Kelverion

RUNBOOK STUDIO

For Microsoft Azure Automation

User Guide

Version 5.6.1

Microsoft Azure Certified

Kelverion Runbook Studio

Kelverion Runbook Studio is Microsoft Azure Certified

Published: November 2023

Feedback

Send suggestions and comments about this document to support@kelverion.com

Contents

Notices 5

Installation and Setup 6

System Requirements 6

Installing Runbook Studio 6

Runbook Studio License 7

Connecting to Microsoft Azure Automation 8

Introduction to Runbook Studio 11

Runbook Canvas 11

Activity Status 11

Link Status 12

Zoom Control 13

Drag and Drop Interaction Modes 13

Explorer Panel 15

The Toolbox Group 15

The Azure Group 15

Properties Panel 19

Connecting to Microsoft Azure 20

Working with Graphical Runbooks 21

Runbook Files 21

Runbook Input and Output 23

Error Handling 23

Logging and Tracing 24

Testing Runbooks 25

Working with Activities 25

Disabling Runbook Activities 26

Retry Behavior 26

Additional Parameters 28

Working with Activity Links 28

The Databus 29

Link Conditions 29

Error Links 30

Cycles 30

Working with Global Assets 30

Publishing Runbooks to Azure 31

Activity Types 32

Command Activity 33

Invoke Runbook Activity 35

Code Activity *35*
Junction Activity *35*
Smart Activity *36*
Generating PowerShell 38
Working with Version Control 40
Getting Started with Version Control 40
Profiles *40*
Connecting to Git Hosting Services *41*
Working with Repositories 43
Initializing a Repository *43*
Cloning a Repository *44*
Recording Changes to a Git Repository 45
Some Git Basics *45*
Committing Changes in Runbook Studio *45*
Undoing Things *46*
Working with Remotes 46
Fetching and Pulling Changes from a Remote *47*
Pushing Changes to a Remote *48*
Working with Branches 49
Creating, Renaming and Deleting Branches *49*
Checking out a Branch *50*
Merging *51*
Resolving Conflicts *52*
Working with Tags 53
Creating Tags *53*
Sharing Tags *54*
Deleting Tags *54*
Working with Stashes 54
Stashing Your Work *54*
Applying Your Stashed Changes *55*
Let us Know How We are Doing? 56

Notices

End of Support Notice for Azure Automation Agent-based Hybrid Workers

Microsoft is ending support for Azure Automation Agent-based Hybrid Workers (Windows and Linux) on August 31, 2024. You must complete migrating existing Agent-based Hybrid Workers to Extension-based Hybrid Workers before August 31, 2024. Moreover, starting October 1, 2023, you will no longer be able to create new Agent-based Hybrid Workers.

Runbook Studio 5.5 introduces support for Extension-based Hybrid Workers. You must upgrade if you are currently using Extension-based Hybrid Workers or planning to start your Agent-based Hybrid Worker migration. Runbook Studio 5.5 will continue support for Agent-based Hybrid Workers, but we cannot guarantee support will continue to August 31, 2024.

For more information see Migrate existing agent-based hybrid workers to extension-based hybrid workers.

End of Support Notice for Azure Active Directory Authentication Library (ADAL)

Microsoft is ending support for Azure Active Directory Authentication Library (ADAL) in June 2023. Runbook Studio 5.4 and all previous versions are dependent on ADAL to authenticate with Azure Automation. Kelverion has added support for Microsoft Authentication Library (MSAL) in Runbook Studio 5.5. We strongly recommend upgrading to Runbook Studio 5.5+ to ensure that Runbook Studio uses security fixes beyond June 2023.

Starting with Runbook Studio 5.5 you must register Runbook Studio in the Azure portal (in previous versions, this was optional). If you have already registered Runbook Studio in the Azure portal, you must update the redirect URI and add permissions to access the **Azure Service Management**, **Azure Automation** and **Microsoft Graph** web APIs.

For more information on creating and App Registration for Runbook Studio or upgrading your existing Connecting to Microsoft Azure Automation.

Federated Subscriptions Update

We resolved an issue where subscriptions that are managed by multiple tenants were not being displayed in the resource panel. ***Important:*** If you are updating from a previous version or Runbook Studio, the first time you start Runbook Studio 5.6 after upgrading you will not see any resources in the left resource panel. To resolve this issue, just Sign-In to Azure normally from Runbook Studio.

Installation and Setup

System Requirements

The Kelverion Runbook Studio has the following system requirements:

- Windows 11, Windows 10, Windows Server 2022, or Windows Server 2019
- Microsoft .NET Framework 4.8
- Azure Subscription containing one or more Automation accounts.
    - Supported Access Control Roles: Owner and Contributor
- 2 GB of disk space
- 4 GB RAM

Important: These system requirements are based on a stand-alone deployment of Runbook Studio for a single user. If you intend to deploy Runbook Studio to a server, so that multiple users can access it concurrently, you should scale your system accordingly.

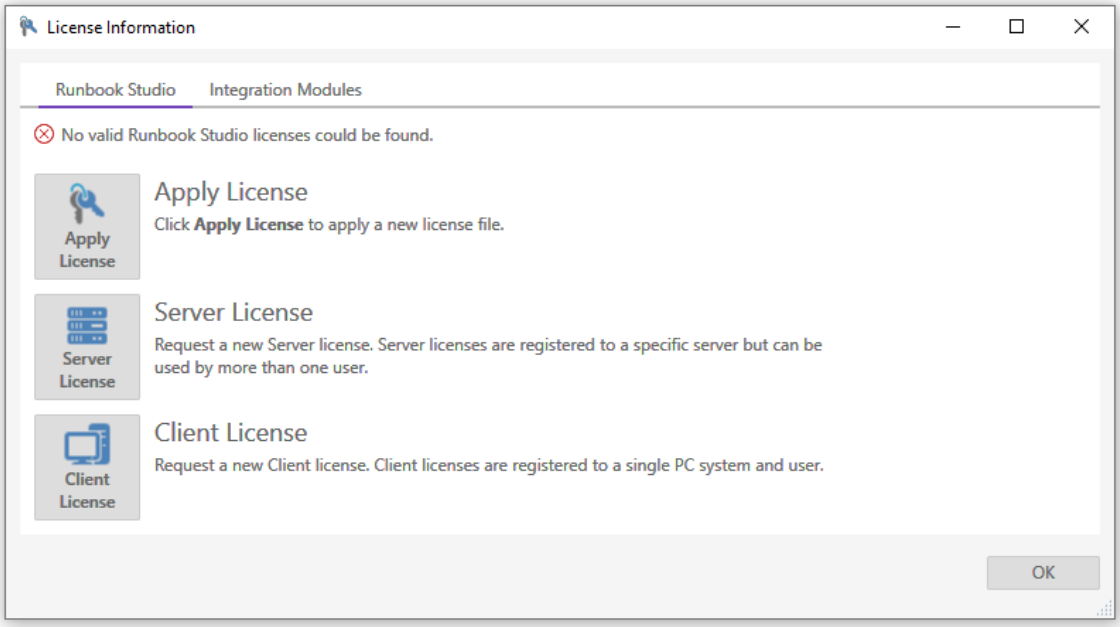
Installing Runbook Studio

Once you have determined that your computer satisfies the installation requirements, you can follow these steps to install the Kelverion Runbook Studio on your computer.

1. Launch the Kelverion Runbook Studio installer (Kelverion.RunbookStudio.msi). The **Kelverion Runbook Studio Setup Wizard** appears.
2. Click **Next**.
3. On the **End-User License Agreement** page, read the Kelverion License Terms, select **I accept the terms in the License Agreement** and then click **Next**.
4. Click **Next** to install Runbook Studio to the default folder.
5. Click **Install**.
6. Click **Yes** to allow the files to be installed.
7. Click **Finish** to exit the installer.

Runbook Studio License

When you launch Runbook Studio, you will be asked to provide a valid license file if one has not been provided already.



To add a new license click **Apply License** and select the license file (*.kasl) that you want to use, click **Open** and then click **OK**. If the license that you provide is valid, Runbook Studio will continue; otherwise, it will close.

If you need to request a new Runbook Studio license, click the **Server License** or **Client License** button, and then complete the License Configuration Form that opens in your web browser.

If you need to manage your license file(s) later, Runbook Studio stores license files at the following location: *C:\Users\<User>\AppData\Local\Kelverion\Runbook Studio\Licenses.*

Starter License

Runbook Studio Starter licenses let you build runbooks for Azure while utilizing advanced features such as smart activities, favorites, drag and drop, color coding and more. There are however some limitations:

- Version control and collaboration features are disabled.
- You can only upload a maximum of twenty-five runbooks to an automation account.

For mor information on our **Professional** and **Client** licenses, please contact sales@kelverion.com for more information.

Connecting to Microsoft Azure Automation

Before you can connect Runbook Studio to Azure Automation, you need to create one or more app registrations in Azure Active Directory App registrations.

Prerequisites:

- An Azure account that has an active subscription. Create an account for free.
- The Azure account must have permission to manage applications in Azure Active Directory (Azure AD). Any of the following Azure AD roles include the required permissions:
    - Application administrator
    - Application developer
    - Cloud application administrator
- Completion of the Set up a tenant QuickStart.

Register Runbook Studio with Azure Active Directory:

1. Sign in to the Azure portal.
2. If you have access to multiple tenants, use the **Directories + subscriptions** filter in the top menu to switch to the tenant in which you want to register the application.
3. Search for and select **Azure Active Directory**.
4. Copy the **Directory (Tenant ID)** to Notepad. You will use the value when setting up your tenant in Runbook Studio.
5. Under **Manage**, select **App registrations > New registration**.
6. Enter a display **Name**, such as Kelverion Runbook Studio.
7. Specify who can use the application, sometimes called its sign-in audience. You will typically select **Accounts in this organizational directory only**.
8. Do not enter anything for Redirect URI (optional). You will configure a redirect URI in the next section.
9. Select **Register** to complete the initial app registration.
10. Select **Overview.**
11. Copy the **Application (client) ID** to Notepad. You will use the value when setting up your tenant in Runbook Studio.

Configure Redirect URIs:

12. Under **Manage**, select **Authentication > Add a platform**.
13. Select **Mobile and desktop applications**.

14. Select the appropriate redirect URIs. This will typically be https://login.microsoftonline.com/common/oauth2/nativeclient.
15. Copy the redirect URL to Notepad. You will use the value when setting up your tenant in Runbook Studio.
16. Click **Configure**.

Configure Application Permissions:

17. Under **Manage**, select **API permissions**.
18. Select **Add a permission > Microsoft Graph**
    a. Select **Delegated permissions**.
    b. Under **Permissions**, check **User > User.Read**.
    c. Click **Add Permissions**.
19. Select **Add a permission > Azure Service Management**
    a. Select **Delegated permissions**.
    b. Under **Permission** select **user_impersonation**.
    c. Click **Add Permissions**.

