WITHYOU
WITHME

RPA Blue Prism Developer Course

Written Guide

Contents

Intent

This document is designed to augment your online learning experience and provide you with the opportunity to review and revise the content that has been discussed during the individual lessons of the course

Feedback

Please contact your WithYouWithMe RPA instructor if you have feedback on this document or any of your WYWM courseware.

Best Practice

Overview

Hi and welcome to the lesson where we will be discussing best practice when building processes and objects in Blue Prism.

In this lesson we'll be taking a step back and considering everything we've discussed so far in the course, but we'll consider the incorporation of best practice and how it applies to what we're going to be doing in Blue Prism.

Design Fundamentals

As we know, Blue Prism is a platform for automating processes and executing them in a secure virtual environment. We also know that this means our automated processes should be able to run unattended and unobserved.

Often due to complexity and time constraints, building a process might require multiple developers working concurrently on the automation solution. We've also identified that due to the number of cases, multiple resources may work on the same list of cases in the control room.

All this means that it's important to learn and reinforce some of the design fundamentals to ensure that we produce robots that are scalable in terms of structure and configuration.

Before we get started on best practices, all of the content we'll be discussing can also be found on the Blue Prism portal, in particular the process and object templates that are available in the materials tab of this lesson. You'll also need to refer to this guide in preparation for the Blue Prism Developer certification exam.

There are a number of steps you can take to ensure that best practices are incorporated into our processes and objects as well as in the application modeller.

Application Modeller

Let's first consider the application modeller. Element nesting is an important part of best practice in the object studio and it ensures that all associated elements are 'nested' under the correct parent element, maintaining the correct parent and child relationships. This makes it easier to identify elements within an object, particularly if multiple developers are working on the same body of work. Proper element nesting practices will also make troubleshooting a lot easier, for example, if one child element is hard to find, maybe try another child of the same parent element?

We touched on this during the object lesson, but a window element should be a parent element to associated child elements within that window. The next window should then be a separate element on the same hierarchical level, and all the elements within this window should be associated child elements. This is parent/child relationship is also how Blue Prism 'sees' elements within an application.

Element naming conventions should also be considered when assigning parent and child elements. First we identify the type of element we've identified i.e. button. Next we specify the element name i.e. submit and finally we annotate the spy mode we used to identify the element i.e. win 32. You can see another example of this in the adjacent image. Also remember that we have several spy modes that we can use, not just Win 32, so it's important that we annotate the mode we used when labeling our elements.

We also want to make sure that no personal data remains on the application modeller as is the case with the object studio. We need to be particularly cautious of the window text attribute, because if there is a value in window text when we identify that element, it will be visible in the value field of that attribute, potentially causing a data security concern.

Business Objects

Let's next look at best practices for business objects. Firstly, each action within an object should have what we call a conditional wait stage at the start of the flow to ensure that the correct element exists before we look to do something with it.

We covered this briefly during the object lessons but it's reiterated here to articulate its importance, and again in the process templated that you can find in the Blue Prism portal and in the materials tab of this course.

As we know, actions should only be called from the process layer, so within business objects, despite the action stage being available, it's important that we don't create actions that call on other actions with other business objects.

It's also important to make use of note stages in both our object and process studio as well as the description fields in our stages. This ensures that there is good visibility across all our stages, objects and processes and they can be picked up by another developer and continuity is ensured.

Exceptions

We've discusses exceptions at length earlier in the course and as we know, it's best practice to avoid exception handling on the object layer. In the business object we only want to generate an exception using and exception stage that will label the exception, then we allow it to bubble up to the process layer where we catch the exception with our recovery stage and blocks.

We also need to make sure that when setting up our exception stages we put informative details in our exception type and exception details sections so the user or process controller, which may not be you, has an understanding of what error has occurred and can take effective steps to start fault finding the error.

Data Items

When considering data items, it’s important to group them together based on the part of the process they pertain to. This will make them easier to find and will make more sense to an incoming developer in the event the developer who designed the process solution is no longer available.

We can also use block stages around data items to highlight them on the process page. If you’re feeling particularly creative you can also change the colour of a block by highlighting it and selecting a fill colour. This can further delineate data items based on their relevance to difference parts of our process and has the added benefit of being aesthetically pleasing.

