



Software Robots - the Virtual Workforce

Surface Automation

TRAINING GUIDE

Version: 1.3

For more information please contact:

info@blueprism.com | UK: +44 (0) 870 879 3000 | US: +1 888 7577476

www.blueprism.com

Contents

1. Introduction	4
2. Training Application and Object Model.....	5
3. Interacting with Application Elements	5
3.1. Text Boxes	5
3.2. Adding Robustness.....	11
3.3. Alternative Text Input Methods	14
3.4. Multiline Textboxes.....	16
3.5. Labels and Relative Positions	18
3.6. Combo Boxes	22
3.7. Check Boxes and Radio Buttons	23
3.8. Tabulated Data	24
3.9. Dialog Boxes	27
4. Further Exercises.....	28
5. Advanced Topics	29
5.1. Fonts	29
5.2. Character Conflicts.....	32
5.3. Performance	34
6. Appendix: Object Model Solution	37
6.1. Object: Thin Client Application – Login	37
6.2. Object: Thin Client Application – Account Details.....	37
6.3. Object: Thin Client Application – Find Account.....	38
6.4. Object: Thin Client Application – Notes List	38
6.5. Object: Thin Client Application – Note.....	38
7. Appendix: Sending Special Keys	39
8. Appendix: Send Key Events	41

The training materials and other documentation ("Training Materials") provided by Blue Prism as part of the training course are Blue Prism's Intellectual Property and Confidential Information. They are to be used only in conjunction with the Blue Prism Software which is licensed to your company, and the Training Materials are subject to the terms of that license. In addition, Blue Prism hereby grants to you a personal, revocable, non-transferable and non-exclusive license to use the Training Materials in a non-production and non-commercial capacity solely for the purpose of training. You can modify or adapt the Training Materials for your internal use to the extent required to comply with your operational methods, provided that you shall (a) ensure that each copy shall include all copyright and proprietary notices included in the Training Materials; (b) keep a written record of the location and use of each such copy; and (c) provide a copy of such record to Blue Prism on request and allow Blue Prism to verify the same from time to time on request.

For the avoidance of doubt, except as permitted by the license or these terms, you cannot (a) copy, translate, reverse engineer, reverse assemble, modify, adapt, create derivative works of, decompile, merge, separate, disassemble, determine the source code of or otherwise reduce to binary code or any other human-perceivable form, the whole or any part of the Training Materials; (b) sublease, lease, assign, sell, sub-license, rent, export, re-export, encumber, permit concurrent use of or otherwise transfer or grant other rights in the whole or any part of the Training Materials; or (c) provide or otherwise make available the Training Materials in whole or in part in any form to any person, without prior written consent from Blue Prism.

© Blue Prism Limited, 2001 - 2015

All trademarks are hereby acknowledged and are used to the benefit of their respective owners.
Blue Prism is not responsible for the content of external websites referenced by this document.

Blue Prism Limited, Centrix House, Crow Lane East, Newton-le-Willows, WA12 9UY, United Kingdom
Registered in England: Reg. No. 4260035. Tel: +44 870 879 3000. Web: www.blueprism.com

1. Introduction

The Surface Automation course allows trainees to learn how to interact with application elements where no element detail is exposed by one of the Blue Prism application connectors. Essential for automating remote applications accessed via Citrix, the techniques are also relevant where traditional interfaces prove inadequate with certain application elements.

The course will focus exclusively on the object layer of Blue Prism and objects are provided for the trainees to work on and complete. The course will explore the most common application elements and the most appropriate way of interfacing with them according to Blue Prism Best Practice.

Finally, the course touches the more advanced topics of font creation, character conflicts, and performance.

The course will be intensive and is for developers of Foundation level who have completed the Introduction to Surface Automation course and are familiar with the basics of:

- Region Editor.
- System Font identification.
- Tracking elements around the screen.
- Controlling a thin client application using mouse clicks and key strokes.

During this course please bear in mind the following tips:

- The Identify Font Automatic feature of Region Editor is not guaranteed to always find your font for you. If it does not identify your font you will need to manually identify the font which may involve some experimentation. **The font in the training application is Arial 9.5.**
- If Recognize Text in a read stage does not work, the most likely causes will be either your font is not set correctly, or otherwise your region is not accurate enough. Play around with the size of your region to try and get your read stage to work.
- To check if a dynamic region is correct, you can use a temporary Read stage to read the image of your region into an Image data item. Once read, you can check the image to see if the region was set correctly.
- Be careful to ensure your regions do not include the border lines from edit boxes around your text.
- Always remember that the application window must be active for Global clicks to work. Elements you want to send Global Send Keys text to must have focus by first tabbing into or click in them.
- Be careful when debugging your work in Object Studio, as Blue Prism may regain focus rather than the training application. It is better to run your objects to breakpoints rather than stepping through with the debugger.

2. Training Application and Object Model

This training course uses a bespoke Windows training application developed by Blue Prism. Although this application will be accessed as a thick client, the course will develop an interface that uses regions rather than Windows controls. This is the interface that would be used to control thin client applications and the more challenging controls within thick clients.

Exercise 2.1.1 Application Familiarization

- Start the **Thin Client Training** application and login. Use whatever login name you wish and this will be the password.
- The application is either launched using a shortcut for the application on your desktop if using a Blue Prism virtual training environment, or by launching the .exe if installed locally.
- Navigate round the application to familiarize yourself with the functionality. Changes to data are not recorded in the underlying database but the application will respond as if they are.
- To navigate through some of the main screens in the training application, take the following steps:
 - Login, with “Account” as the value the combo box in the login screen.
 - Use the Find button and Find Account dialog to search for an Account ID. Some Account IDs you could use are: 24064631484213; 96463268719814; 60273292295243.
 - When in an account, click on the Notes tab.
 - Double-click on any row in the Notes tab table to show the “View Note” dialog.
- Appendix 6 describes an object model solution to support the application functionality. Object “stubs” have been created in Blue Prism and we will build upon these during the course.

3. Interacting with Application Elements

Within this section, we will look at interfacing with different application elements. We will start with the **Thin Client Application – Login** object.

Open the object, access Application Modeler, and launch the application.

3.1. Text Boxes

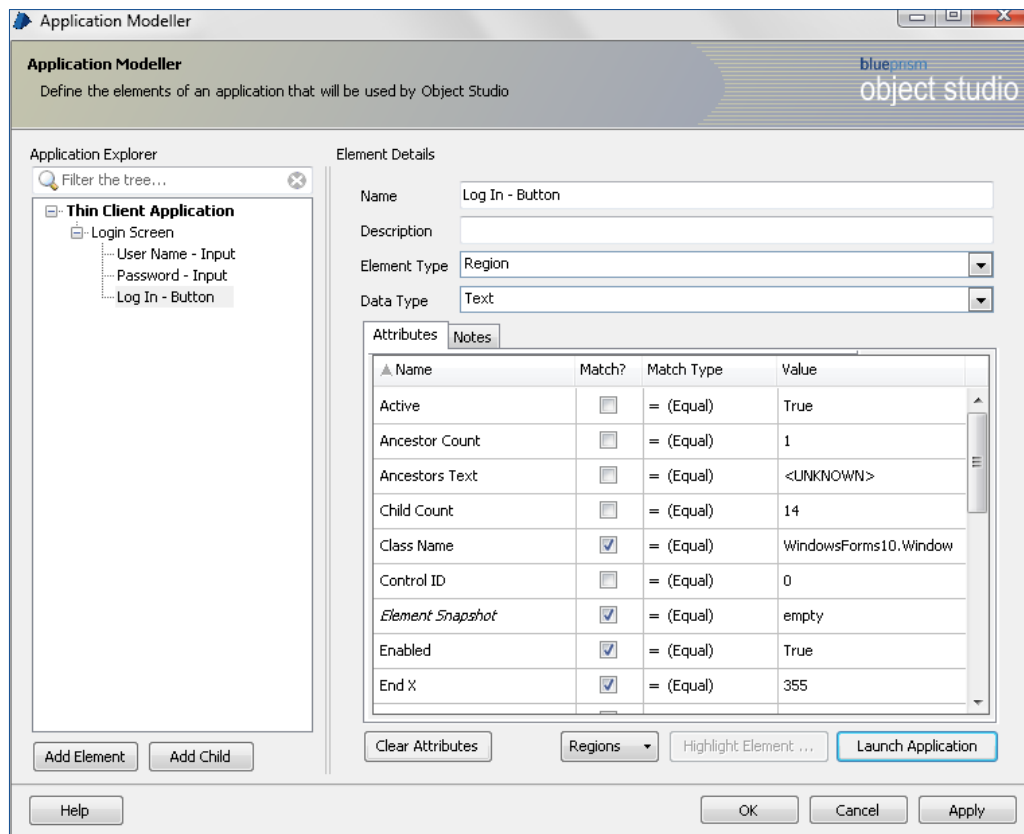
During the **Introduction to Surface Automation** course, you will have learned how to send keystrokes to the calculator application. The calculator has only one place to catch keystrokes (the calculator screen), but with most applications care must be taken when using Global SendKeys as we must be certain that either application focus or the cursor is on the required field or area of the application.

To demonstrate, we will complete the **Login** action in the **Thin Client Application – Login** object.

Application Modeler already contains an element named Login Screen. Use **Identify Element** to select the login screen (remember to use the ALT key to switch to Region mode).

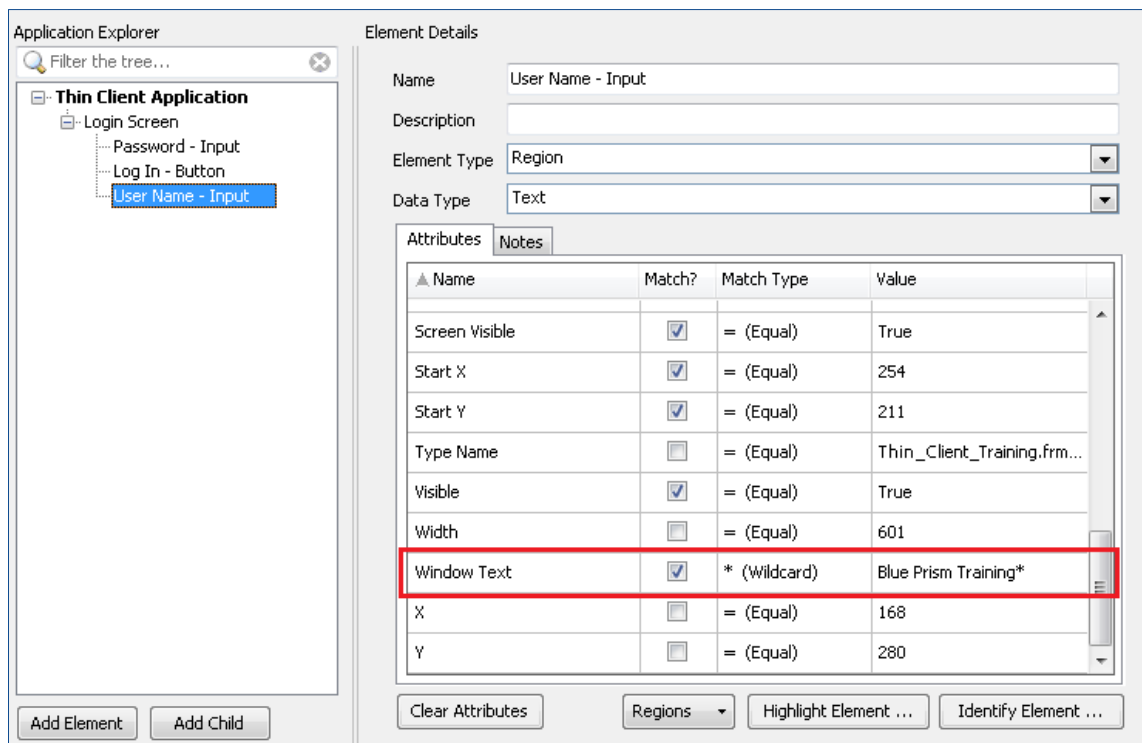
Using the **Region Editor** create regions for User Name, Password, and the Login Button.

Close Region Editor and use **Highlight Element** to verify the regions.



Even though the elements spy successfully, the Window Text attribute that we match on contains the date in the title (Blue Prism Training dd mm yyy). It is an attribute that contains dynamic data that will cause your elements to not be usable on future dates.

We need to match the Window Text **attribute** so let's use a wildcard instead of a date (see below). You will need to do this for all regions within the Login Screen.



Now that you have identified the elements in Application Modeler, you are going to use them in Object Studio to log into the training application.

To set the value of the fields, we must first select the **region** using the cursor (using Global Mouse Clicks) and then send the relevant **keystrokes** to populate the field.