Create a new tenant registration in Runbook Studio:

1. Start Runbook Studio.
2. Click the **File** tab.
3. Click **Azure AD**.
4. Click **Manage Tenants**.
5. Click **Add a Tenant**.
6. In the **Name** field, enter a name to describe the tenant.
7. In the **Directory (tenant) ID** field, paste the directory ID that you copied to Notepad.
8. In the **Application (client) ID** field, paste the application ID that you copied to Notepad.
9. In the **Redirect URI** field, paste the URL that you copied to Notepad.
10. Click **OK**.
11. Click **OK**.
12. In the **Active Tenant** field, select the tenant that you just created.

Note: Smart Connections, which are used to facilitate resource discovery when building runbooks, are associated with a specific Runbook Studio tenant. For multi-tenant deployments, when you switch the active tenant, you will only have access to the Smart Connections in that tenant.

Introduction to Runbook Studio

Runbook Studio is an on-premises Windows application that provides users with an interactive environment for authoring automation runbooks and for managing content in Azure Automation.

The Runbook Studio is organized into four principal areas, the Ribbon Bar (top), the Resources Panel (left), Runbook Canvas (center) and Properties Panel (right).

Runbook Canvas

The **Runbook Canvas** is where you design and build your runbooks. You can open as many runbooks as you need and quickly switch between them by clicking the appropriate tab at the top of the canvas. File operations, such as **Save,** and **Upload** are applied to the runbook that is currently being displayed on the canvas.

You add activities from the nodes in the Resources panel and then connect them with links to define the logic of a runbook. When you click on an activity or link on the canvas the properties used to configure the selected runbook element are displayed in the **Properties** panel. If you click on an empty area of the canvas, the properties used to configure the runbook are displayed in the **Properties** panel.

When you click on an activity node in the canvas and hold the mouse button down, you can drag the activity, or group of activities, to a new position on the canvas. The position of an activity on the canvas does not have any effect on its behavior, although it can help to understand its function within the runbook.

When you update a runbook, the name displayed in the tab at the top of the canvas will be appended with an asterisk (*) to indicate that you need to save your work. To help ensure that you do not lose any unsaved work, Runbook Studio will prompt you to save your work when you try to close a runbook that has been changed.

Activity Status

To assist with runbook authoring, Runbook Studio provides visual feedback on the status and configuration of the activities in the runbook.

An icon is displayed in the top left corner of every activity that indicates whether the activity has been correctly configured. A green arrow indicates that the activity is valid, and a red X indicates that it is invalid. You can hover the mouse over the red X icon to view the problems that need to be corrected.

Invalid Activity

Valid Activity

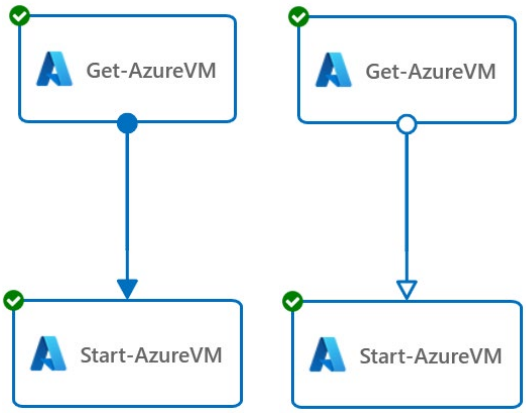
Additional status icons may be displayed in the top left corner of an activity, depending on how it has been configured. The following table describes what the status icons mean.

▲	Indicates that the activity converts exceptions to errors.
⚑	Indicates that the checkpoint has been set up after the activity runs. Checkpoints are only available in graphical PowerShell Workflow runbooks.
↻	Indicates that the activity will repeat until a specified exit condition has been met
💡	Indicates that the activity is a Smart Activity, with support for dynamic discovery

Link Status

The runbook canvas also provides visual feedback on the types of links that connect the activities in your runbook. Valid links are blue and invalid links are red. When you hover the cursor over an invalid link, Runbook Studio will display a tool tip to help you identify and resolve the problem.

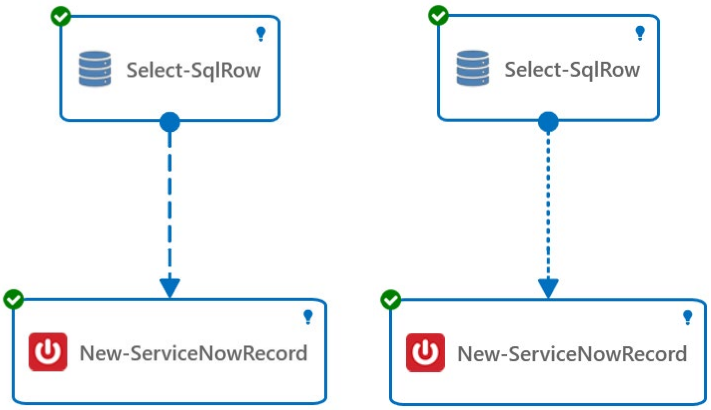
Runbook Studio supports pipeline and sequence links, which are discussed in detail in the Activity Links section of this guide, and they are displayed differently on the runbook canvas. **Sequence** links start with an *Open Circle* end with an *Open Arrow*, whereas **Pipeline** links start with a *Filled Circle* and end with a *Filled Arrow.*



Pipeline Link Sequence Link

Links can also have conditions applied to them and these are displayed differently as well. Links without a condition are drawn with a *solid line* and links that have a condition are drawn with a *dashed line*.

You can also create special links for handling errors. Error links are only followed if the source activity emits an error and will not be followed if the activity emits only normal output. Error links are drawn with a dotted line.



Conditional Link Error Link

Zoom Control

When working with the runbook canvas, you can zoom in to get a close-up view of the activities in your runbook or zoom out to see more of the runbook.

Zoom into a runbook.
In the **Home** tab, click **Zoom In**.

Zoom out of a runbook.
In the **Home** tab, click **Zoom Out**.

Alternatively, in the Runbook status bar use the **Zoom** slider.

You can also click **Scale to Fit** in the **Home** tab or status bar to automatically set the zoom level so that the entire runbook is visible.

Drag and Drop Interaction Modes

The Runbook Canvas has two interaction modes, called **Pan,** and **Select** and you can switch between them by clicking either the *Hand* or *Pointer* button on the toolbar.

When **Pan** is enabled, the cursor changes to a hand . When you click and hold the mouse button in pan mode you can reposition (or pan) the entire runbook canvas. This is helpful when working on large runbooks and you need to reposition the canvas to view a specific area of the runbook.

When **Select** mode is enabled, the cursor changes to a pointer . When you click and hold the mouse button in select mode you can drag a selection box around a group of runbook elements to select them all. You can then move all the selected elements together as a distinct group.

When Select mode is enabled, you can select one or more runbook canvas elements and click the **Cut** or **Copy** button in the main toolbar to copy the elements to the clipboard. You can then click the **Paste** button to paste the elements in any runbook that is currently open. However, when cutting or

copying activities and links from one runbook to another you may have to resolve any runbook resources, such as runbook input parameters, which are not available in the destination runbook.

It is important to note that when you paste runbook activities into a new runbook, some parameters values, such as those that use Activity Output or Runbook Input data sources, may no longer be appropriate and as a result will be reset. For example, if one of the activities that is being copied has a parameter that is assigned and Activity Output data source that references an activity that is not being copied then the value assigned to that parameter will be reset.

Explorer Panel

The **Explorer** panel provides access to resources and tools for building and managing your runbooks and for managing assets in Azure Automation.

- The **Toolbox** group provides cmdlets and runbook control activities that you can use in your runbooks.
- The **Azure** group provides automation assets that you can use in your runbooks as well tools for managing the assets in your Azure Automation accounts.
- The **Repository** group provides tools for working with the current Git repository.
- The **Changes** group provides tools for managing the changes to your current Git repository.

The Toolbox Group

When Runbook Studio launches, it automatically identifies the PowerShell modules that are available on your computer and lists them and the cmdlets they contain in the Toolbox tab of the Resources panel.

If the Toolbox tab does not display a cmdlet that you would like to add to your runbook, you should ensure that the location of the module that contains the cmdlet is specified in the **PSModulePath** environment variable.

To quickly find a cmdlet, or set of similarly named cmdlets, type a keyword in the search box at the top of the On-Premises tab. For example, to find all cmdlets in the Azure module type the keyword *Azure* in the search box. To clear the search and display all available cmdlets, click the **X** button to the right of the search box.

To add a command activity to your runbook, you can **drag and drop** the cmdlet from the Resources panel onto the runbook canvas or right click the activity and click **Add to Runbook**.

The Azure Group

When you connect to Microsoft Azure, Runbook Studio automatically identifies the available Automation Accounts and organizes them by Subscription in the **Azure** tab of the Resources panel.

Runbook Studio lets you work with the following types of global assets.

- Certificates
- Credentials
- Connections
- Runbooks
- Schedules
- Variables
- Webhooks

Tip: To use an asset in your runbook, you can simply drag and drop it onto the runbook canvas.

You can also use the Azure tab to manage the global assets and runbooks in your Automation Accounts. This is particularly helpful when you need to add or edit an asset that is required by the runbook that you are currently building.

Right click on an asset to view the editing options that are available.

You can also open your Azure runbooks in Runbook Studio or download them to your computer. When you open a runbook from Azure, Runbook Studio opens an on-premises copy of the runbook. If you update an Azure runbook that you opened in Runbook Studio you need to explicitly publish the runbook to make the changes available in Azure.

Online and Offline Mode

Runbook Studio supports working with Azure in both Online and Offline modes. When you connect Runbook Studio to Azure (online mode), a secure snapshot of the assets in your automation accounts is created and periodically updated. This means that the changes that you, or someone else, make in the Azure web portal will automatically become available as you work in Runbook Studio.

When Runbook Studio is disconnected from Azure (offline mode), you can still access the resources in your automation accounts using the snapshot that Runbook Studio generated the last time that you were in online mode. Any changes that you make, including adding, editing, or removing automation assets, will be stored on-premises, and then uploaded to Azure the next time you are in online mode.

The following table outlines the icons that Runbook Status uses to notify users as to the synchronization status of an automation asset.

	Indicates that the on-premises and cloud versions of the automation asset are the same based on the last synchronization event.
	Indicates that the on-premises version of the automation asset as new or has changed and that it will be uploaded to the cloud during the next synchronization event.
	Indicates that the automation asset was changed both in the cloud and on-premises. It is up to you to resolve the conflict manually by downloading the cloud version or uploading the on-premises version by editing the asset.
	Indicates that that there was an error synchronizing the automation asset. Right click on the asset to get more information.

Disabling Azure Subscriptions and Automation Accounts

When you connect to Azure, Runbook Studio will automatically download information about the Azure Subscriptions and Automation Accounts that are associated with your credentials. However, you may find that some of these subscriptions and automation accounts are not applicable to the work that you want to do in Runbook Studio, in which case you may find it helpful to disable them. Disabled subscriptions and automation accounts are not kept synchronized and their assets are not available when building runbooks.

