and footer template for generated reports is stored in the site repository. The header and footer designer provides access to the repository in order to enable changing the template. You can create and save additional header/footer templates can be created and saved, and associated with specific report definitions using a script. For more information, see Custom Reporting.

Run Reports

The NotebookReporter class has methods that support running report objects and sending the output to various destinations. The `Symyx.Framework.Reporting.Scrivo.IO.VaultReport` represents a report using a fixed report template and the `Symyx.Framework.Reporting.Scrivo.IO.VaultReportDefinition` object represents a report using a configurable report definition.

The `NotebookReporter.Run` method runs a report template or report definition, depending on the specified parameter. The following examples show how to invoke the `Run` method:

```
// Run a report template
public void RunReportDefinition(VaultReport vaultReport)
  {
    notebookReporter.Run(vaultReport);
  }

// Run a report definition
public void RunReportDefinition(VaultReportDefinition definition)
  {
    notebookReporter.Run(definition);
  }

// Run a report using its VaultID.
public void RunReport(VaultId vaultId)
  {
    notebookReporter.Run(vaultId);
  }
```

Reporting requires specific options before generating the reports. When the `Run` method is invoked, a report options dialog displays allowing the user to choose options.

The `NotebookReporter.Run` methods enable passing of a `System.Windows.Forms.Form` object. The passing the object is optional. If the Form object is passed as an argument to the run methods the Form is used as the owner of any dialogs created during the running of the method.

Report Queue Name Property

A dependency property has two parts:

- An object of the `DependencyProperty` class.
- A string with get and set methods for the `DependencyProperty` object.

The following example defines a `DependencyProperty` public static object named `QueueNameProperty`.

```
public static DependencyProperty
QueueNameProperty = DependencyProperty.Register("QueueName",
typeof(string), typeof(PrintReportActivity), new PropertyMetadata
(@".\Private$\PrintReports"));
```

The following example defines a string named `QueueName` that is exposed to Workflow Designer. The object contains get and set methods for `QueueNameProperty`.

```
[Description("Report queue name")]
[Category("Dependency Properties")]
[Browsable(true)]
[DesignerSerializationVisibility
```

```
     (DesignerSerializationVisibility.Visible)]
public string QueueName
  {
    get { return (string) GetValue(QueueNameProperty); }
    set { SetValue(QueueNameProperty, value); }
  }
```

Report Template Identifier Property

The C# code below defines the report template identifier property. The code defines:

- A `DependencyProperty` public static object named `ReportIdProperty`.
- A string with get and set methods for the `ReportIdProperty` object.

```
public static DependencyProperty
ReportIdProperty = DependencyProperty.Register
  ("ReportId", typeof(string), typeof(PrintReportActivity),
    new PropertyMetadata(@"your report Id"));
/// <summary>
/// Required field: ReportID (definition or template)
/// </summary>
[Description("Report ID (definition or template)")]
[Category("Dependency Properties")]
[Browsable(true)]
[DesignerSerializationVisibility
     (DesignerSerializationVisibility.Visible)]
public string ReportId
  {
    get { return (string) GetValue(ReportIdProperty); }
    set { SetValue(ReportIdProperty, value); }
  }
```

Specify Report Options

The `NotebookReporter.Run` methods accept a `Symyx.Framework.Reporting.Scrivo.ReportOptions` object as a parameter. For example:

```
ReportOptions options = new ReportOptions
  {OutputWriter = "Adobe Acrobat"};
notebookReporter.Run(vaultId, options);
```

You can set the following properties for ReportOptions:

Property	Description
FromPage	Specifies an integer representing the first page to print. The default value is 1.
ToPage	Specifies an integer representing the last page to print. The default value is -1, indicating print all pages.
OutputName	Specifies the name of the printer or the file to use when saving the report output.
OutputWriter	Specifies the name of the writer used to create the output. The default is Print Preview. Available values are: ■ Print ■ Print Preview

Property	Description
	■ Adobe Acrobat ■ Microsoft Word ■ Microsoft Word 2007
OutputOption	Contains options that determine whether an existing output file is overwritten if it already exists. Available values are the enumerated values of `Symyx.Framework.Reporting.Scrivo.OutputDisplayOption` ■ Show - Indicates that the output is displayed. ■ Store - Indicates that the output is written to the OutputStream property. ■ None - Indicates that the output is not displayed.
OutputStream	When the OutputOption is set to Store, the output is written to the OtuputStream property when the report completes. The calling program is responsible for eliminating the stream.
OverwriteOption	Indicates what to do if the file specified in OutputName already exists. The available options are the enumerated values of `Symyx.Framework.Reporting.Scrivo.OutputOverwriteOption`: ■ Prompt requires user interaction to decide whether to overwrite the file. ■ Overwrite replaces the file with the latest content. ■ Cancel does not allow overwriting the file.

Specify Report Run Mode

Some `NotebookReporter.Run` methods accept `Symyx.Framework.Reporting.Scrivo.RunMode` enumeration as a parameter. You can specify the following values:

- Interactive - Reports are run interactively. All messages and dialogs will be shown, where necessary.
- Batch - Reports are run in batch mode. No dialogs will be shown and all messages will be logged rather than displayed.

Specifying User Settings

Some of the `NotebookReporter.RunReportDefinition` and `ExportReportDefinitionToPdf` methods accept a string containing user settings for generating a report as a parameter. Specify the user settings in an XML string using the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<!--Warning - manual changes to this XML should be made with caution, as
invalid entries may cause errors-->

  <UserSettings>
    <Settings PaperKind="Letter" Paper="Letter"
              CustomSize="2159;2794" WatermarkOpacity="24">
      <Margins Left="64" Top="64" Right="64" Bottom="64" />
    </Settings>
    <ExcludedSections />
    <ExcludedSectionGroups />
    <SectionOrder />
    <UserDefinedSectionGroups />
    <PropertyOverrides />
    <DisplayOverrides />
    <ReportOptions />
  </UserSettings>
```

Get User Settings XML String

The Report Builder uses an XML string containing user-specified settings for generating a report.

In the Report Builder, you can reconfigure the report. For information about configuring the report.

To get the XML string:

1. In the Notebook Explorer, right-click the experiment to use as the source for the report, and choose **Run Report**.
2. Select the report to generate, for example, the Audit Trail or Experiment report.

   The report is generated and displayed in Report Builder.
3. After changing report configuration settings, select **Tools** > **Copy Settings to Clipboard**.
4. Paste the clipboard contents to a text editor to get the XML string which specifies the current report user settings.

Display a Report Objects List

The `NotebookReporter.SelectAndDesign` method opens a dialog to enable selecting a report and opening the report in the Report Designer or Report Builder. It is invoked as follows:

```
notebookReporter.SelectAndDesign(new []
  {DesignObjectType.ReportTemplate});
```

The use of an array in the example allows the Vault dialog to show objects of different types such as report templates and report definitions to be selected.

Specify Custom Report Definitions

To specify custom report definitions:

- Run:

```
ReportDefinitionPublisher.exe server domain\user password
localPath:"c:\customreports\expreport.xml"
vaultPath: "site\notebooktemplates64\experiment" /x
```

To extract the default audit trail report:

- Run:

```
ReportDefinitionPublisher.exe server domain\user password
localPath:"c:\customreports\audit.xml"
vaultPath: "site\notebooktemplates64\audit trail" /x
```

To add the custom report definitions:

- Run:

