

Blue Prism Learning

Exception Handling

Training Guide

Revision History

| Date | Revision | Author | Description |
|------------|----------|--------|---|
| 09/01/2014 | 0.1 | JC | Draft document created |
| 20/02/2014 | 1.0 | JC | Revision following peer review |
| 04/04/2014 | 1.1 | JC | Revision following peer review |
| 08/07/2014 | 1.2 | CS | Updated Legal Terms |
| 22/01/2015 | 1.3 | DD | Aligned with the Foundation Course, info that used to be in the Config Guidelines, and our Process Templates. Removed all mention of Components – as they are a separate advanced topic. |

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.

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 | 5 |
| 1.1 | Exception Cases | 5 |
| 1.2 | Visualising Exception Cases | 5 |
| 1.3 | Application Problems | 6 |
| | Exercise 1.3.1 Inducing an Exception | 6 |
| | Exercise 1.3.2 Handling an Exception | 7 |
| | Exercise 1.3.3 Exception Loop | 8 |
| | Exercise 1.3.4 Isolating Exception Handling | 8 |
| | Exercise 1.3.5 Avoiding an Exception | 10 |
| 1.4 | Business Rules | 10 |
| 1.5 | Types of Exception..... | 12 |
| | Exercise 1.5.1 Handling a Business Rule..... | 15 |
| | Exercise 1.5.2 Recovery Mode..... | 18 |
| | Exercise 1.5.3 Accessing Exception Detail | 18 |
| | Exercise 1.5.4 Misusing Exception Functions | 20 |
| 2 | Exception Handling Strategies..... | 21 |
| 2.1 | Retrying | 22 |
| 2.2 | Preserve the type and detail of the current exception | 24 |
| | Exercise 2.2.1 Using 'Preserve the type and detail of the current exception' | 24 |
| | Exercise 2.2.2 Using Resume | 25 |
| 2.3 | Casting..... | 26 |
| | Exercise 2.3.1 Exception handling and casting | 26 |
| 2.4 | Business Objects | 28 |
| 2.5 | Sub-Pages | 33 |
| 2.6 | Processes | 35 |
| 2.7 | Queue Item Results | 37 |
| 3 | Exception Handling Pitfalls | 42 |
| 3.1 | Infinite Re-throw Loop | 42 |
| 3.2 | Infinite Retry Loop..... | 42 |
| 3.3 | Nested Retries | 42 |
| 3.4 | Nesting Exception Handling | 42 |
| 3.5 | Generating another exception without resuming..... | 43 |
| 3.6 | Stepping Past the Resume Stage | 44 |
| 3.7 | Misleading Exception Detail | 44 |

| | | |
|----------|--------------------------------|-----------|
| 3.8 | Missing Exception Detail | 45 |
| 4 | Human Intervention..... | 46 |
| 4.1 | Diagram | 47 |
| 5 | Process Templates..... | 50 |

1 Introduction

This module is intended to supplement the Blue Prism Foundation Training course and is aimed at students who have completed the course and are beginning to put their education into practice.

1.1 Exception Cases

You will recall from the Foundation Training course that Blue Prism uses *work queues* to manage its workload.

- A work queue is a list of cases to be worked by a process.
- A process will iterate through the queue working cases in sequence.
- After working a case a process will update the queue with a result.
- Queue data is used to measure process performance and for generating MI (Management Information).

A process is unlikely to be able to complete every single case and some will be identified as *exception cases*.

1.2 Visualising Exception Cases

It may help to visualise exception cases by thinking of a Blue Prism process simply as a machine – a machine used to do a specific job over and over again.

Without concerning ourselves how this machine works, we can see that cases are fed into one end of the machine and results come out at the other end.

And in an ideal world this machine would be able to do all its work without encountering any difficulties – every case fed into the machine would be completed.

Unfortunately the reality is that the machine will occasionally encounter problems and some cases will not make it all the way through to completion.

These unfinished cases will require the attention of a human being.



1.3 Application Problems

You will recall from the Foundation Training that rather than failing in the event of an exception, Blue Prism can be configured to cope with, or **handle** the exception and try to keep going.

Exception handling is a critical part of any Blue Prism solution and should be designed with the same level of care afforded to the other parts of a solution. Focusing only on the 'happy path' is not sufficient – the 'unhappy path' must also be considered.

Although not every type of exception is recoverable, and sometimes it makes no sense to attempt a recovery, the ability to handle exceptions at least provides us with the opportunity to decide what to do when processing does not perform as expected and the logical flow has strayed from the ideal path.

Most exception handling should be done at process (or component) level. Business objects can contain exception handling but in general they should be kept as simple, reusable pieces of logic. In the following exercises we will look at a scenario where exception handling is commonly used at business object level - when we launch or attach to an application.

Exercise 1.3.1 Inducing an Exception

In this exercise we'll create a business object that cannot handle a problem with an application.

- Create a new object based on the Notepad application.
- Create a new page called **Attach (or Launch)**
- Add a Navigate stage to attach to **notepad.exe**
- Without launching Notepad, run the page and confirm that the following exception message appears.

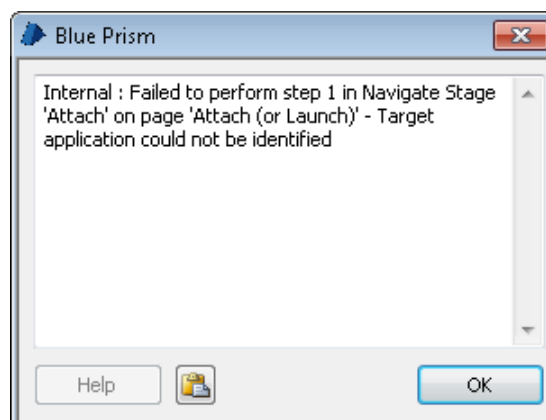


Figure 1 Target application could not be identified

- 'Target application could not be identified' is the business object telling you that it can't find the application to attach to. And in most cases we can assume this is because the application is not running.

Exercise 1.3.2 Handling an Exception

In its current state the business object generates an exception when it tries to attach to a non-existent Notepad. Remember that if left unchecked, an exception will **bubble** upwards towards the main page of the parent process, ultimately bringing the process to a stop.

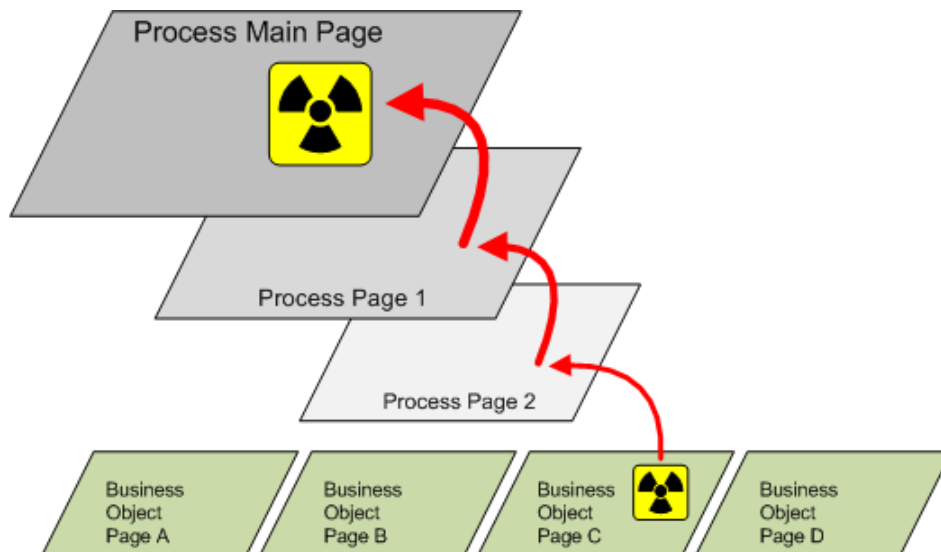
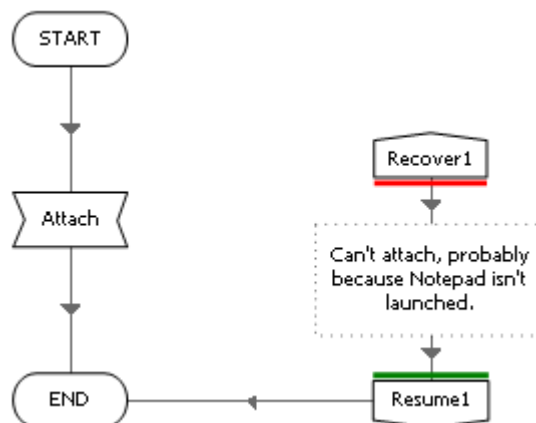


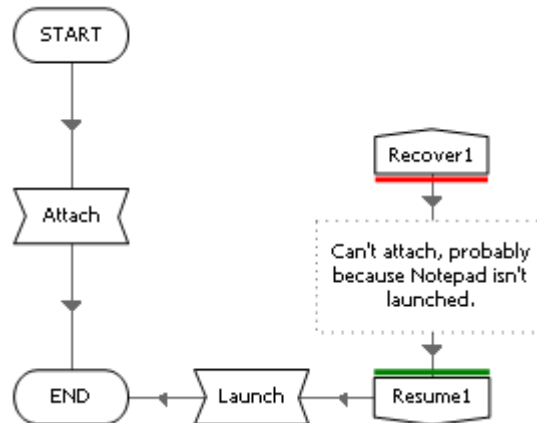
Figure 2 Exception bubbling


In this next exercise we'll handle the exception to prevent this from happening. And because we know (or at least can reasonably assume) that the exception can be interpreted as 'Notepad is not launched', we can add additional logic to remedy the situation.

- Add Recover and Resume stages to **handle** the exception, with your diagram looking like this (NB the note stage isn't necessary).



- Run the page again to confirm the exception handling prevents the message from appearing. This puts the object back on the 'happy path' but does not do anything to improve the situation.
- Add another Navigate to launch Notepad after the Resume stage, like this.



- Run the page and confirm that Notepad is launched.
- Press the Detach  button and run the page again. Now because Notepad is already launched the attach will work.

Exercise 1.3.3 Exception Loop

Run the page again (slowly) to see that although the exception handling works at first, the diagram falls into a loop when it tries to launch Notepad. What do you think is happening?

- The attach failed initially because the business object was already attached, and a business object cannot attach more than once.
- This exception was handled but then the launch failed because Notepad was already running, and once attached, a business object cannot launch again.

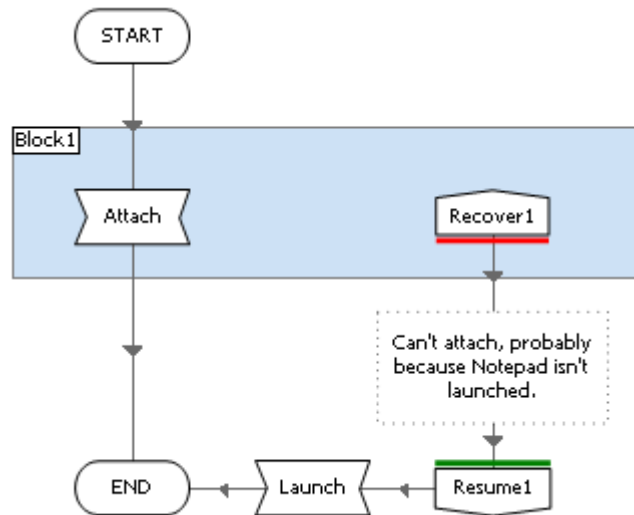
Key Points

- * Launching an application attaches the business object automatically.
- * Once a business object has launched an application, any attempt to launch again will cause an exception.
- * Once a business object has become attached to an application, any attempt to attach again will cause an exception.

Exercise 1.3.4 Isolating Exception Handling

The exception handling itself is now causing a problem, in that the Recover stage is attracting exceptions from the launch point and sending the diagram into a tailspin. A way to remedy this problem is to *isolate* the source of the exception using a *Block*.

- Add a Block around the first Navigate stage and the Recover, making the diagram look like this.



- Run the page again to confirm that the infinite loop does not occur. The launch exception is no longer handled because the Recover stage is now only concerned with exceptions occurring *inside* the Block. You should now see an exception message at the launch stage.

