Blue Prism

Excel Automation Guide

Date	Revision	Author	Description
2017/11/03	1.0	JT	Initial Version

Published by:

Blue Prism Limited
Centrix House
Crow Lane East
Newton-le-Willows
WA12 9UY, UK
Registered in England; Reg. No. 4260035
www.blueprism.com
Tel: 0870 879 3000

Contents

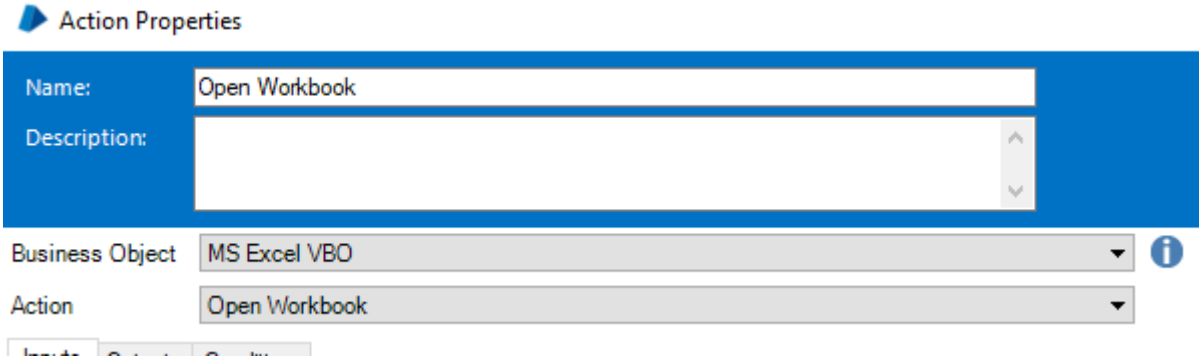
1 Introduction

This document acts as a useful source of advice and various helpful tips to Blue Prism developers who are attempting to automate Microsoft Excel. This document can be referred to when seeking best practices relating to interfacing with Excel and to troubleshoot issues encountered when automating Excel.

Excel is the recommended spreadsheet software for Enterprise RPA. Excel is extremely capable software widely used by many organisations across the world, featuring extensive API functionality. This comprehensive API is leveraged by Blue Prism in the form of the MS Excel VBO which is bundled with Blue Prism. There are however situations where the MS Excel VBO cannot be used, such as when Excel is being accessed over virtualisation software like Citrix. This guide contains advice to cater for both occasions.

2 MS Excel VBO

Microsoft Excel features a comprehensive API which is utilised by Blue Prism in the form of the MS Excel VBO. This chapter outlines some general best practices relating to using the MS Excel VBO as well as other performance and application tips. For more detail on the MS Excel VBO and its actions, click the "i" button next to the Business Object dropdown when you have selected MS Excel VBO.



2.1 Opening a workbook using the MS Excel VBO

In most situations, using the MS Excel VBO to launch Excel and open a specified workbook is the most appropriate method. This means the object will be attached to the newly created instance of Excel and you will be able to use its actions immediately. The *Create Instance* action will create a new instance and output the handle of that instance. The handle is used to identify the created instance in subsequent actions. Each instance of Excel created will have a different handle number.

The *Open Workbook* action can then be used to open a specified workbook. It is possible to use the *Open Workbook* action without using Create Instance first, since *Open Workbook* will automatically create a new instance if one does not already exist.

If your process uses Excel throughout processing, opening and closing several workbooks for example, it is worth considering creating a single instance of Excel at the start of the process and only closing it when the process has finished working. Doing so will minimise memory usage and yield a small reduction in case processing time.