```
ReportDefinitionPublisher.exe server domain\user password
localPath:"c:\customreports\expreport.xml"
vaultPath: "<repository\folder>\customdefinition" /a
```

To collect the definition settings using Report Builder:

1. From Notebook Explorer, select an experiment and run a default report on an experiment or audit trail.
2. If required, adjust the report using the report options and document option properties.

   When making document option property changes, you can suspend automatic report generation by choosing **File** > **Tools** > **Suspend automatic report regeneration**.
3. Select **File** > **Tools** > **copy Settings to clipboard**.
4. Paste this content into a text file and save the file for future use.
5. Modify the report definition XML file
6. Using a text editor, open the `reportsettings.xml` from the report printer folder.

   The default user settings in the XML file are:

```
<?xml version="1.0" encoding="utf-8"?>
<!--Warning - manual changes to this XML should be made with caution, as
invalid entries may cause errors-->
  <UserSettings>
    <Settings PaperKind="Letter" Paper="Letter, 8 1/2 x 11"
              CustomSize="2159;2794" WatermarkOpacity="24">
    <Margins Left="64" Top="64" Right="64" Bottom="64" />
    </Settings>
    <ExcludedSections />
    <ExcludedSectionGroups />
    <SectionOrder />
    <UserDefinedSectionGroups />
    <PropertyOverrides />
    <DisplayOverrides />
    <ReportOptions />
  </UserSettings>
```

7. Replace the default user settings in the `reportsettings.xml` file with your custom user settings. Save the `reportsettings.xml` file.
8. Get the custom report definition ID from Notebook Explorer
9. In Notebook Explorer, open the custom report definition that you uploaded in step 2.
10. Display the properties to get report definition ID.
11. Copy the definition ID to the clipboard.
12. Open the `Symyx.ReportPrinter.exe.config` file and modify the `ReportDefinition` setting and any associated settings in the XML to reference the definition ID you retrieved in the previous step. For example:

```
<add key="ReportDefinition.b24ae005-499d-43d1-8b27-94d2c661a044"
value=".\reportsettings1.xml"/>
```

13. Save the `Symyx.ReportPrinter.exe.config` file.

Run Reports Using a Report Definition

You can use the `NotebookReporter.RunReportDefinition` methods with `Symyx.Framework.Reporting.Scrivo.IO.VaultReportDefinition` objects. The following code shows an example:

```
public void RunReportDefinition(VaultId vaultId)
  {
    ReportOptions options = new ReportOptions
    {OutputWriter = "Microsoft Word"};
    notebookReporter.RunReportDefinition(vaultId, options);
  }
```

The method supports the same `Symyx.Framework.Reporting.Scrivo.ReportOptions` and `Symyx.Framework.Reporting.Scrivo.RunMode` options as the `NotebookReporter.Run` methods.

For more information on the possible formats for OutputWriter and other report options, see Specify Report Options.

Run the Report Printer

You can run the Report Printer program, `Symyx.ReportPrinter.exe`, to generate PDF reports on experiments.

The following example generates a PDF named `testreport.pdf` using the document ID and report template ID specified in the `docId` and `reportId` parameters.

For example:

```
Symyx.ReportPrinter.exe
-user:vm-vault66\vault.admin
-pass:password
-endpoint:VM-VAULT66
-docId:Document.5cee92a7-b072-4082-8d72-2a5514bf51ec
-reportId:ReportTemplate.95815a62-a809-49ab-b1e9-edb78629d65b
-reportPath:"C:\reports\testreport.pdf"
```

Reports for Large PDF Output

For reports that generate larger PDF output, use the `NotebookReporter.ExportReportDefinitionToPdf` method, for example:

```
public void ExportToPDF(VaultReportDefinition reportDef,
string userSettings)
  {
    ReportOptions options = new ReportOptions
      {OutputWriter = "Microsoft Word"};
    notebookReporter.ExportReportDefinitionToPdf
      (reportDef, options, userSettings);
   }
```

The `NotebookReporter.ExportReportDefinitionToPdf` method allows specifying the `RunMode` and passing a `System.Windows.Forms.Form` object. Passing the object is optional. If the Form object is passed as an argument to the method, the Form is used as the owner of any dialogs created while running the method.

Notes:
- The `ExportReportDefinitionToPdf` methods only support report definitions (configurable reports).
- Reports created using report templates are not supported.
- Users should only generate huge PDFs using report definitions.
- If the settings specified in the `userSettings` parameter are configured to use section by section (rather than experiment by experiment) reports, then the `ExportReportDefinitionToPdf` method reverts to calling the `RunReportDefinition` method. This could result in out of memory errors.

For more information on the possible formats for OutputWriter and other report options, see Specifying Report Options and Specifying User Settings.

Create and Upload a Report Example

The following sample IronPython script creates a custom toolbar button that prompts a user for an experiment, creates a report for the selected experiment, and saves the report to a Vault repository.

To run this script:

1. Create an experiment template with a File Section.
2. Add the example script to the `OnApplicationLoaded` event of the experiment.
3. Check in the experiment template.
4. Create a report template for the experiment template from the previous step, and check it in.
5. Create an experiment using the experiment template you just created, and check it in.
6. Open the experiment you just created.

   You should see an **Upload Report** button that was added by the example script you added to the `OnApplicationLoaded` event.
7. Click **Upload Report**.
8. Select the report template you created.
9. Select the experiment template you created.
10. Select a folder destination for your report.
11. Close the experiment. In Notebook Explorer, you should see the uploaded report in the folder you specified.

Experiment Template Source Example

```
import System from System.Windows.Forms
import DialogResult from System.Windows.Forms
import MessageBox from System.Windows.Forms
import MessageBoxButtons from System.Windows.Forms
import MessageBoxIcon from System.Windows.Forms
import Cursor from System.Windows.Forms
import Cursors from System.Windows.Forms
import ToolStripButton from System.Windows.Forms
import ToolStripSeparator from System.IO
import File from System.IO
import Stream from System.IO
import Path from System.IO
import Directory from System.IO
import SeekOrigin from Symyx.Windows.Controls
import AsynchronousTreeLoadStrategy from Symyx.Framework.IO
import StreamCopier from Symyx.Framework.Vault
import VaultObject from Symyx.Framework.Vault
import VaultObjectType from Symyx.Framework.Vault
import VaultObjectTypes from Symyx.Framework.Vault
import VaultId from Symyx.Framework.Vault
import VaultUri from Symyx.Framework.Vault
import DataScope from Symyx.Framework.Vault
import VaultIdPrefixes from Symyx.Framework.Controls
import WorkspaceTreeNodeFactory from Symyx.Framework.Controls
import SelectFromLoadableTreeDialog from
       Symyx.Framework.Reporting.Scrivo
import ReportOptions from Symyx.Framework.Reporting.Scrivo
import OutputDisplayOption from Symyx.Framework.Reporting.Scrivo
import RunMode from Symyx.Framework.Reporting.Scrivo.IO
import VaultReportDefinition from
       Symyx.Notebook.Reporting.Scrivo.Helpers
import VaultSettingsHelper from Symyx.Notebook
import Document from Symyx.Notebook.Dialogs
import WorkspaceBrowseDialog from
       Symyx.Notebook.Reporting.Scrivo
import NotebookReporter

def get_data(sender, args) :
    args.DataDescriptor.DataObject = editor.Document