The steps that your flow will need to take to log in using Surface Automation will be:

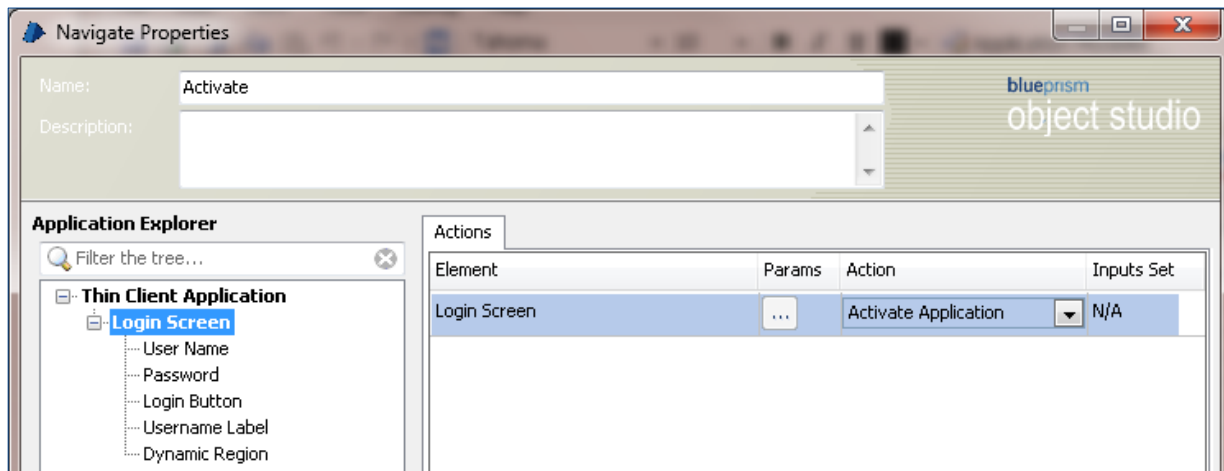
- Set the initial values for two data items in your **Login** action; one for User name and one for Password (you can them to be anything so long as they have the same value).
- Add a wait stage to wait for the log in window (it is best practice to have wait stages at the start and end of all your actions).
- Activate the log in window so that it has focus. Windows always needs to have focus if you are using any **Global** interfaces such as Global Send Keys or Global Mouse Clicks.
- Wait a tiny amount of time (for example, 0.1 seconds) to give the Activate time to work. Without waiting for activate to work your action may not work when running at full speed in control room.
- Click in the User Name edit box to give it focus.
- Send the User Name by sending keystrokes.
- Click in the Password edit box to give it focus.
- Send the Password by sending keystrokes.
- Click on the Log In button.
- Wait for the log in window to have gone.

Exercise 3.1.1 Start Building the Login Action

Use the following instructions to build the **Login** action including all the above steps:

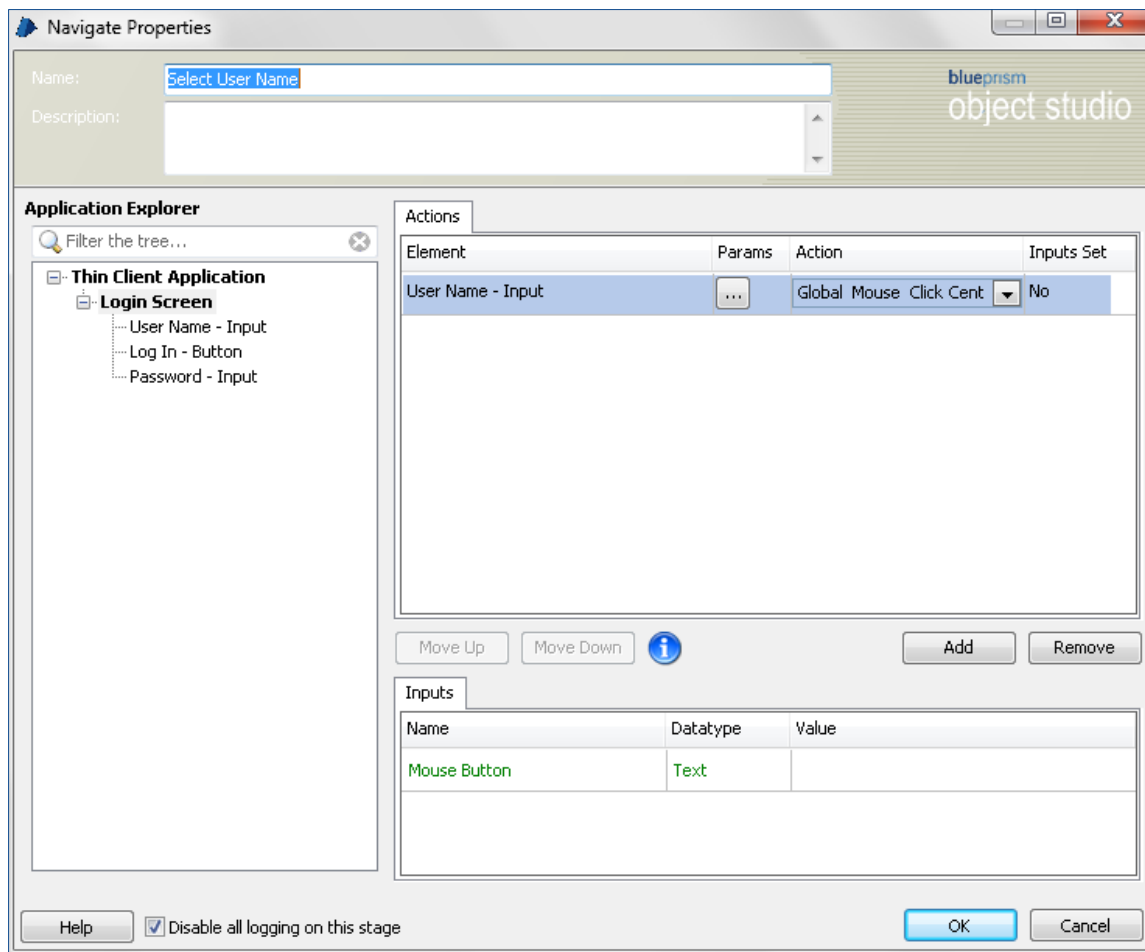
Add a Wait stage to wait for the Login window to exist.

Add a Navigate stage named **Activate** to activate the Login Window. Specify the **Login Screen** element and Activate Application as the Action (see below).



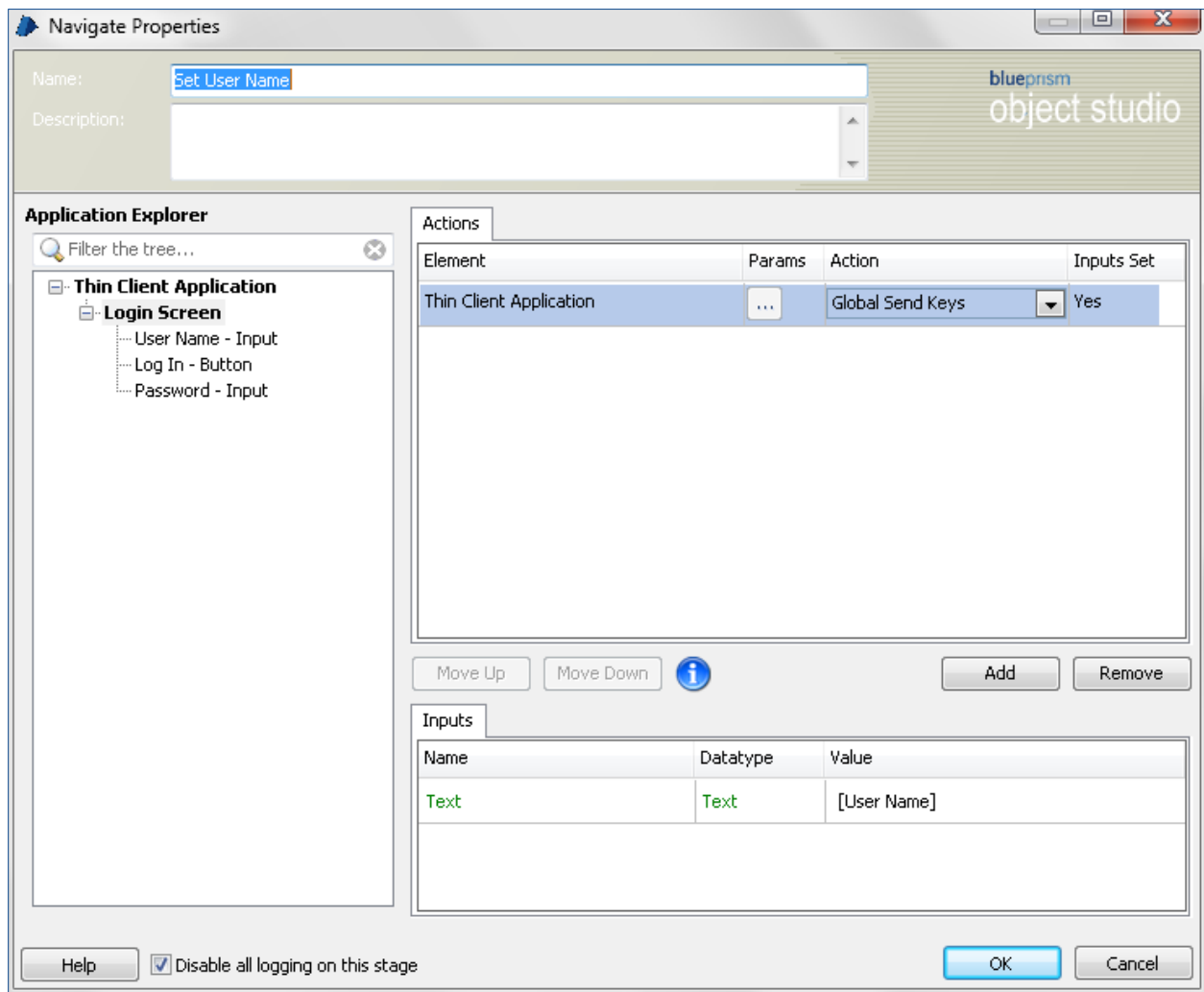
Add a wait stage, not waiting for any elements, but simply waiting for 0.5 seconds to give the Login Screen time to become active.

Add a Navigate stage named **Select User Name**. Specify the **User Name** element and **Global Mouse Click Center** as the Action (see below).

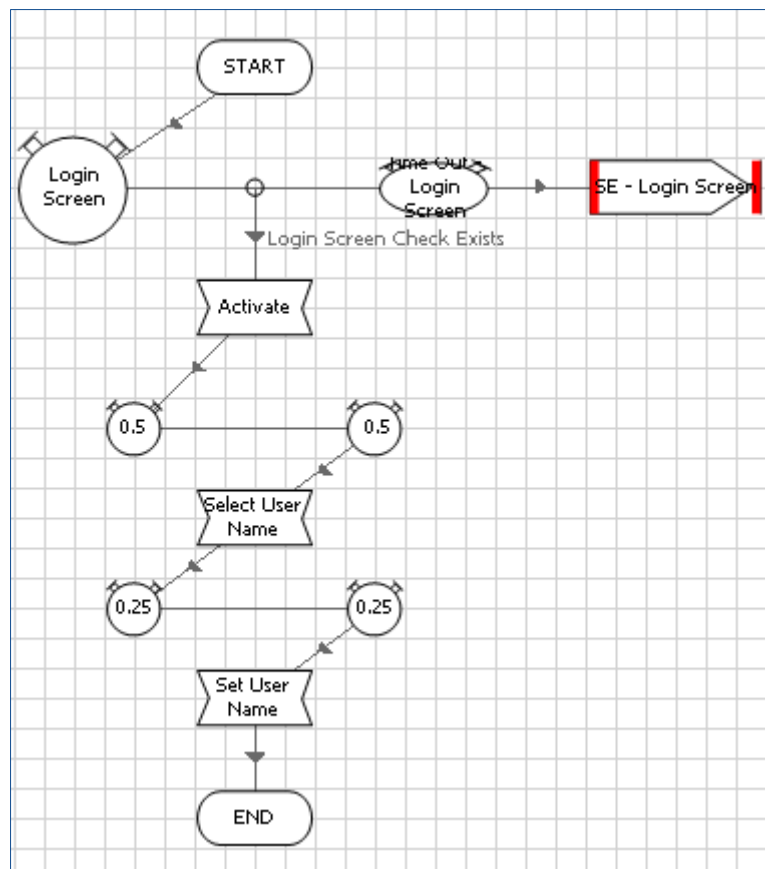


Add a wait stage, not waiting for any elements, but simply waiting for 0.5 seconds to give the Login Screen time to become active. This is required because some elements are not instantly ready to receive input after a mouse click.

Add a Navigate stage named **Set User Name**. Specify the **Thin Client Application** as the element and **Global Send Keys** as the action. Set the **User Name** as the Input parameter for the **Global Send Keys** action.



The process flow will focus the window, select the User Name field by clicking in the center of it before sending the current value of the [User Name] data item.



Run your action to verify that the user name is input to the field.

Exercise 3.1.2 Complete the Login Action

- Complete the action so that the Password is input and the Log In button clicked.

3.2. Adding Robustness

You have now built the login action, but there are potential issues that you may come across in future when using Surface Automation. These issues are:

- What if there is already text in the edit box you want to send text to?
- What if text is being sent to the application by Blue Prism too quickly, causing keys not to be sent correctly?
- What if there is a risk that you could be updating the incorrect account, or entering the incorrect values?
- What if Send Keys does not work for your application?

In the following exercises, we will look at how you will solve these potential problems:

Clearing Text and Special Keys

For the Login Screen of the training application, when we set the field values we know that the text box will be blank as the application has just started. However, when using some text boxes, it is necessary to:

- Clear the text first if there is a chance it may be populated.