Finally, it’s important that we don’t “hard code” variables like usernames and passwords into a process or object. It’s best practice to store variables in data items and fill these data items with environmental variables or inputs as relevant.

Scalability, Recoverability and Usability

Overview

Hi and welcome to the lesson where we will be discussing scalability, recoverability and reusability in Blue Prism.

In this lesson we'll continue to step back and consider everything we've discussed so far in the course as we did the best practice lesson, but this time we'll build on our knowledge of Blue Prism and discuss the concept of scalability, recoverability and reusability when building bots.

Scalability

A common oversight when designing an automated process is not considering that the process will need to run concurrently on multiple machines or resources. Although Work Queues prevent multiple process instances from accessing the same queue item, the steps outside the case working loop may need special attention if the process is to be run on more than one resource.

For example, let's consider a process that starts by consuming a daily input file and then loading a work queue. If three instances of the process are running concurrently what will happen? Will all three instances try and read the file and load the queue at the same time?

Well they would if we didn't manage our process to plan for the use of multiple resources. In the case we just illustrated, we could use an environmental lock.

Environmental Locks

Environmental locks are a feature that enable a permission to be shared between processes and objects. Process instances can be made to compete for permission to ensure that only one instance can perform a particular step.

For example, imagine that the first step of a process is to read a file and load a queue. Three instances start our process at the same time and all then "compete" for possession of an environmental lock.

There can only be one winner and this means that only one process will be able to load a queue, therefore stopping the of duplication of work by having three processes loading a queue with the same data simultaneously.

Once the environmental lock has been released the 'loser' instance are able to access the queue in this case and proceed with the processing of the data in the queue.

In order to use environmental locks you will need to import the environment locking VBO.

To understand more about environmental locks open Process Template 1 - Basic and have a play around with applying them within a process.

Multi Object Design

Another important factor of scalability is considering the use of single use or multi object design. When we use the term, single object design, we're referring to the design idea that a single object is created specifically for a particular application. A single object design is appropriate for a small scale operation or a proof of concept project, where only a single developer is working on a project.

In the vast majority of cases we should look to employ multi object design which is to build more than one object per application.

Although there are no hard and fast rules on how many objects may be required per application, a good rule to work by is to build a single object for each screen of the application that's being automated.

This design is more scalable and efficient, particularly when considering more complex applications, and mitigates the risks that we see with single object design.

Ultimately we should look to employ multi object design if the object we're building is going to be used in the production environment or where more than one developer will be building an object or process in Blue Prism.

Recoverability

Recoverability is the capacity of the solution to handle problems and return to normality within our process. It is unrealistic to assume that our process will always remain on the happy path, so we should always include the provision of recovery logic.

In terms of system recovery, we have learnt about exception handling earlier in the course, but it's important to summarise from a system recovery perspective.

When applications aren't behaving as expected and control has been lost, the error from an object and resulting exception can be managed with exception handling, but the applications must still be brought back under control at some stage if the process is to be continued at some point.

On some applications this can be easy, for example on a web application it might be a case of navigating directly to the home URL, however in some cases navigating back is not so simple and it may be necessary to restart the application.

Each application must be careful assessed in anticipation of including a recoverability section in the solution, and although system recovery is not explicitly captured in the solution design document or the process design instruction the developer will need to appreciate the concept when configuring robust solutions.

If we then move on to case recovery, this is where we describe the mechanisms that must be in place to retry cases after exceptions have been created.

We've touched on this previously during the retrys lesson but it's important to note the difference between system and business exceptions, and in particular those that we want to retry and those that we don't.

It's important to note that we only want to retry system exceptions, because retrying business exceptions will likely return the same result over and over again, not through any fault of Blue Prism or our application. The WYWM RPA Analyst course elaborates more on business and system exceptions if you need to refresh your knowledge in this key area.

Reusability

Finally let's discuss reusability. Reusability is a key tenant of RPA solution design that we've touched on throughout this course. By thoughtfully designing objects we can create a library of reusable logic that can be built up and called upon across multiple processes.

This will reduce the effort and maintenance overheads and to make our objects reusable we need to ensure that they are free of business process logic and object pages (or actions) should be small and execute discrete singular process tasks where possible.

An example of this was in our notepad object we had a single page that launched notepad. Because there was no other logic within this object, we are able to reuse this action in a multitude of other processes that may start by opening the notepad application.