def write_stream_to_file(stream, file_name) :
    temp_folder = Path.GetRandomFileName()
                  Directory.CreateDirectory(temp_folder)
    temp_file = Path.Combine(temp_folder, file_name)
    bytes = System.Array.CreateInstance(System.Byte, stream.Length);

stream.Read(bytes, 0, stream.Length)
File.WriteAllBytes(temp_file, bytes)
return temp_file

def run_report(report_vault_uri) :
    options = ReportOptions()
    options.OutputOption = OutputDisplayOption.Store
    options.OutputWriter = 'Adobe Acrobat'
    notebook_reporter = NotebookReporter()
    notebook_reporter.DataRequired += get_data

report = active_workspace.Get(report_vault_uri, DataScope.All) if
report.VaultId.Prefix == VaultIdPrefixes.ReportDefinition :
notebook_reporter.RunReportDefinition(report, options,
VaultSettingsHelper.RetrieveUserSettingsForReportDefinition(report.Title),
RunMode.Batch)

if report.VaultId.Prefix == VaultIdPrefixes.ReportTemplate :
notebook_reporter.Run(report, options, RunMode.Batch)
notebook_reporter.Dispose() return options.OutputStream

def create_report_experiment(pdf_file_path) :
    template_vault_uri = browse_vault(VaultObjectTypes
        (System.Array[VaultObjectType]
          ([VaultObjectType.DocumentTemplate])
         ),
         VaultObjectType.DocumentTemplate,
         "Browse for Experiment Template")
if template_vault_uri == None : return None

template = active_workspace.Get(template_vault_uri, DataScope.All)
document = Document.Create(template)
document.Title = editor.Document.Title + " Report"

file_paths = System.Array.CreateInstance(object, 1)
file_paths[0] = pdf_file_path
fileSection = document[0]
file_packages = fileSection.BuildersManager.BuildExternalFilesPackages(file_
paths,False) fileSection.Files.InsertRange(0, file_packages, True)
return document

def upload_experiment_to_vault(document) :
    folder_vault_uri = browse_vault_folders()
    if folder_vault_uri == None :
return folder = active_workspace.Get(folder_vault_uri,DataScope.All)
                 active_workspace.Add(document, folder)
return "Uploaded report to document '" + document.Title + "' in folder '" +
folder.Title + "'"

def export(sender, e) :
    Cursor.Current = Cursors.WaitCursor
    report_vault_uri = browse_vault(VaultObjectTypes
        (System.Array[VaultObjectType]
          ([VaultObjectType.ReportTemplate,
           VaultObjectType.ReportDefinition])
         ),
         VaultObjectType.ReportTemplate,
         "Browse for Report")
if report_vault_uri == None :
   Cursor.Current = Cursors.Default
   MessageBox.Show("Upload canceled.", editor.Title, MessageBoxButtons.OK,
                    MessageBoxIcon.Information)
return

output_stream = run_report(report_vault_uri) if output_stream == None :
  Cursor.Current = Cursors.Default
  MessageBox.Show("No report to upload.", editor.Title,
MessageBoxButtons.OK,
                  MessageBoxIcon.Information)
return
```

```
temp_pdf_file = write_stream_to_file(output_stream,
                editor.Document.Title + " Report.pdf")

document = create_report_experiment(temp_pdf_file) if document == None :
  File.Delete(temp_pdf_file)
  Cursor.Current = Cursors.Default
  MessageBox.Show("Upload canceled.", editor.Title, MessageBoxButtons.OK,
                  MessageBoxIcon.Information)
return

message = upload_experiment_to_vault(document) if message == None :
message = "Upload canceled."

File.Delete(temp_pdf_file)
  Cursor.Current = Cursors.Default
  MessageBox.Show(message, editor.Title, MessageBoxButtons.OK,
                  MessageBoxIcon.Information)

def browse_vault(vaultObjectTypes, default_vault_object_type, title) :
  dialog = WorkspaceBrowseDialog(vaultObjectTypes,
           VaultObjectType.ReportTemplate,
           WorkspaceTreeNodeFactory.RepositoryTypes.PrivateAndVault)
  dialog.Text = title
  dialog.AllowReadOnlyOpen = False

if dialog.ShowDialog() != DialogResult.Cancel
and dialog.SelectedItems.Count > 0:
  vaultObject = dialog.SelectedItems[0] dialog.Dispose()
return vaultObject.VaultUri

dialog.Dispose() return None

def browse_vault_folders():
    factory = WorkspaceTreeNodeFactory
              (active_workspace,
               VaultObjectTypes.Repository | VaultObjectTypes.Folder,
               WorkspaceTreeNodeFactory.RepositoryTypes.Vault)
    strategy = AsynchronousTreeLoadStrategy(factory)
    dialog = SelectFromLoadableTreeDialog("Select Folder", strategy)
     if dialog.ShowDialog() == DialogResult.OK :
        selected_node = dialog.SelectedNode dialog.Dispose()
return selected_node.GetObject[VaultId]()

dialog.Dispose() return None

tool_strip = editor.GetToolStrip('standardToolStrip')
tool_strip_item1 = ToolStripButton
                   ('Upload Report', None, export, 'uploadToolStripButton')

tool_strip.Items.Add(ToolStripSeparator())
tool_strip.Items.Add(tool_strip_item1)
```

Remove the Notebook Reporter from Memory

The `NotebookReporter` class implements the `IDisposable` interface. When the Notebook Reporter is no longer needed, you must remove the Notebook Report to free memory and resources using the `IDisposable` interface. Before disposing of the Notebook Reporter, you should remove the `DataRequired` event handler:

```
notebookReporter.DataRequired -= notebookReporter_DataRequired;
notebookReporter.Dispose();
```

Chapter 5: Interactive Protocols

Interactive Pipeline Pilot Protocols

If you have template editor permissions, you can configure a dynamic toolbar item to run interactive BIOVIA Pipeline Pilot protocols that displays a sequence of interactive web pages, before returning data files to BIOVIA Workbook. You can view a protocol in a browser dialog that enables user interaction.

You must grant Workbook users the Pipeline Pilot **RunProtocol** application permission.

Reaction Condition Dynamic Toolbar Item

If you have template editor permissions, you can create a template, then create a Reaction Condition dynamic toolbar item, and assign the `RunProtocol` permission to the dynamic toolbar item. Scientists can browse and attach a protocol to the experiment. When the protocol is attached to the dynamic toolbar item, the Script is automatically populated.

All the protocols in a case-insensitive folder named Interactive are available as Interactive protocols.

The interactive folder can reside anywhere in the path such as `myFolder\INTERACTIVE` or `myFolder\interactive\reaction-protocols`.

The scientists set the input data and output data in the protocol.

Enabling Interactive Pipeline Pilot Protocols

The protocol should place all the output files in the first job directory. On completion of the protocol, use a post execution script to inform BIOVIA Workbook that the protocol has completed, for example:

```
"window.external.RaiseGenericEvent('Finish','');"
```

BIOVIA Workbook downloads the result files when the protocol fires the `Finish` event.

The following script cancels the protocol:

```
"window.external.RaiseGenericEvent('Cancel','');"
```

Sequence Protocol

A Sequence protocol returns a series of interactive web pages before returning data files to BIOVIA Workbook. Any interaction on the page is handled by the Sequence protocol, and could return subsequent pages for display and further interaction. The protocol must trigger a `Finish` event to notify BIOVIA Workbook that the protocol has completed. When the protocol is completed, the final result files are transferred to BIOVIA Workbook.

Disabling Interactive Protocols

By default, the `InteractiveProtocolEnabled` configuration setting is set to *true* so that the protocols in the Interactive folder open in the browser window, and allow interactivity. You can set the `InteractiveProtocolEnabled` property to *false*, so that the protocols run as non-interactive protocols.

To disable interactive protocols:

1. Upgrade the templates or set up the template mappings.
2. Create a backup copy of the `Symyx.Notebook.Application.exe.config` file. For example, as `Symyx.Notebook.Application.exe.configbackup`.
3. Edit the `Symyx.Notebook.Application.exe.config` file as follows:

   ```
   <setting name="InteractiveProtocolEnabled" serializeAs="String">
     <value>False</value>
   </setting>
   ```

4. Save and close the file.

Chapter 6: Managing Workbook Using Import and Export

This chapter provides information and support for managing importing and exporting with Workbook.

Exporting and Importing

You can use export and import to:

- Reduce the effort associated with validating the BIOVIA Workbook system configuration.
- Promote system configuration items such as templates, and forms from one server to another.
- Transfer the current version of the item is exported to a Contract Research Organization (CRO).

  However, you cannot transfer electronic records as defined in 21 CFR 11. Audit history is not exported. Information relating to the persons that authored the content is not preserved.

IMPORTANT!

- You can export an experiment or an experiment template with a File section containing files.
- You cannot import the experiment or experiment template with a File section containing files. This is a known limitation.

  To workaround this limitation you can:

  1. Save the files in the File section to a location on your computer.
  2. Delete the files from the source experiment or template.
  3. Export the experiment or template.
  4. Import the experiment or template.
  5. Import the files from your computer back into Workbook.

Configuration Best Practices

Proper configuration management is critical to a customer's validation efforts. You should provision three environments:

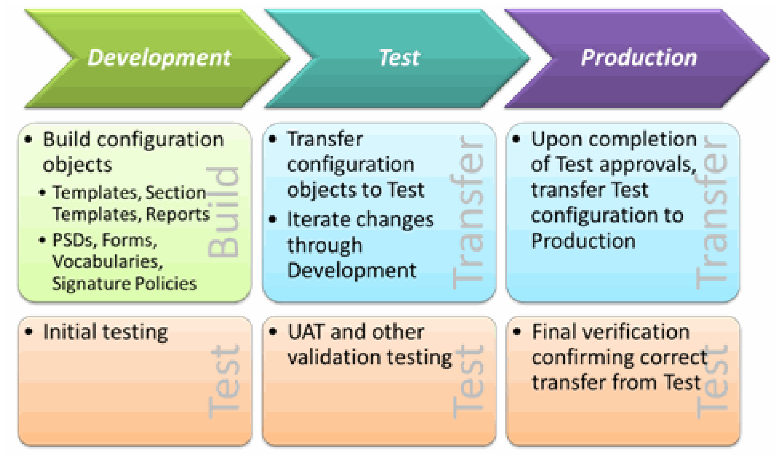
- Production

  An environment used for actual work. This environment is strictly controlled to ensure that change is introduced in a manner that is consistent with the customer's standard operating procedures.
- Test

  A controlled environment meant to mimic the production as closely as possible. This environment is used to prove that pending changes work correctly and that the deployment procedures are correct. This environment should be reserved exclusively for testing.
- Development

  An environment used for new development and initial testing. Making mistakes in development does not affect systems in active use. When new development is ready for final testing, it can be staged for promotion to the Test environment.

The following graphic illustrates the recommended approach to building and deploying BIOVIA Workbook configuration items between environments.


Development
Test
Production
Build configuration objects
Templates, Section Templates, Reports
PSDs, Forms, Vocabularies, Signature Policies
Build
Transfer configuration objects to Test
Iterate changes through Development
Transfer
Upon completion of Test approvals, transfer Test configuration to Production
Transfer
Initial testing
Test
UAT and other validation testing
Test
Final verification confirming correct transfer from Test
Test

Items are built only in the development environment. Items are deployed to test and then production, using the export/import to ensure that what was developed is exactly what was tested and deployed to production.

Exporting

Exportable Items

You can export the following items:

- Document Templates
- Section Templates
- Forms
- Operations
- Vocabularies
- Property Set Definitions
- Signature Policies.

You can select one item at a time for export. When exporting an item, any dependencies the item relies on are also exported. You can transfer a fully-functional Document Template to the destination system.

The most recently checked-in version of each item is exported. You must grant users the `TransferTemplates` application permission to use the export command.

Exclusions from Dependency Export

The export functionality detects dependencies referenced by its Vault ID. Export does not detect dependencies referenced by title. You must load the dependencies using other methods such as by publishing a VOZIP package.

Pipeline Pilot Client protocols are not included in the export because the protocols reside on the Pipeline Pilot. You must use Pipeline Pilot Client to manually move the protocols.

Stripping Properties During Export

For properties in the source code that you do not want in production, place the text `#DELETE#` in the Category field. The property remains in the source system but is not transferred to the destination system.

Importing

Importable Items

During import determine whether any of the items in the exported file exist in the destination system. The import code performs this resolution process when the export file is opened.

You must grant users the `TransferTemplates` application permission to use the import command.

The BIOVIA Workbook scientist views the results of the resolution process in the import dialog. When possible, the import process suggests appropriate import actions such as Add, Update, or Relink. The import process does not suggest an action when one is not clear and prevents actions that could create data integrity issues.

The original resolution result indicates whether the item was found in the destination system. The system first tries to match on Vault ID and then attempts to match on Vault Path. If the attempts yield a match, the original result is Missing. When the result is missing, the scientist can browse to an existing item if they know that it exists in the destination system.

An OK in the status indicates the item is ready to import. Ambiguous is displayed when more than one item is a potential match. Ambiguous indicates the user must make an additional selection. Incompatible is displayed when a PSD or Form cannot be imported due to a data integrity constraint.

All items are suitable for import when the status is OK. Audit entries are created in the destination system for each imported item. Upon completion of the import, the user can save a copy of the Import Session Report.

Import Guidelines

When importing a signature policy from a `*.voexp` file into BIOVIA Vault Server, import the policy in the root level of the Site repository. If the `.voexp` file is imported into a subfolder, you cannot edit the signature policy.

Property Set Definitions for Import Operations

Property Set Definitions and Forms are subject to additional data integrity checks to ensure that an import does not adversely affect the destination system.

All shared properties, those that exist in both the source and target PSD, must use the same data type and have the same indexing behavior. The import cannot result in the removal of a property in the target system. The following table describes the actions available to the user in various PSD import scenarios by examining a single PSD containing properties1 – 3 in various combinations.

Source	Target	Available Import Actions	Notes
Property1, Property2	Property1, Property2, Property3	Relink	Update is not applicable because it removes the property on the target.
Property1, Property2, Property3	Property1, Property2	Update	Relink is not applicable because an imported item might rely on Property3
Property1, Property2	PSD does not exist	Add	

Source	Target	Available Import Actions	Notes
Property1, Property2	Property1, Property2	Update, Relink	Either option is valid; the user selects the appropriate option.
Property1(Integer), Property2	Property1(Date), Property2	None	Property1 is a different data type. The PSD was not reconciled, and the import is blocked

Data Integrity Rules for Importing

Property Set Definitions and Forms are subject to data integrity checks to ensure an import does not adversely affect the destination system.

Forms are checked to ensure that all widgets in the destination form exist in the form to be imported. The data types of all widgets must match.

Import Session Report

All activity is recorded in the Import Session Report. You can save the Import Session report when the import session is completed and is also attached to the audit entries generated for each imported item.

Import session reports can be imported easily into Microsoft Excel for analysis or formatting.

The XML format is:

```
<?xml version="1.0" encoding="utf-8"?>
  <ImportSessionReport>
    <ImportSession>
      <ImportSessionID></ImportSessionID>
       <TargetServer></TargetServer>
       <ImportedOn></ImportedOn>
       <ImportedByUsername></ImportedByUsername>
       <ImportedByUserID></ImportedByUserID>
       <SourceFile></SourceFile>
       <SourceFileCreationDate></SourceFileCreationDate>
       <SourceServer></SourceServer>
    </ImportSession>
    <ItemsToImport>
      <ImportItem>
        <ItemCategory></ItemCategory>
        <ResolutionStatus></ResolutionStatus>
        <Type></Type>
        <Title></Title>
        <SourceVaultPath></SourceVaultPath>
        <TargetVaultPath></TargetVaultPath>
        <ImportAction></ImportAction>
        <SourceVaultID></SourceVaultID>
        <SourceVersion></SourceVersion>
        <TargetVaultID></TargetVaultID>
        <TargetVersion></TargetVersion>
        <OriginalStatus></OriginalStatus>
        <OriginalTargetVaultPath></OriginalTargetVaultPath>
      </ImportItem>
    </ItemsToImport>
  </ImportSessionReport>
```

Import Session Report Data Elements

The following table lists each of the Import Session Report data elements and provides a description for each element.

Element	Description
<?xml version="1.0" encoding="utf-8"?>	Specifies the character encoding used in the report
<ImportSessionReport>	Defines a complete session report
<ImportSession>	Contains the metadata for a single import session
<ImportSessionID></ImportSessionID>	Unique Identifier for a single import session
<TargetServer></TargetServer>	The server to which the items are imported
<ImportedOn></ImportedOn>	The date and time (in GMT) the import session started
<ImportedByUsername></ImportedByUsername>	The account name of the user performing the import
<ImportedByUserID></ImportedByUserID>	The unique identifier of the user performing the import
<SourceFile></SourceFile>	The full filepath of the file used as the source for the import
<SourceFileCreationDate></SourceFileCreationDate>	The date and time (in GMT) the source file was created
<SourceServer></SourceServer>	The name of the server from which the source items were exported
</ImportSession>	Ends the import session metadata definition
<ItemsToImport>	Contains the metadata for each item imported during a single import session
<ImportItem>	Contains the metadata for a single item that was imported
<ItemCategory></ItemCategory>	Indicates whether the item was a Primary or Dependency. Primary items were selected by exporting user, while dependencies were automatically include by the system to ensure that all necessary items are present on the target server.
<Type></Type>	The type of VaultObject being imported

Element	Description
<Title></Title>	The title of the VaultObject being imported
<SourceVaultPath></SourceVaultPath>	The VaultPath of the item being imported as it existed in the source system
<TargetVaultPath></TargetVaultPath>	The location of the imported VaultObject in the target system
<ImportAction></ImportAction>	The action taken on the imported item Valid values are Add, Update, and Relink. Add indicates that a new VaultObject was added to the target system. Update indicates that a new version was created for an existing VaultObject in the target system. Relink indicates that the existing version of a VaultObject in the target system was used.
<SourceVaultID></SourceVaultID>	The unique identifier of the VaultObject that was exported from the source system
<SourceVersion></SourceVersion>	The version number of the VaultObject that was exported from the target system
<TargetVaultID></TargetVaultID>	The unique identifier of the imported VaultObject in the target system
<TargetVersion></TargetVersion>	The version number of the imported VaultObject in the target system
<OriginalStatus></OriginalStatus>	Indicates whether the item was found on the target server Valid values are Found, Missing, and Ambiguous. Found indicates a suggested match was found. Missing indicates that the system was unable to find a match. Ambiguous indicates that the system found more than one possible match.
<OriginalTargetVaultPath></OriginalTargetVaultPath>	The suggested location for the imported VaultObject
</ImportItem>	Ends the metadata for a single item that was imported
</ItemsToImport>	Ends the metadata for each item imported during a single import session
</ImportSessionReport>	Ends the complete session report definition

Import Troubleshooting

The primary cause of import failures is security. Verify that the user performing the import has Update Properties and Write Data for each item that is updated or added. The import process terminates when it is unable to write to the destination system.

When Property Set Definitions or Forms are incompatible, the user must review the source and destination objects to determine the problem.

Data type differences issues are difficult to resolve when importing property set definitions. To resolve import issues in forms, add the form to the destination system to mitigate the data type incompatibility.

For help troubleshooting and resolving incompatibility issues, contact Dassault Systèmes Customer Support.

Chapter 7: Using Extended Units

This chapter provides information and support for using extended units with Workbook.

Workflow for Using the Extended Units Utility

BIOVIA recommends thoroughly testing custom units before the units putting the units into production.

The following steps provide an overview for using the Extended Units Utility:

- Run the Extended Units Utility using the -dump option.
- Copy the `ExtendedUnits_Example1.xml.txt` template file and save it with a different name.
- Edit the copy of the file in an text editor and save it.
- Rerun the utility to publish your customized unit categories and units.

Extended Units Utilities Files

The following table lists the files for extending units. These files are located in the `C:\Program Files (x86)\BIOVIA\Vault Administration\ExtendedUnitsUtility` folder on the server or client machine on which the Vault Administration Console is installed.

File name	Description
`ExtendedUnitsUtility.exe`	The Extended Units Utility executable file. Run this utility from a command window on the Vault Server machine.
`ExtendedUnits_Example1.xml.txt`	XML file template that you can use to create new units in an existing units category.
`ExtendedUnits_Example2.xml.txt`	XML file template that you can use to create new unit category, units, and unit groups.

Extended Unit Utility Required Parameters

The following table lists the required parameters. All of the parameters are case sensitive.

If the parameter value is not entered, the error message Not enough input to run the utility appears.

Parameter	Command line argument	Default value	Description
Name	-u	n/a	Specifies the domain\administrator user name on the Vault server for the installation of the Extended Units Utility
Password	-p	n/a	Specifies the password of the user account specified above.
Server	-s	n/a	Specifies the name of the Vault server where the Extended Units Utility resides.

Parameter	Command line argument	Default value	Description
File name	-f	n/a	(Optional) Not required the first time the Extended Units Utility is run. After the you have created a customized XML file, enter that unique file name. The file is then published.
Dump file name	-dump	n/a	Provides a name for generated dump file. You can select any file name. Create an unique name. Existing files are overwritten. You can include the path to where you want the file saved. If you do not enter a path, the file is saved in the folder where the utility resides, for example, MMC.

Run the Extended Units Utility

To run the Extended Units utility:

1. Open a command prompt window on a client computer.
2. Change the directory to the folder containing the BIOVIA Workbook.
3. In the \MMC folder, type:

   ```
   ExtendedUnitsUtility.exe -u domain\username -p password -s servername -
   dump filename
   ```

   For example:

   ```
   ExtendedUnitsUtility.exe -u xyz\greg.green -p Good2Day! -s server1.2k8 -
   dump c:\temp\UnitsFile.xml
   ```

Add Unit Categories, Units, and Unit Groups

Use the ExtendedUnit_Example2.xml.txt template to add a unit category with units and unit groups.

Note: Do not make changes to the xs:schema section of this file. Your changes can corrupt the XML file.

1. Open the `ExtendedUnit_Example2.xml.txt` file and save it with unique name.
2. Close the `ExtendedUnit_Example2.xml.txt` file, and open the file you just saved.
3. Create the `CFU Unit` category.
   a. Using the `-dump` option in the `ExtendedUnitsUtility.exe` to output all the existing definitions to a file.
   b. Determine an unused negative integer for the new unit category to use as the primary key such as the SymyxID is -15000.
   c. Use unique Name, Symbol, and Type entries.

      For example, ColonyFormingUnits, CFUPERML, and CFU.

4. Create a unit.
   a. For SymyxID, enter a unique negative number as the primary key, for example, -15000.
   b. For Symbol, Name, and LongName, enter unique names.

      For example, CFUPERML, CFU/mL, and Colony Forming Units per milliliter.
   c. Enter zero (0) for Dimensions, Factor, and Offset for non-computable units.
5. Create a UnitGroup.

   The UnitGroup is used to link Unit and UnitCategory together..
   a. For ID, enter a unique, negative number that has not been used before, for example, -15000. This is the primary key
   b. For Unit, enter the negative number that is defined in Unit.SymyxID, for example, -15000. This links the UnitGroup to the Unit.
   c. For UnitCategory, enter the number that is defined in UnitCategory.SymyxID, for example, -15000.
6. Repeat the steps to add the unit Colony Forming Units per gram.

After the file is published, the new unit category, units, and unit groups display in the BIOVIA Workbook Property Set editor.

Add Unit Category and Units Example

In this example, a new unit category, ColonyFormingUnits (CFU), and two new units, Colony Forming Units per mililiter (CFUPERML) and Colony Forming Units per gram (CFUPERGRAM), are added.