There are many different ways of clearing all text from a text box finding the best way for an application will require some manual experimentation. Potential methods of clearing text include any of the following:

- Sending Home, Shift+End, Delete Keystrokes.
- Sending Ctrl+A and Delete Keystrokes.
- Sending multiple Delete keystrokes (this option should only be used if the maximum number of characters in an edit box is a low).
- Right-clicking in an element to use a Context Menu with Select All and Delete options

To update the Login action to clear User Name text box, we need to send **special keys**. Special keys are sent enclosed in braces { }. The Shift key is used by sending the + character.

A full list of Special Keys can be found in Appendix – Sending Special Keys. You can send multiple special keys by adding a number at the end of the braces e.g., {DEL 10} will send 20 delete keystrokes.

The example below shows how we send the HOME key followed by 20 Delete keystrokes followed by the User Name.

Inputs		
Name	Datatype	Value
Text	Text	"{HOME}{DEL 20}" & [User Name]

Exercise 3.2.1 Clear Text from the Send Password Stage

Edit the text we input to the Password field so that it deletes 20 characters (in the same way as we have shown in the above screenshot for the User Name input).

Test this by populating the User Name field and then running the Login action.

- Tip: You may have to change where you send your Global Click to within the Password edit box so that your Delete keystrokes are sent to the correct place.

Ensuring Global Send Keys work – keystroke delay

Sometimes Global Send Keys does not work in a reliable and consistent way due to the behavior of the application or environment you are interfacing with.

Blue Prism users have previously experienced some applications where the Global keystrokes were received with either missing characters or with characters in a different order to which Blue Prism sent them.

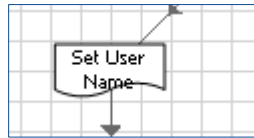
To add robustness to your Global Send Keys, **Blue Prism Best Practice recommends adding a 0.1 second keystroke between each character sent.**

The **Thin Client Application – Login** object that you have been working on has a sub-page action named **Send Keys** that has not yet been used. If you look at this sub-page you will see that it contains a loop that sends characters with a 0.1 second delay between each character.

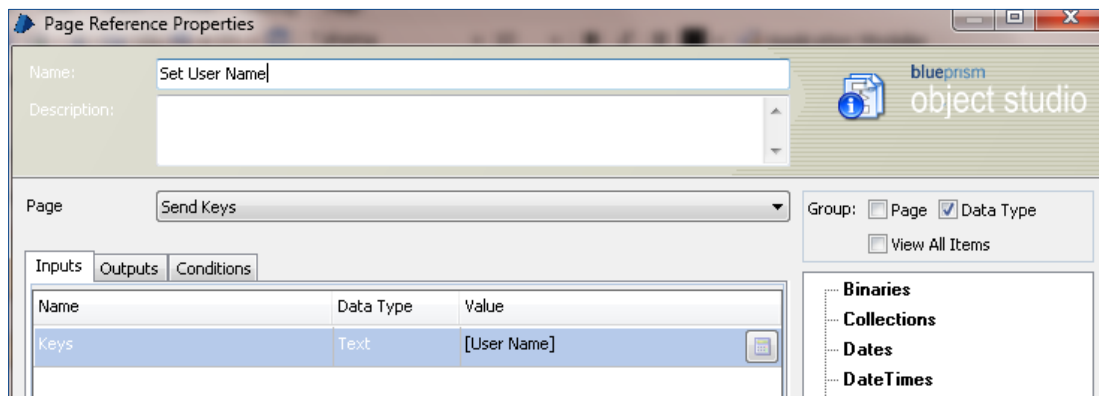
Exercise 3.2.2 Add Keystroke Delays to the Send User Name Stage

Remove the navigate stage from your **Login** action flow that sets the User Name field using Send Keys.

In its place, put a Page stage that calls the Send Keys action.



Amend your new Page stage to rename it “Set User Name”, and to set the **Keys** input parameter to be the User Name data item.



Test your action. You may not notice any difference to the performance of your Login action, but it is now running using a more robust and reliable method.

- In your future “real-world” development it is recommended that you always use keystroke delays when using Send Keys or Send Key Events.
- The default delay of 0.1 seconds has been found to be reliable for the majority of applications, but it may be increased if your testing finds Send Keys to not be sending keystrokes reliably.

Validating all Send Keys Keystrokes

Very rarely, the keystroke delay technique described above may be found to not guarantee that your Surface Automation Send Key inputs work consistently and reliably. During testing you may find that some characters are still not sent correctly, or are not inputted into the application in the correct order.

Where an issue still exists with Send Keys not sending keystrokes correctly, you will need to build in a read stage to read back the text you entered to ensure it is correct.

- It is only recommended that this method of validating all text entry is used if a reliability issue has been found with Send Keys during your testing.

Exercise 3.2.3 Validation Text Entry

Extend the Login action so that it will read the contents of the User Name field and confirms it is correct.

- ★ *Tip: Think back to the Calculator exercises in the Introduction to Surface Automation Training. We had to populate the region, identify the font and then use the Recognize Text action.*

★ *Tip: The font in the training application is Arial 9.5*

Integrity Checks

When using Surface Automation, it is good practice to ensure that you are on the correct screen, in the correct case or account, and that any high risk inputs, for example: financial transaction updates are entered correctly.

Even where testing has shown that validating text entry, as in the previous exercise, is not required, be mindful of adding additional reads the risk is deemed high of being in the incorrect account or entering the incorrect value for an update.

3.3. Alternative Text Input Methods

Send Key Events

For some applications, including systems presented by Citrix servers, the Send Keys actions that you have used so far might not work. For these applications, Send Key Events should be used instead.

Send Key Events is a similar interface, but has slightly different syntax when sending control keys. The syntax to use for Send Key Events is described in 8. Appendix – Send Key Events.

Using Clipboard to input text

For some elements, especially where large amounts of text needs to be entered, using the Windows Clipboard to paste text can be a better and faster option than using Send Keys or Send Key events.

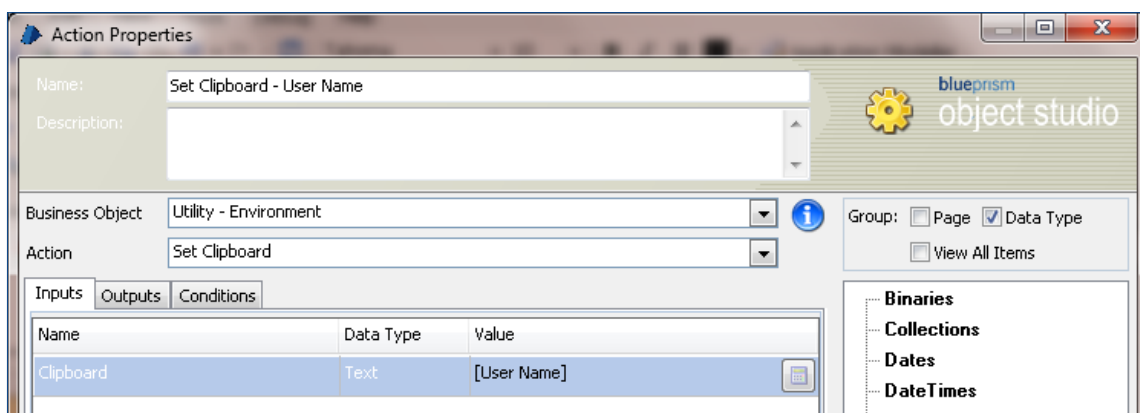
In the next exercise, we are going to replace the logic you implemented previously to enter the User Name using Send Keys, to instead use Send Key Events to paste the User Name.

Exercise 3.3.1 Using Clipboard Instead of Send Keys

Change your Login action to use Windows Clipboard to set text:

- Add an action stage before to your Set User Name stage. Select the Utility – Environment Business Object and the Set Clipboard action. Set the Clipboard input parameter to the action to be your User Name data item.

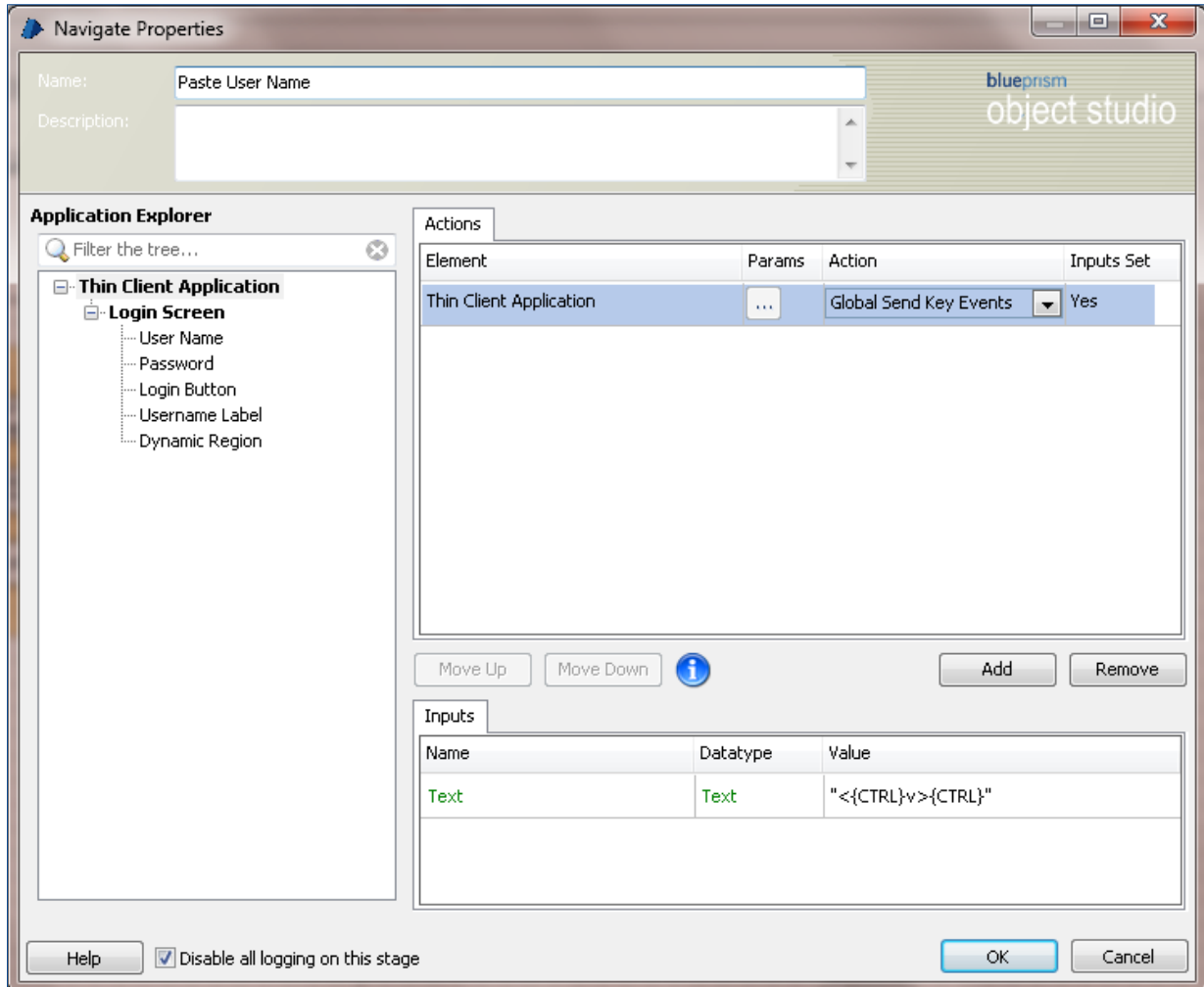
Your action stage properties should look like the following image:



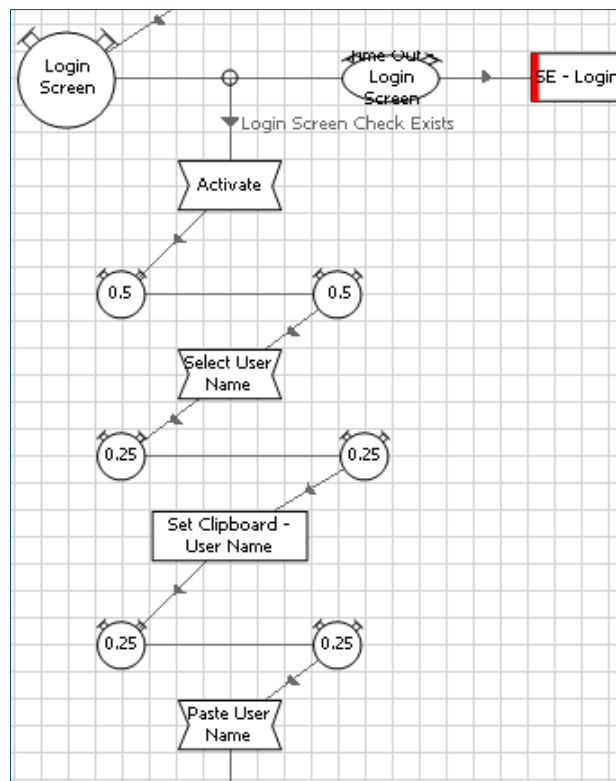
- Add a small wait for 0.25 seconds after your new action stage.
- Amend your Set User Name navigate stage. Change the action used in the navigate stage from Global Send Keys to instead use the Global Send Key Events action.