To disable an Azure Subscription or Automation Account, **right click** on the application item in the **Azure Resources Panel** and select **disable**. To enable a disabled subscription or automation account, **right click** on the item and select **enable**. When you re-enable a subscription or automation account, all the changes that have occurred while you were disconnected will be automatically synchronized.

Update/Delete Conflicts

Most of the time Runbook Studio will be able to synchronize your on-premises changes to automation assets with changes made by yourself or other users in Azure. However, there will be times, especially when working in offline-mode for an extended time, that Runbook Studio will not be able to automatically synchronize your on-premises changes because of changes made in Azure.

The most common synchronization conflict occurs when changes are made to an automation asset in both Runbook Studio and in Azure within the time that the last synchronization event occurred. Other sources of conflict occur when an asset is deleted in Runbook Studio and updated in Azure or vice-versa.

When Runbook Studio detects a synchronization conflict it will flag the asset with a red exclamation icon and provide a tooltip with details of the conflict. To resolve the conflict simply right click on the asset and select one of the available options. For example, to resolve conflicting changes to an automation asset you may decide to **edit** the changes made in Runbook Studio and override the changes that were made in the cloud. Alternatively, you decided to **download** the asset from the cloud and override the changes that were made in Runbook Studio.

Runbook Assets

Runbook assets are managed a little differently in Runbook Studio than other automation assets. When you select a runbook asset in the Azure Resources panel, the properties of the runbook are displayed in the Properties panel along with options for managing the automation assets that are associated with the runbook, such as job schedules and webhooks.

To schedule a runbook job, simply select the **Job Schedules** group, click **Add Schedule**, and enter the appropriate details in the dialog that is provided. Similarly, to create a webhook, simply select the **Webhooks** group, click **Add Webhook** and enter the appropriate details in the dialog that is provided.

You do not need to be connected to Azure to view the published or draft version of a runbook asset. Runbook Studio stores a snapshot of the versions that were available the last time you were signed into Azure.

View the published or draft version of a runbook:

1. In the **Resource** panel, click the **Azure** tab.
2. Find the desired runbook in the resources tree either by expanding the tree or using the search field.
3. Right click the runbook.
4. Click **Open** and then click **Published** or **Draft** to view the desired version.

5. An on-**premises copy** of the runbook will be displayed on the runbook canvas.

Note: A draft version of a runbook is only available if the runbook state is New or In Edit. Similarly, a published version of a runbook is only available if the runbook state is In Edit or Published.

Webhooks

A *webhook* allows you to start a particular runbook in Azure Automation using a single HTTP request. This allows external services, such as GitHub, Azure Log Analytics, or custom applications to start runbooks without implementing a full solution using the Azure Automation API.

Add a Webhook.

1. In the **Home** tab, click **Sign In**. Sign into Azure.
2. In the **Resources** panel, select the **Azure** tab.
3. In the resources tree, select the desired runbook.
4. In the **Runbook** panel, select the **Webhooks** tab.
5. Click **Add Webhook**.
6. In the **Name** field, enter a unique name for the webhook.
7. In the **Expiration** field, select an expiration date and time.
8. In the **Run on** field, select where the to run the runbook (could be Azure or hybrid worker)
9. Click the **Copy** symbol and paste the webhook URL somewhere safe.
10. If the runbook has one or more input parameters, select the **Parameters** tab, and enter the desired inputs.
11. Click **OK**.

Remove a Webhook.

1. In the **Resources** panel, select the **Azure** tab.
2. In the resources tree, select the desired runbook.
3. In the **Runbook** panel, select the **Webhooks** tab.
4. Find the webhook you want to delete and click the ✕button.
5. Click **Yes**.

Schedules

To schedule a runbook in Azure Automation to start a specific time, you link it to one or more schedules. A schedule can be configured to either run once or on a reoccurring hourly or daily schedule. You can also schedule runbooks to run weekly, monthly, on specific days of the week or days of the month, or a particular day of the month. A runbook can be linked to multiple schedules, and a schedule can have multiple runbooks to it.

Create a new schedule.

1. In the **Home** tab, click **New Asset** and click **Schedule**.

2. Type a **Name** and optionally a **Description** for the new schedule.
3. Specify the time that the schedule starts in the **Starts** field. Note that the start time is in UTC.
4. Check **Recurs every** to create a reoccurring schedule. Leave **Recurs every** unchecked to create a schedule that runs once.
5. For reoccurring schedules, enter an interval and select hour, day, week, or month.
6. For schedules the reoccur weekly, check the days on which the schedule will occur.
7. For schedules the reoccur monthly, select whether the schedule occurs on specified days of the month of a specified day of the week. For the former, select the days of the month that the schedule occurs and for the latter select the relative day of the week.
8. Optionally, you can create a schedule that expires by checking **Expires** and entering the desired expiration date and time. Expiration date and time is in UTC.
9. Click **OK**.

Linking a schedule to a runbook

1. In the **Resource** panel, click the **Azure** tab.
2. In the resources tree, select the runbook that you want to schedule.
3. In the **Runbook** panel, click the **Job Schedules** tab.
4. Click **Add Schedule**.
5. Select the **Schedule** that you want the runbook to reoccur on.
6. Select whether the runbook should **Run on** Azure or a hybrid worker.
7. Enter values for any runbook input parameters. You must enter values for all required input parameters.
8. Click **OK**.

Properties Panel

The **Properties** panel is where you configure your runbook, the activities in your runbook and the links that connect them. The Properties panel is also where you assign a name to your runbook, and if necessary, add input parameters to control the runbook when it runs.

The options that are available on the Properties panel depend on which element, if any, is currently selected on the runbook canvas. For example, when you select a command activity on the canvas, the Properties panel will display options for selecting a parameter set and for assigning parameter values. When you select a link, the Properties panel will display options for configuring how the link will behave.

When no runbook elements are selected, the Properties panel will provide you with options that you can use to configure the runbook itself, such as its name, description, and input parameters.

The properties panel is organized into tabs, with each tab containing a set of related options. To access the options in a tab, simply click the appropriate tab at the bottom of the Properties panel.

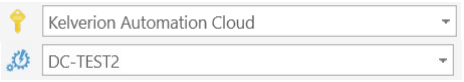
Connecting to Microsoft Azure

Runbook Studio lets you build runbooks offline using the resources in your on-premises environment, however at some point you will want to connect Runbook Studio to Azure to access and manage the assets in your automation accounts and to publish your runbooks.

Before you can sign into Azure you must register one or more tenants and then select which tenant you want to be active. Once this information is provided you can connect Runbook Studio to Azure by clicking **Sign In** on the toolbar. When you have successfully signed into Azure you can select the Azure tab on the library panel to access information about the automation accounts in your Azure subscriptions.

Since Runbook Studio provides you with access to all your automation accounts, you need to specify the *Active Automation Account* before you can publish a runbook. Also, when an active automation account is specified, Runbook Studio can assist you when configuring activities that use global certificate, credential, connection, and variable assets.

Setting the active automation account



1. In the **Home** tab, select a subscription.
2. And then select an automation account.

Note: When assigning data sources to the properties of an activity, the names of available automation assets are retrieved from the active automation account.

Working with Graphical Runbooks

The Kelverion Runbook Studio provides runbook authors with the tools to build graphical runbooks using facilities that are available in your on-premises environment and then lets you publish your runbooks to any of your Azure Automation Accounts.

All runbooks have the following properties.

Property	Description
Name	A unique name to identify the runbook. Runbook names can only contain letters, numbers, underscores, and dashes. The name must begin with a letter.
Runbook Type	The type of runbook, either **Graphical** or **Graphical PowerShell Workflow**. The former is based on Native PowerShell whereas the latter is based on PowerShell Workflow.
Description	An optional description of the runbook.
Input Parameters	An optional set of parameters that can be used to invoke the runbook.
Output Types	An optional set of output types that can be used to provide guidance when invoking the runbook from another runbook.

You can access the properties for a runbook by clicking on an empty area on the runbook canvas.

Runbook Files

You can create or open as many runbooks as you require and quicly switch between them by clicking the appropriate tab at the top of the runbook canvas. Kelverion Runbook Studio allows you to create and manage both graphical and scripted runbooks, however it is specficially designed to helps users author graphical runbooks.

Property	Description
Graphical	Based on Windows PowerShell and edited completely in Runbook Studio or the Azure portal.
Graphical PowerShell Workflow	Based on Windows PowerShell Workflow and edited completely in Runbook Studio or the Azure portal.
PowerShell	Text runbook based on Windows PowerShell script.
PowerShell Workflow	Text runbook based on Windows PowerShell script.
Python	Text runbook based on Python.

When you create a new graphical runbook, you have the option to create a **Graphical** runbook, which is based on native PowerShell, or a **Graphical PowerShell Worflow** runbook, which is based on

PowerShell Workflow. Each runbook type has advantages and disadvantages so you should carefully consider what type you require as you will not be allowed to change it afterwards.

For example, if your runbook must support checkpoints and the ability to run activities in parallel, then you should choose **Graphical PowerShell Workflow**. Alternatively, if your runbook will include Code activities that will make use of PowerShell features not supported by PoweShell Workflow, then you should choose **Graphical**.

Create a new runbook Document

1. In the **Home** tab, click **New Runbook**  .
2. Click the runbook type that you want to create.

Open an existing runbook Document

1. In the **Quick Access Toolbar**, click **Open**  , or press CTRL+O.
2. Select the file you want to open and click **Open**.

When building your runbooks, you can save your work at any time as a Kelverion Runbook File (*.Runbook). To help ensure that you do not lose any unsaved work, Runbook Studio will prompt you to save your work when you try to close a runbook that has been changed.

Save a runbook file on your device for the first time.

1. On the **Quick Access Toolbar**, click **Save,**  or press CTRL+S.
2. Type a name for the runbook document and click **Save**.

Save an existing document as a new document (Save As)

1. Open the runbook document that you want to use as the basis for a new runbook.
2. Click **File**, and then click **Save As**.
3. Type a name for the document, and then click **Save**.
4. Edit the document the way that you want.

Important: Whether it is a runbook you created from scratch in Runbook Studio or a runbook that you created online in Azure and then opened in Runbook Studio, you should be aware that you are working on an **on-premises copy of the runbook**, which is not actively connected to any of your runbooks in Azure.

For example, when you right click on a graphical runbook on the Azure library and click **Open**, Runbook Studio downloads the runbook's data and creates a new on-premises copy. This copy is not linked to the online version in any way other than that it initially shares the same name. After you have opened an Azure runbook, you can save the runbook as a file on your computer, so that you can edit it locally, or publish a draft to any of your automation accounts.

Important: As the *Name* property is used to uniquely identify a runbook in an Azure Automation Account, care should be taken when publishing a runbook to an automation account that is different from the one that you downloaded it from. If you are not careful, it is possible to overwrite an

otherwise unrelated runbook that simply shares the same name as the runbook you are trying to publish.

Runbook Studio will warn you when it detects a runbook with the same name as the runbook you are trying to publish and then ask you whether you would like to continue before publishing the runbook to Azure.