```
<!-- CFU is the new UnitCategory-->
<UnitCategory>
  <SymyxID>-15000</SymyxID>
  <!-- SymyxID is the primary key. Use a unique negative that was not
       used for UnitCategory.SymyxID -->
  <Name>ColonyFormingUnits</Name>
  <!-- Name is a unique name for UnitCategory.Name -->
  <Symbol>CFU</Symbol>
  <!-- Symbol must be unique, not used before for
       UnitCategory.Symbol -->
  <Type>CFU</Type>
  <!-- CFU must be unique, not used before for UnitCategory.Type-->
</UnitCategory>

<!-- Colony Forming Units per milliliter. -->
<Unit>
  <Symbol>CFUPERML</Symbol>
  <!-- Symbol must be unique, not used before for Unit.Symbol -->
  <SymyxID>-15000</SymyxID>
  <!-- SymyxID is the primary key.This number must be a negative
       number, not used before for Unit.SymyxID -->
  <Name>CFU/mL</Name>
  <!-- Name must be unique, not used before for Unit.Name -->
  <LongName>Colony Forming Units per milliliter</LongName>
  <!-- LongName must be unique, not used before for Unit.LongName -->
  <Dimensions>0</Dimensions>
  <!-- As a unit that does not participate in calculations,
        this is zero -->
```

```
    <Factor>0</Factor>
    <!-- As a unit that does not participate in calculations,
         this is zero -->
  </Unit>

  <UnitGroup>
    <ID>-15000</ID>
    <!-- ID is the primary key. This must be unique, not used before
         for UnitGroup.ID -->
    <Unit>-15000</Unit>
    <!-- Unit refers to the Unit.SymyxID. -->
    <UnitCategory>-15000</UnitCategory>
    <!-- -15000 is the number defined for CFU category -->
   </UnitGroup>

  <!-- Colony Forming Units per gram. -->
  <Unit>
    <Symbol>CFUPERGRAM</Symbol>
    <!-- Symbol must be unique, not used before for Unit.Symbol -->
    <SymyxID>-15001</SymyxID>
    <!-- SymyxID is the primary key. This must be a negative number, not
         used before for Unit.SymyxID -->
    <Name>CFU/g</Name>
    <!-- Name must be unique, not used before for Unit.Name -->
    <LongName>Colony Forming Units per gram</LongName>
    <!-- LongName must be unique, not used before for Unit.LongName -->
     <Dimensions>0</Dimensions>
    <!-- As a unit that does not participate in calculations,
         this is zero -->
    <Factor>0</Factor>
    <!-- As a unit that does not participate in calculations,
         this is zero -->
    <Offset>0</Offset>
    <!-- As a unit that does not participate in calculations,
         this is zero -->
  </Unit>

  <UnitGroup>
    <ID>-15001</ID>
    <!-- ID is the primary key. This must be unique, not used before
         for UnitGroup.ID -->
    <Unit>-15001</Unit>
    <!-- Unitrefers to the Unit.SymyxID. -->
    <UnitCategory>-15000</UnitCategory>
    <!-- -15000 is the number defined for CFU category -->
  </UnitGroup>