- Change the input text in the navigate stage to be the syntax for sending paste keystrokes using Send Key Events. This needs to be: "<{CTRL}v>{CTRL}".

Your navigate stage properties should look like the following image:



Your new flow should look like the flow in the following image:



Test your new flow to ensure the User Name is populated correctly.

- Not all application elements will have the clipboard enabled. If clipboard is disabled in an application then font recognition is your only option.
- Keystrokes are not your only option to use the clipboard. Sometimes menu or context menu options may be available if the CTRL-C and CTRL-V keyboard shortcuts are not available.

3.4. Multiline Textboxes

View Note

Name

GUIDERIUS

Type

Information

Note

Fear no more the heat o' the sun,
 Nor the furious winter's rages;
 Thou thy worldly task hast done,
 Home art gone, and ta'en thy wages:
 Golden lads and girls all must,
 As chimney-sweepers, come to dust.

Cancel

Using the same technique we can read from multiline textboxes. The only difference is we set the Split Lines to True.

Inputs		
Name	Datatype	Value
Colour	Text	
Background Colour	Text	
Split Lines	Flag	True

The text in the region is returned to the text data item with carriage returns and line feeds

Exercise 3.4.1

- Manually navigate the application so that a Note is open as shown above
- Attach the **Thin Client Application – Note** object.
- Complete the **Get Note** action to wait for a relevant screen element and then read the Note field. Don't forget to add your activate navigate stage to focus the View Note window.

Using Clipboard to Read Text

As well as being an option for sending text, using the clipboard to copy text is a good alternative option for reading text from applications. Especially where character conflicts exist in a Font (see 5.2 Character Conflicts), using the clipboard to copying text from fields may be the best option.

To use the clipboard to read text from an element, the following steps need to be taken:

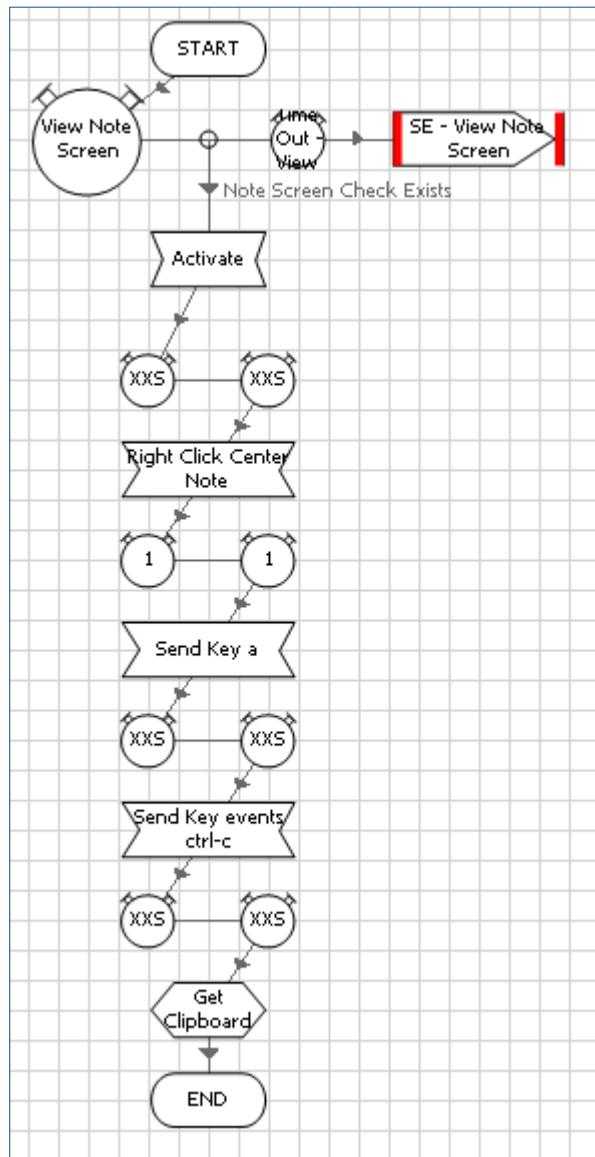
- Focus the element you want to read from, by sending a Global Click into the element.
- Send keystrokes to select all the text in the element. This can often be done by sending the CTRL-A keystrokes.
- Send keystrokes to copy all the selected text. This is done by sending the CTRL-C keystrokes.
- Use a Calculation stage with the GetClipboard() function to retrieve the clipboard and store the result in a data item

Exercise 3.4.2 Using Clipboard Instead of Recognize Text

Replace the logic you implemented in exercise 3.4.1, to instead use Send Key Events to select all the text in the multi-line note, and then copy the text.

- First focus the element and select the text.
 - Usually, using Send Key Events keystrokes can be used to select all text in an element ("`<CTRL>a<CTRL>`") but this Note element is an example of where CTRL-A does not seem to work.
 - Experimentation found that sending a Right-Click displayed a context menu, and then sending "a" keystroke using Send Keys selected all the text.
 - A right mouse click is sent instead of a left-click can be sent by simply setting the Mouse Button Input to "Right" in a Navigate stage.
 - Then Send key Events syntax for copying the selected text to clipboard will be: "`<CTRL>c<CTRL>`".
- Finally use a Calculation stage to get text from the Windows clipboard. The expression required in a calculation stage to retrieve the clipboard into a data item is GetClipboard().

Your new flow should look something like the image below (notice how Wait stages have been put between the different actions to give them time to act):



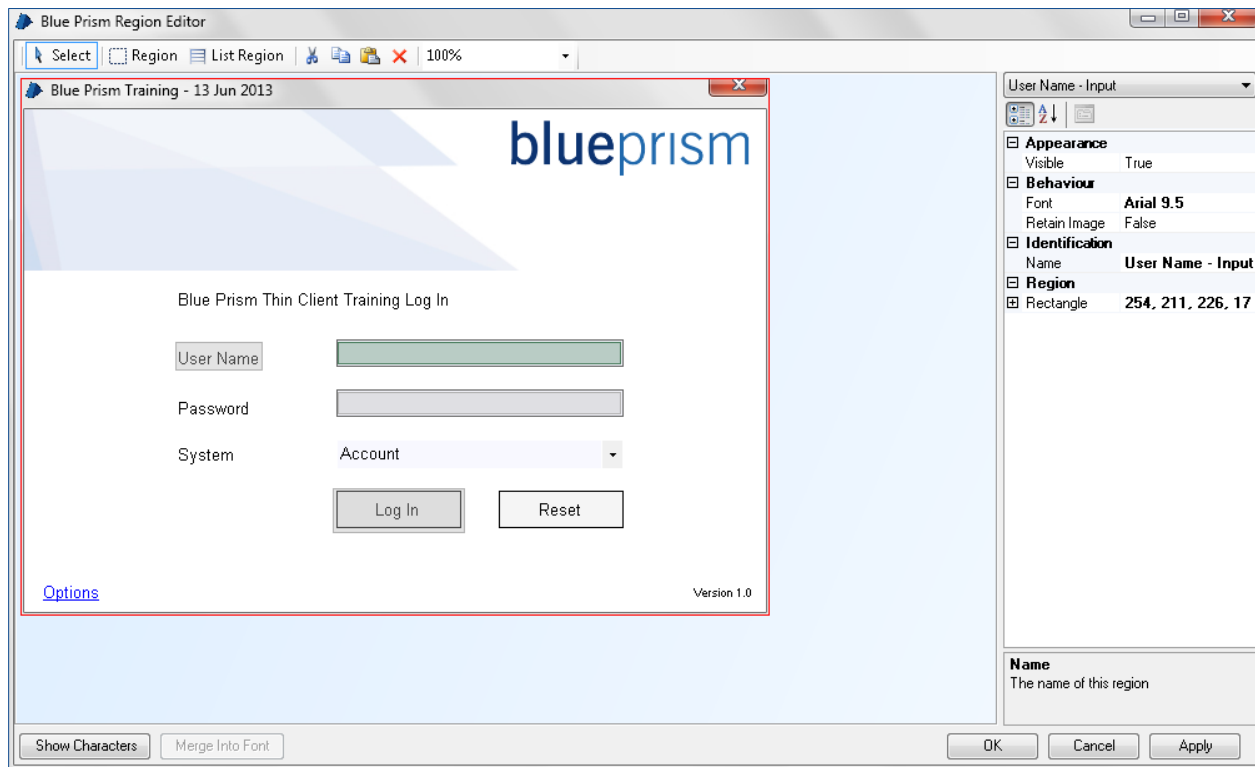
3.5. Labels and Relative Positions

In order to interact safely with applications, it is Blue Prism best practice to first qualify the position of each element before interacting with it. Most fields within an application will have a label. By first determining the exact position of the label, we can determine the relative position of the element we want to interact with.

- Before interacting with application elements, first read an element label and confirm that its value is what we expect.
- If it is, we can assume that the element we want to interact with is in the correct position.
- If we can't read the label from its expected region, then we must search for an image of the label to return the results of its location. Using these results, we can determine the position of the elements we want to interact with.
- You can either base the location of all your screen elements on their relative position to a single label or image, or for more dynamic applications, you can base the location of all screen elements to the location of their own related label.

Let's determine the position of the **User Name - Input** element **region** we created using the **User Name label**.

Create a new region to capture the user name label.

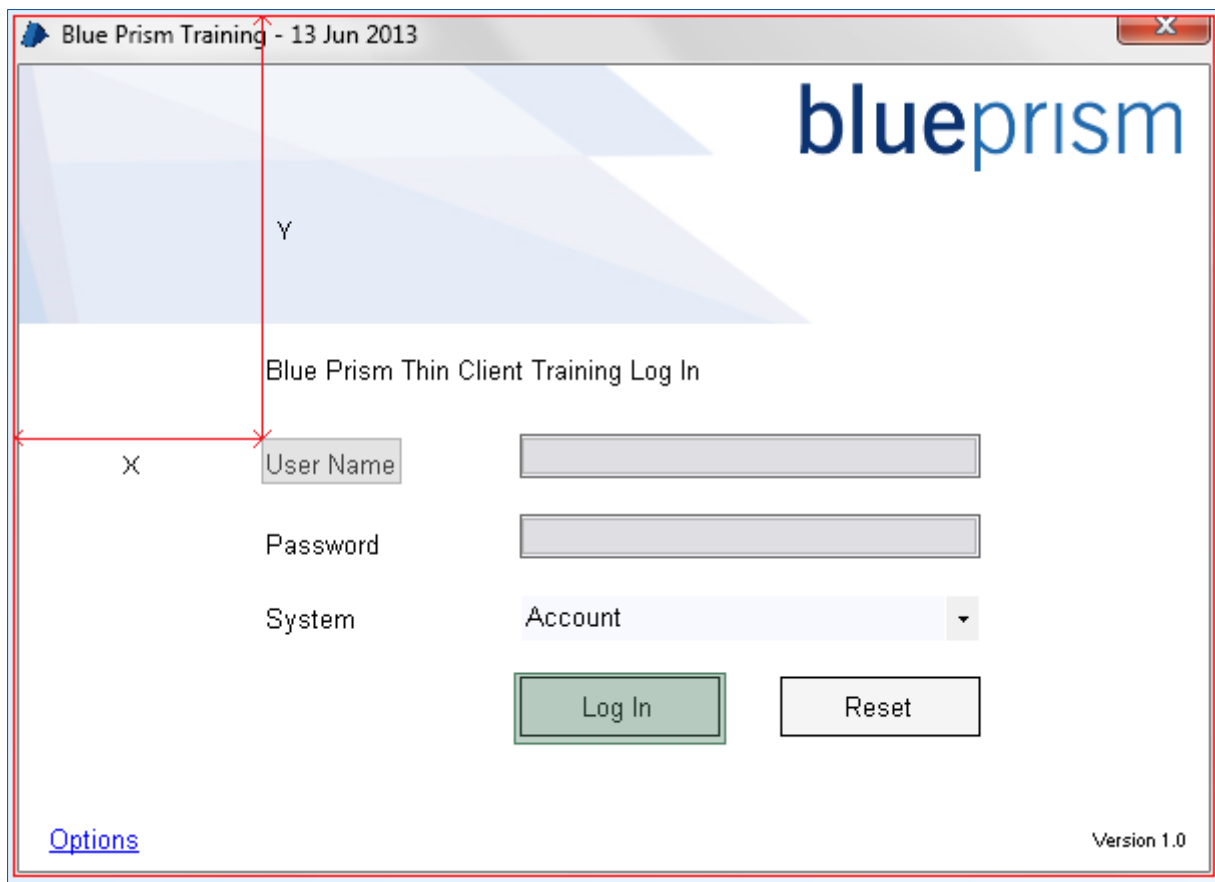


Expand the rectangle in the property list to see the exact location of the label.

Rectangle	124, 211, 69, 22
X	124
Y	211
Width	69
Height	22

The X and Y coordinates refer to the region's position within its container: X = 124, Y = 211. The container is the outer red line in the screenshot. Your values may be different depending on the exact position of the region you spied. The values are unimportant - it's their relative value to the coordinates of the User Name input box that is important.