Runbook Input and Output

Your runbooks may require input to control their behavior. For example, if you are designing a runbook that needs to start a virtual machine you may need to provide information, such as the name of the virtual machine.

Add an input parameter to a runbook:

1. In the **Runbook Properties** panel, click **Input and Output**.
2. Click **Add Input**.
3. Type a **Name** and an optional **Description**.
4. Specify a **Type** and if the input is **Mandatory**.
5. Optionally, specify the input **Has a default value** and type an appropriate value.
6. Click **OK**.

To edit an input parameter, click the **Edit** button . To delete an input parameter, click the **Delete** button .

As a runbook author, you can use *Output Types* to provide consumers of your runbooks with aid in using them their own runbooks. When you add a child runbook and it has one or more output types defined, the runbook asset data source parameter in any child activities will provide the ability to browse the properties of the output types(s).

Add an output type to a runbook.

1. In the **Runbook Properties** panel, click **Input and Output**.
2. Click **Add Output**.
3. Specify the *fully qualified name* of the output type (ex. System.String, System.DateTime, Microsoft.Azure.Commands.Automation.Model.AutomationAccount, etc.).
4. Click **OK**.

To edit anoutput type, click the **Edit** button . To delete an output type, click the **Delete** button .

Error Handling

Error handling is an important consideration when building runbooks. Enabling error handling involves configuring your activities to convert terminating exception to errors and using error links to connect your activities to error handling logic. **Important:** error handling is only supported by Graphical Runbooks.

To build error handling in your runbook, you must identify the activities in your runbook that can generate exceptions. These could be terminating exceptions, such as a non-existent cmdlet, or non-terminating exceptions, such as permission issues. Select each activity, and in the **Properties** pane, enable the **Convert exceptions to errors** option. This option configures the activity to convert terminating exceptions into non-terminating errors that you can manage.

Next, you need to connect the error generating activities to the error handling logic in your runbook and you do this with error handling links. To create an error link , add a link from one of your activities that converts exceptions to errors to your error handling activities in your runbook and in the **Properties,** pane enable to **Error link** option.

Tip: Activities that convert exceptions to errors display a small triangle indicator in their top, right corner and error links are displayed using a dotted line.

Logging and Tracing

The **Logging and Tracing** panel contains options that can be enabled to help you diagnose problems with your runbooks. By default, logging and tracing should be disabled to improve performance.

Verbose Records

The Verbose message stream contains general information about the operation of a runbook. Since the Debug stream is for interactive sessions and therefore not available to runbooks, you should use the Verbose Stream to report diagnostic information. You can write verbose messages using the *Write-Verbose* cmdlet.

To include verbose messages in the job history you must enable the **Log verbose records** option. When this option is enabled, all verbose messages will be displayed when you test your runbook in either Runbook Studio or the Azure portal.

Progress Records

When enabled, progress records are included in the job history before and after each activity in the runbook runs. To include progress records in the job history, you must enable the **Log progress records** option. Progress records are not stored when you run a published version of your runbook in Azure and not when you test a draft runbook in Runbook Studio or the Azure portal.

Activity Level Tracing

For graphical runbooks, additional logging is available in the form of activity-level tracing. There are two levels of tracing: **Basic** and **Detailed**. Tracing information is written to the **Verbose** stream, so you must enable Verbose logging when you enable tracing.

When **Basic Tracing** is enabled, you can see the start and end times of each activity in the runbook as well as information about retries, such as the number of attempts and the start time of each retry

attempt. When **Detailed Tracing** is enabled, you get Basic tracing plus input and output data for each activity.

Testing Runbooks

Runbook Studio lets you test the runbooks in your Azure Automation Accounts and view the output that is being generated in real time.

Testing a runbook that is already uploaded to Azure:

1. In the **Resources** panel, click the **Azure** tab.
2. In the resources tree, find the runbook that you want to test.
3. Right click the runbook and click **Test Draft**.
4. Specify whether to **Run on** Azure or a hybrid worker.
5. Click **Start**.
6. If the runbook has one or more input parameters, type the desired inputs, and click **OK**. You must provide values for all mandatory runbook inputs.

*Tip: You can view the output from the most recent test job by clicking **View last test**.*

You can suspend, resume, or stop a runbook test job by clicking the **Suspend**, **Resume** or **Stop** button, respectively. Only Graphical and PowerShell runbooks can be paused.

You can also test a copy of the runbook that you are currently working on in the Runbook Studio runbook canvas. This will upload a draft version of the runbook to the active automation account and then start a new test job. To upload and test a runbook from the runbook canvas, do the following:

1. In the **Home** tab, click **Sign In** and follow the instructions to sign into Azure.
2. Set the active automation account to the account that you want to upload the runbook to.
3. Click **Test**  .

Working with Activities

Activities are the building blocks that you will use to construct your runbooks. An activity can be to call a PowerShell cmdlet, run a child runbook or execute PowerShell code.

All activities have the following properties:

Property	Description
Label	A unique label that identifies the activity in the runbook. Runbook Studio will provide a default name for each activity, but you can provide your own labels to make their role in the runbook more obvious.
Description	An optional description of the activity. Providing a description is a fantastic way to let everyone understand the function of the activity in the runbook.

Checkpoint	Indicates whether a checkpoint is set in the runbook workflow after the activity runs. Checkpoints are only available for Graphical PowerShell Workflow runbooks. If the runbook uses Azure cmdlets, you should follow best practices and follow a check-pointed activity with an *Add-AzureRMAccount* in case the runbook is suspended and restarts from this checkpoint on a different worker.
Color	Optionally, you can assign a color to the activity to signify context and meaning.

In addition, some activities will also have additional properties, such as:

- Discovery options (unique to Kelverion Smart Integration Modules)
- Parameter sets.
- Mandatory and/or optional parameters
- Retry options.

Once you have added an activity to your runbook you can drag the activity and place it anywhere on the runbook canvas that makes sense to your runbook. Although the location of the activity on the canvas does not affect its behavior, choosing a convenient location can help express its role and how it is associated with other activities in the runbook.

When you select an activity in the runbook canvas, its properties will be displayed in the **Properties** panel.

Disabling Runbook Activities

Runbook Studio provides runbook authors with the ability to selectively disable activities and their incoming and outgoing links. Temporarily disabling activities in a runbook is something that runbook authors may find useful when designing and testing their runbooks as it lets you investigate specific paths through your runbooks while ignoring others.

When the graphical runbook is uploaded to Azure, all disabled activities and links will be excluded. Only the enabled activities and links will be visible in the Azure version of the graphical runbook.

To disable an activity in a graphical runbook, **right click** on the activity and select **disable**. The activity and all incoming and outgoing links will become greyed out to indicate that they are disabled. To enable a disabled activity, **right click** on the activity and select **enable**.

Retry Behavior

Some activity types can be configured to run multiple times until a particular condition, which you specify, is satisfied. You can use the retry behavior options to configure activities that should run multiple times, which are error prone or may need more than one attempt for success.

When you enable retry for an activity, you can configure the runbook to wait a specified number of minutes or seconds before running the activity again. If no delay is specified the runbook will run the activity again, immediately after it is completed.

The retry condition lets you specify a PowerShell expression that the runbook will evaluate after each time the activity runs. If the result of the expression is true the activity does not run again, and the runbook moves on to the next child activity in the runbook.

When defining the retry conditions for your activity, you can take advantage of a global variable called **$RetryData**. Specific information about the last time the activity ran can be accessed using the following properties.

Property	Description
NumberOfAttempts	Number of times that the activity has ran.
Output	Output that was generated by the activity the last time that it ran.
TotalDuration	Time elapsed since the activity was started.
StartedAt	Time in UTC when the activity was first started.

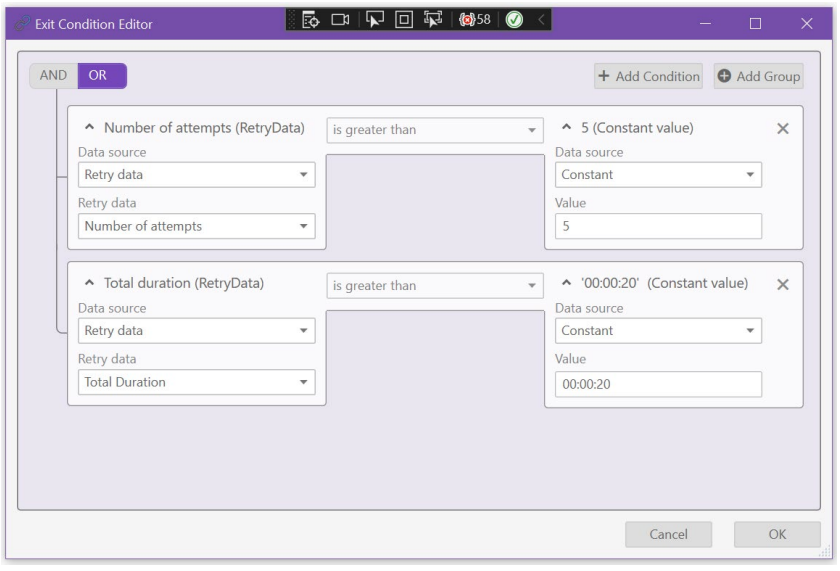
The following are some examples of activity retry conditions.

```
# Run the activity exactly five times
$RetryData.NumberOfAttempts -eq 5

# Run the activity until it produces some output
$RetryData.Output.Count -ge 1

# Run the activity until at least 2 minutes has elapsed
$RetryData.TotalDuration.TotalMinutes -ge 2
```

Runbook Studio also provides a **Condition Editor** that you can use to build retry exit conditions graphically, without having to manually write PowerShell. Also, graphical exit conditions are automatically updated whenever you change an activity label or runbook input name.



Click **Add Condition** to add a new exit condition and click **AND** or **OR** to specify how a group of conditions are logically connected. When **AND** is selected, all the connected conditions must evaluate to **true** to exit the retry loop. When **OR** is selected, only one of the connected conditions must evaluate to **true** to exit the retry loop. For complex exit conditions, click **Add Group** to add a nested condition group. The graphical condition will be automatically concerted to PowerShell code when the runbook is uploaded to Azure Automation.

Additional Parameters

Some activities also provide you with the ability to specify additional PowerShell parameters that you can use to control the behavior of the activity.

For example, to output detailed information about the operation performed by an activity, you would specify **-Verbose:$True.**

Working with Activity Links

Links are used in a runbook to determine the sequence in which the activities in your runbook are executed, and to facilitate the flow of data from one activity to the next.

You can create a link between two activities by hovering the cursor on the bottom of the source activity until it changes to the **link** cursor and then dragging and dropping the new link to a destination activity.

When adding a link from a command activity, Runbook Studio will check to see if the activity declares an output type. If an output type is declared, Runbook Studio will add a Pipeline link. Otherwise, it will add a Sequence link.

When you select a link on the runbook, **Link Properties** will be displayed on the Properties panel. All links have the following properties.