  </UnitsDataSet>
```

ExtendedUnits_Example Files

The `ExtendedUnit_Example1.xml.txt` and `ExtendedUnit_Example2.xml.txt` files are located in the `C:\Program Files (x86)\BIOVIA\Vault Administration\ExtendedUnitsUtility` folder. These XML files are templates that you can use to create customized unit categories, units, and unit groups.

The following table contains the parameters that you can customize and their values.

Parameter	Comments
Unit	
Symbol	Name of your customized unit. Use unique names in uppercase, for example, XGRAM. Do not use names specified as a Unit.Symybol.
SymyxID	This is the primary key. Use an unique negative number. Do not use a negative number that was used as a Unit.SymyxID.
Name	Specifies the name that displays in the unit list. Use an unique name, for example, xg. You cannot use a name that was used as a Unit.Name.
LongName	A longer, more descriptive name. This name must be unique, for example, x-gram, and cannot have been used before as a Unit.Longname.
Dimensions	Search for this parameter in the dump file that was generated when the **.**exe was run using the -dump option.
Factor Offset	Determine the base.Unit and use the formula newUnit = baseUnit*factor + offset For example, if the baseUnit is Kg, then (xg = kg*0.001 + 0)
UnitGroup	
ID	This is the primary key. This must be negative number, for example: -14000, that has not been used before as a Unit.Group
Unit	The same negative number as SymyxID.
UnitCategory	Use the same number as the existing UnitCategory For example, 256, the SymyxID for the unit category Mass. Search for this parameter in the dump file that was generated when the .exe was run.
UnitCategory	
SymyxID	This is the primary key. This must be negative number that has not been used before, for example, -15000, and cannot have been used before as a Unit.SymyxID.
Name	Specifies the name that displays in the unit list. This name must be unique, for example, ColonyFormingUnits, and cannot have been used before as a UnitCategory.Name.

Parameter	Comments
Symbol	This has to be unique, for example, CFU, and cannot have been used before as a UnitCategory.Symbol.
Type	This has to be unique, for example, CFU, and cannot have been used before as a UnitCategory.Type.
Unit	
Symbol	This has to be unique and cannot have been used before as a UnitCategory.Symbol, for example, CFUPERML.
SymyxID	This is the primary key. This must be negative number that has not been used before, for example, -15000, and cannot have been used before as a Unit.SymyxID.
Name	Specifies the name that displays in the unit list. Select an unique name, for example, CFG/mL, that was not used before as a Unit.Name.
LongName	A longer, more descriptive name. This name must be unique, for example, Colony Forming Units per milliliter, and cannot have been used before as a Unit.Longname.
Dimensions	As a unit that does not participate in calculations, this is zero (0).
Factor Offset	As a unit that does not participate in calculations, these are zero (0).
UnitGroup	
ID	This is the primary key. This must be negative number, for example: -15000, that has not been used before as a UnitGroup.ID.
Unit	The same negative number as defined for Unit.SymyxID, for example, -15000.
UnitCategory	This is the same negative number as defined for CFU category, for example, -15000.

ExtendedUnit_Example1.xml.txt

If you add a unit to the Mass-metric category or the Volume-metric category, you also must add this unit to the main Mass or Volume categories. An error occurs if you have not added the sub-category unit to its main category.

The `ExtendedUnit_Example1.xml.txt` file is a template for add custom units within an existing unit category.

To add custom units:

1. Open the `ExtendedUnit_Example1.xml.txt` file and save it with unique, descriptive name to a location of your choice, for example, `C:\temp\CustomMassUnits.xml`.
2. Close the `ExtendedUnit_Example1.xml.txt` file, and open the file you just saved, for example, `CustomMassUnits.xml`.

Do not make changes to the xs:schema section of this file, changes corrupt the XML file.

3. Scroll to the `<UnitCategory>` section to see the parameters that you can customize by editing the parameters.
4. In another window, open in a XML editor the dump file that was created when you ran the `ExtendedUnitsUtility.exe` file. See Run the Extended Units Utility.

   This file contains all of the existing unit definitions.
5. In the dump file, search for and copy the unit categories that you want customize, for example, VOLUME and MASS as shown in the examples.

   The parameters of the `UnitCategory` that you use must exactly match the parameters found in the dump file. An error displays if there are discrepancies.
6. Return to the file that you are customizing and paste these parameters into `<UnitCategory>` section.
7. Enter your custom parameter values to the `Unit` and `UnitGroup` sections.

   For more information on the parameters, see the table.
8. After you have completed customizing the file, save it and rerun the `ExtendedUnitsUtility.exe` to publish your customized Units.
9. Optionally, after you have rerun the `.exe` file using the `-dump` option, you can open and scroll through the dump file to verify that your customized Units have been published.
10. After files are published, the customized units display in the BIOVIA Workbook Property Set editor.

Custom Units Examples

Example 1

In this example, KILOLITER (a custom unit) is added to the Volume(metric) UnitCategory. When the new custom unit is added to the Volume(metric) category, the new unit must also be added to the Volume category.

```
<Unit>
  <Symbol>KILOLITER</Symbol> <!-- Custom unit KILOLITER  -->
  <SymyxID>-9001</SymyxID>   <!-- Unique SymyxID -9001 in Unit -->
  <Name>kL</Name>
  <LongName>kilo liter</LongName>
  <Factor>1</Factor>
  <Dimensions>2</Dimensions>
  <Offset>0</Offset>
</Unit>

<UnitCategory>
  <SymyxID>15872</SymyxID> <!-- UnitCategory 15872 is VOLUME METRIC -->
  <Name>volume (metric)</Name>
  <Symbol>VOLUME_METRIC</Symbol>
  <Type>VolumeMetric</Type>
</UnitCategory>

<UnitGroup>
  <ID>-9001</ID>
  <Unit>-9001</Unit>  <!-- Refer to KILOLITER Unit.SymyxID is -9001-->
  <UnitCategory>15872</UnitCategory>
</UnitGroup>
```

```
<!-- Add into Volume category. Refer to the dump file and get VOLUME
category information -->
<UnitCategory>
  <SymyxID>512</SymyxID>
  <Name>volume</Name>
  <Symbol>VOLUME</Symbol>
  <Type>Volume</Type>
</UnitCategory>

<!-- Create a new UnitGroup for Unit -9001(KILOLITER) -->
<UnitGroup>
  <ID>-9002</ID>      <!-- Unique negative ID in UnitGroup -->
  <Unit>-9001</Unit>  <!-- Refer to KILOLITER Unit.SymyxID is -9001-->
</UnitGroup>

<!-- 512 is VOLUME UnitCategory.SymyxID -->
<UnitCategory>512</UnitCategory>
```

Example 2

In this example, DECAGRAM (a custom unit) is added into Mass(metric) UnitCategory. When new custom unit is added to Mass(metric) category, you must add the new unit to the Mass category.

```
<Unit>
  <Symbol>DECAGRAM</Symbol>  <!-- Custom unit DECAGRAM  -->
  <SymyxID>-9005</SymyxID>   <!-- Unique SymyxID -9005 in Unit -->
  <Name>dg</Name>
  <LongName>deca gram</LongName>
  <Factor>0.01</Factor>
  <Dimensions>1</Dimensions>
  <Offset>0</Offset>
</Unit>

<UnitCategory>
  <SymyxID>15616</SymyxID>  <!-- UnitCategory 15616 is MASS METRIC -->
  <Name>mass (metric)</Name>
  <Symbol>MASS_METRIC</Symbol>
  <Type>MassMetric</Type>
</UnitCategory>

<!-- Add into MASS category. Refer to the dump file and get MASS
category information -->
<UnitCategory>
  <SymyxID>256</SymyxID>
  <Name>mass</Name>
  <Symbol>MASS</Symbol>
  <Type>Mass</Type>
</UnitCategory>

<UnitGroup>
  <ID>-9005</ID>
  <Unit>-9005</Unit>  <!-- DECAGRAM unit. Unit.SymyxID is -9005 -->
  <UnitCategory>15616</UnitCategory>
```

```
</UnitGroup>

<UnitGroup>
  <ID>-9006</ID>
  <Unit>-9005</Unit>  <!-- DECAGRAM unit. Unit.SymyxID is -9005 -->
</UnitGroup>

<UnitCategory>256</UnitCategory></UnitGroup>
```

ExtendedUnit_Example1 XML Schema

Do not make changes to the xs:schema part of this file, changes corrupt the XML file.