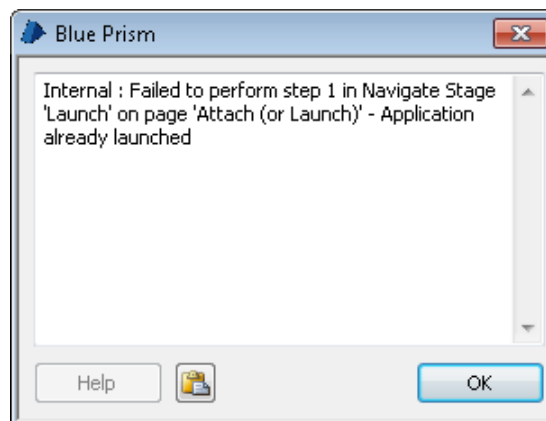


Figure 3 Application already launched

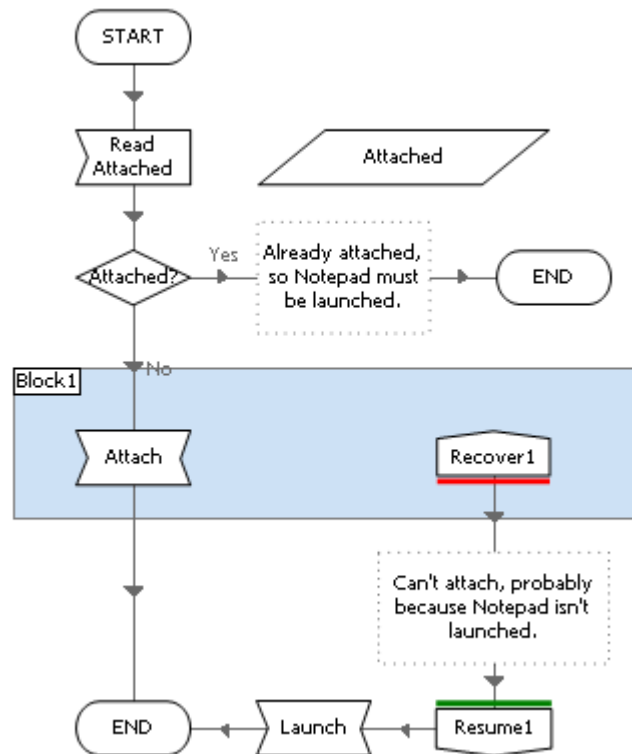
Key Points

- * Although very useful, exception handling can itself cause problems if not well laid out.
- * By default a Recover stage will attract any exception occurring on its page, and this can sometimes lead to an infinite loop.
- * A Block is a mechanism for isolating exception handling to a specific area and is a good way to prevent an infinite loop.

Exercise 1.3.5 Avoiding an Exception

Sometimes we can make checks to avoid scenarios where an exception might occur. In this example we know we can't attach or launch more than once, so to finish off our page we shall add in some logic to see if we are already attached.

- Add in a Read stage, a Decision and an End stage before the first Navigate so that the page looks like this.

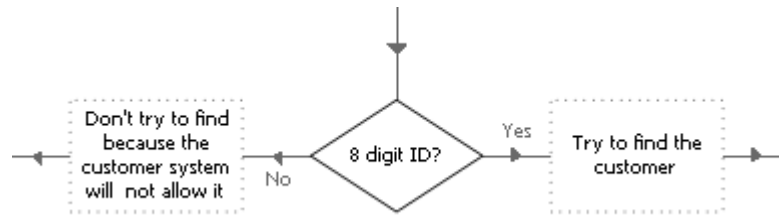


- Use the Read stage to set a Data Item called Attached, and edit the Decision expression to use the Data Item to divert the flow away from the Navigate stage.
- ★ *Tip: Use the root (top) element in the application model and select the 'Is Connected' read operation.*

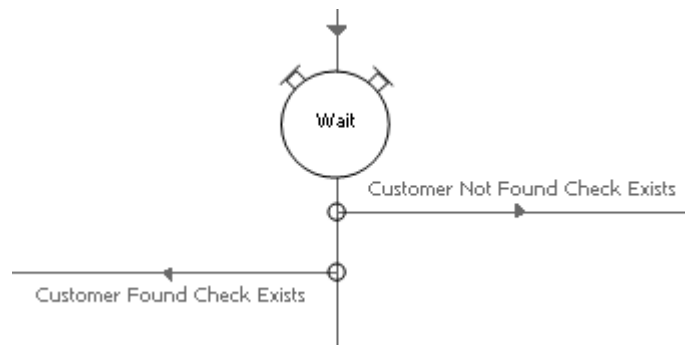
1.4 Business Rules

The previous section looked at handling an exception caused by the behaviour of an application, namely when Notepad is not in an ideal state for attaching and launching. This is an example of how Blue Prism can be made to react to the *situation* it finds itself in and take corrective measures, i.e. 'the attach failed, so assume Notepad isn't running and proceed to launch'.

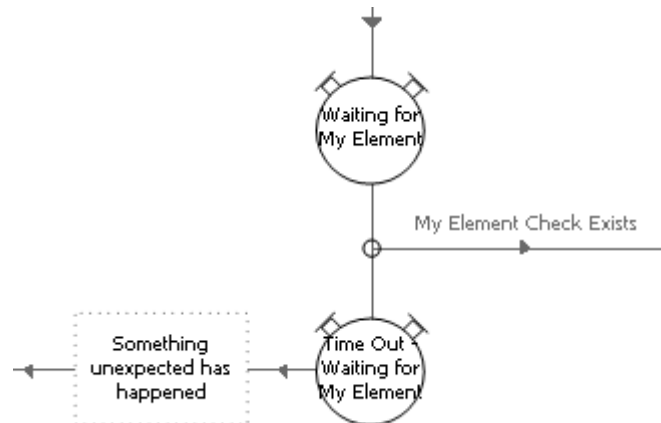
Blue Prism can also be made to react to the *data* it is provided with. For example, if the first step in a procedure was to find a customer record using an 8 digit ID, it would make no sense for Blue Prism to attempt the procedure if it has (mistakenly) been given only 6 digits. It would be better if Blue Prism contained a rule that said 'check the ID is 8 digits before attempting to find the customer'.



This is an example of validation logic that can be used to filter out 'bad' cases – ones that Blue Prism should not attempt to complete. Similarly, if the ID was 8 digits but the customer system displayed a 'record does not exist' message, Blue Prism would be unable to complete the case.



These types of exception, based on 'business rules', should be seen as different from a technical error. When a case is not completed due to such logic it should not be seen as a problem – Blue Prism is behaving according to the rules. However, a case that did not flout any business logic but could not be completed should be seen as an error.



1.5 Types of Exception

So the term 'exception' is used to describe a problem Blue Prism encounters while it is running, but not every exception should be thought of as 'bad'. Blue Prism enables us to invent as many types of exception we want and normally we use at least two.

System Exception

We know that Blue Prism is used to automate other applications, and as such is dependent on the behaviour of these applications. Again, in a perfect world these applications would always work faultlessly, never crashing, freezing, stalling, running slowly or doing anything they weren't supposed to do.

If this was the case, then in theory a faultless Blue Prism process could be constructed. But sadly this theory isn't reality, and we can and should expect some issues with the applications we will be automating.

A term we use in Blue Prism for this kind of application-based problem is *system exception*.

Business Exception

As well as technical issues with our applications, it is not impossible that the cases fed into Blue Prism contain some sort of problem that makes it impossible for the case to be worked.

Similarly, a process may be set up to deliberately disregard certain types of case. For example, if a process was designed to only work the cases of adult patients, it may contain a rule to check the date of birth and deem juvenile cases as 'out of scope'.

A broad term we use for this kind of rules-based rejection is *business exception*.

Key Points

- ★ Exceptions can have different types.
- ★ Not all exceptions are bad; some can represent 'out of scope' cases.

Other Types of Exception

There is no restriction on how you choose to classify exceptions and you may want to use terms other than 'System' and 'Business'. However, Blue Prism recommends that the number of exception types used is kept to a minimum to ensure ease of understanding and support.

Blue Prism recommends the other potential exception types:

| Exception Type | Description |
|----------------------|---|
| Validation Exception | The data input into the process has been validated and failed (i.e. sort code 5 digits instead of 6). |

| | |
|-------------------------------------|---|
| <p>System Exception Try Once</p> | <p>The same as any other system exception but marked as one that should not be retried.</p> <p>If an exception occurs on an update stage in an action, i.e. a Click Confirm Button stage, Blue Prism may have no way of knowing if the update was done or not (i.e. the system becomes unresponsive) and should therefore not retry the update.</p> <p>Such an exception must be reported to be manually investigated and not re-attempted.</p> |
| <p>Login System Exception</p> | <p>Occurs if an exception occurs attempting to log into a system.</p> <p>When a login system exception occurs, no work queue items should be marked with an exception (any current case should be put back into the queue – pending and untagged). Support staff should be notified by email or alert.</p> <p>If logic has been specified in the SDD and Security Policy documents to allow automated password changes, the exception detail should be interrogated for the text 'Password Change Required' and a Password Change component used.</p> |
| <p>System Unavailable Exception</p> | <p>If a System Exception or Internal exception occurs the process should check, if possible, that the required systems are still available.</p> <p>If a system is not available, it is a System Unavailable Exception and no work queue items should be marked with an exception (any current case should be put back into the queue – pending and untagged). Support staff should be notified of this type of exception by email or alert. The process should keep attempting to start/use the system periodically until it is available again.</p> |

However you decide to label exceptions, it is important to differentiate between exceptions for 'in scope' cases and 'out of scope' cases. As mentioned above, if a process was designed to ignore cases for juvenile patients, these sorts of exceptions should not be seen as errors. The

cases are not complete, but they have been worked correctly. If this distinction is overlooked, MI reports may give an overly negative impression of the performance of the process.

As mentioned previously, the types of exception used, what they mean and how they will be handled should be considered an integral feature of any Blue Prism solution.

Internal Exception

'Internal' is the type of exception that isn't generated by an exception stage, and we have already seen an example in the previous exercise where we first attempted to attach to Notepad. In simple terms, the internal exception is used by Blue Prism to say 'there is a problem and I can't do this part'.

So the exception from our earlier exercise, 'target application could not be identified', is Blue Prism saying 'I can't find Notepad'. You may also recall some of these internal exception messages from your training.

| Internal Exception Message | Interpretation |
|---|---|
| Already connected to an application | The business object is already attached and cannot be launched or attached again. |
| Not Connected | The business object is unattached and must be attached before it can perform any other actions. |
| Unable to match any windows with the query terms | The element of the application model can't be spied anymore. |
| Syntax error | An expression containing an error has been found. |
| Missing data | A data item is being referred to in an input field or expression incorrectly. |
| Stage xxx does not exist | A data item referred to in an output or 'store in' field cannot be found. |
| The business object xxx does not support the action yyy | A business object page has been renamed (or unpublished or deleted) but the action stage is still using the old name. |
| Failed to find stage linked from stage xxx | There is a link missing and dead end was reached. |
| Decision did not result in a yes/no answer | The result of an expression in a decision was not a Flag data type (i.e. True or False). |

| | |
|--|--|
| Cannot perform + operation when the left-hand value is empty | A number data item has no current value. |
|--|--|

This isn't a complete list of internal exceptions and no doubt you will come across others. The important thing to remember is that the internal type is an exception Blue Prism itself generates, and all other types are generated by you using the exception stage.

Exercise 1.5.1 Handling a Business Rule

In this exercise we'll create a process that can handle 'dirty data', i.e. cases that cannot be completed because they contain incomplete or invalid information.

- Create a new work queue with **Patient ID** as the key field.
- In a new process create a collection containing the fields **Patient ID** (data type text) and **Date of Birth** (data type date).
- Populate about 10 rows with data. Use 8 digits for most Patient IDs but also use 6 and 7 digit numbers on some rows too. For Date of Birth use mainly dates for an adult but also use some 'under 18' dates on some rows.
- Use the **Internal – Work Queues** object to add new items to the queue. Re-run the process a few times so you have plenty of new items in the queue.
- Modify the process to use the **Get Next Item** and **Mark Completed** actions, so it looks like the diagram below.

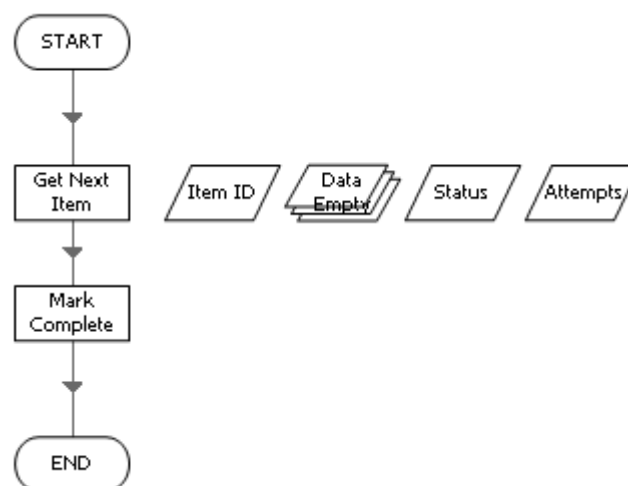
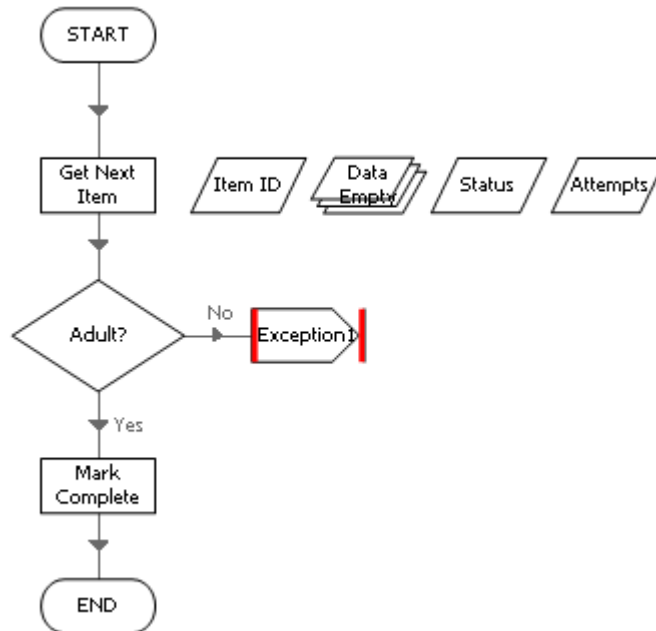


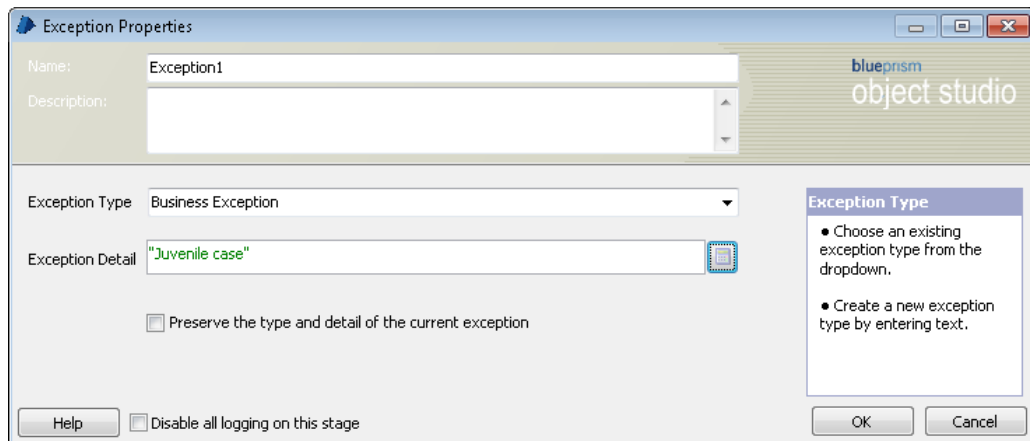
Figure 4 Mark Complete

- Run the process and then look in Control Room to confirm that one item in the queue has been completed.