The screenshot below shows where the X and Y coordinates are taken from:



The properties of the User Name **Input Box** are:

Rectangle	254, 211, 226, 17
X	254
Y	211
Width	226
Height	17

In this example we have the following X, Y values:

	X	Y
Label	124	211
Input Box	254	211

The horizontal (X) start position of the Input Box is 130 pixels to the right of the Label (254-124)

From this we can work out that relative to the label:

$$\text{Input Box (X)} = \text{Label (X)} + 130$$

$$\text{Input Box (Y)} = \text{Label (Y)}$$

Close Region Editor and return to Application Modeler.

The X, Y, Width and Height attributes in Application Modeler refer to the container and not the region. The regions attributes are Start X, End X, Start Y, and End Y.

The attribute properties of the **User Name – Input** include the following:

Attribute	Value	Description
X	168	This is the X position on the screen of the container. It is NOT the X position of the region
Y	280	This is the Y position on the screen of the container. It is NOT the Y position of the region
Height	431	This is the height of the container. It is NOT the height of the region
Width	601	This is the width of the container. It is NOT the width of the region
Start X	254	This is the X value of the left most point of the Input Box region. (X value from Region Editor)
End X	480	X value from Region Editor + Width value from Region Editor (254 + 226)
Start Y	211	This is the Y value of the top most point of the Input Box region. (Y value from Region Editor)
End Y	228	This is the Y value of the bottom most point of the Input Box region. Y value from Region Editor + Height value from Region Editor (211 + 17)

★ Remember your values may be slightly different depending on the size and location of your regions.

The X, Y, Height, Width values of the region we captured in the Region Editor only affect the Start X, Start Y, End X, End Y attributes in Application Modeler.

To allow our **User Name– Input** region to move around the screen relative to the position of the label we can make the Start X and End X coordinates dynamic.

Start X	<input checked="" type="checkbox"/>	Dynamic	254
Screen Visible	<input checked="" type="checkbox"/>	= (Equal)	True
Font Name	<input checked="" type="checkbox"/>	= (Equal)	Arial 9.5
End Y	<input checked="" type="checkbox"/>	= (Equal)	228
End X	<input checked="" type="checkbox"/>	Dynamic	480

Once we have determined the coordinates of the User Name Label we can set the Start X and End X values of the **User Name – Input** region in our navigate stage.

Exercise 3.5.1 Setting a User Name

Update your Login action to determine the location of the User Input label and use the returned coordinates when setting the User Name – Input value. To do this you will need to:

- Create an image data item that contains an image of the User Name – Label.
- Get the bounds of the container (screen).
- Create a dynamic region to return the snapshot of the screen (Remember set the wild card on the Window text property).
- Use the Utility - Image Manipulation to return the location of the label by searching for it within the screen.
- Determine and use the correct coordinates in the dynamic attribute values of the navigate stage to set the user name.

- For applications that change often or elements with “floating windows” that cannot be identified as a window element in Application Modeler, It is best practice to always use label images and dynamic regions to accurately find field locations.

3.6. Combo Boxes

Selecting items within combo boxes can be achieved in a variety of ways. The method you adopt will depend upon the application that you are interfacing with and how it responds to keystrokes within the combo box field.

Some combo boxes allow you to simply type in the value you want. Treat these elements as text boxes.

Alternatively, there are four possible options:

Option 1 – Searching for the Value by Entering it into the Combo Box

- Select the combo to set focus.
- Key the first letter of the item to be selected.
- Check it matches required value. If not, key the second letter, then third letter, etc.
- Keep retrying with a maximum number of attempts.

Option 2 – Searching for the Value by Entering the First Letter into the Combo Box

- Select the combo to set focus.
- Key the first letter of the item to be selected.
- Check it matches required value. If not, key the first letter again.
- Keep retrying with a maximum number of attempts.

Option 3 – Searching for the Value Using the Down Key

- Select the combo to set focus.
- Send a down key to the application.
- Check the value in the combo box matches the required value. If not, send a further down key.
- Keep looping until you see the same value twice (you’ve reached the bottom of the list) or until you’ve exceeded a maximum number of attempts.

Option 4 – Using the Drop-down List

- Click the drop-down element of the combo box to expose the selectable values.
- Return the list of visible values and their screen positions to Blue Prism
- Select the required value if it is visible.
- Scroll down the list until the required value appears or the same values appear twice (you've reached the bottom of the list) or until you've exceeded a maximum number of retries.

Exercise 3.6.1 Titles

The Thin Client Application supports Options 3 and 4.

- Click on the New button to navigate to the Add Account screen.
- Edit the **Add Account** action in the **Thin Client Application - Account Details** object.
- For now, only populate the Title field (we will complete the remainder of the object later).
- Capture the title using Option 3 and then Option 4 from above. Test using the title "Professor".

Note: it is possible to simply use a navigate stage to set the text in this combo box, but we are asking you to instead use Option 3 and Option 4 above to experience Surface Automation alternatives.

- Selecting the best method if interfacing with a Combo Box requires experimentation with using the application manually and experimenting within Object Studio.
- Interface with a Combo Box using Option 1. If this isn't possible, use Option 2, then Option 3, and finally Option 4
- Use field labels where possible to confirm the combo box is the one we wish to interface with.
- After setting the value of the Combo Box, always read its value to confirm that it has been set correctly.

3.7. Check Boxes and Radio Buttons

Setting check boxes and radio buttons is done by clicking the region captured in **Region Editor**. You will need to capture a region for each check box and radio button option. In our example application on the account page, this would be Male and Female for the Gender field and True or False for the Verified field.

When reading the value of a check box, you will need to create a **snapshot** in **Region Editor** for each state (checked and unchecked) for each option (i.e., Male and Female).

In the following example, we want to read the details of an account. This includes the Gender and Verified fields. We will need to create four elements:

- Account Details – Male.
- Account Details – Female.
- Verified – True.
- Verified – False.

We need four elements because we can't capture for example a Male and Female account holder in one screenshot. Hence we need a screenshot for each scenario.

In each element we create a region and capture an element snapshot. This will allow us to determine the Gender of an account holder and if the account has been verified.

✱ *When creating the region zoom in to make sure the region is tight against the element.*

The following screenshot shows our Gender Male element and the Element Snapshot

Element Details

Name: Gender Male

Description:

Element Type: Region Data Type

Attributes Notes

Name	Match?	Match Type	Value
Active	<input type="checkbox"/>	= (Equal)	True
Ancestor Count	<input type="checkbox"/>	= (Equal)	1
Ancestors Text	<input type="checkbox"/>	= (Equal)	<UNKNOWN>
Child Count	<input type="checkbox"/>	= (Equal)	6
Class Name	<input checked="" type="checkbox"/>	= (Equal)	Window
Control ID	<input type="checkbox"/>	= (Equal)	0
Element Snapshot	<input checked="" type="checkbox"/>	= (Equal)	58x20
Enabled	<input checked="" type="checkbox"/>	= (Equal)	True

Buttons: Clear View... Import.. Export..

Small window titled 'Male' with a checkbox and label 'Male'.

The images we have captured are only valid when the account is locked as all the elements are grayed out. If you wanted to read the values before saving the account details, you would need to repeat the exercise and capture the elements when they are not grayed out.

Exercise 3.7.1 Get/Add Account Details

Within the *Thin Client Application - Account Details* object:

- Update the *Get Account Details* action to read the Gender and Verified fields when the application is locked.
- Update the *Get Account Details* action to read the Gender and Verified fields when the application is unlocked.
- Update the *Add Account Details* action to set the Gender and Verified fields. Remember to verify the fields have been set correctly by reading the field values.

- Include option labels when reading the state of a checkbox or radio button. This will confirm the checkbox or radio button is the one we wish to interface with. In the example above, we check if the account holder was male by confirming a ticked checkbox image and "Male" label and not just the radio button.
- After setting the value of a checkbox or radio button always read its value to confirm that it has been set correctly.

3.8. Tabulated Data

Returning data from a table can be achieved with the use of a dynamic region. If we know the height of the row, the width of the column, and the font then we can navigate around a table to return data to a Blue Prism collection.

Consider the following screen from our Thin Client Application. We want to return all the data from the table so that we can determine the correct account is and then select it.

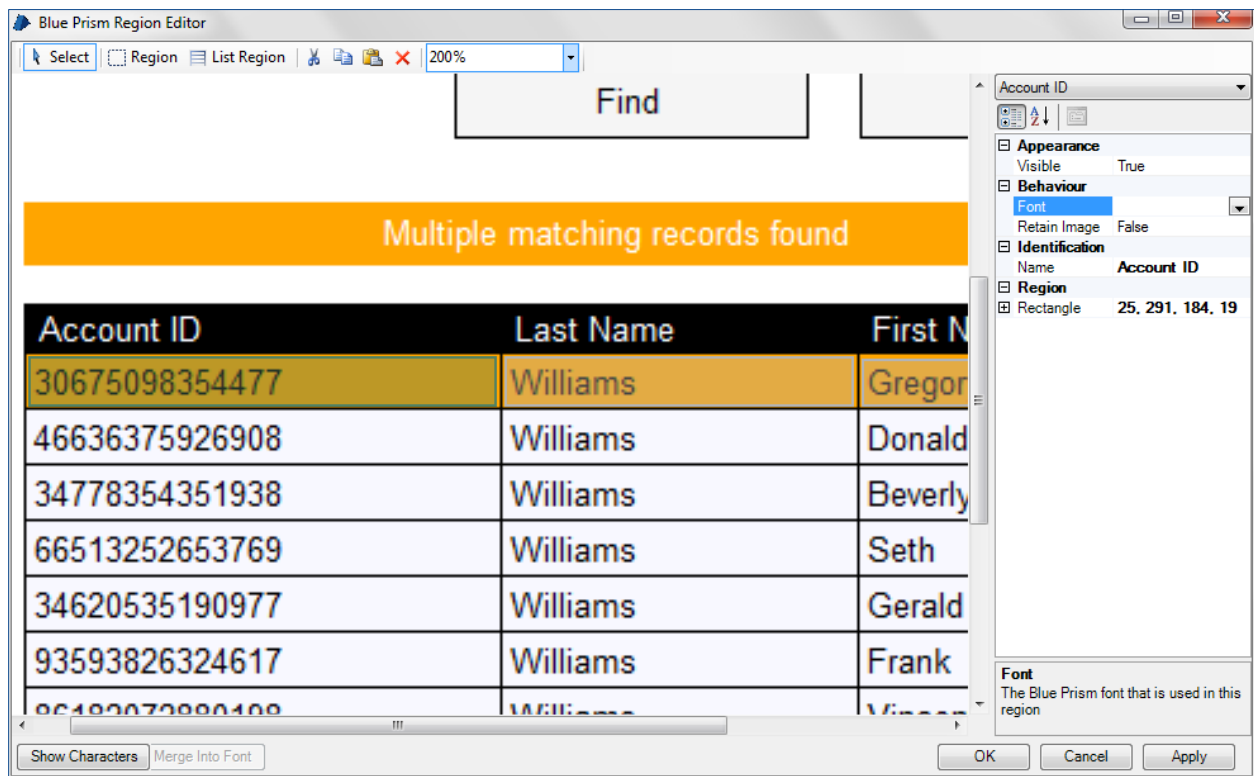
Account ID	Last Name	First Name
30675098354477	Williams	Gregory
46636375926908	Williams	Donald
34778354351938	Williams	Beverly
66513252653769	Williams	Seth
34620535190977	Williams	Gerald
93593826324617	Williams	Frank
86182072880198	Williams	Vincent
62409570850685	Williams	Lisa
28245891846443	Williams	Eve
21955645627760	Williams	Jesse

Using **Region Editor** we can identify the column widths and the column heights. But we do not know how many rows the table will include.

We need to build an object that will accommodate the whole table. If there are too many records to fit in the window a scroll bar will appear allowing you to scroll down to further records. This functionality will need to be catered for in our business object.

The screenshot below shows how we have created regions for the columns; Account ID, Last Name, and First Name.