```
<UnitsDataSet xmlns="http://tempuri.org/UnitsDataSet.xsd">

<xs:schema id="UnitsDataSet"
targetNamespace="http://tempuri.org/UnitsDataSet.xsd"
xmlns:mstns="http://tempuri.org/UnitsDataSet.xsd"
xmlns="http://tempuri.org/UnitsDataSet.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault="qualified"
elementFormDefault="qualified">

<xs:element
name="UnitsDataSet"
msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
.
.
.
</xs:schema>

<!-- Use the following section of this example file to help you customize
Unit categories and Units.-->

<UnitCategory>
<!-- Use the -dump option in ExtendedUnitsUtility.exe to output all the
existing definitions to a file -->
  <SymyxID>256</SymyxID> <!-- The primary key in the dump file -->
  <Name>mass</Name>      <!-- Get this from the dump file -->
  <Symbol>MASS</Symbol>  <!-- Get this from the dump file -->
  <Type>Mass</Type>      <!-- Get this from the dump file -->
</UnitCategory>

<Unit>
<!-- xgram is the equivalent to gram. -->
  <Symbol>XGRAM</Symbol>
  <!-- Symbol Must be unique, not used before for Unit.Symbol -->
  <SymyxID>-14000</SymyxID>
  <!-- SymyxID is the primary key. This number must be a negative
       number, not been used before for Unit.SymyxID -->
  <Name>xg</Name>
  <!-- Name must be unique, not used before for Unit.Name -->
  <LongName>x-gram</LongName>
  <!-- LongName must be unique, not used before for Unit.LongName -->
  <Dimensions>1</Dimensions> <!-- Get this from the dump file -->
  <Factor>0.001</Factor>
  <!-- The baseUnit is Kg. Therefore,
          newUnit = baseUnit*factor + offset (xg = Kg*0.001 + 0) -->
  <Offset>0</Offset>
</Unit>

<UnitGroup>
  <ID>-14000</ID>
  <!-- ID is the primary key. This must be unique, not used before
       for UnitGroup.ID -->
  <Unit>-14000</Unit> <!-- This refers to the Unit.SymyxID. -->
 <UnitCategory>256</UnitCategory>
 <!-- 256 is the number defined for Mass category -->
</UnitGroup>

<Unit>
  <Symbol>XMILLIGRAM</Symbol>
  <SymyxID>-14001</SymyxID>
  <Name>xmg</Name>
  <LongName>x-milligram</LongName>
  <Dimensions>1</Dimensions>
  <Factor>0.000001</Factor>
  <Offset>0</Offset>
</Unit>

<UnitGroup>
  <ID>-14001</ID>
  <Unit>-14001</Unit>
  <UnitCategory>256</UnitCategory>
</UnitGroup>

<Unit>
  <Symbol>XMEGAGRAM</Symbol>
  <SymyxID>-14002</SymyxID>
  <Name>xMg</Name>
  <LongName>x-megagram</LongName>
  <Dimensions>1</Dimensions>
  <Factor>1000</Factor>
  <Offset>0</Offset>
</Unit>

<UnitGroup>
  <ID>-14002</ID>
  <Unit>-14002</Unit>
  <UnitCategory>256</UnitCategory>
</UnitGroup>

</UnitsDataSet>
```

ExtendedUnit_Example2.xml.txt

```
<UnitsDataSet xmlns="http://tempuri.org/UnitsDataSet.xsd">

<xs:schema id="UnitsDataSet"
targetNamespace="http://tempuri.org/UnitsDataSet.xsd"
xmlns:mstns="http://tempuri.org/UnitsDataSet.xsd"
xmlns="http://tempuri.org/UnitsDataSet.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault="qualified"
elementFormDefault="qualified">

<xs:element name="UnitsDataSet"
            msdata:IsDataSet="true"
            msdata:UseCurrentLocale="true">

<xs:complexType>

<xs:choice minOccurs="0" maxOccurs="unbounded">

<xs:element name="Dimensions">
  <xs:complexType>
    <xs:all>
      <xs:element name="SymyxID" type="xs:int" />
      <xs:element name="Symbol" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Length" type="xs:int" minOccurs="0" />
      <xs:element name="Time" type="xs:int" minOccurs="0" />
      <xs:element name="Mass" type="xs:int" minOccurs="0" />
      <xs:element name="LuminousIntensity"
                  type="xs:int" minOccurs="0" />
      <xs:element name="ElectricCurrent" type="xs:int" minOccurs="0" />
      <xs:element name="Mole" type="xs:int" minOccurs="0" />
      <xs:element name="ThermodynamicTemperature"
                  type="xs:int" minOccurs="0" />
      <xs:element name="ReferenceUnit" type="xs:int" minOccurs="0" />
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="Unit">
  <xs:complexType>
       <xs:all>
      <xs:element name="Symbol" minOccurs="1">
        <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="SymyxID" type="xs:int" />
      <xs:element name="Name" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="LongName" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Factor" type="xs:double" minOccurs="1" />
      <xs:element name="Dimensions" type="xs:int" minOccurs="1" />
      <xs:element name="Offset" type="xs:double" minOccurs="1" />
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="UnitCategory">
  <xs:complexType>
    <xs:all>
      <xs:element name="SymyxID" type="xs:int" />
      <xs:element name="Name" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Symbol" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Description" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Type" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
            <xs:minLength value="1" />
            <xs:pattern value=".*[^\s].*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="UnitGroup">
  <xs:complexType>
    <xs:all>
      <xs:element name="ID" msdata:AutoIncrement="true"
                  type="xs:int" />
      <xs:element name="Unit" type="xs:int" minOccurs="1" />
      <xs:element name="UnitCategory" type="xs:int" minOccurs="1" />
    </xs:all>
  </xs:complexType>
</xs:element>

</xs:choice>

</xs:complexType>

<xs:unique name="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:Dimensions" />
  <xs:field xpath="mstns:SymyxID" />
</xs:unique>

<xs:unique name="Unit.SymyxID" msdata:ConstraintName="Constraint01"
           msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:Unit" />
  <xs:field xpath="mstns:SymyxID" />
</xs:unique>

<xs:unique name="Unit.Symbol" msdata:ConstraintName="Constraint02">
  <xs:selector xpath=".//mstns:Unit" />
  <xs:field xpath="mstns:Symbol" />
</xs:unique>

<xs:unique name="Unit.Name" msdata:ConstraintName="Constraint03">
  <xs:selector xpath=".//mstns:Unit" />
  <xs:field xpath="mstns:Name" />
</xs:unique>

<xs:unique name="Unit.LongName" msdata:ConstraintName="Constraint04">
  <xs:selector xpath=".//mstns:Unit" />
  <xs:field xpath="mstns:LongName" />
</xs:unique>

<xs:unique name="UnitCategory.SymyxID" msdata:ConstraintName="Constraint05"
           msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:UnitCategory" />
  <xs:field xpath="mstns:SymyxID" />
</xs:unique>

<xs:unique name="UnitCategory.Name"
           msdata:ConstraintName="Constraint06">
  <xs:selector xpath=".//mstns:UnitCategory" />
  <xs:field xpath="mstns:Name" />
</xs:unique>

<xs:unique name="UnitCategory.Symbol"
           msdata:ConstraintName="Constraint07">
  <xs:selector xpath=".//mstns:UnitCategory" />
  <xs:field xpath="mstns:Symbol" />
</xs:unique>

<xs:unique name="UnitGroup.ID" msdata:ConstraintName="Constraint08"
           msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:UnitGroup" />
  <xs:field xpath="mstns:ID" />
</xs:unique>

</xs:element>

<xs:annotation>
  <xs:appinfo>
    <msdata:Relationship name="DimensionsUNIT"
                         msdata:parent="Dimensions"
                         msdata:child="Unit"
                         msdata:parentkey="SymyxID"
                         msdata:childkey="Dimensions" />
    <msdata:Relationship name="UnitCategoryUnitGroup"
                         msdata:parent="UnitCategory"
                         msdata:child="UnitGroup"
                         msdata:parentkey="SymyxID"
                         msdata:childkey="UnitCategory" />
    <msdata:Relationship name="UNITUnitGroup"
                         msdata:parent="Unit"
                         msdata:child="UnitGroup"
                         msdata:parentkey="SymyxID"
                         msdata:childkey="Unit" />
  </xs:appinfo>
</xs:annotation>
```

```
</xs:schema>
```