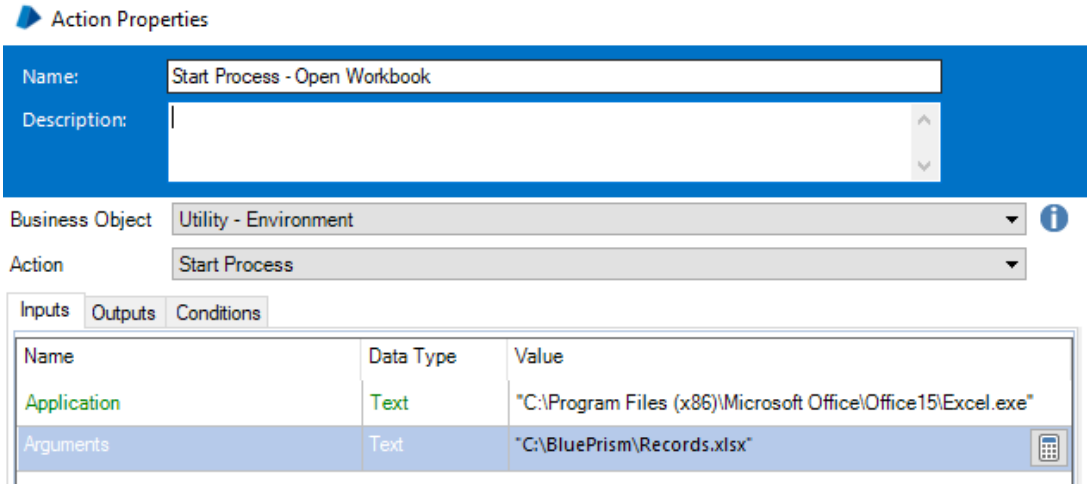
If a popup window appears after a workbook is opened which causes Blue Prism to hang you can use the *Enable Events* input of the *Create Instance* action. It will need to be set to *False*. This should prevent such popups from appearing. See chapter 2.8 of this document for more information.

2.2 Attaching the MS Excel VBO to an open workbook

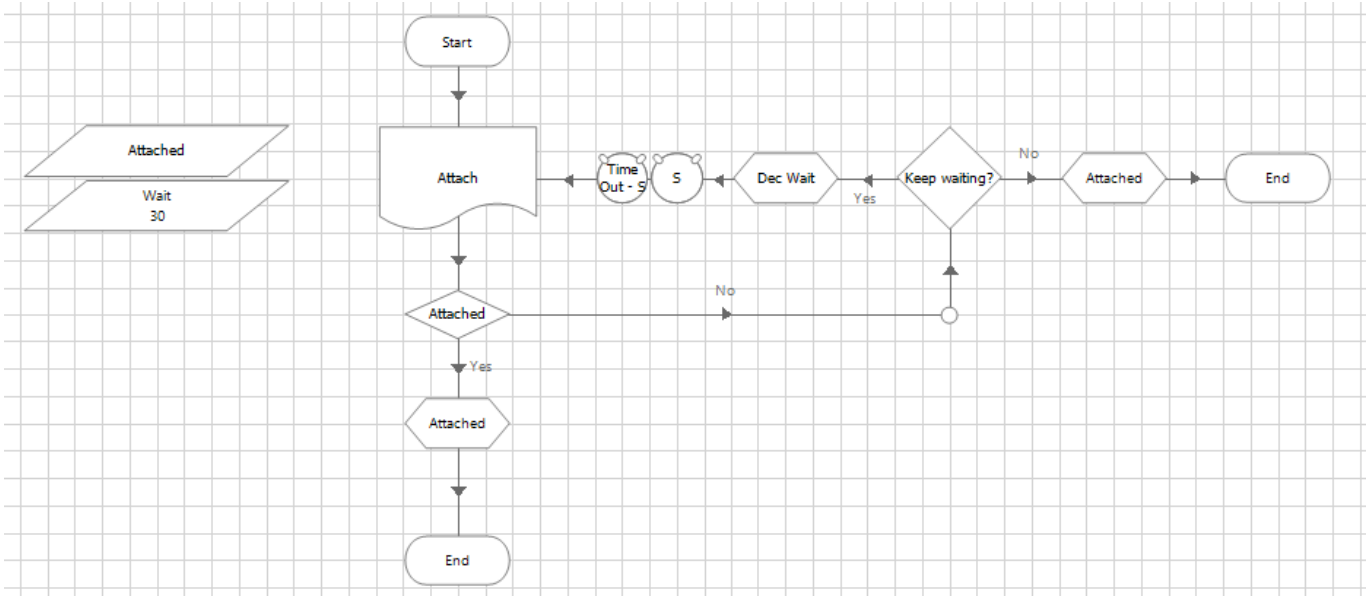
Sometimes a separate application is used to open a workbook. For example, a workbook and its contents may be created at runtime by a host application and then presented on the screen as a running instance of Excel. To use the MS Excel VBO it will need to be attached to the new instance. This can be achieved by using the *Attach* action, which will attach to the first instance of Excel it finds.