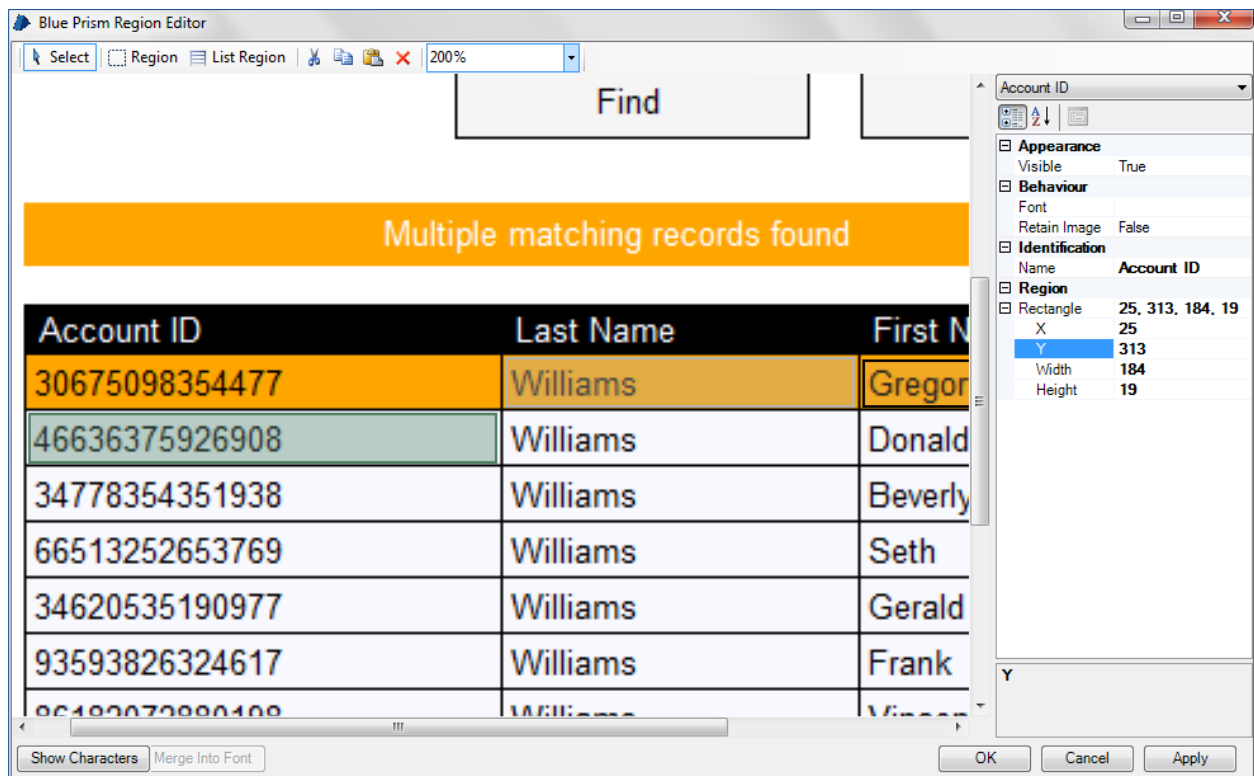
★ Notice that we have zoomed into 200%. Defining regions can be a lot easier if you zoom in.



We will make the Start Y and End Y coordinates of each region dynamic in Application Modeler. This will allow us to move the regions down the column in order to read each cell.

To move the regions down the table, we will need to increment the Start Y and End Y values by a specific number of pixels. To determine how many pixels we need to move down, manually amend the Y value of the Account ID region. As you change this value in Region Editor, the region will move down the screenshot; when it is in the correct position make a note of the Y value before returning it to its original value.

The screenshot below shows the Account ID over the second cell in the column.



The Y value we had to set to position the region in the correct place was 313.

The starting position was 291. Hence we need to increase the Start Y and End Y values for the dynamic regions by 22 pixels (313 – 291).

Exercise 3.8.1

In the **Thin Client Application - Find Account** object, complete the Get Results action to read the results table to a Blue Prism collection.

- Create dynamic regions for each column.
- Navigate the regions down the table to return the cell values to a Blue Prism collection.
- Determine how to recognize the end of the list of records.
- Determine how to continue if the search results extend to further pages. (To populate the table with more than one page of results search for Surname “w*”).

3.9. Dialog Boxes

- It is often possible to make the dialog box the parent container when using Citrix. This can be done by moving the main application window away from behind the dialog box. This will allow the Blue Prism spy tool to recognize the dialog box so that you don’t need to track it around the screen.

4. Further Exercises

Exercise 4.1.1

Complete the “Thin Client Application – Login” object to recognize an incorrect password or login ID warning message.

Exercise 4.1.2

In the “Thin Client Application – Account Details” object, complete the “Get Account Details” action (remember to first confirm that you are on the correct screen).

Exercise 4.1.3

In the “Thin Client Application – Account Details” object, complete the “Navigate to Notes” action, the “Navigate to Find” action, and the “Close” action.

Exercise 4.1.4

In the “Thin Client Application – Notes List” object, complete the “Get Notes” action. This will need to return in a collection the note details of all notes that meet the input requirements (Date Range, Type).

5. Advanced Topics

5.1. Fonts

Identifying Fonts

When identifying system fonts, we have used the **Automatic** tab that allows Blue Prism to return the best matches. From this you can select and test the returned fonts to determine the most appropriate.

If no fonts are returned, you can use the **Manual** tab to return samples from all available system fonts and manually select one. Alternatively you can create a font (see 5.1.3 Creating a Font).

Qualifying Fonts

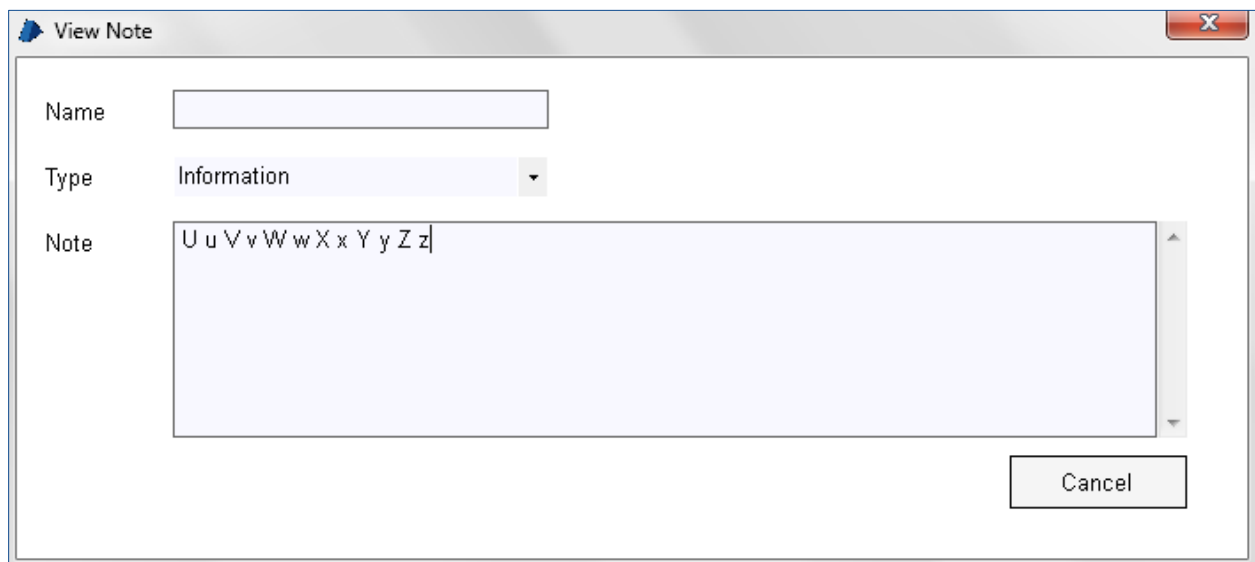
Irrespective of whether the font was found automatically or manually, an important part of the interface configuration is to qualify the font. This means proving all the letters (upper and lower case), numbers and symbols match correctly. This may not always be possible and workarounds may be required (5.2 Character Conflicts)

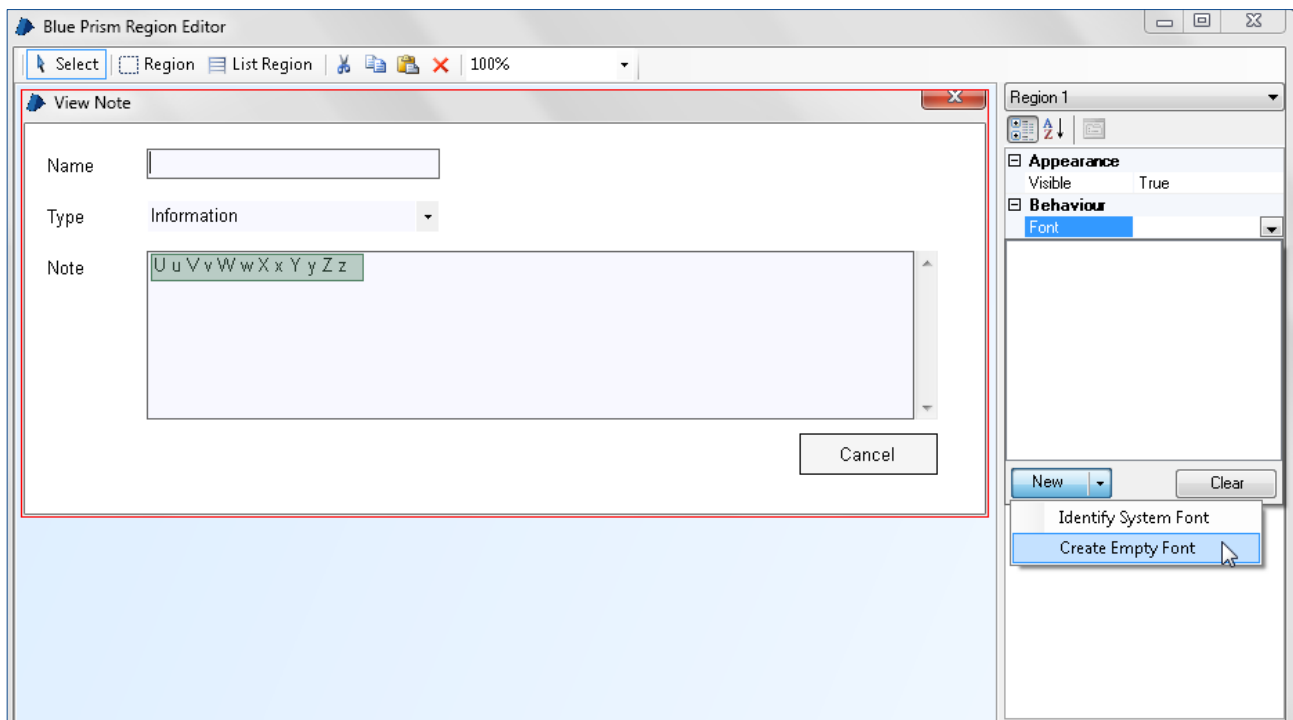
To qualify a font:

- Read as much text from labels as you can to provide a wide selection of alphanumeric characters and symbols.
- Find an area of the application where you can enter text (e.g., note field or large input box) and read it within Blue Prism to confirm values.

Creating a Font

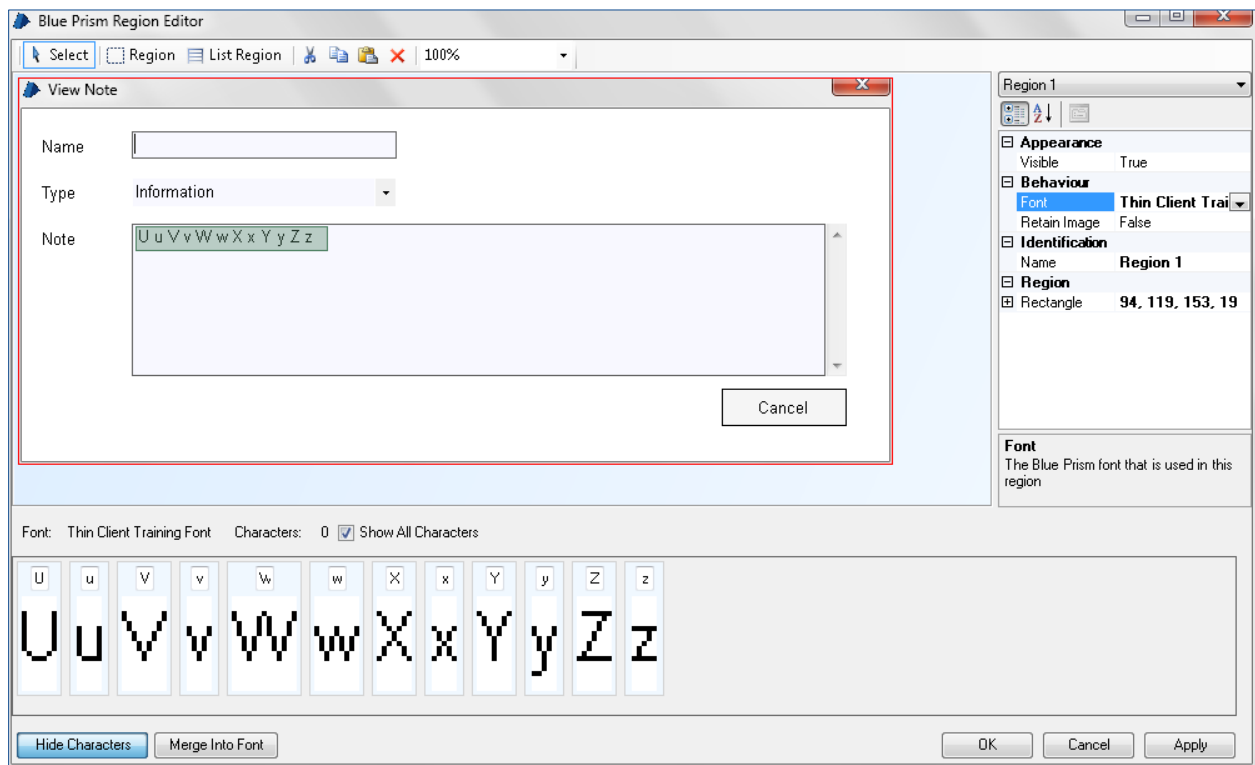
Where a font can't be identified or successfully qualified you can create a font. Consider the following screenshot. We have navigated the process to the Note screen and entered a range of characters.