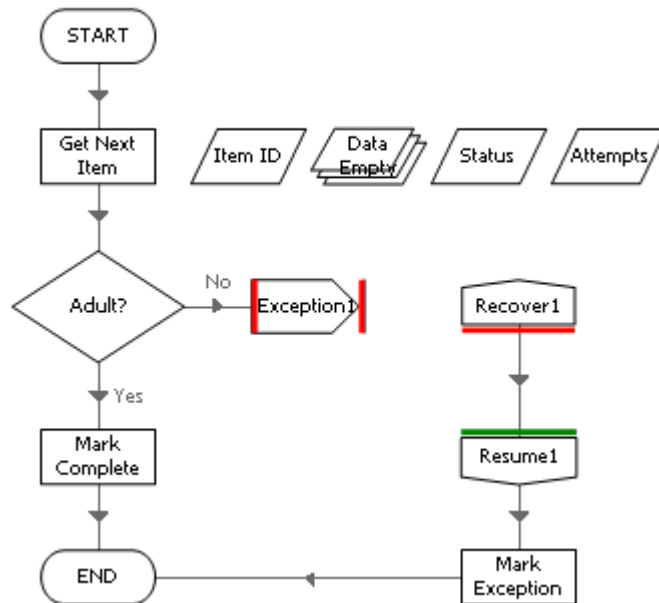
- Introduce a Decision stage to check the value of Date of Birth. If the patient is an adult, mark the case complete, otherwise throw an exception. Your diagram should look something like this.



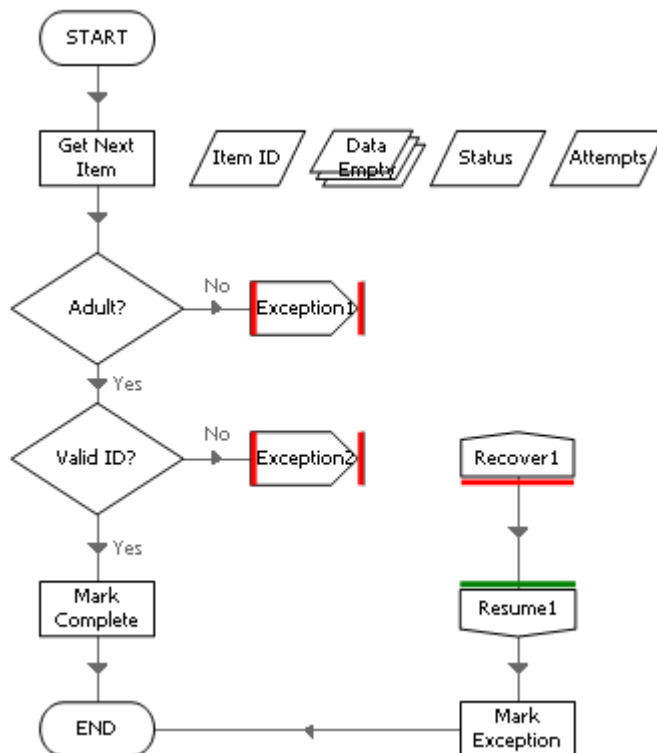
- Open the properties of the exception stage and set Exception Type to **Business Exception** and Exception Detail to **"Juvenile Case"** (note the quotes – this is an expression).



- Add Recover and Resume stages to handle an exception and use the **Mark Exception** action from the **Internal – Work Queues** object to update the queue with the expression **"My exception reason"**.



- Re-run the process until you get an exception case. The go back to Control Room to confirm the queue has been updated with the exception reason.
- Add another decision to check that the Patient ID is valid, i.e. 8 digits. Run the process again but see that the same exception message is generated every time.

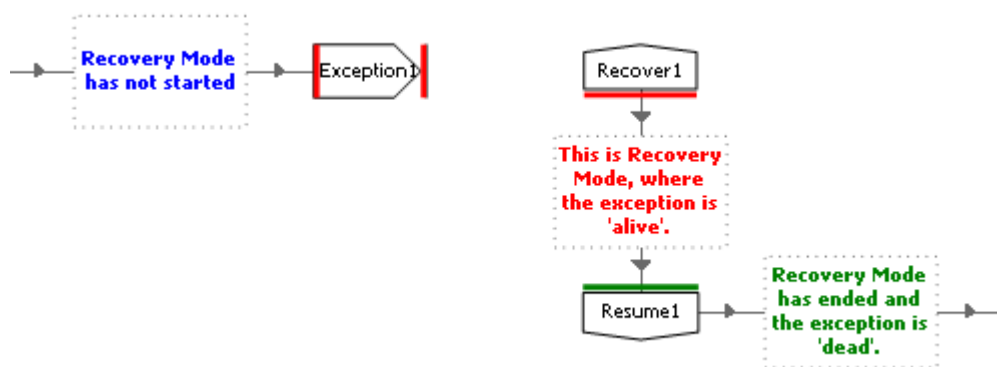


This is because we are using a fixed value for the Exception Reason input parameter of the Mark Exception stage - regardless of the nature of the exception we are naively just saying 'My exception reason' each time.

Obviously this isn't ideal, and there is a better way to update the queue by using information extracted from the exception during what is known as **Recovery Mode**.

Exercise 1.5.2 Recovery Mode

As we have seen, the Recover stage attracts exceptions rather like a magnet. Once the flow of a process (or business object) has reached a Recover stage it is said to be in **Recovery Mode**.

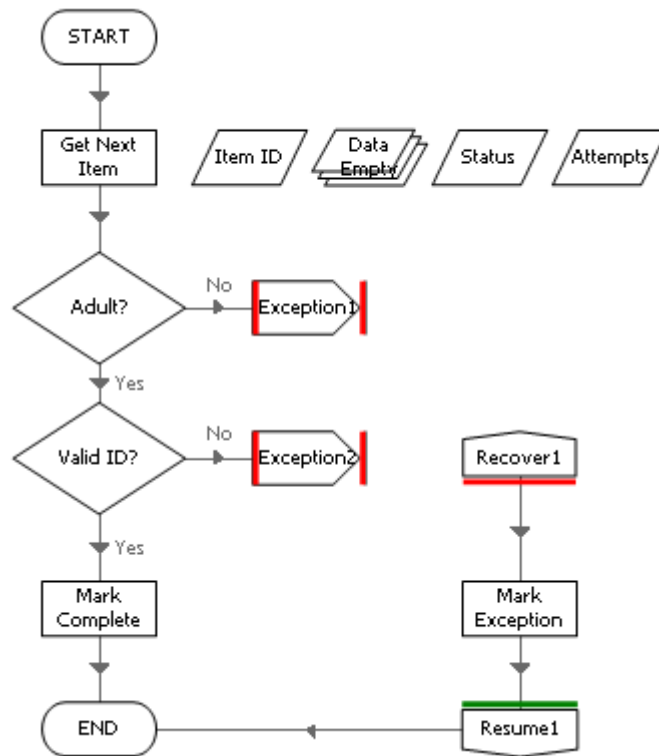


The Resume stage neutralises, or kills off, the exception and moves the process out of Recovery Mode. Once past the Recover stage, the process is back into 'normal' running mode.

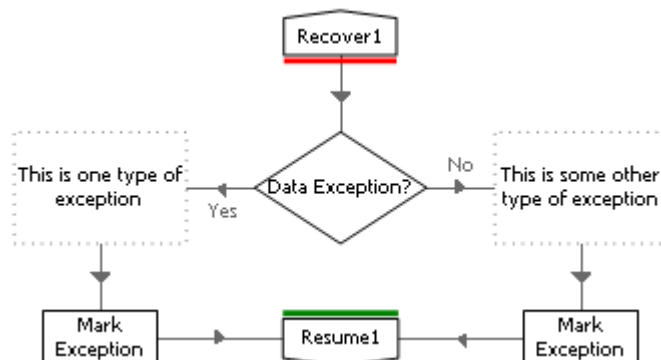
Exercise 1.5.3 Accessing Exception Detail

Another way to improve exception handling is to inspect or examine an exception and then make some sort of decision based on its detail. Blue Prism provides functions for use as part of an expression for doing just that.

- Return to your exception handling diagram from the previous exercise and open the properties of each Exception stage. Make sure they have **"Invalid ID"** and **"Juvenile Case"** in their respective Exception Detail fields.
- Now move the Mark Exception stage in between the Recover and Resume stage, i.e. to a Recovery Mode position.
- Amend the expression used for the Exception Reason input parameter to be **ExceptionDetail()**.

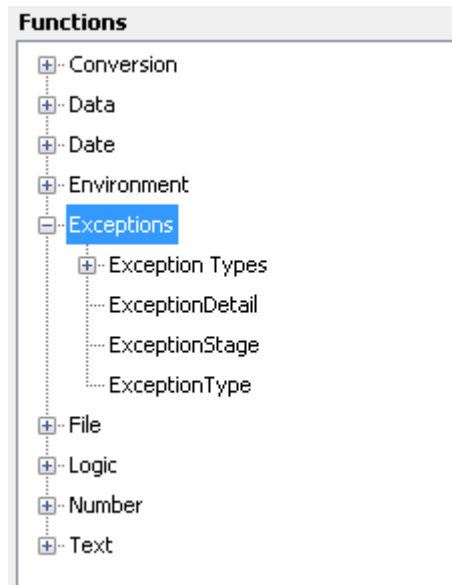


- Re-run the process and look at the work queue. You should see that the terms 'Invalid ID' and 'Juvenile Case' are now being used to update the queue.
- Change the Exception stage thrown after the '8 digit' decision so that it's Exception Type is **Data Exception**
- Add a new decision with the expression **ExceptionType()="Data Exception"** just after the Recover stage. Then make a second copy of the Mark Exception stage and link the decision to each one, like this.



- Notice how the exception handling flows one way or the other, depending on the type of exception thrown. Although this doesn't look too impressive at first, you should be able to see the potential of having a Recover/Resume arrangement that can react differently according to the exception it has caught.

Note that the expression functions (found in the properties forms where an expression is required) must only be used during Recovery Mode and will themselves generate exceptions if they are used anywhere else.



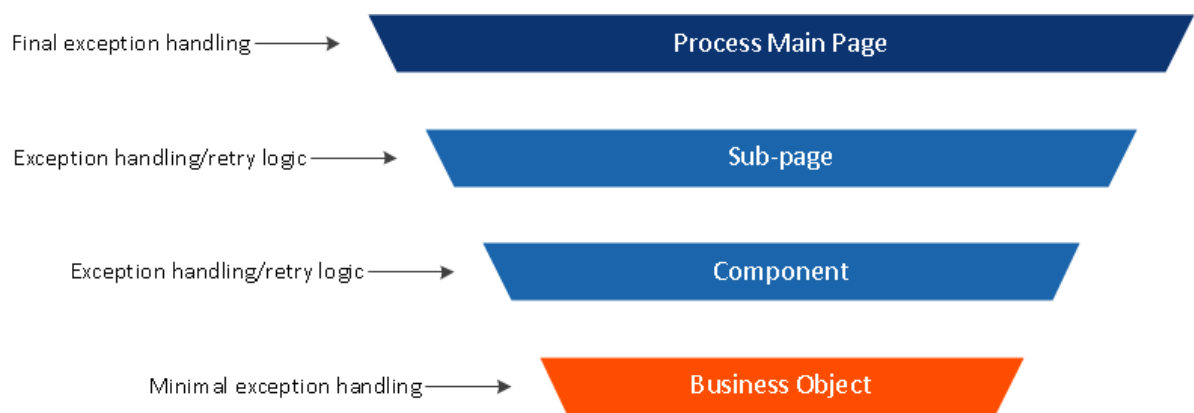
Exercise 1.5.4 Misusing Exception Functions

- To confirm how expression functions should not be used, introduce a deliberate error into your diagram by using one of the functions outside recovery mode.

2 Exception Handling Strategies

Anticipating and dealing with problems is crucial to a successful Blue Prism solution and an exception handling strategy should be devised at the design stage. Regarding exception handling as an 'extra' to be stuck on just prior to testing is a mistake that can lead to confusing diagrams and a poorly performing solution.