Property	Description
Type	When **Pipeline** is selected the destination-activity is run once for each output object that is generated by the source object. When **Sequence** is selected the destination-activity runs only once and it receives an array of objects containing all the output objects that were generated by the source activity.
Label	An optional label that will be displayed next to the link on the runbook canvas.
Description	An optional description to help describe the link. Providing a link with a description is especially helpful when the link is conditional.
Color	An optional color that can be used to imply meaning to the link.
Style	An optional style that controls how the link appears.

A graphical runbook will start all activities that do not have an incoming link and run them in parallel. An activity can be the source activity for more than one destination activity, in which case each outgoing link from the source activity will be processed in parallel.

The Databus

Any data that is output by an activity with an outgoing link is written to the runbook's **databus**. A destination activity can use the databus to access the output from any previous activity in the runbook.

How the data is written to the databus depends on the type of the outgoing link. If the link is a **pipeline**, the output data is added to the data bus as multiple, individual objects. If the link is a **sequence**, the output data is added to the databus as a single array of objects.

When a destination activity is connected to a source activity through a pipeline link, you can use the **Activity output** data source to assign output from the activity to a specific parameter. When you select the Activity output data source, Runbook Studio will provide you with a list of source activities whose data can be accessed. You can also specify an optional **Path** to access a specified property of the output object.

Data from the databus can also be used in PowerShell script code, such as the code used to define a link condition, the code used in a PowerShell expression data source, or the code used to configure a Code activity. When writing script code, the **$ActivityOutput** global variable can be used to access the output from a previous activity.

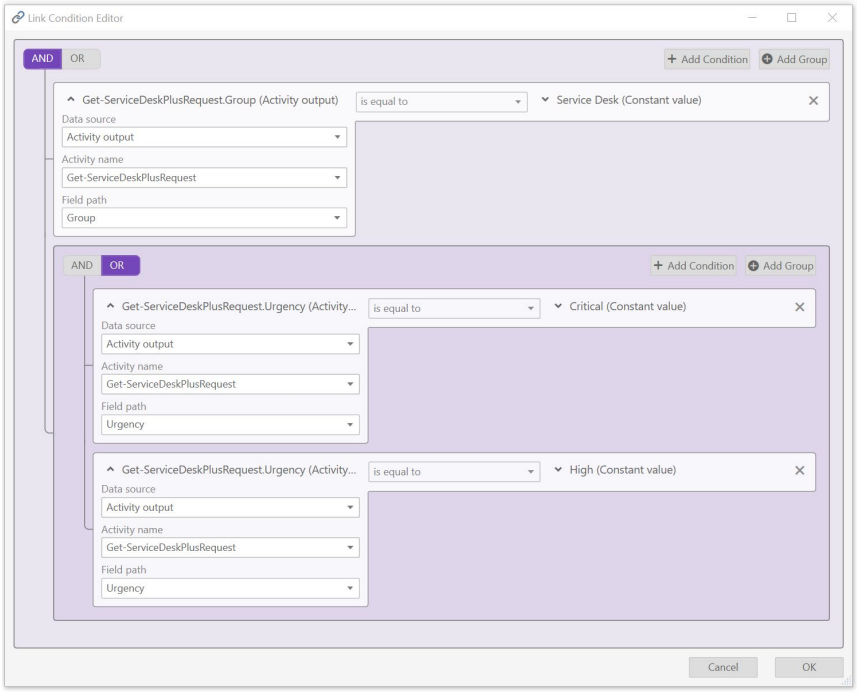
The **$ActivityOutput** variable is a *hashtable,* and you use the **label** assigned to an activity to access its data. If the output from the activity is an object, you can also access a specific property. For example:

```
$ActivityOutput['Activity Label']
$ActivityOutput['Activity Label'].PropertyName
```

Link Conditions

When you specify a condition on a link, the destination activity runs only if the condition resolves to True. For a **pipeline link**, you must specify a condition for a single object. The runbook evaluates the condition for each object output by the source activity. It then runs the destination activity for each object that satisfies the condition. For a **sequence link**, the runbook only evaluates the condition once, since a single array containing all objects from the source activity is returned. Because of this, the runbook cannot use a sequence link for filtering, like it can with a pipeline link. The sequence link can simply determine whether the next activity is run.

To add condition to a link, select the **Condition** tab, enable the **Apply condition** option, and then click **Build in Editor** to open the condition editor. In the editor, click **Add Condition** to add a new link condition and click **AND** or **OR** to specify how a group of conditions will be logically connected. When **AND** is selected all the connected conditions must evaluate to **true** for the condition to pass. When **OR** is selected, only one of the connected conditions for the condition to pass. For complicated conditions, clock **Add Group** to add a nested condition group.



Alternatively, you can use the text editor to manually write the link condition using PowerShell code. Just make sure that your expression evaluates to **$true** or **$false**.

Error Links

When an activity is configured to convert exceptions to errors, you can add outgoing error links to support custom error handling in your runbook.

To make an error link, select the link and in the Link Properties panel set the **Error Link** option to **True**. Error links will only be followed when the source activity generates an exception, and the activity has been configured to **convert exceptions to errors**. Error links will not be followed if the source activity emits standard output (i.e., no error records).

You can access details of the errors emitted from an activity using **$ActivityError ['<source activity>'].ExceptionMessage**.

Cycles

A cycle is created when a destination activity links back to its source activity or to another activity that eventually links back to its source. Cycles are not supported in Azure Automation and Runbook Studio will warn you when you attempt to publish a runbook in which cycles are detected.

Working with Global Assets

When authoring runbooks, you will often find yourself using the global assets that you have defined in Azure Automation. Runbook Studio makes it easy to use the assets from any of your Azure automation accounts in your runbooks.

The simplest way to consume global assets in your runbooks is to use the appropriate data sources when you configure the parameters of your activities. The following steps outline how to assign a global connection asset to a connection parameter of a command activity.

1. In the **Home** tab, specify the active automation account.
2. Add the desired activity to the runbook canvas.
3. Select the activity. Click **Parameter Sets** and select the desired parameter set.
4. Click **Parameters** select the desired connection parameters.
5. Select the **Connection asset** data source and select the desired connection asset from the collection of connection assets that are associated with the active automation account.

Alternatively, you can use the *Get-AutomationConnection* activity to retrieve a connection asset for use in your runbooks.

1. Add a *Get-AutomationConnection* activity to your runbook.
2. Select the *Get-AutomationConnection* activity and then click **Parameters**.
3. Click the **Name** parameter and select the **Constant** data source.
4. Type the name of the connection asset you want to retrieve.
5. Add an activity that requires a connection asset as a parameter.
6. Connect the *Get-AutomationConnection* activity to a child activity.
7. Select the child activity.
8. Click **Parameter Sets** and then click the desired parameter set.

If you have connected Runbook Studio to Azure, you can also just drag and drop the desired resource from the Azure Library onto your runbook. If you want to set the value of a global variable, right click the variable on the Azure Library and click **Add set variable to canvas**.

Publishing Runbooks to Azure

The runbooks you build in Runbook Studio are stored as files on your computer and are not connected to the runbooks in your Azure Automation Accounts. To make your runbooks available to Azure, you must **publish a draft copy** to an Azure Automation Account.

You can publish drafts of your runbooks to as many automation accounts as you like, but care should be taken to ensure that the cmdlets and assets used by the runbook are available in the automation account that you are publishing too.

The first step in publishing your runbooks is to set up the *Active Automation Account*, and there are a couple of ways that you can do this. The first is to select the desired **Subscription** and **Account** in the main toolbar and the second is to right click on an automation account in the Azure Library and click **Set as active account**.

Once you have set an active automation account, click the **Upload** button  in the toolbar. If Runbook Studio detects any problems that may prevent the runbook from running in Azure, it will display an error report. You must fix all errors before the runbook can be published.

If the Azure Automation Account that you are publishing already contains a runbook with the same name as the on-premises runbook that you are publishing, Runbook Studio will ask you to confirm that you want to overwrite the runbook in Azure. **Care should be taken to ensure that you are publishing your runbooks to the correct automation account and that you are not accidentally overwriting a runbook with the same name in another account**.

When a graphical runbook is uploaded from Runbook Studio to Azure Automation, the runbook is converted to a format that is compatible with Azure. This conversion means that the version of the graphical runbook that you view in Runbook Studio can be quite different than the version that you view in Azure Automation. For example, discoverable smart activities are translated to command activities and color formatting is lost altogether.

Starting with Runbook Studio 3.1, runbook features that are unique to Runbook Studio will be preserved when you upload and download graphical runbooks to and from Azure Automation. The Runbook Studio runbook is still converted to a format that is compatible with Azure Automation; however, the unique Runbook Studio features are persisted and restored when the runbook is downloaded and opened in Runbook Studio.

Graphical runbooks that are created in Runbook Studio and uploaded to Azure Automation should not be edited using the Azure Automation Portal. Any changes that you make to a graphical runbook in the Azure Automation Portal will be lost when the graphical runbook is downloaded and opened in Runbook Studio.

Activity Types

Runbook Studio supports several diverse types of activities, and each one has a specific role in the runbooks that you will build.

The activity types supported by Runbook Studio include:

Activity Type	Description
Command Activity	Invoke a specific PowerShell cmdlet using parameters that you specify.
Invoke Runbook Activity	Invoke another runbook in your automation account, optionally using parameters that you specify.
Junction Activity	A control activity that waits until all incoming branches have completed.
Smart Activity	A special type of command activity that lets you interactively discover the resources in your enterprise environment. Smart activities are unique to Kelverion Smart Integration Modules and Runbook Studio.

Code Activity	Accepts Native PowerShell or PowerShell Workflow code to provide complex functionality that would otherwise not be available.

Command Activity

Command Activities are used when you want to invoke a PowerShell cmdlet in your runbook. Cmdlet activities appear as a rectangle on the runbook canvas.

Runbook Studio automatically provides you with access to the cmdlets that are available on **your computer**.

The cmdlets that are available in Runbook Studio will differ from the cmdlets that have been imported to a particular Azure Automation Account. Consequently, before publishing a runbook to Azure, you should verify that the automation account that you are targeting contains the required modules and cmdlets.

Parameter Sets

Windows PowerShell uses parameter sets to enable a single cmdlet to perform different actions for different scenarios. Each parameter set exposes different mandatory and optional parameters.

All cmdlets have at least one parameter set, and most cmdlets have multiple. By default, Runbook Studio will automatically select the *default parameter set,* if one is specified or *the first parameter set* if one is not.

To view the parameter sets for a cmdlet activity, select the activity on the canvas and then select the **Parameter Sets** tab on the **Properties** panel.

If the cmdlet is available on your computer, Runbook Studio will display the syntax of the parameter set to help you select which one you require.

The current parameter set determines which parameters you can access. You can change the current parameter set by selecting it in the list, however any parameters that you have already configured will be lost.

Parameters

When you specify a value for a parameter you must select a **data source**. Depending on the type of data source that is selected you may be provided with additional options. For example, if you select the Activity Output data source you must also select the activity whose output will be assigned to the parameter and optionally the path to a specific property of the output.