2.3 Opening a workbook using the Start Process action

If a workbook has a component such as a custom add-in that does not function properly when the workbook is launched via the MS Excel VBO, you can try using the *Start Process* action found within the *Utility – Environment* object instead. You will need to specify the file path to the Excel executable as the Application input and the file path to the workbook you wish to open as the Arguments input.



If you are using the *Start Process* action to open a workbook you will need to attach the MS Excel VBO object (just as with any other object) to the workbook before you can use it, as discussed in chapter 2.2. If the workbook takes some time to open, you will need to ensure it has finished loading before you attempt to work with it. To cater for this, you can create some logic that attempts to attach to Excel for a specified period of time. An example is below (attach page contains the *Attach* action):



2.4 Closing Excel

There are multiple ways of closing Excel, some by using the MS Excel VBO. The *Close Workbook* action gives you the option of saving changes to the specified workbook, but the instance will continue to exist, useful if your process works with multiple workbooks as discussed in chapter 2.1 of this document. The *Close Instance* action also has an input to specify whether changes are to be saved or not, but it will also close the instance of Excel, not just the open workbook.

The *Close All Instances* action should generally be avoided since it can cause an "RPC Server Is Unavailable" error as discussed by this Knowledgebase article named *RPC Server Is Unavailable (Exception from HRESULT: 0x800706BA)*. Some generic logic to close all open instances of Excel shouldn't be necessary if individual instances are properly managed.

Alternatives to using the MS Excel VBO to close Excel include creating a separate object to attach and close Excel, either by spying the "close" button, sending a keyboard shortcut such as ALT + F4 or using a Terminate stage. It is recommended Excel is closed "cleanly", either by using the VBO or closing the application as a human might. Using actions such as *Kill Process* to close instances of Excel is known to sometimes cause RPC Server or memory issues.

2.5 Performance

2.5.1 Effects of displaying a workbook on the screen

The performance of Excel automation using the MS Excel VBO can vary depending on how the object is implemented and executed.

An example of this is the performance impact of the *Show* action. This action is often used to display a newly created instance of Excel and opened workbook on the screen. When the workbook is displayed on the screen the speed at which the MS Excel VBO can manipulate the workbook is greatly reduced, even though to the human eye it still appears very swift.

Displaying the workbook on the screen has benefits such as for demonstrating and aiding development and testing, although it is not necessary once the process in development has passed all testing phases and is ready for deployment into a production environment. Therefore, it is recommended that workbooks are not displayed on the screen during production running to quicken processing time.

Data security should also be considered when workbooks are displayed on-screen if the workbooks in question contain sensitive data.

2.5.2 Control Room and Process Studio performance differences

As you will know, the speed at which a process, and the objects it uses, executes varies between Control Room and Process Studio. This is especially the case when using the MS Excel VBO. Control Room running will be much faster than that experienced when using Process Studio, even if the "Step Out" button is used to start running. This is also the case when working with loops and the collections and the Utility – Collection Manipulation object, both of which are often used in conjunction with the MS Excel VBO.

2.5.3 Testing and timing considerations

When testing the automation of Excel or measuring timing metrics it's important to consider the performance impacts of the already mentioned approaches. For example, a complex part of a process that needs to read and manipulate a large amount of data held within a workbook displayed on the screen could take an hour to complete when being run in Process Studio. However, this same part may only take minutes when it is started through Control Room with the workbook hidden from the screen. Therefore, any performance testing should be conducted on processes running in Control Room, with Excel hidden from the screen when the MS Excel VBO is being used.