As mentioned earlier, exception handling tends to be done above the business object layer. This simplifies the business object design and avoids adding complexity that might hinder the reusability of the object. The top layer of a solution is the main page of the parent process, and often this page will use a block as part of a 'final' exception handler that protects lower layers and catches exceptions that bubble up.

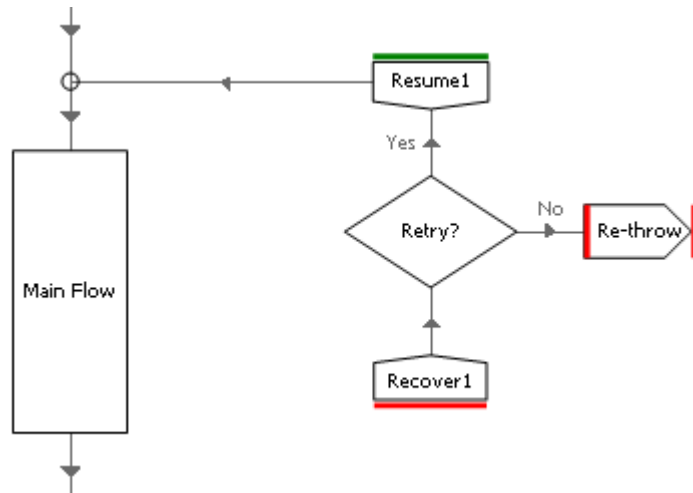


Key Points

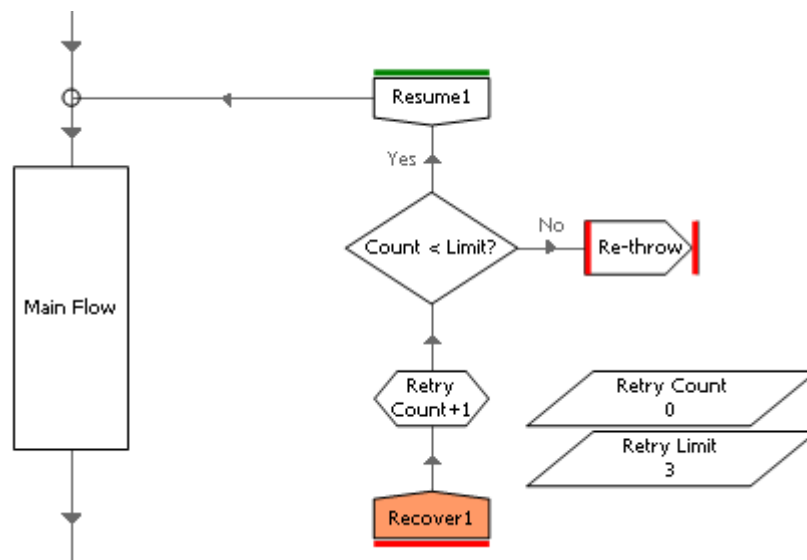
- * Business objects usually contain very little (or no) exception handling.
- * Exception handling and retrying tends to be done on layers above the object layer.
- * The top main page generally uses an exception block to cover the layers beneath it.

2.1 Retrying

Retrying simply means to recover an exception and then steer the flow back into the main part of the diagram in the hope that the problem will be alleviated by another attempt.



The decision whether to retry is normally governed by the exception details and the number of retries that have been attempted. To avoid retrying too many times data items are often used to monitor the number of iterations around the exception handling loop.

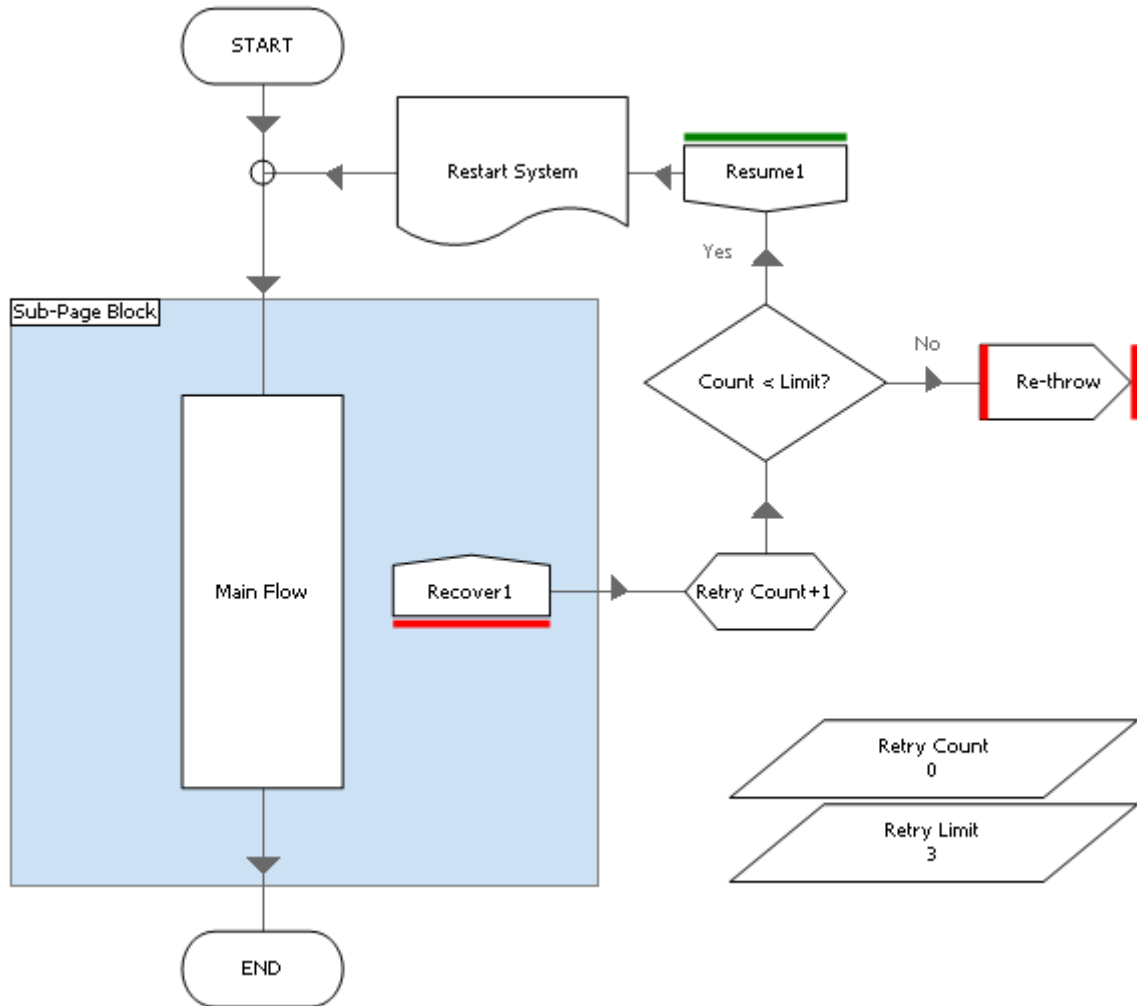


The expression in the Retry? decision in the Blue Prism Process Templates is as follows:

[Retry Count]<[Retry Limit] AND (Lower(ExceptionType())="system exception" OR Lower(ExceptionType())="internal")

This decision ensures the flow only retries if the Retry Count has not been reach and if the exception type is either a System Exception or an Internal Exception. For example, you would not want to retry for Business or Try Once exceptions.

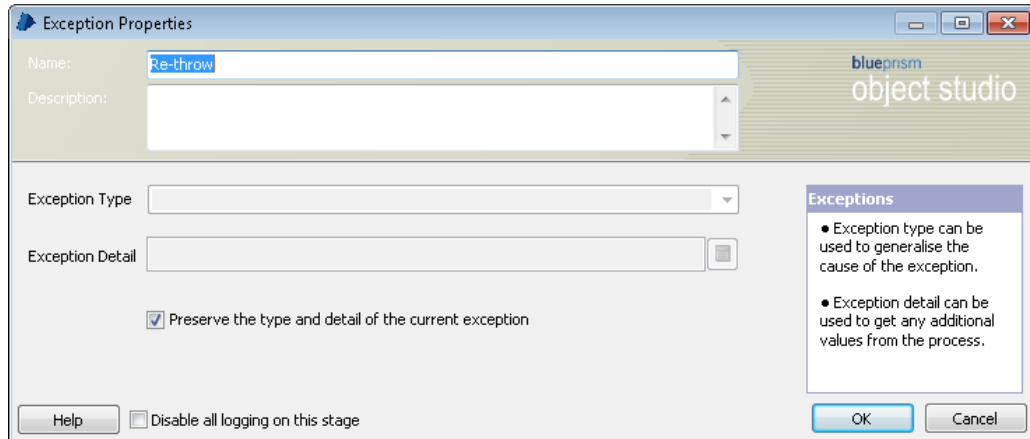
Finally, below we have a retry loop as it looks in the sub-pages of the best practice Blue Prism Process Templates.



After the Resume stage, the flow needs to do something to get the application ready to retry the main flow again. In this instance it is simply calling a page that restarts the application being used. A Block has been added to ensure that an infinite loop cannot occur should an error happen between the Recover and Resume stages.

2.2 Preserve the type and detail of the current exception

When the decision to stop retrying is reached, it is important to understand the role of the 'Preserve the type and detail of the current exception' checkbox.

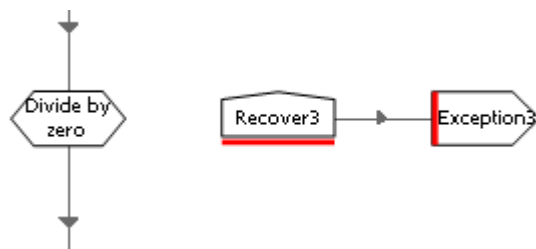


When this checkbox is ticked, the exception stage simple 're-releases' the current exception and allows it to bubble upwards as if it had never been recovered in the first place.

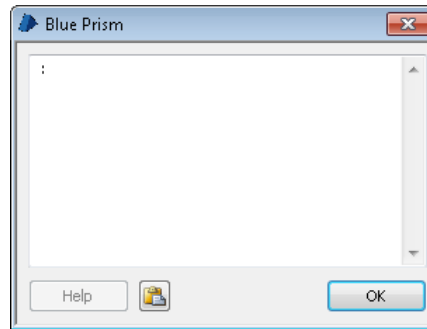
When this checkbox is not ticked, the exception stage generates a *new exception*, and importantly, *keeps the current exception alive*. This means that by misusing the 'preserve' checkbox can lead to problems.

Exercise 2.2.1 Using 'Preserve the type and detail of the current exception'

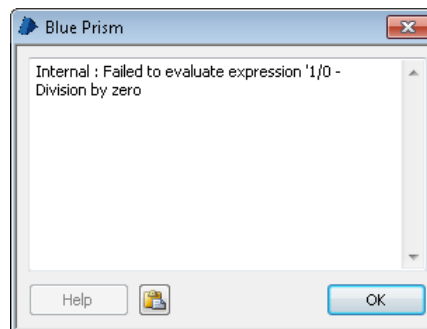
- Recreate the 'divide by zero' problem from the training course.
- Add a Recover and an Exception stage, as shown below. Leave the properties of the Exception stage untouched.



- Run the page and see that this (nearly blank) exception message is displayed.



- Now open the Exception stage properties and tick the 'Preserve..' checkbox. Re-run the page and see that this exception message is displayed.



Without the 'Preserve...' setting the Exception stage generates a new exception. We didn't enter any Exception Type or Exception Detail in the Exception stage properties and that is why the message was (nearly) blank.

Critically, the original exception - the one caught by the Recover stage - is still alive. And although this won't appear to present a problem in this simple example, when this mistake is present in a more complex, multi-layer solution, unhandled 'zombie' exceptions like this will cause problems.

With 'Preserve...' switched on, the exception caught by the recover stage is re-released (or re-thrown) by the exception stage, and that is why the message was 'Division by zero'.

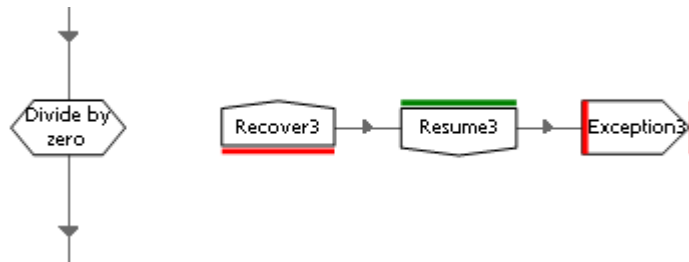
Key Points

- * The current exception is released when the 'Preserve...' checkbox is ticked.
- * A new exception is generated when the 'Preserve...' checkbox is not ticked.

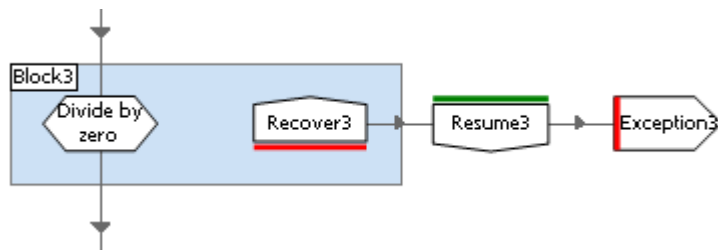
Exercise 2.2.2 Using Resume

The previous exercise demonstrated the use of the Exception stage as part of Recovery Mode, i.e. between a Recover and Resume. Here we will see what happens when the Exception stage is used outside Recovery Mode.