You must configure all mandatory parameters. To view the optional parameters that are associated with an activity, click **Optional** at the top of the Parameters tab.

Several factors determine the data sources that are available to a parameter, and these include the parameter's data type, whether it is linked to another activity and whether the runbook has any input parameters.

Runbook studio supports the following data sources.

Data Source	Description
Activity output	Specify activity whose output will be assigned to the parameter. You may also provide an optional Path to select a specific property of the output objects that are generated by the activity. Available when the activity is linked to a source activity.
Not configured	Clears any value that was previously configured. You must configure all mandatory parameters.
Certificate asset	Specify the name of the global certificate asset that will be used to provide a value for the parameter. If you have connected to Azure and selected a Subscription and Automation Account on the toolbar, the data source will provide the names of the certificates that are available.
Credential asset	Specify the name of the global credential asset that will be used to provide a value for the parameter. If you have connected to Azure and selected a Subscription and Automation Account on the toolbar, the data source will provide the names of the credentials that are available.
Constant	Specify a constant value to assign to the parameter. Available for parameters that have the following data types: • String • DateTime • Boolean • Char • Byte • SByte • Int16 • Int32 • Int64 • UInt16 • UInt32 • Uint64 • Decimal • Double • Float • SwitchParameter When assigning a constant DateTime value, Runbook Studio assumes the value is in UTC.
Connection asset	Specify the name of the global connection asset that will be used to provide a value for the parameter. If you have connected to Azure and selected a Subscription and Automation Account on the toolbar, the data source will provide the names of the connections that are available.
Empty string	An empty string will be assigned to the parameter. Available when the parameter is type *System.String*
Null	A null ($null) value will be assigned to the parameter. Available when the parameter type is a reference type.
PowerShell expression	Specify a *simple* PowerShell expression whose output will be assigned to the parameter.

	You can use variables in the expression to access the output of an activity or a runbook parameter.
Runbook input	Specify the name of the runbook input parameter whose value will be assigned to the parameter. Available when the runbook has one or more input parameters.
Variable asset	Specify the name of the global variable asset that will be used to provide a value for the parameter. If you have connected to Azure and selected a Subscription and Automation Account on the toolbar, the data source will provide the names of the variables that are available.

Invoke Runbook Activity

The **Invoke Runbook** activity is used to invoke another child runbook. If the runbook that you want to invoke has parameters, you can access them by selecting **Parameters** on the Properties panel.

Using the Invoke Runbook activity to invoke child runbooks is an important runbook authoring tool as it allows you to share commonly used runbook logic, such as error handling, across multiple runbooks.

Code Activity

The **Code** activity is a special type of activity that can be used in a runbook to run PowerShell or PowerShell Workflow code, depending on the runbook type. This control provides functionality that might not be available by other means.

Code activities cannot accept parameters, but they can use variables to access activity output and runbook input parameters. Any output from a code activity is added to the databus. If a Code activity does not have any outgoing links, its output is added to the runbook's output.

You can view and modify the PowerShell code that will be executed by the activity by selecting the Code activity on the canvas and then selecting the **Code** tab on the Properties panel.

Junction Activity

A **Junction** is a special activity that will wait until all incoming branches are completed. This allows a runbook to run multiple activities in parallel and then ensure that they have all been completed before continuing.

Junction activities can have more than one incoming link, however **only one incoming link can be a pipeline**. The number of incoming sequence links is not constrained.

Smart Activity

Smart Activities are a special type of command activity developed by Kelverion and only available in Runbook Studio. Smart activities support interactive discovery as well as dynamic parameters, filters, and outputs. Smart activities are identified by the light bulb icon in the top right corner of the activity.

For example, the *Insert-SqlRow* activity in the *Kelverion Integration Module for Microsoft SQL Server*, can connect to a selected database in your environment and automatically provide you with the names of the tables that you can modify. When you select a target table, Runbook Studio will dynamically generate parameters based on the columns that are in the table that you selected.

When you publish a runbook to Azure, smart activities are converted to Code activities that contain PowerShell code that maps the dynamically generated parameters and filters onto the static parameters of the cmdlet that will be invoked by the activity.

Connecting Smart Activities to Your Environment

Smart connections are a unique feature of Runbook Studio and are used to configure Smart Activities so that they can connect to your IT environment and facilitate resource discovery.

Adding a Smart Activity

1. On the ribbon, select **Home**. Click **Smart Connections**, or press CTRL+SHIFT+C.
2. Click **New**.
3. Type a unique **Name** and optional **Description**.
4. Select a **Connection Type**.
5. Type appropriate values the provided connection fields. You must type values for all mandatory fields.
6. Click **OK**.

Important: Smart activity connections are not global connection assets, although they may share similar configuration properties. Smart activity connections are used by Runbook Studio to facilitate discovery. If any of the smart activities in your runbook require a connection to your systems when run in Azure, you must create the appropriate global connection assets in Azure and then configure the activities to use them.

When you save a runbook that contains one or more smart activities, the runbook file that is generated will also contain connection information. This will let you open the runbook file on another instance of Runbook Studio where the connection information has not yet been defined. However, since the runbook file is not secure, sensitive connection information, such as passwords, will be excluded and will have to be re-entered before you can use the runbook.

Alternatively, you can export smart connection information so that it is available to other instances of Runbook Studio. Smart connection export files are encrypted with a password that you specify

and as a result may contain sensitive connection information, such as passwords. When you import a smart connection export file into Runbook Studio, all connection information will be restored exactly as it was originally entered.

Exporting Smart Connections

1. On the toolbar click **Smart Connections**. The Smart Connections dialog is displayed.
2. Click **Export All**. The Export Smart Connections dialog is displayed.
3. Click the **File Path** Browse button (...) and enter the file name you want to create.
4. In the **Password** box, type a password to encrypt the file.
5. In the **Confirm Password** box, enter the password again.
6. Click **OK**.

Importing Smart Connections

1. On the toolbar click **Smart Connections**. The Smart Connections dialog is displayed.
2. Click **Import**. The Import Smart Connections dialog is displayed.
3. Click the **File Path** Browse button (...) and select the file you want to import.
4. In the **Password** box, type the password that was used to export the file.
5. Optionally, select **Overwrite existing connections** to override the connection if it already exists.
6. Click **OK**. The imported smart connections should be displayed in the list.

Smart Discovery

When you add a Smart Activity to a runbook, Runbook Studio replaces the standard cmdlet activity Parameter Sets tab, with a special **Discovery** tab.

At the top of the Discovery tab is a **Connection** box that lets you select the connection that will be used to connect to the system whose resources you want to discover. For example, this could be a connection to a Microsoft SQL Server Database, a BMC Remedy ARS server or a ServiceNow instance.

After you select a connection, the smart activity will provide you with additional discovery options.

When you have provided values for all discovery options, Runbook Studio will dynamically populate the **Properties** and optional **Filters** tabs with options that you can use to configure the activity.

Adding a Smart filter to a Smart activity:

1. In the runbook canvas, select the desired smart activity.
2. If not already done so, click the **Discovery** tab and configure the activity.
3. Click the **Filters** tab.
4. Click **Add Filter**.
5. Select a **Filter** and **Operation**.

6. Select a **Date source** and configure it accordingly.
7. Click **OK**.

Note: The Filters tab will only be visible if the selected smart activity supports filter conditions.

Smart Parameters

Smart activities dynamically generate parameters by integrating with external systems using the discovery options that you provided, and these parameters can be accessed by selecting the **Parameters** tab.

For information on configuring parameters refer to the Parameters section of the Cmdlet Activity.

Smart Filters

To help you build runbooks that retrieve complex data from external systems, some smart integration modules will provide activities that let you define filters.

If a smart activity supports filters, the Activity Properties will include a **Filters** tab. To add a filter, select the Filters tab and click **Add Filter**.

To make changes to a smart filter, double click on on the filter to expand its properties. Any changes that you make will be automatically saved.

To remove a filter, select the filter and click the **Remove** button.

When applying multiple filters to the output generated by an activity, only objects that satisfy every filter condition will be included.

Smart Output

Data that is generated by an activity with an outgoing link is written to the data bus and any destination activity can assign data from the data bus to its parameters using the **Activity output** data source. In many instances, you will want to select a specific property of an output object, and this can be difficult if the object contains many properties and/or is a PSObject with dynamically generated properties.

Smart activities make using the data bus easier, because they provide Runbook Studio with the set of output properties that can be accessed based on the discovery options that you entered.

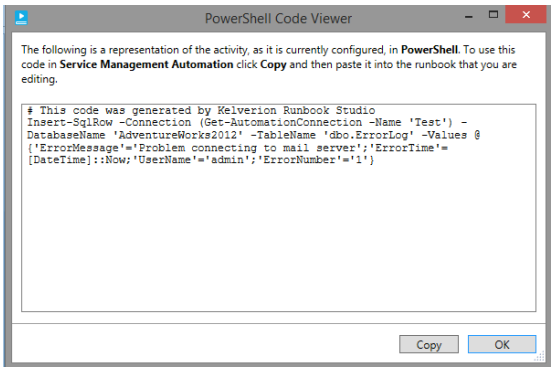
When you use the **Activity output** data source to assign an output from a smart activity to a parameter, click the down arrow in the **Field path** box to see the list of properties that are available.

Generating PowerShell

Kelverion Runbook Studio is primarily focused on helping users build graphical runbooks for Azure Automation. Furthermore, the advanced discovery features provided by the activities in Kelverion's catalog of smart integration modules are designed to work with Runbook Studio and as a result they can be difficult to configure when building runbooks in native PowerShell or PowerShell Workflow.

To help users of Service Management Automation use Kelverion's integration modules in their runbooks, Runbook Studio provides the ability to generate PowerShell code snippets from any activity in your graphical runbook. This is particularly helpful with Kelverion's Smart activities because Runbook Studio will automatically manage translating the dynamically generated parameters and/or filters in your activity into the correct PowerShell code needed to invoke the cmdlet.

To generate PowerShell code for an activity, simply right click the activity and select **View PowerShell Code**. Runbook Studio will generate the code required to invoke the activity and then display it in a dialog for you to review.



To use the PowerShell code in Service Management Automation or a PowerShell editor such as PowerShell ISE, just click **Copy** and paste the text into the body of your script.

Working with Version Control

Version control management is a system for recording changes to a file or collections of files over time so that you can recall specific versions later. Runbook Studio now includes version control to help you manage your runbook projects. Runbook Studio version control system is based on the extremely popular Git framework.

Git is a distributed version control system, where clients do not just check out the latest snapshot of files from a remote server; rather, they fully mirror the repository, including its full history. When your team collaborates on a project using Git, everyone has a clone of the repository, and each clone is a full backup of the data. This means that most of the time when you are working with Git you are working on your own local collection of files, but it also means that your data is incredibly disaster proof.

Runbook Studio enables you to do many of the things you can do with Git, including initializing and cloning repositories, creating, and checking out branches, committing changes and pushing and fetching changes to remote repositories. The following sections provide an in-depth guide to working with Git in Runbook Studio so that you can take advantage of version control in all your runbook projects.