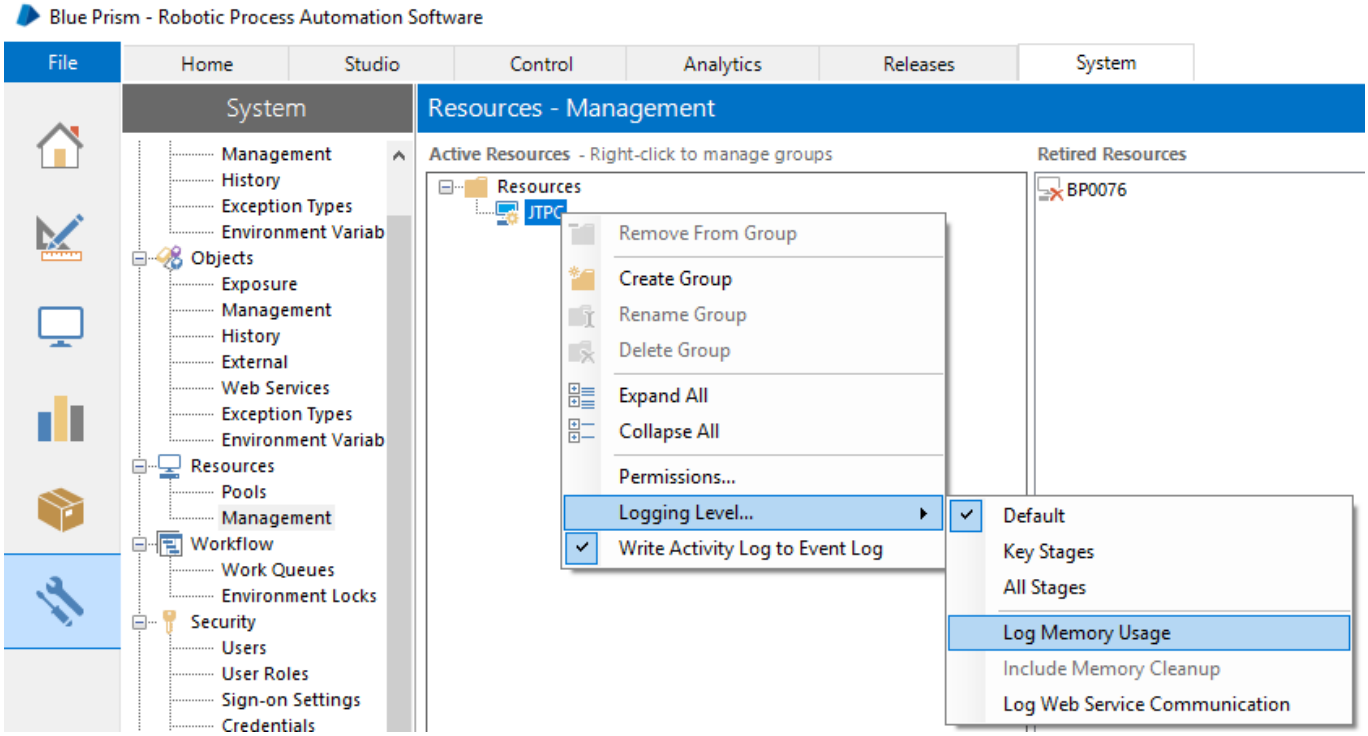
2.5.4 Memory usage

Issues can be caused by high memory usage when working with workbooks containing large amounts of data, for example, thousands of rows and hundreds of columns. Reading full datasets such as this into Blue Prism collections can exacerbate memory usage problems. Blue Prism collections are essentially .net datatables. If you search for "datatable out of memory" in a search engine, you'll see hundreds of thousands of results; this is the underlying issue with memory problems involving large collections. The resolution is to reduce the amount of data you store in a collection, or simply not use such large collections at all. There are many ways to mitigate this problem:

- Try reducing the data you read into a collection. Does your process really need it all? Try using actions such as *Get Worksheet Range As Collection* to read a certain range of cells. This re-orchestration of the solution to split up the handling of large datasets can resolve memory issues.
- Try to avoid producing copies of the same collection in different places, such as different actions in an object or sub-pages in a process containing inputs or outputs which are large collections.

- Delete unnecessary data from a collection using actions found within the Utility – Collection Manipulation object such as *Delete Column* or *Delete Field* at the earliest opportunity.
- Force the .net garbage collector to execute, freeing up memory for use. This can be achieved via a Code Stage using code such as *System.GC.Collect()*. This will only work if a collection has not got data in it so logic to clear it will be required first.
- Create a macro to perform the data manipulation you require and run this macro using the *Run Macro* action. If this method is chosen, you should consider the robustness, security and maintainability of the macro since it will be external to Blue Prism.
- Use OLEDB to directly query the workbook to extract only the required data. See the OLEDB chapter of this document or the *Guide to using OLEDB* found in the Guides section of the Portal.

If you are unsure as to which part of a process is causing memory problems you should ensure logging is enabled including memory usage logging, which can be enabled from System Manager. You shouldn't enable parameter logging if large collections are being used as inputs or outputs, this is unnecessary and will increase log sizes and possibly impact Blue Prism database performance.