- Change your diagram so that the Exception stage is after the Resume stage.



- Re-run and see that the page falls into an exception loop. This is because the Resume stage has neutralised the original exception and the Exception stage generates a new one, which in turn is caught by the Recover stage.
- Add a block as shown below. See that it prevents the infinite loop by isolating the Recover stage so that it only has responsibility for exceptions originating inside the block.



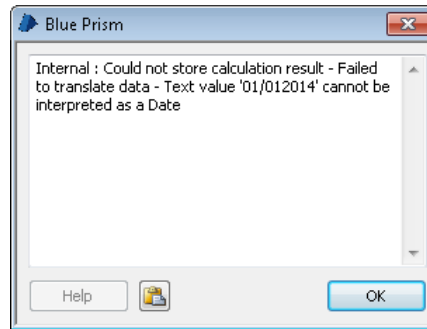
2.3 Casting

Recall from the training course that 'casting' is a means of using one data type to set a different data type. In Blue Prism terms, this means taking one value and using it to populate a data item of another data type, e.g. using the text expression "1234" to populate a number data item.

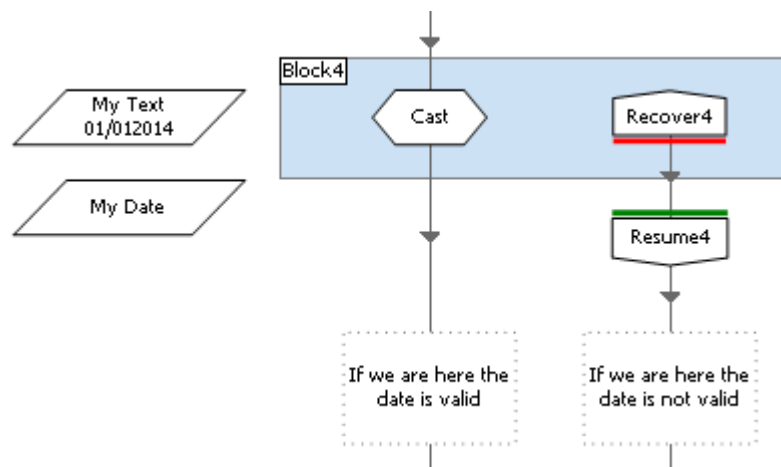
The problem with casting is that it can go wrong if there is no 'fit' between the data types, e.g. "£12.34CR" will not 'fit into' a number data item because of the non-digit characters. Blue Prism does provide a number of functions to test values (e.g. IsNumber) but often exception handling is used to test the compatibility of a value.

Exercise 2.3.1 Exception handling and casting

- Create a text data item called *My Text* with initial value *01/012014* (note the deliberate error).
- Create date data item called *My Date* with no initial value.
- Create a calculation with My Date as the 'Store In' and *[My Text]* as the expression (dismiss any data type complaints the Calculation properties form may present to you).
- Run the page to see the following exception message.



- Add exception handling around the calculation like this.

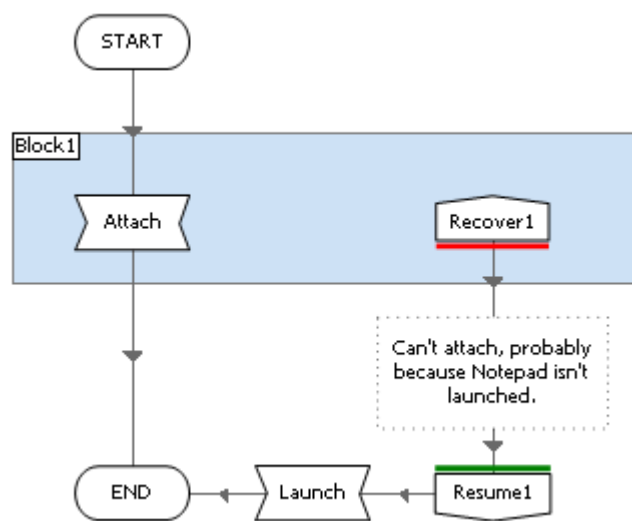


- Run the page and see that the failed cast operation is handled with more control.
- This kind of 'exploratory' casting is often a good way to validate data. Yes, there are other ways to check, but because the 'fit' required for a cast has to be exact, using exception handling like this is often the safest strategy.

2.4 Business Objects

As mentioned above, business objects tend not to have much in the way of exception handling. This isn't a rule as such, but in general business object pages are kept simple (and reusable) and the logic dictating what to do in the event of an exception is left to whoever is using the business object.

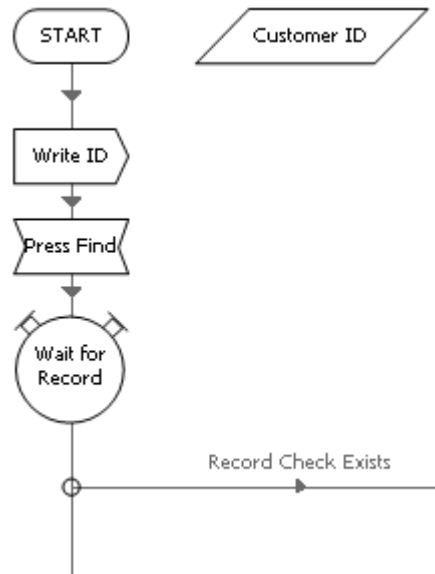
As we saw in the first few exercises, using exception handling around an attaching Navigate stage is often used as a means to detect whether an application is running – if the attach fails, it can be reasonably assumed that the application is not running. Used like this, exception handling can provide the decision whether to attach or launch.



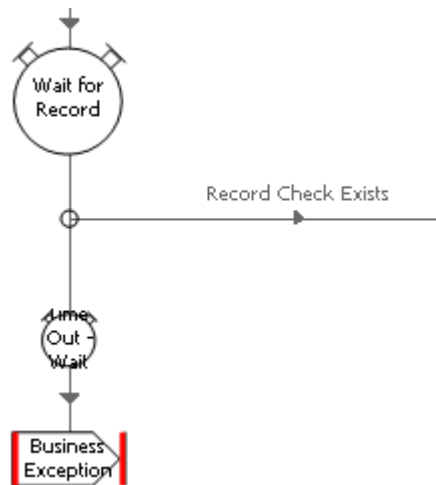
Passing Results as Exceptions

An exception is naturally associated with a negative outcome and we know that it will bubble upwards through the layers of a solution - as we have been discussing, this is how a problem at the application layer can find its way up to the work queue. Knowing this, it is possible to use exceptions to communicate results as well as problems.

Imagine a business object page called 'Find Customer' that has an input parameter called 'Customer ID'. The page will write the ID to a field on the screen, press a button and wait for the customer record to appear.



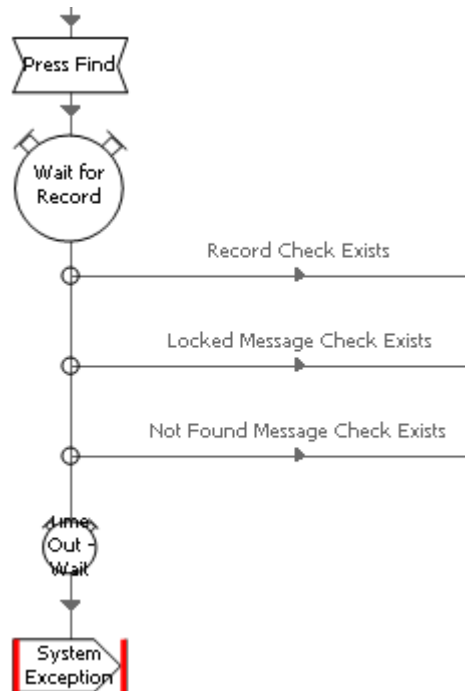
As you know, effort should be made to have business objects anticipate all scenarios, so what should happen if the customer could not be found? A basic solution would simply be to throw an exception from the end of wait stage, like this.



The exception type could be 'Business Exception' and the exception message could be 'Customer Not Found'. But is this correct?

The wait stage checks for the existence of the Record element until the time out expires. At that point, logically all we can be sure of is that the element **could not be found**. However the example above assumes this to mean 'customer does not exist', and this may be incorrect.

A more robust technique would be to have a wait stage check for more than just the 'happy path'. For example, imagine that the application might also display the message 'Record not found' or 'Record is currently locked by another user' when the button was pressed.



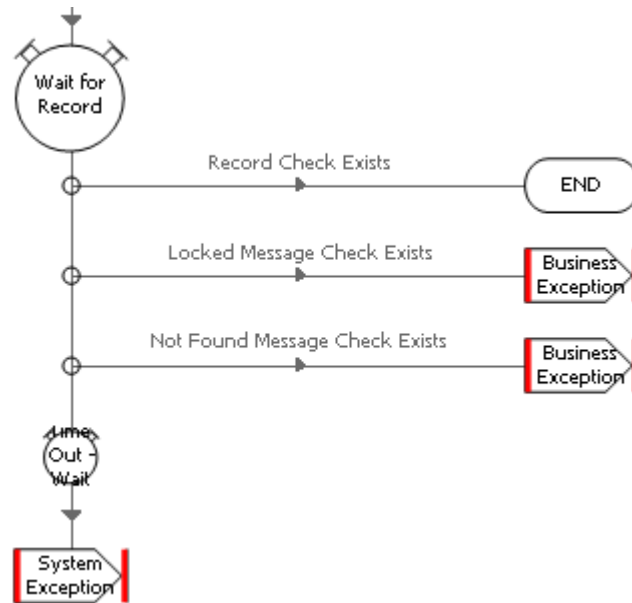
See that in this example a System Exception is thrown after the time out. Rather than inferring a time out to mean 'customer does not exist', here we are simply (and more accurately) saying 'none of the above elements could be found'. Critically, the business object is not passing judgement on what that means, it is merely stating a fact. The decision on what to in that scenario - the exception handling - is left to the user of this business object.

Output Parameters or Exceptions

In the example above there are four outcomes to the Find Customer action.

- Customer found
- Customer not found
- Record locked
- None of the above

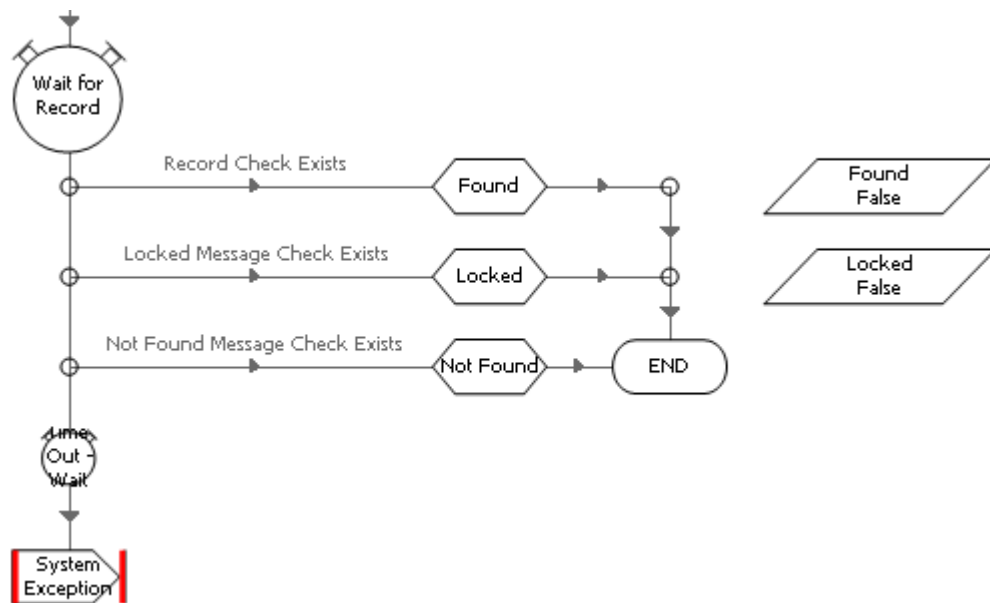
The 'none of the above' outcome is catered for by the System Exception after the time out. But what of the other outcomes? It is perfectly possible to use more exception stages for the other 'negative' outcomes, like this.



Although this idea will work, consideration should be given to the impact it has on the users of the business object - anyone wanting to use this page must be aware of these potential outcomes and make provision for them.

This means that users have to either consult the business object documentation (which in turn must be accurate) or study the diagram to find out about these exceptions. And once they have done that, they must construct exception handling appropriate to their needs.

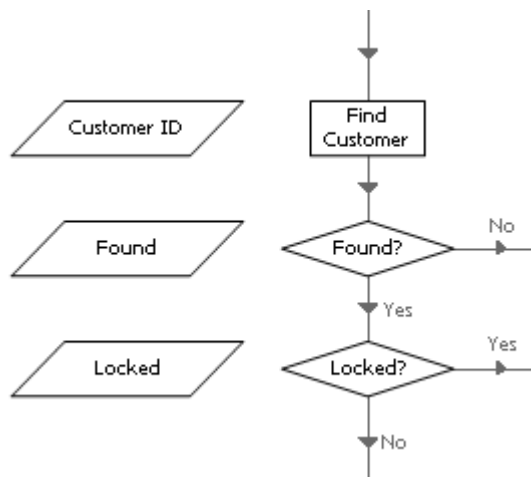
To make the outcomes more visible to users, output parameters could be used instead, maybe like this.