Troubleshooting and Common Faults

Overview

Hi and welcome to the lesson where we discuss troubleshooting.

In this lesson we'll build on our knowledge of Blue Prism and we'll discuss the key elements of fault finding which are an essential skillset of the RPA Developer in industry and they'll also come in handy when it comes time to do the final assignment of this course.

What is Troubleshooting?

Firstly, why is troubleshooting so important? Well troubleshooting is the systematic process you use to solve your problem when you are stuck or your bot is producing an error. So it's important that we understand how to troubleshoot, not only for when we encounter issues during the build phase, but also when maintaining bots that have already been deployed.

First let's discuss some helpful functionality that we refer to as a breakpoint.

Breakpoints

Pausing our process can be handy, particularly for troubleshooting, but for very large processes, it could take quite some time to reach a particular step and if we don't pause at exactly the right time we could miss a critical piece of information.

Therefore, a better way of making sure we stop a process at a particular point is by using what we call a breakpoint. We use breakpoints to pause a process at a certain point. We can do this by right clicking the stage that we'd like our process to stop at and selecting breakpoint. This will cause the process to stop when it gets to this stage and the stage will be highlighted with a red rectangle.

As we mentioned, stopping our process at a particular point will allow us to review our data items and other stages at a particular point. Reviewing a data item's current values will often allow us to understand why a process may not be functioning as we want them to.

It's worth noting that breakpoints will only take effect when we're running this process in the process studio as we're doing here. If this process was being run from our control room with the process studio closed, our process would ignore this breakpoint and continue on. We'll also learn more about the control room and the production environment in later lessons.

Syntax Errors

We'll now discuss some common faults, the first of which is syntax errors. In Blue Prism we know that it's always preferable to drag and drop values into expression windows but as we also know this is not always possible. In the event that we want to extract some information from a collection that has not yet been populated, we're unable to drag that particular value into the relevant expression window because it doesn't exist yet and so in this case we need to type the identifying syntax manually.

This is however prone to errors and as we discussed in earlier lessons, capitalisation, spelling and any other typographical errors including missing inverted commas around text data will induce an error. For this reason it's essential to review all syntax issues before we finalise our work and make sure that we cross check where possible. As a general rule this is also a good place to start looking if an error is produced that relates to collections.

Hidden Collections

Another place to look if your process can't find a data item or collection is the hide from other pages check box. Unfortunately this box will be ticked as a default and if left ticked this data item or collection will not be visible to other stages on any pages other than the page where it's located.

What's the Process Actually Doing?

A final point that relates to the collections, data items, write stages and any other part of our process or object where data or a value is being moved from one place to another.

A great place to start when troubleshooting an issue that relates to the movement of a value or data is to understand what each individual stage is doing and asking 'what am i expecting this stage to do'? It's important that you carry out this little activity before Blue Prism reached the stage so you can compare what you expected to happen against what actually happened.

Step Functionality

This can however be difficult if we're using the play button and watching the process flow through with limited control. For this reason a preferred option is stepping through a process using the step function.

The step function allows us to step through each stage of our process and actions within our objects. By stepping we're able to see exactly what steps are being carried out by Blue Prism and where errors are being produced.

Further, we can also step over and step out of process pages and action stages if we don't have a need to fault find a particularly part of our process.

Stepping functionality can also be used after a breakpoint for particularly long processes where stepping would be unnecessarily time consuming. For example, if I wanted to troubleshoot a particular part of a long and complicated process that was after several pages of processing, I could use a breakpoint to stop my process at a particular point and once that breakpoint was reached I could use the step functionality to slowly step through the steps I'd like to troubleshoot.

Reviewing Errors

Review of identified errors using the error symbols is also an essential part of fault finding and errors should be referred to prior to running a process. A simple rule to apply in the case of error messages is that they should always be investigated. Clicking the error symbol will open an error window that will provide some basic details about the error and the stage (and page if relevant) that the error relates to.

If we click on the error line Blue Prism will take us to the stage on the page that the error has occurred.

It is often the case that we will have errors as we are building our processes or objects, however errors should not be common place when our process is complete and they should certainly not be present when we put our process into production.

Google

Finally let's discuss Google. Google is a fantastic resource that will answer the vast majority of your questions. Simply typing questions into google and using Blue Prism as either the prefix or suffix will typically direct you to online discussion forums that are closely related to the issue you're encountering.