2.6 Macros

2.6.1 Running a macro

It is possible to use the MS Excel VBO to run macros embedded in a workbook. This can be achieved with the *Run Marco* action. You simply need to specify the name of the macro to be executed and the handle of the Excel instance. Depending on how the macro has been created the name will be one of the following:

- Just the macro name e.g. MyMacro
- The workbook and macro name e.g. MyWorkbook.xls!MyMacro
- The workbook, worksheet and the macro name e.g. 'MyWorkbook.xls!'MyWorksheet.MyMacro

2.6.2 A macro is causing Blue Prism to hang

In some situations, a running Macro requires a response from the user before it can complete. Such macros can cause Blue Prism to "hang" since the *Run Macro* action will not complete until it has received a response from Excel indicating the user has responded. If this is the case, you will need to use an action which creates a parallel thread while running the macro, meaning control is passed back to Blue Prism as soon as the macro is executed. Blue Prism can then be used to provide a response to the macro. You can request a copy of the *Run Macro with Parallel Thread* action from Blue Prism Support if you do not already have it in your environment.

2.7 Extending the MS Excel VBO with new functionality

The scope of the Excel API is vast and therefore the Excel VBO provided by Blue Prism will likely never cover all the functionality exposed by the Excel API. The VBO will continue to be built on but might not always provide all the features of Excel you require. If the MS Excel VBO is missing some API functionality you would like to use, you can extend the capabilities of the MS Excel VBO by creating your own actions or by modifying existing actions. If you inspect the actions within the object, you'll notice the majority of actions are code stages which use various parts of the Excel API. Some knowledge of coding and the Excel API will be necessary if you wish to extend the object and Blue Prism will not be able to support the development of new actions or any processes or objects that are affected by MS Excel VBO modifications. There are some recommendations however:

- Make sure you create a duplicate object of the MS Excel VBO and rename the duplicate something like *MS Excel VBO – Extended*, and make modifications or append new actions to this object, without modifying the original. This eliminates the risk of detrimental effects to the original object and any processes that use it.
- Ensure you complete the object and action documentation such as page descriptions, preconditions and post-conditions as well as input and output descriptions.

A technique often used is to create a macro within Excel, record the functionality you need, and then examine the produced code. This code can then be applied to a code stage.

More information on how to extend the MS Excel VBO including an example is located within a document named Extending the MS Excel VBO. This is available on the Blue Prism Portal.

If possible, use the actions within the Excel VBO to achieve your requirements. This will be easier to maintain and support than a newly created bespoke code stage.

2.8 Popup windows and other GUI elements

When opening a workbook, you might see popup appear, perhaps a warning to update some links in the workbook. Such popups may cause Blue Prism to hang. The most appropriate solution would be to set the *Enable Events* input of the *Create Instance* action to false, this should prevent popups like this appearing.

You may find yourself in a situation where a GUI element or popup window cannot be dealt with by the MS Excel VBO. In these situations, the best approach is usually to create an object that attaches to Excel or the popup window and then performs the required interactions. As with any other Windows application, you will need to spy elements such as buttons and popup windows with either Win32 or Active Accessibility mode, possibly a mixture of both.

Alternatively, solutions leveraging the Excel API could be investigated, although as mentioned previously in this document this should generally be a last resort.

2.9 Export to PDF

Excel Worksheets can be exported to PDF by extending the MS Excel VBO. The Knowledgebase article named *How do I export to PDF file using the MS Excel VBO?* explains how to do this.

3 OLEDB

The OLEDB libraries allow Blue Prism to interact with many data files including Excel workbooks via a subset of SQL. This means workbooks can be queried to extract data and written to using the Data OLEDB object bundled with Blue Prism. This has advantages in both speed and memory efficiency as the entire file does not need to be fully loaded into memory.

OLEDB should be used when working with worksheets that contain large amounts of data, perhaps a table with a hundred columns and thousands of rows for example. Loading this entire data set into a Blue Prism collection using the MS Excel VBO could cause memory usage issues. Using OLEDB will negate these problems and result in a faster automation. OLEDB can be used to extract only the required data (columns and rows) from a larger dataset.