For more information on version control and Git refer to Getting Started – What is Git? in the online version of the Pro Git book.

Getting Started with Version Control

Before you can start working with Git in Runbook Studio there is some basic setup you need to do, such as creating a user profile and authenticating Runbook Studio so that it can connect to remote Git hosting providers, such as Azure DevOps and GitHub.

Profiles

Runbook Studio uses profiles to manage user information. You can create multiple profiles to manage different projects and work environments. You must create at least one profile before you can commit changes to a Git repository.

Create a new profile:

1. On the ribbon, go to the **File** tab.
2. On the File menu select **Version Control**.
3. On the Version Control menu, select **Profiles**.
4. Click **New Profile**.
5. In the **Profile Name** box, enter a unique name for the profile.
6. In the **Name** box, enter your name.
7. In the **Email** box, enter your email address.

8. Click **OK**.

Connecting to Git Hosting Services

Runbook Studio allows you to connect to several hosted and self-managed Git hosting providers, including Azure DevOps, GitHub, GitHub Enterprise, GitLab, GitLab Self-Managed and Bitbucket.

Connecting to Azure DevOps

Runbook studio uses Personal Access Tokens to authenticate access to Azure DevOps.

Create a personal access token in Azure DevOps:

1. Sign into your organization in Azure DevOps (https://dev.azure.com/{yourorganization}).
2. From your home page, open your user settings, and select **Personal access tokens**.
3. Under **Security**, select **Personal access tokens**, and then select **+ New Token**.
4. Name your token, select the organization where you want to use the token and then choose a lifespan for your token.
5. Select the scopes for this token. Select **Custom defined** and scroll down to **Code**. Enable **Read & write**.
6. Click **Save**.
7. Click to copy the token to your clipboard. For security reasons, after you navigate off the page, you will not be able to see the token again.

Authenticate Runbook Studio:

1. On the ribbon, go to the **File** tab.
2. On the File menu, select **Version Control**.
3. On the Version Control menu, select **Remote Hosts**.
4. On the Remote Hosts panel, select **Azure DevOps**.
5. Enter your organization's **Host Domain** and your **Personal access token**.
6. Click **Connect**.

Connecting to GitHub or GitHub Enterprise

Runbook Studio uses Personal Access Tokens to authenticate access to GitHub.

Create a personal access token in GitHub:

1. Sign-in to GitHub.
2. From your home page, open your user settings and select Settings.
3. On the Personal settings menu, select Developer settings.
4. On the Developer settings menu, select Personal access tokens.

5. Click Generate new token.
6. In the Note box, name your token.
7. Select the scopes for this token. You must select **repo**.
8. Click **Generate token**.
9. Click  to copy the token to your clipboard. For security reasons, after you navigate off the page, you will not be able to see the token again.

Authenticate Runbook Studio:

1. On the ribbon, go to the **File** tab.
2. On the File menu, select **Version Control**.
3. On the Version Control menu, select **Remote Hosts**.
4. On the Remote Hosts panel, select **GitHub** or **GitHub Enterprise**.
5. Enter your **Personal access token**.
6. If authenticating GitHub Enterprise, enter the **Host Domain**.
7. Click **Connect**.

Connecting to GitLab and GitLab Self-Managed

Runbook studio uses Personal Access Tokens to authenticate with GitLab.

Create a personal access token in GitLab:

1. Sign-in to GitLab.
2. Click on our profile in the toolbar and select **Settings**.
3. In the left menu, click **Access Tokens**.
4. In the **Name** box, enter a descriptive name for your token.
5. In the **Expires at** box, enter the date that the token expires.
6. Select the **API** scope.
7. Click **Create personal access token**.

Authenticate Runbook Studio:

1. On the ribbon, go to the **File** tab.
2. On the File menu, select **Version Control**.
3. On the Version Control menu, select **Remote Hosts**.
4. On the Remote Hosts panel, select **GitLab** or **GitLab Self-Managed**.

5. Enter your **Personal access token**.
6. If authenticating GitLab Self-Managed, enter the **Host Domain**.
7. Click **Connect**.

Connecting to Atlassian Bitbucket

Runbook studio uses Bitbucket credentials to authenticate with Bitbucket.

Authenticate Runbook Studio:

1. On the ribbon, go to the **File** tab.
2. On the File menu, select **Version Control**.
3. On the Version Control menu, select **Remote Hosts**.
4. On the Remote Hosts panel, select **Bitbucket.**
5. Enter your Bitbucket **Username** and **Password**.
6. Click **Connect**.

Working with Repositories

A Git repository is a file location that acts as a virtual storage of your project. A repository allows you to save versions of your runbooks, which you can access when needed. With Runbook Studio you can initialize new Git repositories, open existing repositories, and clone existing repositories from remote hosts, such as Azure DevOps and GitHub.

For a detailed explanation of Git repositories see Git Basics – Getting a Git Repository in the online version of the Pro Git book.

Initializing a Repository

With Runbook Studio you can initialize new Git repositories to convert an existing, un-versioned collection of runbook files or to initialize a new empty repository. Most other Git features are not available outside of an initialized repository, so this is usually your first step.

Initialize a new Git repository:

1. On the ribbon, select **Version Control**.
2. Click **Init**.
3. Select the **Profile** you want to use to initialize the repository.
4. In the **Name** box, enter the name of the new repository.
5. In the **Initialize In** box, enter the repository location. Alternatively, click **Browse** to open the Browse for Folder dialog.

6. Optionally, have Runbook Studio initialize the repository with a .gitignore file and/or a Readme file. Git ignore files define the files that you do not want to track in your Git repository.
7. Click **Initialize**.

Open a local Git repository:

1. On the ribbon, select **Version Control**.
2. Click **Local** and select **Open a repository**.
3. Select the folder containing the Git repository you want to open.
4. Click **OK**.

Cloning a Repository

If you want to collaborate on a project that has been set up in a central repository, you can use the Git clone operation to get your own working copy. With Runbook Studio, you can clone Git repositories using a URL or by selecting from a list of your organization's repositories stored in Azure DevOps or GitHub.

When you clone a repository, the latest version of the remote repository files on the master branch will be pulled down and added to a new folder on your computer. This folder will contain the full history of the remote repository and a newly created master branch.

Clone a Git Repository using a URI:

1. On the ribbon, select **Version Control**.
2. Click **Clone** and select **Clone with URL**.
3. In the **Location** box, enter the parent folder that will contain the repository. Alternatively, click **Browse** to open the Browse for Folder dialog.
4. In the **URI** box, enter the URL of the repository you want to clone.
5. Click **Clone**.

Clone a Git repository from git hosting provider:

1. On the ribbon, select **Version Control**.
2. Click **Clone** and select a cloud or on-premises hosting provider.
3. In the **Location** box, enter the parent folder that will contain the repository. Alternatively, click **Browse** to open the Browse for Folder dialog.
4. If cloning from GitHub Enterprise, select the desired **Organization**.
5. In the **Repository** box, select the repository you want to clone.
6. Click **Clone**.

Recording Changes to a Git Repository

Now that you know how to initialize and clone Git repositories you are ready to start making changes and committing snapshots of those changes to your Git repositories. Typically, you will commit your work whenever the collection of runbooks in your project reaches a state that you want to record and share with others.

Some Git Basics

The files in your Git repositories have three main states: *modified*, *staged,* and *committed*. Modified means that you have modified the file but have not committed it yet. Staged means that you have marked a modified file, in its current version, as ready to be included in the next commit snapshot. Committed means that a snapshot of your file is safely stored in your Git repository.

Each Git project has three main sections: the *working tree*, the *staging area,* and the *Git directory*. The working tree is a checkout of one version of your project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify. The staging area stores information about what will go into your next commit. The Git directory is where Git stores the metadata and object database for your project.

The basic Git workflow goes something like this:

1. You modify files in your working tree.
2. You selectively stage the changes that you want to include in your next commit.
3. You do a commit, which takes the files from the staging area and stores them as a permanent snapshot in your Git directory.
4. Repeat.

Note that when you commit your work in Runbook Studio, the changes are stored in your local Git repository. To share your work with others you must push it to a remote repository.

Committing Changes in Runbook Studio

Runbook Studio makes it easy to stage the runbook files that you have modified and commit them to your Git repository. Let us say you have been working on multiple runbooks in your project and you are ready to commit those changes to your Git repository.

1. On the Resources Panel, select **Changes**.
2. Select the changes that you want to include in the commit. Recent changes are selected by default.
3. Click the **Commit changes**. The Commit dialog appears.
4. In the **Profile** box, select the profile that you want to commit with.
5. In the **Summary** box, enter a summary of the commit. The summary should be fifty characters or less and no longer than seventy-two characters.

6. Optionally, in the **Description** box, enter a description of the commit.
7. Click **OK**.

Undoing Things

Occasionally, you may neglect to include one or more changes in a commit. To amend a previous commit. Note that you should only amend a commit that you have not pushed to a remote repository.

1. On the Resources Panel, select **Changes**.
2. Select the changes that you want to make.
3. Select **Amend**.
4. Click **Amend previous commit**.

If you decide that you do not want to commit the changes that you have made to one or more of your runbooks, you can discard them and return them to the version in your most recent commit.

Discarding changes to a runbook:

1. On the Resources Panel, select **Changes**.
2. Right click on the modified file you want to reset and select **Discard changes**.

Using the Git revert operation, you can go one step further and undo the changes in a previous commit. When you revert to an existing commit, Git creates a new commit that reverses the changes that were included in the commit. With Runbook Studio you can revert the last commit that you made on a branch or go back farther using into the repository's history and revert a specific commit.

Reverting the last commit on a branch:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch with the commit you want to undo and click **Revert**.

Reverting a commit in the Git history:

1. On the ribbon, select **Version Control**.
2. Click **History**  .
3. On the left panel, right click on the commit you want to undo and select **Revert**.

Working with Remotes

Remote repositories are versions of your repositories that are hosted on the Internet or somewhere on your network. Collaborating on a Git project requires managing remote repositories and pushing and pulling data to and from them when you need to share work.

For a detailed explanation of remotes, refer to Git Basics – Working with Remotes in the online version of the Pro Git book.

Add a remote repository:

1. On the **Resources** panel, select **Repository**.
2. Right click on **REMOTE** and select **Add remote**.
3. In the **Name** box, enter the name of the remote.
4. In the **URL** box, enter the URL used to push to and pull from the remote.
5. Click **OK**.

Note that when you clone a remote repository, a remote repository with the name *origin* is automatically created for you.

Fetching and Pulling Changes from a Remote

When you fetch data from a remote repository, Git downloads all the data from the remote that you are currently missing, including all the branches from the remote.

Fetch changes from a remote repository:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **REMOTE**.
3. Right click on the remote you want to fetch from and select **Fetch**.

It is important to note that fetching from a remote only downloads data to your local repository and does not merge it with any of your work or modify on what you are currently working. To access the data that was downloaded from the remote, you must manually merge it with your current work.