It's very rare that any issue you're experiencing is being encountered for the first time and using resources like google and discussion boards can save you time and resources in faultfinding. Written and video advice can also be very useful but be careful downloading content like home-made IBOs and VBOs from unknown sources.

Advanced Exception Handling

Overview

Hi and welcome to the lesson where we discuss advanced exception handling.

In this lesson we'll build on our our discussion of exceptions and cover some more advanced concepts that relate to exception handling and dealing with anomalies within our processes or applications.

Review of Exception Handling

When discussing exception handling it's important to remember that it is a key part of an RPA developer's job in industry because you only need to build a bot once, but once that bot's deployed you need to be continually maintaining it to ensure it still providing the same operational capability in what is generally a changing environment.

To recap on the concepts we've covered in earlier lessons, as a general rule, we use exception stages to label exceptions and we use recovery logic, that includes recover and resume stages, to get our process back on its feet and in a position where it can either retry the part of the process that produced the error, or terminate the process if required.

We also learnt that exceptions can bubble up from the object layer to the process layer and also from a subordinate process page to the main page in line with the picture seen on the slide.

Finally, we also explored the concept of retry logic and how we incorporate retry logic for particular exceptions. The concept of retrys also incorporated the use of circular paths to count the number of attempts we've made at a particular step, before we use an exception stage to 'throw' an exception.

Now that we've rehashed the basic let's discuss some of the pitfalls associated with exception handling, firstly is the infinite rethrow loop.

Infinite Rethrow Loop

An infinite Re-throw loop is when exception handling logic throws an exception back to itself and falls into into a recurring cycle of throw, recover, throw recover etc etc, and this will only be resolved by the intervention of a developer. When a controlled rethrow occurs in the control room we can simple terminaste the process and investigate further, however if a rethrow loop occurs when we're in the process studio we rely on the pause button to stop our process from continuing on a never ending throw/recovery loop. In the event the pause button is inaccessible, generally because the loop is running too fast, then there may be no other way to stop our process other than using the task manager to kill the process. As you can imagine, if we haven't saved our work when we encounter an infinite rethrow loop this will quickly become very frustrating.

Because exception throwing and recovery logic is not physically linked, it is by nature more difficult to conceptualise, however we can avoid suffering the fate of an infinity rethrow loop by using blocks throughout our process.

Infinite Retry Loop

A similar trap when considering retry logic is an infinity retry loop. An infinity retry loop is when exception retry logic doesn't have a way of breaking out of the loop and so it continues to loop around forever. We've seen so far in the course that we can use a counter to add to a data item and when a predetermined number of loops is reached, the process exits the loop and generally throws an exception. It's important to make sure this logic is function correctly or we will find ourselves in an infinite retry loop and our process will need us to intervene or it will loop around forever.

Nested Exception Handling

We also want to avoid a concept called nested exception handling. Nested exception handling typically occurs when we try and have a block within a block as you can see. As we know, problems occur when we have an exception during recovery mode and Blur Prism therefore has no concept of nested recovery modes, the process is either in recovery mode or it isnt and block therefore should never be nested together or overlapping.

Just as we wanted to avoid generating an exception while in recovery mode using nested blocks, we also want to avoid it by throwing an exception when in recovery mode. If we want to generate a new exception that has a more informative message than what was provided by default, then the correct method would be to use a recover stage, followed by a recover stage, then throw another exception as required.

Advanced Work Queues

Overview

Hi and welcome to the lesson where we discuss advanced Work Queue Management.

In this lesson we'll build on our our discussion of Work Queues and cover some more advanced concepts that relate to the operation of the work queue and how they're employed in industry, particularly from the perspective of the process controller.

Review or Work Queues

Firstly let's review what we've learned about wok queues buy starting with the basic layout of work queues and the control room.

First you'll remember that we can select a work queue from the queue management tab within our control room and by clicking on the work queue we want to look at we can see the queue contents in the button half of the screen.

We also learnt that we can create a queue from the system tab by clicking the work queues item within the workflow heading. Once we've selected work queues we can input the work queue name, the key name, which denotes the displayed name on our work queue page, as well as max attempts, whether the work queue is encrypted or not,

Tags

Just as we used key name in the settings tab to choose what is displayed in the work queue, we can also add tags to items within a queue to make them easier to identify in much the same was as we marked exceptions and marked compete.