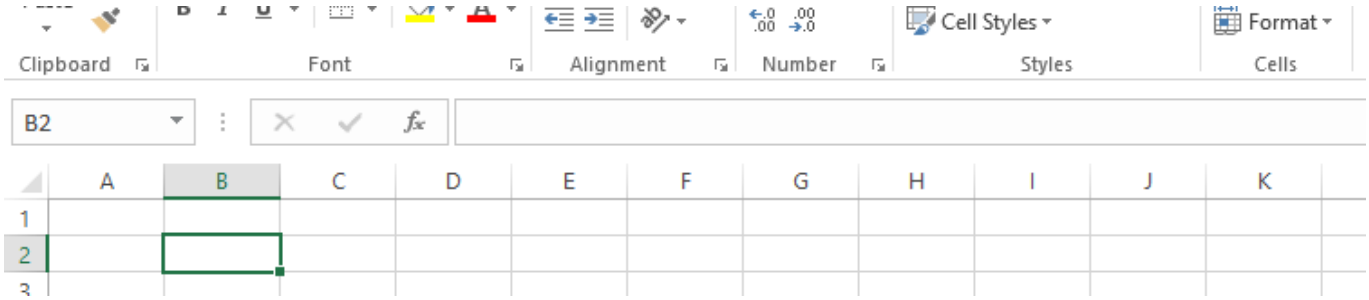
For more information on how to use OLEDB, read the *Guide to using OLEDB* found in the Guides section of the Portal. Knowledgebase articles on this subject also exist, such as the article named *How do I configure the OLEDB VBO to connect to an Excel file?*

4 Automating Virtual Instances of Excel

If you need to automate an instance of Excel that is running via Virtualisation software such as Citrix, you will need to use Surface Automation techniques, as effectively, an image of Excel is being displayed on the screen. The MS Excel VBO will not work as Blue Prism is not deployed on the target environment i.e. alongside the running instance of Excel. This chapter provides some tips that may help in this situation. Completion of the Surface Automation training is strongly recommended before attempting development.

4.1 Reading and Writing Data

As with any thin client application Blue Prism cannot interface with directly, to write data you will need to use Global Send Keys. The recommended way of writing would be to set the clipboard and then paste the value into Excel. Pasting into the Formula bar at the top of the worksheet is a good place to do this. The dropdown box next to it can be used to select the desired cell beforehand.



Similarly, cell values can be read by selecting the cell and copying its value from the Formula bar. CTRL + A does not highlight the entire text, so try using Send Keys to send SHIFT + HOME, SHIFT + END and then CTRL + C. This should highlight and copy the entire value. Blue Prism can then retrieve the data from the clipboard. OCR could also be used to read text.

4.2 Useful Keyboard Shortcuts

There are many keyboard shortcuts within Excel which you may find useful when automating Excel over virtualisation software, some of which are below:

- F9 to refresh the workbook
- Shift + directional arrows to highlight cell ranges and text.
- Alt + F8 opens the *Choose Macro* window.
- Shift + Space to select the entire row
- Ctrl + Space to select the entire column

Research Excel keyboard shotcuts prior to development to save development effort.

4.3 Ensuring Consistency

Steps can be taken to ensure the presentation of Excel remains constant, some of which include:

- Consistent screen resolution
- Keep windows maximised if possible
- Activate windows to make sure they are foregrounded
- If possible, ensure workbook formats remain consistent.

4.4 OLEDB

Can the Excel file be accessed directly via OLEDB? If it can, development effort could be reduced.

5 Solution Design Considerations

Blue Prism solution design is a separate topic with related documentation found on the Portal. There are however some specific considerations to be made when designing solutions involving automation of Microsoft Excel.

- When working with large volumes of data, consider using OLEDB to query the workbook to extract only the required data. Avoid using collections containing vast amounts of data. OLEDB can also be used to write data to workbooks.
- Where possible, keep the Excel user interface hidden from the screen. This will increase the speed of the automation.
- Avoid opening and closing instances of Excel repeatedly, instead reuse the same single instance during processing.
- Try to use the Excel VBO as much as possible as an alternative to using the Excel user interface. For example, rather than sending a mouse click to a button that starts a macro, try using the Run Macro action. Rather than filtering tables of data in Excel like a human user would, try reading the data into Blue Prism via the Excel VBO or OLEDB and performing filtering within Blue Prism. This should result in faster, more robust automations. There could be situations where it is more suitable to use the user interface however, so judgement is required.