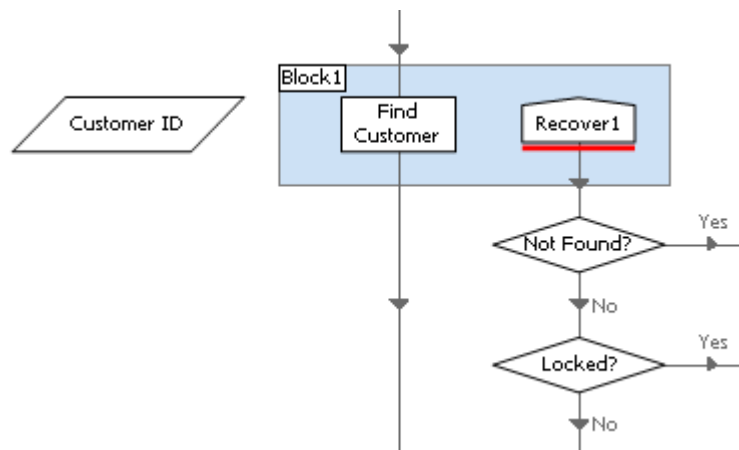
The outputs are clearly visible to the business object user in the properties of an Action stage.

| Inputs | | | Outputs | | | Conditions | | |
|--------|-----------|----------|---------|-----------|----------|------------|-----------|----------|
| Name | Data Type | Store In | Name | Data Type | Store In | Name | Data Type | Store In |
| Found | Flag | Found | | | | | | |
| Locked | Flag | Locked | | | | | | |

And because the outputs can only be True or False, the process logic using the Find Customer action is unambiguous.



In contrast, using the 'output-less' version of Find Customer is not as easy. First the user must know (or remember) that the business object can throw different types of exception. Then they need to open Object Studio, examine the diagram and then create expressions to test for the exception details during a recover sequence.

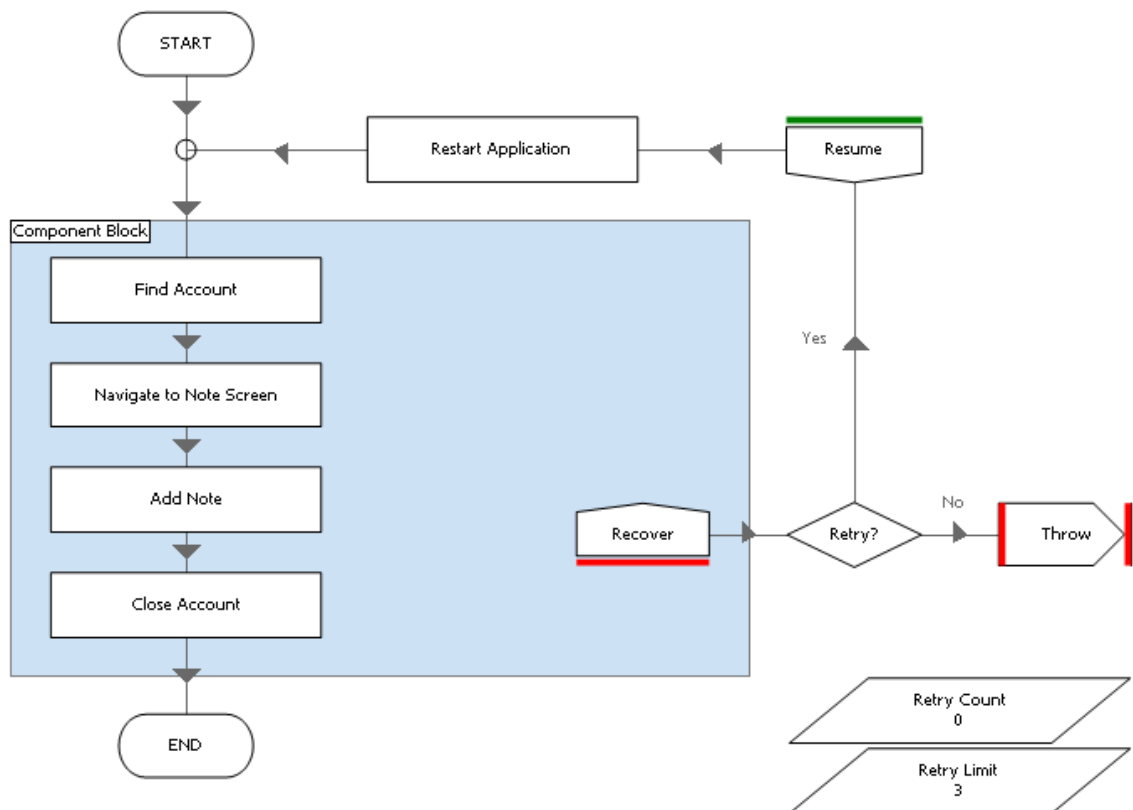


2.5 Sub-Pages

Sub-Pages

Sub-Pages of your process should, if possible, have a single task or function such as retrieving information from one system, or updating another system. Any sub-page that interfaces with a system should have retry logic (as discussed above in section 2.1).

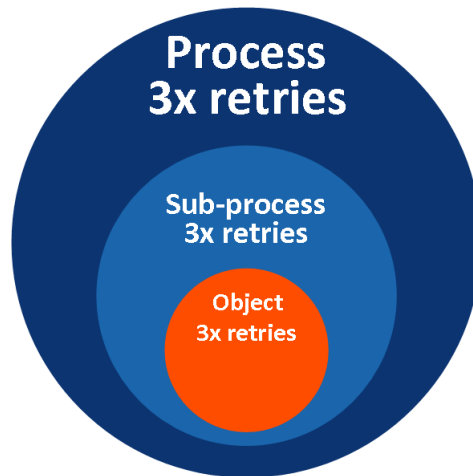
An example would be a sub-page called 'Add Note' which navigates to the screen in the application where a note is added, adds the note, and finally performs any navigation required to return to the main screen of the application ready for the next task.



Sub-processes

Retry Loops

When using components, sub-processes and objects care must be taken not to compound any retry loops by inadvertently nesting them together. The potential for concentric retrying can be difficult for a novice to spot, particularly when child diagrams are themselves calling yet more children.



A simple way to avoid doing this is to make sure a child capable of retries is not used on the retry path of the parent.

A slightly more complex technique could be to use an input parameter to dictate to the child the number of retries it is allowed to perform.

Testing Retry Loops

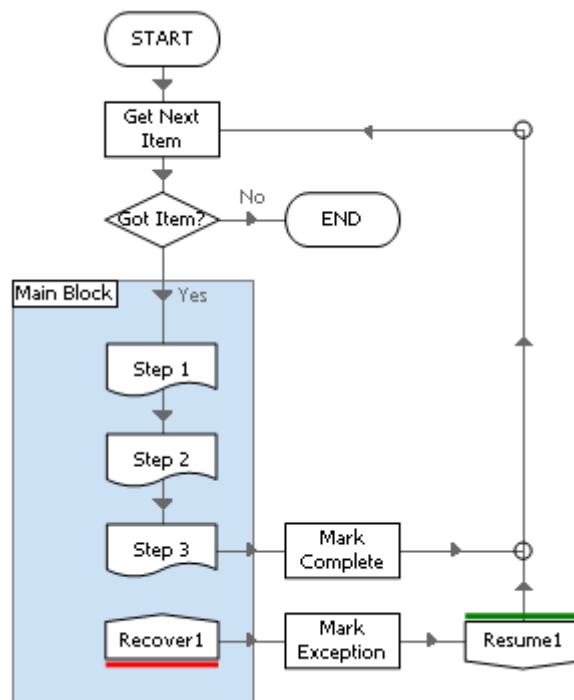
It is good practice to test your Processes to ensure that they can cope if something unexpected occurs in application.

Whilst your Process sub-page is running, manually interfere with the application it is using. Manually navigate to a different screen or exit the application. If your retry logic is working correctly, your flow should be able to recover from your manual intervention and continue.

2.6 Processes

A process will usually use a queue to manage its workflow – as a place for new work, to control the distribution of cases to Resource PCs and to store results. And regardless of what happens to a queue item while it is being worked, exception handling should be used to make sure the queue is updated with a result.

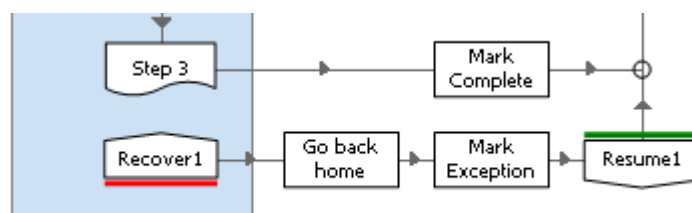
A common arrangement is to use an exception block on the main page of a process covering the part of the path where an item is worked, rather like the diagram below.



This is a simple yet powerful design to protect a process against termination and control queue items. Any exception occurring within the main block will be handled by the Recover stage. And because we know that exceptions bubble upwards, we are free to generate exceptions from any of the pages inside the block, safe in the knowledge that they will be caught.

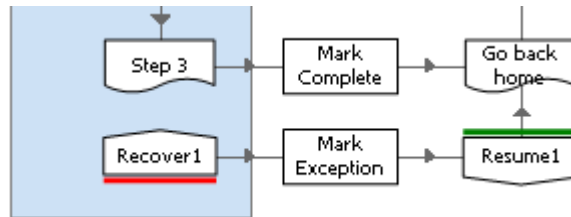
Knowing that the main block provides a safety net, we must also consider the path outside the block. An exception occurring in this unprotected area will terminate the process, so care must be taken to assess the whole process for the risk of unhandled exceptions.

For example, suppose that in the event of an exception a process was to return the target application to the 'home' screen in readiness for the next case. The following arrangement could work, provided there were no problems.



But consider what would happen if the 'Go back home' stage failed. The additional exception wouldn't be handled and would terminate the process before the queue had been updated. This would mean the queue item would be given the default (and rather unhelpful) exception reason 'automatically set exception at clean up'.

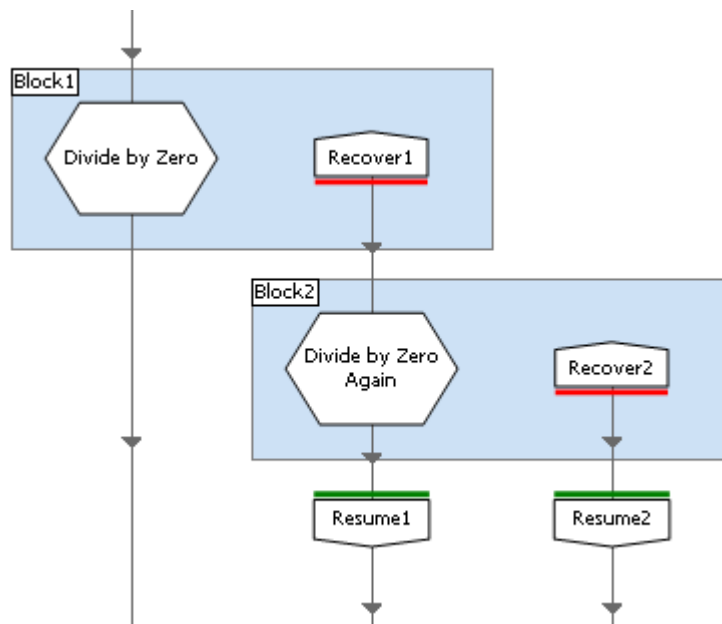
A more robust design would be to update the queue and move past the Resume stage before attempting to remedy the situation, maybe like this.



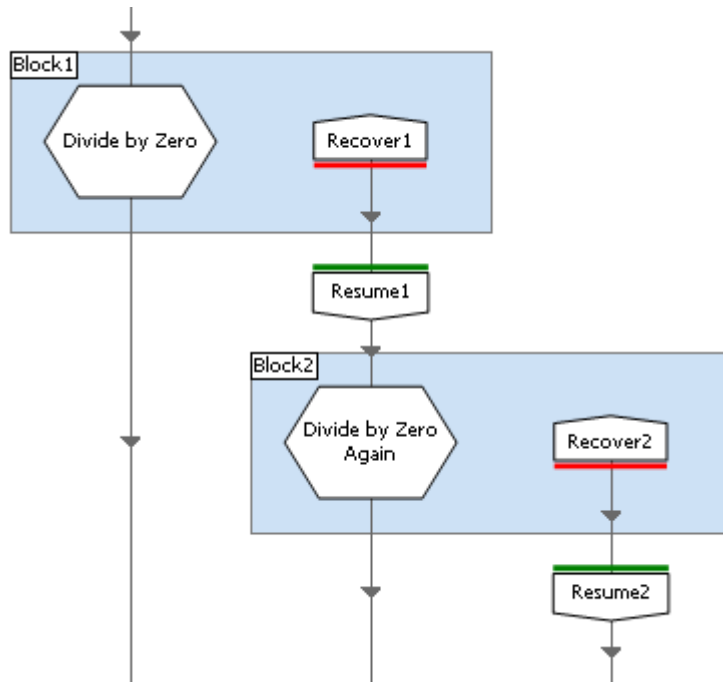
Notice that a 'Go back home' page has been created. This would enable a separate exception handling sequence to be put on this page specifically for the purpose of controlling the application in between cases.

Blocks in Recovery Mode

Exceptions occurring during Recovery Mode cannot themselves be recovered. Although the example below may seem logical it will not work - rather like the forgetting to tick the 'Preserve...' check box, the second 'divide by zero' produces another exception that goes unhandled despite the presence of the second Block.