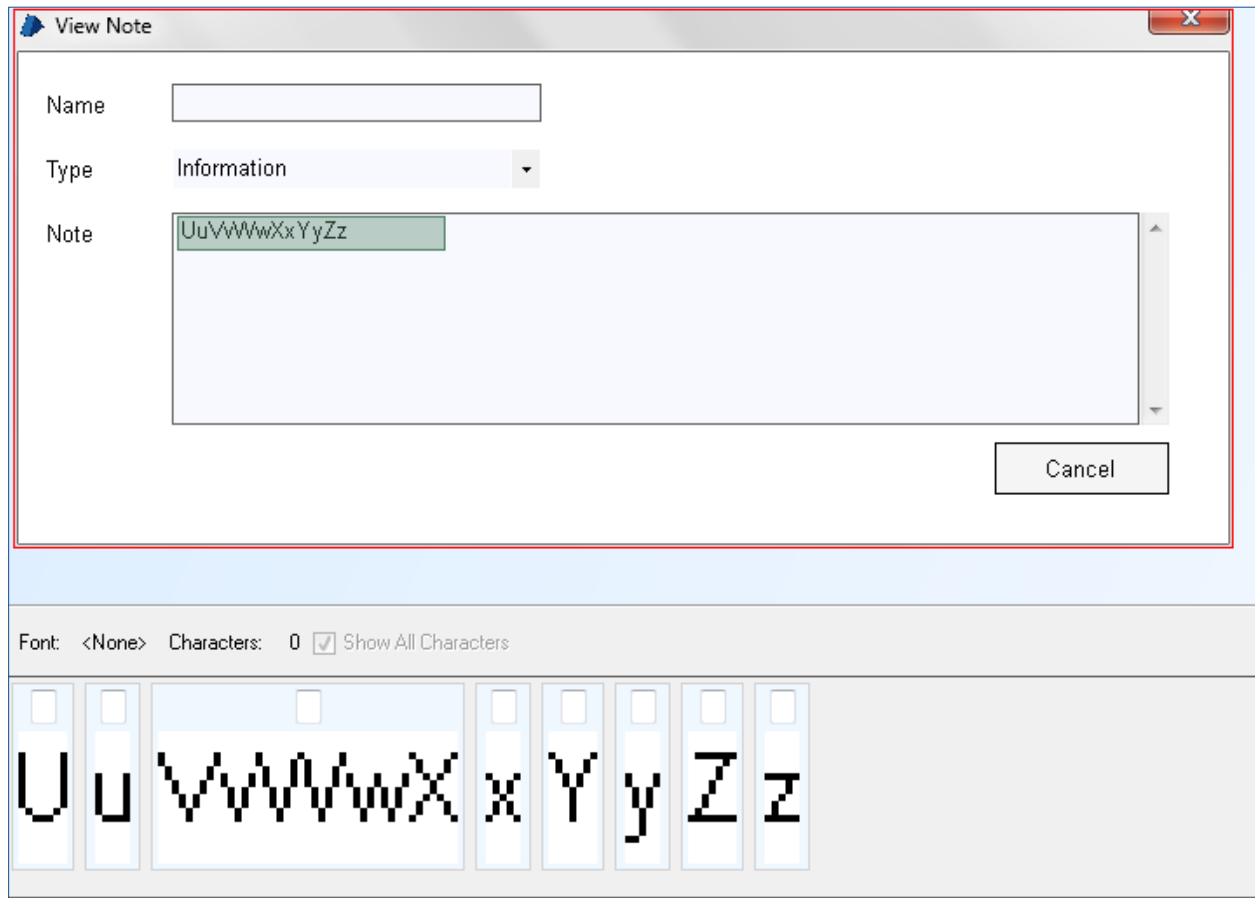


We have created a region to wrap some of the text on the application. From the Font property we select **Create Empty Font**. After supplying a name for the font, we click the **Show Characters** button at the foot of the **Region Editor**.

Blue Prism separates the distinct values and presents them to you so that you can set the characters. Above each character is an input box for you to type the character and then click **Merge Into Font**

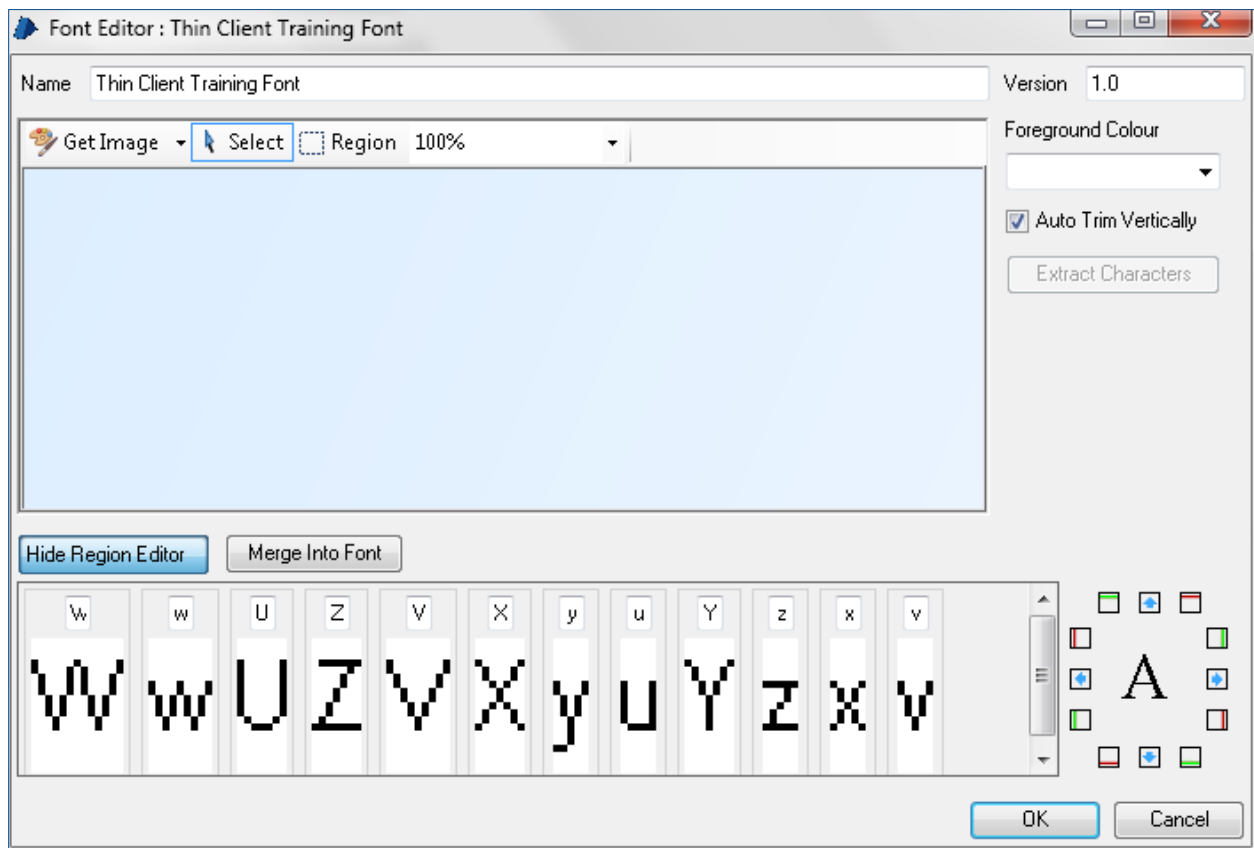


You could repeat the exercise for further characters and symbols. Notice how we put spaces between each character. If we hadn't, the font kerning would have provided the following results:



Blue Prism has been unable to distinguish the VvWwX characters.

Fonts can also be edited in System Manager – System – Fonts as well as in Region Editor.



The interface is similar to region Editor but allows you to select font color and adjust the whitespace around each character.

Font File

Once a font has been qualified, it can be exported to other Blue Prism Resources via a Blue Prism [font file](#). You can export a font to a [font file](#) in System Manager or as part of a Blue Prism Release. This allows your font to be migrated to other Blue Prism Resources.

The Blue Prism font file can also be edited in a text editor if you need to further fine tune your font by setting space widths.

For example if when recognizing text you notice that some words are concatenated this may be due to the space width being too large. In the font file you will find the spacewidth property e.g.,

```
<spacewidth value="4" />
```

Adjusting the value here from 4 to 3 may solve the issue in this example. Once edited, you will need to save the file and import it to Blue Prism via System Manager.

5.2. Character Conflicts

When recognizing text from an application you may occasionally be confronted by character conflicts. The most common one being upper case i and lower case L which may appear the same "l".

Find	Save	Close	New
------	------	-------	-----

Account	Notes
---------	-------

Agent	Date	Note	Type
DOLABELLA	04 May 2006	Caesar, I shall.	i
OCTAVIUS CAESAR	30 Dec 2005	Wherefore is that? and what art thou that darestAppear thus to us?	g
DERCETAS	22 Aug 2005	I am call'd Dercetas;Mark Antony I served, who best was worthyB...	g
OCTAVIUS CAESAR	13 Apr 2005	What is't thou say'st?	g
DERCETAS	28 Nov 2004	I say, O Caesar, Antony is dead.	!
OCTAVIUS CAESAR	13 Jul 2004	The breaking of so great a thing should makeA greater crack: the...	g
DERCETAS	24 Feb 2004	He is dead, Caesar:Not by a public minister of justice,Nor by a hi...	g
OCTAVIUS CAESAR	03 Oct 2003	Look you sad, friends?The gods rebuke me, but it is tidingsTo wa...	i
AGRIPPA	11 May 2003	And strange it is,That nature must compel us to lamentOur most ...	i
MECAENAS	13 Dec 2002	His taints and honoursWaged equal with him.	!
AGRIPPA	13 Jul 2002	A rarer spirit neverDid steer humanity: but you, gods, will give us...	i
MECAENAS	06 Feb 2002	When such a spacious mirror's set before him,He needs must se...	i
OCTAVIUS CAESAR	28 Aug 2001	O Antony! I have follow'd thee to this: but we do lanceDiseases in ...	g

Consider the top note in the above example. In the phrase "I shall" the first letter is different to the last two letters yet is visually identical. Although this is no problem for a human to interpret, for a robot, we need specific rules.

The rules you implement to interpret character conflicts such as this will depend on the context of your character recognition.

In this instance, you could validate and clean the text using the following Regex rules implemented in a Blue Prism Code stage:

Code
<i>I at the start of an upper case word probably I</i>
Unclean = Regex.Replace(Unclean, "(\b)l([A-Z]+)(\b)", "\$1I\$2\$3")
<i>I after upper case at end of a word probably I</i>
Unclean = Regex.Replace(Unclean, "([A-Z])l(\b)", "\$1I\$2")
<i>I after two upper case probably I</i>
Unclean = Regex.Replace(Unclean, "([A-Z]{2})l", "\$1I")
<i>II in a word probably II</i>
Unclean = Regex.Replace(Unclean, "([a-z,A-Z])II", "\$1II")
<i>I after upper case and before small case probably I</i>
Unclean = Regex.Replace(Unclean, "([A-Z])I([a-z])", "\$1I\$2")

I after small case probably I

```
Unclean= Regex.Replace(Unclean, "([a-z])I", "$1I")
```

A single I is probably an I

```
Unclean= Regex.Replace(Unclean, "(\\b)l(\\b)", "$1I$2")
```

There is no hard and fast logic that can be applied and you will need to thoroughly test any logic you implement.

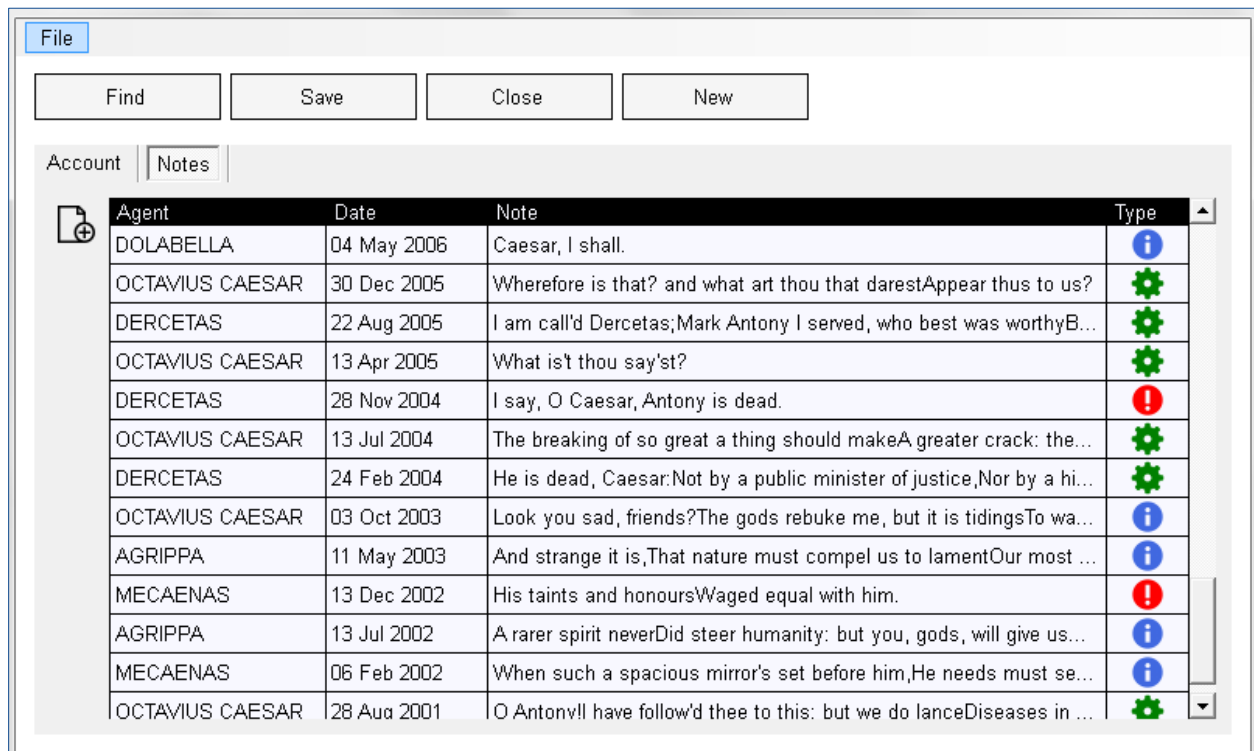
5.3. Performance

Blue Prism Best Practice suggests that before interacting with an element you first confirm that it is the correct element by using image recognition to reference nearby elements such as labels. Searching for images within images as we have learned takes time. If we have 10 fields to complete on a form, searching for labels to confirm the positioning of elements could increase the time taken to complete the form by as much 300%.