In short, a tag is a keyword or term assigned to an item within a work queue that categorises or groups that item. The primary reason we would look to do this would be to easily view or report on work queue items with a specific tag, or to provide a process more control over how work queue items are locked or worked.

Some examples could be to tag a line item with a customer tag that associates a specific order with a customer, we could note the way a process was completed (in the event there were multiple ways a process could be completed) or we could identify the computer name if an item needs to know the robot that added it to the work queue.

Good news is that adding tags to a data line within a work queue is as easy as marking an item as complete or marking it as an exception. We simply use the Work Queues IBO and select Tag item. Once selected we'll be prompted to provide the item ID (As was the case with mark complete) as well as the tag we'd like to provide in an expression window.

We can also untag items in exactly the same way should our process require a line item be untagged. If we're untagging items we simply put the tag that we'd like to remove in the tag field of our input section and this will removed the tag that we've specified.

Adding tags to a work queue item

The tag feature can be particularly useful when we're getting the next item from our work queue especially if we want to process some categories of data item before others. Let's say that we had a customer that weren't paying their invoice and we wanted to make sure that we didn't process their order. We could tag all the orders associated with this company as 'Bad Credit". Then when it comes time to run our orders process we could add some extra details to our Get Next item Action stage to make sure that all orders that were tagged as having bad credit weren't processed.

In this case we if we provide the value "-Bad Credit" in the tag field, making sure to use inverted commas, our get next item action stage would pull the next item from the work queue, providing it didn't have the tag Bad Credit.

Similarly, if for some reason we only wanted to process the items that were tagged as bad credit, we would need to put "+Bad Credit" in the Tag filter and then our get next item action stage would only pull items tagged as bad credit from the work queue.

We can also set multiple filters if multiple tags are being used, in exactly the same way, we just need to seperate each filter with a semi colon.

So, given the example -Bad Credit;+Priority order, what items would be called from the work queue?

We'll good question you might say... The answer would be that the action stage will only pull items from that work queue that are not tagged with Bad Credit but are tagged with priority order.

Filtering Items with tags

One final point on Tags. We can also view and filter our tags in the queue management tab of the control room. At first glance the tag filter looks like a text filter but it works a little differently. We can use the tags filter to search for items that have specific tags or those that do not have specific tags in exactly the same way that we just discussed using the + and - values.

If I wanted to review a work queue and see only the orders that didn't have bad credit, I could type "Bad Credit" and prefix it with a minus and this would provide a list of all the line items in a work queue that hadn't been tagged with bad credit.

Setting Priority of Work Queue Items

Another function within the work queue to set the priority of line items within a queue. By default, the order in which the next item is selected from a work queue is based on the first in first out principle, meaning the items are pulled from the queue in the same order that they were loaded, however because this is only the default we can modify the priority of line items as required.

When selecting the next item in a queue based on priority, Blue Prism will select the lowest number first so items with a priority of 3 will be selected before items with a priority of 7. If items have the same priority number the first in first out rule will apply.

It's worth noting that Because the priority is an optional parameter if no priority is selected the priority of an item will default to 0.

We can assign a priority in much the same way as we did when we considered tags. When using the Add to Queue within the Work Queues IBO we can input a priority that applies to the entire queue being uploaded.

We can also use the set priority action within the work queues IBO to assign priorities to individual line items and in this case we need to also provide the item ID of that line item in the inputs section.

Encrypting Work Queues

Finally, let's expand on what we learnt about encrypting work queues in the earlier lessons. As a bit of background, the data held by each line item is stored in plain text on the database as a default, however our queue can be configured to encrypt this data automatically when it is saved to the queue and decrypted when it is received from the queue.

As we know, we can enable this capability by checking the encrypted check box when we create or manage a work queue in the system tab. Once we check the encrypted box a drop down list is available which allows us to select the key we're using to encrypt. There's not too much more to it than that.

If the selected key can't be found then it will be appended with unresolved key and any work or operations within this queue will likely fail.

It's also worth noting that only the item data is encrypted. If items already exist in the queue before it is set to encrypted, only new items will be encrypted when added to the queue. If encryption is turned off after an item has been encrypted that item will remain encrypted, although get next item action stages will continue to work, they will just be pulling encrypted item data from the work queue, which dependant on the process may or many not result in a process error.