Merge changes from a remote repository:

4. On the **Resources** panel, select **Repository**.
5. If necessary, click › to expand **LOCAL**.
6. Right click on the branch you want to merge with and select **Checkout**.
7. If necessary, click › to expand **REMOTE**.
8. If necessary, click › to expand the remote you want to merge from.
9. Right click on the remote branch you want to merge and select **Merge**.

If your current branch is set up to track a remote branch you have the option of performing a pull operation which automatically fetches from the remote and merges the changes into your current branch. Setting up your local branches to track a remote branch and then pulling from the remote is the best strategy for collaborating with others.

Pulling changes from a remote repository:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to update and select **Checkout**.
4. Right click on the branch and select **Pull from**.

Alternatively,

1. On the ribbon, select **Version Control**.
2. Checkout the branch you want to merge with.
3. Click **Pull**

In Runbook Studio, you can easily set up a local tracking branch by checking out the remote branch you want to work with – this will automatically create a local branch and configure it to track the remote branch. Alternatively, you can manually assign the upstream branch of a local branch to facilitate tracking.

Set upstream branch:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to update and select **Set upstream**.
4. Select the **Remote** repository you want to collaborate with.
5. In the **Branch name** box, enter the name of branch you want to track.
6. Click **OK**.

Pushing Changes to a Remote

When you are at a point where you want to share your runbooks with others, you must push your data upstream to a remote. Before you can push to a remote, you must be up to date with the remote, so you might have to pull from the remote first.

Pushing changes to a remote:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch and select **Push to**.

Alternatively,

1. On the ribbon, select **Version Control**.
2. Checkout the branch you want to push.
3. Click **Push**.

Working with Branches

A branch in Git is simply a lightweight, moveable pointer to a specific commit. Branches allow for independent lines of development with each branch having its own working directory, staging area and commit history.

For a detailed explanation refer to Git Branching – Branches in a Nutshell in the online version of the Pro Git book.

Creating, Renaming and Deleting Branches

Having a dedicated branch for new work is the Git way of doing things and it makes it quite easy to try new experiments without the fear of destroying existing functionality. Branches also make it possible to work on several unrelated runbooks at the same time and to facilitate collaboration with others.

Create a new branch:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to branch from and click **Create branch here**.
4. In the **Branch name** box, enter the name of the branch.
5. Click **OK**.

Alternatively,

1. On the ribbon, select **Version Control**.
2. Checkout the branch you want to branch from.
3. Click **Branch** .
4. In the **Branch name** box, enter the name of the branch.
5. Optionally, in the **Branch from** box, select where you want to branch from.
6. Click **OK**.

Sometimes, you may find it necessary to rename an existing branch.

Rename a branch:

1. On the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to rename and select **Rename**.
4. In the **New name** box, enter the new name for the branch.
5. Click **OK**.

Once you have finished working on a branch and have merged it into the main code base, you are free to delete it without losing any history. You may also want to delete a branch that contains a failed experiment that you do not want to share. In any case, deleting branches that are no longer required will make your Git repository easier to manage.

Delete a local branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to delete and select **Delete**.
4. Click **Yes**.

You can also delete a local branch that you have pushed to a remote repository.

Delete a remote branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **REMOTE**.
3. If necessary, click › to expand remote that contains the branch you want to delete.
4. Right click on the branch you want to delete and select **Delete**.
5. Click **Yes**.

Checking out a Branch

In Git, the term *checkout* refers to the act of switching between different versions of your project. Checking out a branch updates the files in your working directory to match the version stored in the branch as well as letting Git know that it should record all new commits on that branch.

When you create a new branch, Runbook Studio automatically checks out the new branch so that it becomes the current branch. You can also check out an existing branch at any time and make it the current branch.

Checkout a local branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on a branch and select **Checkout**.

Alternatively,

1. On the ribbon, select **Version Control**.
2. From the box, select the branch you want to check out.

When collaborating on a team, it is common to utilize shared remote repositories. Each remote repository contains its own set of branches that can be downloaded locally by performing a fetch

operation. When checking out a remote branch, Runbook Studio automatically creates a local tracking branch that is setup to push and pull to and from the remote branch.

Checkout a remote branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **REMOTE**.
3. If necessary, click › to expand the desired remote.
4. Right click on the remote branch and select **Checkout**.

Merging

When you have finished working on the runbooks in a branch you need to merge the changes into the main code branch. Merging is Git's way of putting the forked history back together again.

For more information on merging see Git Branching – Basic Branch and Merging in the online version of the Pro Git book.

Note that the merge operations in Runbook Studio involve merging changes into the current branch. The current branch will be updated to reflect the merge, but the branch you are merging from will be unchanged. Consequently, you will usually find yourself checking out branches, such as the *master* branch, and then merging another branch with it.

Merge a local branch into the current branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. If necessary, right click on the branch you want to merge into and select **Checkout**.
4. Right click on the branch you want to merge and select **Merge**.

Note that Runbook Studio will only let you merge a branch into the current branch if it includes commits that are ahead of the current branch.

You can also merge a remote branch into the current branch, which you may need to do if you have fetched changes from a remote Git repository. However, in most cases you will find yourself checking out the remote branch, in which case Runbook Studio automatically creates a local tracking branch for you.

Merge a remote branch into the current branch:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **REMOTE**.
3. If necessary, click › to expand the desired remote.
4. Right click on the remote branch and select **Merge**.

Resolving Conflicts

When collaborating on runbook projects it will eventually happen that both you and your colleagues have committed the overlapping changes that cannot be automatically resolved by Git. Runbook Studio will notify you when conflicts are encountered, and it is up to you to manually resolve them.

Sometimes conflicts will require you to choose which version of the conflicted runbook, either yours or theirs, you want to keep and other times you will need to manually modify one of the conflicted runbooks to combine the changes made by yourself and others.

Resolve merge conflict by choosing your version of the runbook:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Conflicted Files** list, right click on the conflicted file and select **Keep my version**.

Resolve merge conflict by choosing their version runbook:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Conflicted Files** list, right click on the conflicted file and select **Keep their version.**
3. Note that the conflicted file is not listed in **Staged Files**.

You cannot open conflicted runbooks files in Runbook Studio using the standard File menu because the conflicted file is no longer a valid runbook file – in fact, Runbook Studio will warn you if you try to do this. Instead, Runbook Studio lets you view both versions of the conflicted runbook file so that you determine how the runbooks differ.

Open your version of the runbook:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Conflicted Files** list, right click on the conflicted file and select **Open mine**.

Open their version of the runbook:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Conflicted Files** list, right click on the conflicted file and select **Open theirs**.

You can also update either your version or their version of the runbook and then save the modified runbook. When you save the modified runbook, it becomes the current version of the runbook; however, when you do this, you need to mark the conflict as resolved.

Manually resolve a conflict:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Conflicted Files** list, right click on the conflicted file and select **Mark resolved**.

When you have resolved all conflicted files, you must commit the merge to your Git repository.

Commit the merge:

1. On the **Resources** panel, select **Merge Conflicts**.
2. In the **Commit Message** box, enter a short subject for the commit.
3. Click **Commit merge**.

You can also return your Git repository to the unconflicted state it was previously in by simply aborting the merge.

Abort a merge:

1. On the **Resources** panel, select **Merge Conflicts**.
2. Click **Abort merge**.

Working with Tags

Tags are used by Git to highlight specific points in a repository's history as being important. Typically, you will use tags to mark release points in your runbook project (v1.0, v2.0, etc.).

For more information on tagging see Git Basics – Tagging in the online version of the Pro Git book.

Creating Tags

Runbook Studio lets you create two types of tags: *lightweight tags* and *annotated tags.* Lightweight tags can be thought of as a branch that does not change – it is just a pointer to a specific commit. Annotated tags are full objects in the Git database, and they include information about the user that created the tag as well as a descriptive message.

Create a lightweight tag:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to tag and click **Create tag here**.
4. In the **Name** box, enter the name of the tag.
5. Click **OK**.

Create an annotated tag:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **LOCAL**.
3. Right click on the branch you want to tag and click **Create annotated tag here**.
4. In the **Name** box, enter the name of the tag.
5. In the **Message** box, enter a description of the tag.
6. Click **OK**.

Sharing Tags

When you want to share a tag with others that you are collaborating with, Runbook Studio lets you push it to your remote repositories. Typically, you will only push annotated tags to your remotes and keep lightweight tags private.

Push a tag to a remote:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **TAGS**.
3. Right click on the tag you want to share and select **Push**.

Deleting Tags

Runbook Studio lets delete tags that are no longer required, and you can choose to remove them locally as well as from your remote repositories. Deleting tags that are no longer required makes your local and remote repositories easier to manage.

Delete a tag locally:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **TAGS**.
3. Right click on the tag you want delete and select **Delete locally**.

Delete a tag from a remote:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click › to expand **TAGS**.
3. Right click on the tag you want to delete and select **Delete tag from remote**.
4. Select the remote you want to delete from or select **Delete from all remotes.**

Working with Stashes

Stashing lets you store work that you are not yet ready to commit so that you can switch to another branch. For example, let us say you are working on a new runbook, but you must fix a bug in another runbook. Stashing takes the modified files in your working directory, including modified files, new untracked files and staged changes and stores them so that you can apply them later.

Stashing Your Work

When you create a stash, the current state of your working directory is stored and returned to a clean state. Note that you can only create a stash when you have changes to your current branch.

You are not limited to a single stash. You can stash several times to create multiple stashes as required. You can think of the Git stash as a stack of unfinished changes, in which you push changes onto the stack and then pop them off the stash when you are ready to work on them again.

Creating a stash:

1. On the ribbon, select **Version Control**.
2. Click **Stash**  .
3. Optionally, click **Override default message** and in the **Message** box, enter a brief description of the stashed changes.
4. Click **OK**.

Applying Your Stashed Changes

When you are ready to restore your stashed changes, you have the option to *pop* or *apply* the stash. When you pop a stash, the stored changes are applied to your working directory and the stash is removed from the stash. When you apply a stash, the stored changes are applied to the working directory, but the stash is not removed.

Popping the most recent stash:

1. On the ribbon, select **Version Control**.
2. Click **Pop**  .

Runbook Studio also lets you selectively pop and restore stashes other than the most recent one as well as drop stashes from the stack.

Selectively pop a stash:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click to expand **STASHES**.
3. Right click on the stash you want to apply and click **Pop Stash**.

Selectively apply a stash:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click to expand **STASHES**.
3. Right click on the stash you want to apply and click **Apply Stash**.

Selectively drop a stash:

1. In the **Resources** panel, select **Repository**.
2. If necessary, click to expand **STASHES**.
3. Right click on the stash you want to apply and click **Drop Stash**.
4. When prompted as to whether you want to drop the stash, click **Yes**.

Let us Know How We are Doing?

If you encounter a problem while working with Runbook Studio or have an idea for making Runbook Studio even better, we would like to hear from you.

1. Click the **File** tab and then click **About**.
2. Click **Send Feedback**.

Alternatively, you can send us an e-mail at *studiofeedback@kelverion.com*

We look forward to hearing from you.