Given that it is unlikely that the form will have changed without Blue Prism being alerted by internal change control, you may be wondering how such an overhead can be justified. This is the trade-off between quality and performance and needs to be assessed on a process-by-process, screen-by-screen basis.

Given that you will inevitably have to search for images within the screen especially if you are having to scroll around an application, there are methods you can use to speed up the search.

Suppose we wanted to find the top information icon on the following screen:



We can use the **Get Sub-Image** action in our **Utility – Image Manipulation** object to find the following sub-image with the entire screen as the search area:



The time taken to find the image is approximately 6 seconds.

The **Get Sub-Image** action searches from the top left to the bottom right of the screen making vertical sweeps equal to the width of the sub-image. The search moves the sub-image one pixel at a time then compares each pixel in the sub-image with relative pixels on the screen.

Knowing this, we can improve search performance using the following techniques:

Minimize the background color in the search image

With the way that Blue Prism searches, you can improve performance by minimizing the amount of background color around the top and left of your search image. If we crop our image to create the following search image:



The time taken to find the image is approximately 2 seconds

Reduce search area

Instead of searching the entire screen, you can reduce your search area to where you expect to find the search image before searching the entire screen.

If we reduce our search area to include only the Type column:



Searching for the original uncropped image is reduced from 6 seconds to 0.2 seconds.

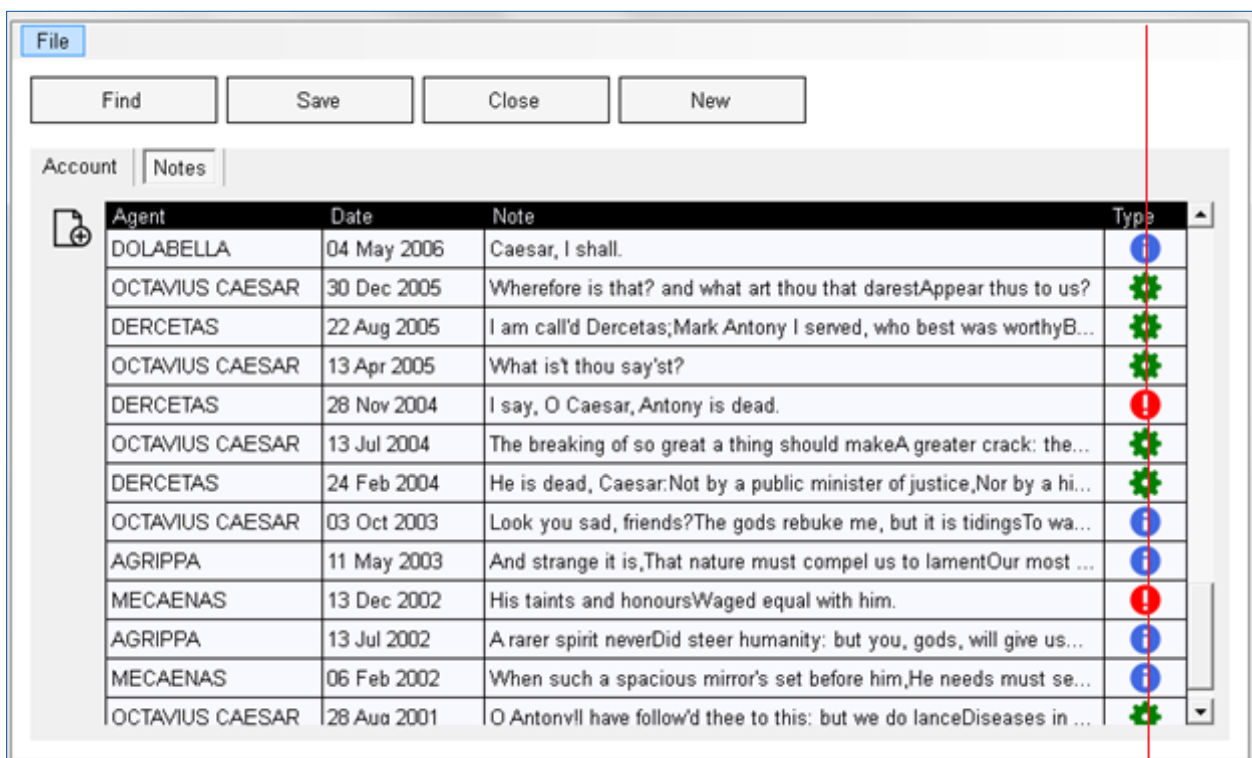
Minimize the size of the search image and the search area

Ultimately we can increase performance to its maximum by searching (initially) for the smallest image within the smallest search area.

In this case we would search for a blue pixel within a search area the height of the screen but one pixel wide.

We can do this by using dynamic regions or by using one of the find pixel actions within the **Utility – Image Manipulation** object.

We are effectively searching down the red line in the image below and checking the color of each pixel until a blue one is found.



Agent	Date	Note	Type
DOLABELLA	04 May 2006	Caesar, I shall.	!
OCTAVIUS CAESAR	30 Dec 2005	Wherefore is that? and what art thou that darestAppear thus to us?	!
DERCETAS	22 Aug 2005	I am call'd Dercetas;Mark Antony I served, who best was worthyB...	!
OCTAVIUS CAESAR	13 Apr 2005	What is't thou say'st?	!
DERCETAS	28 Nov 2004	I say, O Caesar, Antony is dead.	!
OCTAVIUS CAESAR	13 Jul 2004	The breaking of so great a thing should makeA greater crack: the...	!
DERCETAS	24 Feb 2004	He is dead, Caesar:Not by a public minister of justice,Nor by a hi...	!
OCTAVIUS CAESAR	03 Oct 2003	Look you sad, friends?The gods rebuke me, but it is tidingsTo wa...	!
AGRIPPA	11 May 2003	And strange it is,That nature must compel us to lamentOur most ...	!
MECAENAS	13 Dec 2002	His taints and honoursWaged equal with him.	!
AGRIPPA	13 Jul 2002	A rarer spirit neverDid steer humanity: but you, gods, will give us...	!
MECAENAS	06 Feb 2002	When such a spacious mirror's set before him,He needs must se...	!
OCTAVIUS CAESAR	28 Aug 2001	O Antony! I have follow'd thee to this: but we do lanceDiseases in ...	!

The image will be found instantly.

Using a combination of these techniques will enable you to dramatically improve performance when searching for images.

6. Appendix: Object Model Solution

6.1. Object: Thin Client Application – Login

Action	Start/End Screen	Inputs/Outputs
Launch	Start Screen: None End Screen: Log In Screen	Inputs: None Outputs: None
Login	Start Screen: Log In Screen End Screen: Log In Screen (Fail) Account Screen (Success)	Inputs: <ul style="list-style-type: none"> • User Name (text) • Password (password) • System (text) Outputs: None

6.2. Object: Thin Client Application – Account Details

Action	Start/End Screen	Inputs/Outputs
Get Account Details	Start Screen: Account Screen End Screen: Account Screen	Inputs: None Outputs: <ul style="list-style-type: none"> • Account ID (text) • Last Name (text) • Middle Name (text) • Title (text) • Gender (text) • House Number (text) • Street (text) • City (text) • County (text) • Postcode (text) • Verified (flag)
Add Account	Start Screen: Log In Screen End Screen: Log In Screen (Fail) Account Screen (Success)	Inputs: <ul style="list-style-type: none"> • User Name (text) • Password (password) • System (text) Outputs: None
Navigate to Notes	Start Screen: Account Screen	Inputs: None

	End Screen: Notes List Screen	Outputs: None
Navigate to Find	Start Screen: Account Screen	Inputs: None
	End Screen: Find Screen	Outputs: None
Close	Start Screen: Account Screen	Inputs: None
	End Screen: None	Outputs: None

6.3. Object: Thin Client Application – Find Account

Action	Start/End Screen	Inputs/Outputs
Find Account	Start Screen: Find Screen End Screen: Account Screen	Inputs: <ul style="list-style-type: none"> Account ID (text) Last Name (password) First Name (text) Outputs: <ul style="list-style-type: none"> Found (flag)

6.4. Object: Thin Client Application – Notes List

Action	Start/End Screen	Inputs/Outputs
Get Notes	Start Screen: Notes List Screen End Screen: Account Screen	Inputs: <ul style="list-style-type: none"> Start Date (date) End Date (date) Type (text) Outputs: <ul style="list-style-type: none"> Notes (collection) <ul style="list-style-type: none"> Agent (text) Date (date) Note (text) Type (text)
Navigate to Add Note	Start Screen: Notes List Screen End Screen: Note Screen	Inputs: None Outputs: None

6.5. Object: Thin Client Application – Note

Action	Start/End Screen	Inputs/Outputs
Get Note	Start Screen: Note Screen End Screen: Notes List Screen	Inputs: None Outputs: <ul style="list-style-type: none"> Note (text)
Add Note	Start Screen: Note Screen End Screen: Notes List Screen	Inputs: <ul style="list-style-type: none"> Name (text) Type (text) Note (text) Outputs: None

7. Appendix: Sending Special Keys

Use SendKeys to send keystrokes and keystroke combinations to the active application. Each key is represented by one or more characters.

The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses () have special meanings to SendKeys. To specify one of these characters, enclose it within braces ({}). For example, to specify the plus sign, use "{+}". To specify brace characters, use "{{}" and "}}". Brackets ([]) have no special meaning to SendKeys, but you must enclose them in braces.

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes in the following table:

Key	Code
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}

F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}
Keypad add	{ADD}
Keypad subtract	{SUBTRACT}
Keypad multiply	{MULTIPLY}
Keypad divide	{DIVIDE}

To specify keys combined with any combination of the SHIFT, CTRL, and ALT keys, precede the key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

To specify that any combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down SHIFT while E and C are pressed, use "+(EC)". To specify to hold down SHIFT while E is pressed, followed by C without SHIFT, use "+EC".

To specify repeating keys, use the form {key number}. You must put a space between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times; {h 10} means press H 10 times.

If Send Keys does not work with your application, Send Key Events should be tried instead (see the next Appendix).

8. Appendix: Send Key Events

Send Key Events differs from Send Keys in that it uses a lower level method of sending the keys, and therefore is more likely to work with some applications. A specific example of this is a Citrix client, which will not respond to Send Keys but will work with Send Key Events.

Send Key Events sends all normal characters, e.g., "a", "9", just by sending that key in a similar way to Send Keys. A key down followed by a key up of that keyboard character is sent.

Special keys are enclosed in {braces}. This allows things such as {HOME} and {ALT} to be sent.

The < and > characters are used to modify the NEXT key to be just a key down or key up respectively.

Since "{", "}", "<" and ">" are special characters, to send these characters not as special characters they must be enclosed in braces themselves, as follows: {{ }}, {<} and {>}

Examples:

"hello{RETURN}" - sends HELLO and presses return.

"<{SHIFT}a>{SHIFT}" – sends an upper case A.

"<{CTRL}A>{CTRL}" - presses the CTRL key down, then presses A, then releases the CTRL key

"<{ALT}AB>{ALT}" - presses the ALT key down, then presses A, then B, then releases the ALT key - i.e., does ALT-A, ALT-B

"<{ALT}A>{ALT}B" - presses the ALT key down, then presses A, then releases the ALT key and pressed B - i.e., does ALT-A, B

"<{CTRL}<{SHIFT}{ESCAPE}>{SHIFT}>{CTRL}" - presses CTRL and Shift, then Esc, then lets go of CTRL and Shift"

"<{CTRL}<{ESCAPE}E>{ESCAPE}>{CTRL}" – sends the Windows key (CTRL and Esc)

Care should always be taken with the Syntax of Send Key Events. A down keystroke for special keys must always be followed by an up keystroke.