For this reason effort must be made to eliminate the possibility of exceptions occurring during Recovery Mode. Although trivial, the example below demonstrates how.



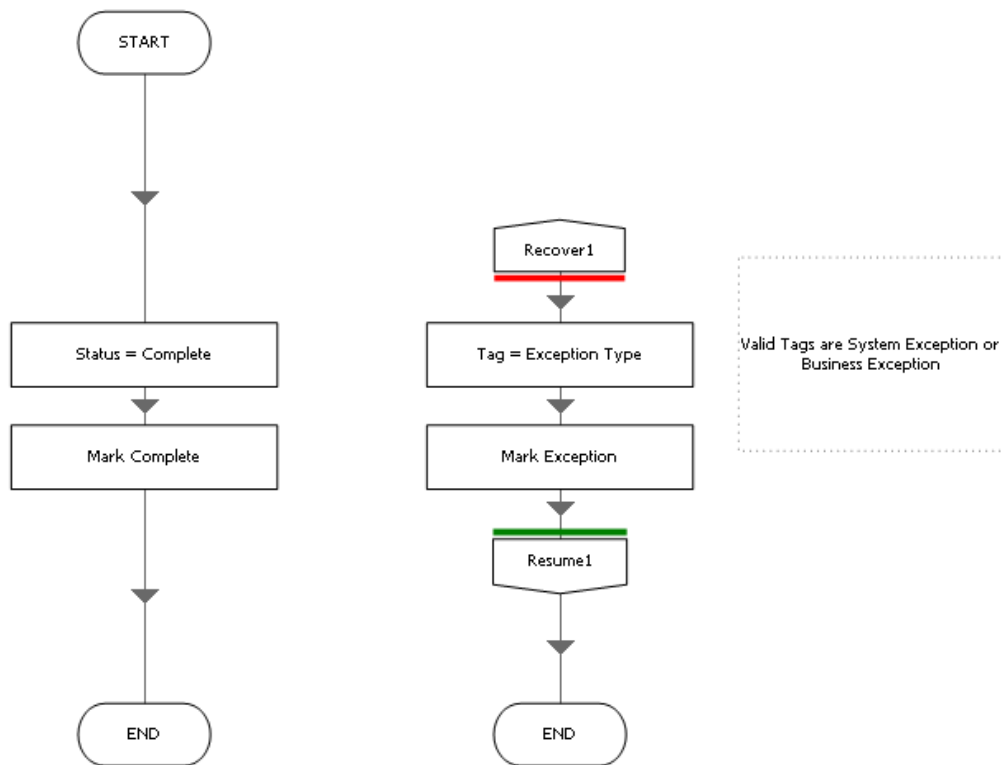
2.7 Queue Item Results

As we have seen, results are applied to a queue item with the Mark Complete and Mark Exception actions. There are also a number of other ways to update an item before a result is applied.

- Tag Item – to append a tag to the item
- Un-tag Item – to remove a tag from the item
- Set Data – to overwrite the item's data collection

A common use of these actions is change the item's tags to match the result. The left hand side of the following diagram illustrates the 'complete' path; the right hand side is the 'exception' path. Tags are a useful way to facilitate MI reporting by labelling items by exception type.

The standard Blue Prism Performance Report requires that all exception items are tagged as either 'System Exception' or 'Business Exception'. If you are using any other exception types in your process (i.e. Validation Exception), you must ensure that you only use one of these two tags.



Monitoring Consecutive Exceptions

In the event of a serious problem with an application it may be better for a process to stop rather than carry on working. For example, if there are 1000 items in the queue and the target application is down, it makes no sense to work all items and have them all marked as exception 'application is unavailable'.

A better solution would be to count consecutive occurrences of an exception and bring the process to a stop when a sensible limit is reached. In other words, if say 3 cases in a row fail for the same System Exception, then treat this as a sign that something is wrong, notify administrator (maybe via email) and stop.

Checking for a System Unavailable Exception

You may decide to use a special exception type to signal a terminal problem and have your process look out for any of these exceptions. For example, if you had a System Unavailable Exception exception type your exception handling could contain the expression ***ExceptionType()="System Unavailable Exception"*** to check whether to stop the process.

A System Unavailable Exception may be thrown by an action or a sub-page if you are unable to launch your application. If such an exception occurs it is not related to an issue with a Work Queue item that might be currently being worked, you may therefore want to simply unlock the item (i.e. by Referring it) rather than marking it as an exception.

Notifications

Instances of some exceptions may warrant a notification being sent out. In a similar way to the 'abort' check described above, you could issue notifications when certain exceptions are recovered.

However remember not to bombard people with emails as you develop and test your process.

Queue Item Retries

Recall from the training course that a Blue Prism work queue can be set up to enable an exception item to be reworked. It does this by creating a clone of the item and injecting it into the queue at the same position as the original.

The **Mark Exception** action has flag inputs called **Retry** and **Keep Locked** for controlling this feature. If a new item is created, its ID will be returned in the **New Item ID** output parameter, and if **Keep Locked** was set to True, the process will have control of the new item.

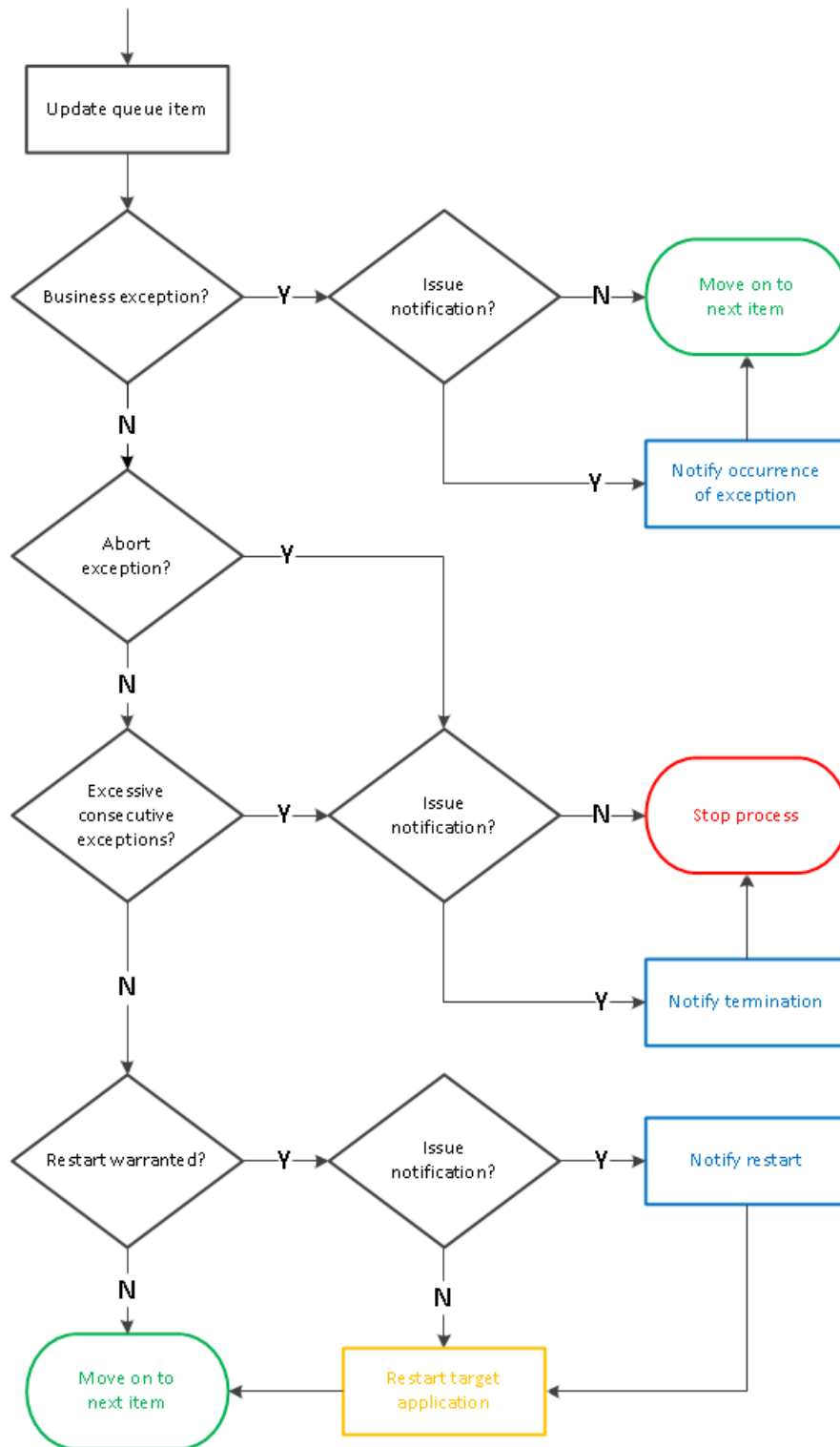
| Name | Data Type | Value |
|------------------|-----------|-------|
| Item ID | Text | |
| Exception Reason | Text | |
| Retry | Flag | |
| Keep Locked | Flag | |

Consideration should be given as to whether a new item should be worked immediately or whether it should be left unlocked (maybe for another machine to pick up) or even **deferred** for a while so as to provide a bit of 'breathing space' between items.

You will usually not require Queue Item Retries unless there is a unique design reason. All exception handling logic is usually done within your process (i.e. with retry loops) rather than using additional attempts on the Work Queue.

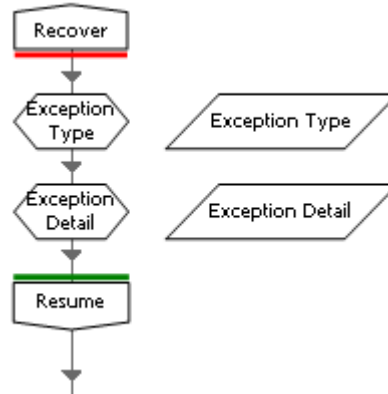
Updating an Exception Item

The following illustration summarises the sort of decisions you may wish to take as the result of an exception.



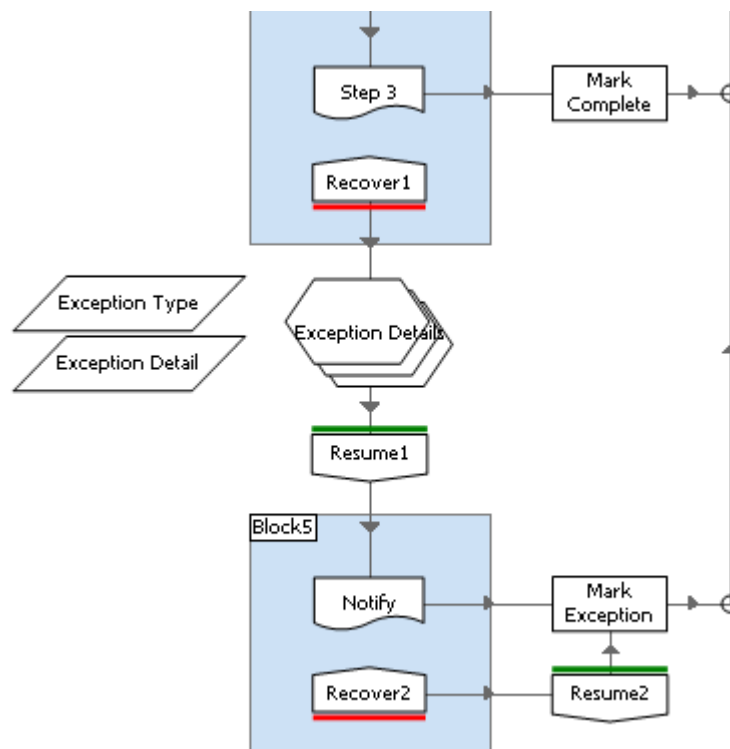
Capturing Exception Details

It can be a good tactic to minimise the time spent in Recovery Mode by capturing the details of an exception in data items and then moving directly to the Resume stage.



The advantage of doing this is that the risk of another exception occurring during Recovery Mode is eliminated and any activities performed after catching the exception can be protected by a secondary piece of exception handling.

In the example below a notification step is performed before updating the queue, but if this step was to fail the second Recover would catch the exception. Then the data items containing details of the first exception would be used to update the queue.



Although the secondary exception handling does nothing to improve the situation, it does at least prevent the notification error from terminating the process.

3 Exception Handling Pitfalls

There are a number of common pitfalls with exception handling that can trip up even the most experienced Blue Prism users. An outline of the most common problems, some which we have discussed above, is as follows.

3.1 Infinite Re-throw Loop

This is when exception handling logic throws an exception back to itself and falls into a never ending cycle of throw-recover-throw-recover. When this occurs in Control Room the process must be terminated. When the diagram is open, if the pause button does not work (because the loop is too fast) often the only option is to kill Blue Prism using Task Manager. And if you haven't saved your work, this can be annoying.

The main cause of an infinite loop is bad use (or no use) of blocks. By its nature exception handling is not linked together by lines like the rest of the diagram, and this can make these potential loops hard to perceive.

3.2 Infinite Retry Loop

This is when exception retry logic does not contain any mechanism to break out of the loop. Normally a counter is used to limit the number of retries that will be performed and without this retrying could go on for ever.

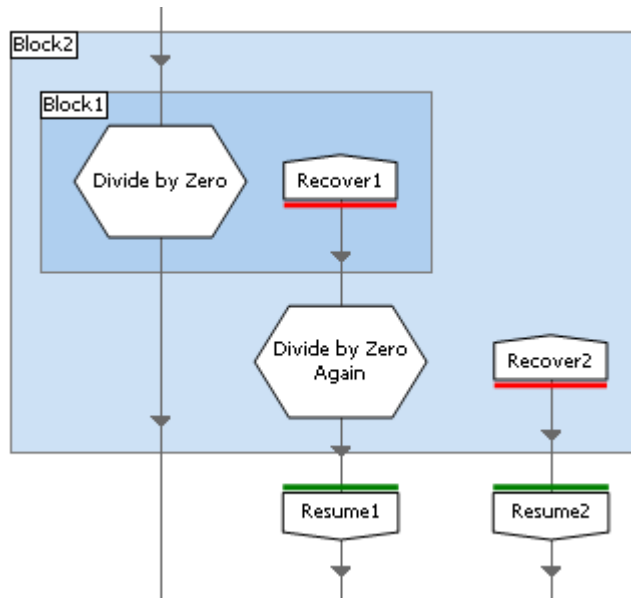
3.3 Nested Retries

This is when one sequence of retry logic sits within another and the numbers of retries in each multiply together. Retry logic is often limited to 3 loops but if multiple sequences have been inadvertently nested together there could be 9 (3x3) or even 27 (3x3x3) loops.

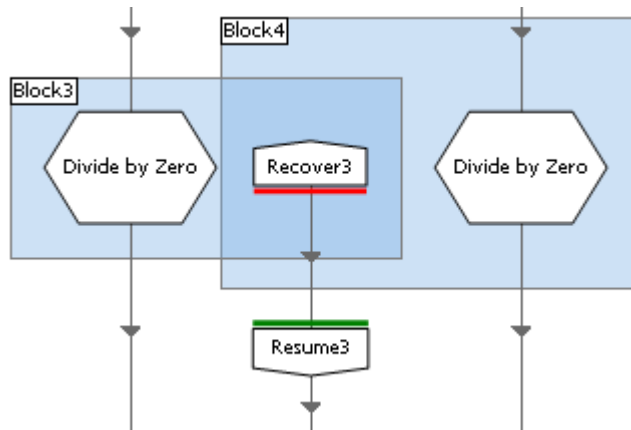
This is usually because the sequences exist on different pages or in different diagrams. This physical separation often makes the whole arrangement harder to visualise, particularly if you are not familiar with all of the diagrams.

3.4 Nesting Exception Handling

As discussed above, problems occur when exceptions occur during Recovery Mode. Blue Prism has no concept of multiple, 'nested' Recovery Modes - Recovery Mode is either on or off. Blocks should not be nested together like this.



Nor should they be overlapped, like this.

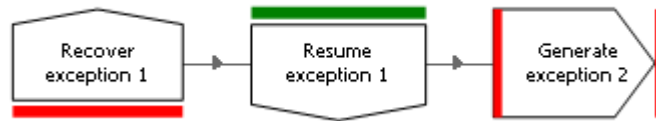


3.5 Generating another exception without resuming

When a Recover stage is followed by an Exception stage, failure to tick the 'Preserve...' checkbox will result in a second exception.

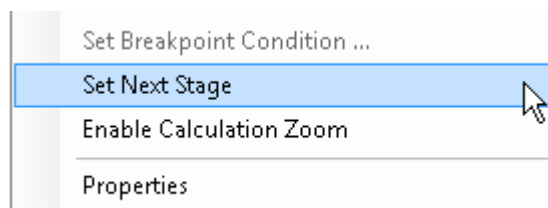


If the intention is to generate a new exception (perhaps with a more informative message) then the correct method is to use the Resume stage to eliminate the first exception.



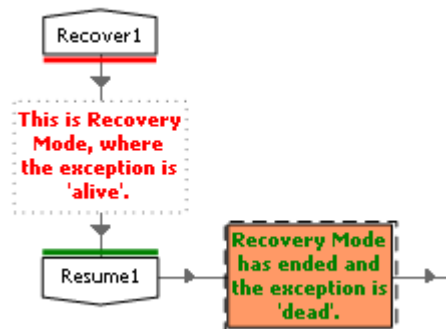
3.6 Stepping Past the Resume Stage

When stepping through a diagram it is not uncommon to find an exception has landed you at a Recover stage, and if the Recover stage is located some distance away from your original position it is tempting to go straight back using 'Set Next Stage' on the mouse menu.



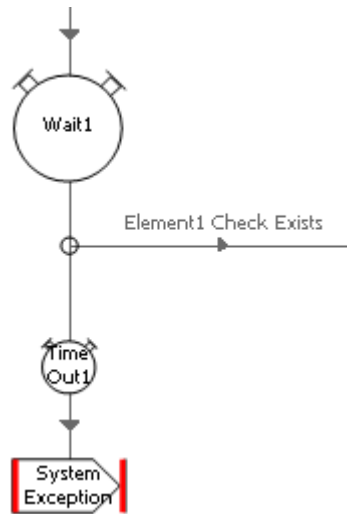
This urge should be resisted because, as we have been discussing, the exception is still 'alive' at the Recover stage – we are in Recovery Mode. Simply moving to another position does not stop Recovery Mode and the exception is still active.

The safer option is to step past the Resume stage to come out of Recovery Mode before moving to another position.



3.7 Misleading Exception Detail

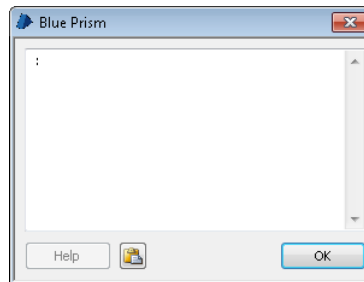
Copy and Paste is a great timesaver in Blue Prism but can result in sloppy detail, particularly in business objects. As you are aware, the time out end of most Wait stages will lead to an Exception stage, probably type 'System Exception' with a message such as 'Failed waiting for something'.



Because Wait stages and Exception stages are so commonly used in a business object, time can be saved by using Copy and Paste. Unfortunately it is very easy to forget to modify the Exception stage and leave the same message in many places. Ideally each exception message should be unique so that the task of retracing the root cause of a problem is made easier.

3.8 Missing Exception Detail

If the properties of an Exception stage are left blank then it will generate an exception with no detail. This can have a knock-on effect on any subsequent exception handling or result in a queue item having no exception detail.



4 Human Intervention

When an exception results in an incomplete case, the case must be referred to a human being.

The SDD (Solution Design Document) may stipulate that referrals are communicated automatically as part of the process, or a simpler, manual procedure may be specified. Either way, cases that Blue Prism did not complete, for whatever reason, should not go undeclared.

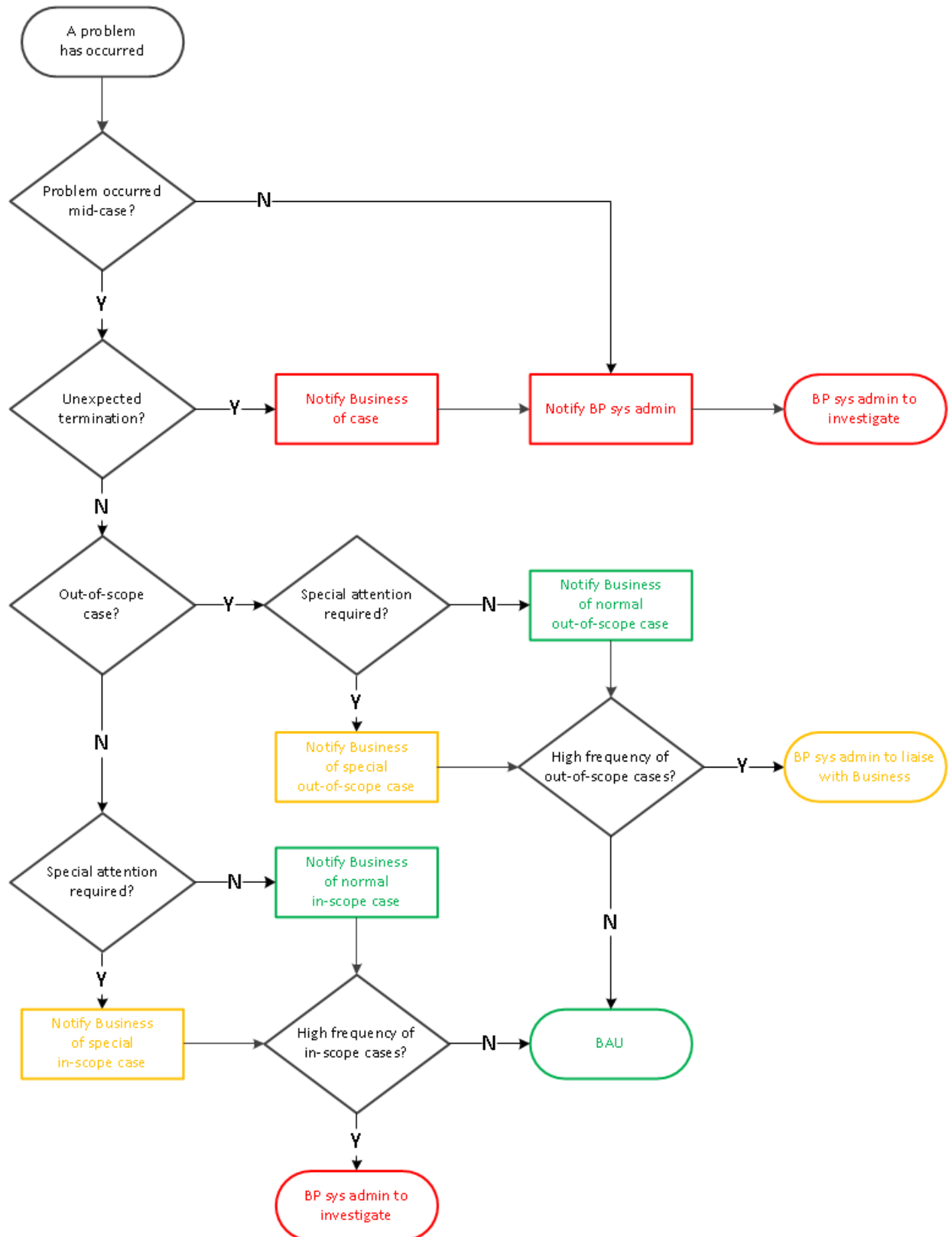
If the exception indicates an out-of-scope case (e.g. a business exception) then Blue Prism is working normally and the referral should be carried out as agreed in the solution design. This will involve passing the case detail on to a team in the Business for manual reworking, perhaps as part of a daily report. Some types of exception may warrant special (or rapid) attention in the form of an individual notification to alert key individuals.

Exceptions indicating a fault (e.g. system or internal exceptions) should also be brought to the attention of the local Blue Prism team. Subsequent investigation into the nature and frequency of these errors may prompt changes or repairs to a solution. It is unlikely that such problems will be completely eliminated, but regular monitoring of exception levels should be done to maintain steady performance.

The work queue will normally be the main source of performance information and session logs will also provide micro-detail of the steps taken when working a case. Similarly, the logs should provide detail of more serious problems like process terminations.

4.1 Diagram

The following diagram illustrates a typical exception referral procedure.



Problem occurred mid-case?

If the problem occurred while working a case then that case will need attention.

Unexpected Termination?

Normally a process will be configured with logic such as a 'stop time' to bring it to a natural completion. If a process terminates unexpectedly then there has been a problem which the Blue Prism system administrator should investigate.

Out-of-scope case?

If an exception case has been deemed out-of-scope in accordance with the solution design, then Blue Prism is working normally.

Special attention required for out-of-scope case?

The solution design may stipulate that some types of out-of-scope case require non-standard attention, perhaps because of an SLA (Service Level Agreement) or the unusual nature of the case. The normal notification procedure should be applied to all other out-of-scope cases.

Notify special out-of-scope case

Special cases may be announced in a different way to others, perhaps by sending an email to a particular mail box.

Notify normal out-of-scope case

The standard notification procedure for out-of-scope cases may simply be a report produced at the end of each day.

High frequency of out-of-scope cases?

The solution design should indicate the anticipated frequency of out-of-scope cases. If these cases appear more often than expected, then the BP system administrator should liaise with the Business to investigate the root cause.

Special attention required for in-scope case?

Some technical failures may be deemed more serious than others and necessitate particular interest.

Notify special in-scope case

If, for example, a target application hangs and consecutive system exceptions occur, the Blue Prism system administrators should be alerted.

Notify normal in-scope case

As with normal out-of-scope cases, in-scope case may simply be announced via a report distributed to the stakeholders agreed in the design.

High frequency of in-scope cases?

Achieving a steady performance should be the goal of any Blue Prism solution. Exception levels are unlikely to be zero but they should at least be stable. Any jump in the failure rate is indicative of some sort of change that requires scrutiny from the Blue Prism system administrator.

5 Process Templates

The Blue Prism Process Templates available on the Blue Prism Portal are intended as a base for starting any new process. Using the templates helps to ensure that you create processes that adhere to best practices you have been taught during your training.

The Blue Prism Process Templates include the following features that have been outlined in this guide:

- Retry loops on sub-pages
- A Block and Recover stage on the Main Page to ensure the Work Queue is updated with a result if there is an exception thrown up from a sub-page
- A Mark Item as Exception page which evaluates exceptions that occur to take the appropriate action (such as adding the correct tag and monitoring concurrent exceptions).

The Process Templates are distributed with Instructions to assist you in their use.