Reference Guide

Process building and debugging

Document Revision 1.0

Trademarks and copyrights

Contents

Introduction

This guide is intended to supplement Blue Prism development Best Practice and provide guidance on solution build strategy for delivery teams and in particular, developers. This document assumes the reader has completed the Blue Prism developer course (Foundation Training plus mandatory guides) and is familiar with solution design concepts.

Solution Build Strategy

It is important that the delivery team cultivates a strategy for building Blue Prism RPA solutions. Projects should be delivered collaboratively rather than as personalised, solo effort of an individual; rather, the team should work together according to the agreed strategy. The strategy will not appear fully formed overnight but will evolve over time as a team's experience and skill grows.

Deciding, defining and agreeing what to build, how and when requires discussion, demonstration, consensus and planning contribution from the business, IT, the RPA team leadership and the delivery team. A full discussion of the Robotic Operating Model and Delivery Methodology is outside the scope of this document, other than to say that successful delivery is founded on careful planning and agreement of requirements, scope, design, test strategy, risk acknowledgment, success criteria and operational and support models.

Business Objects

Designing and creating business objects is relatively easy when each object page has a specific, understandable purpose. As well as discouraging single 'mega' objects in favour of a series of smaller objects, Best Practice also advises to try and keep object pages simple. Complex, multipurpose pages tend to be harder to build, test, maintain and importantly, re-use.

Conceptually, the pages of a business object can be thought of as 'mechanical parts' that are used to create bigger machines, i.e. processes.

Although it's normal to start building a solution by creating objects, a common beginner's mistake is to spend too long in Object Studio before starting on the process, and this strategy is almost certain to lead to problems. The risk in concentrating on objects is that the developers forget that all the parts need to work together, and even though the individual parts may seem perfect, they don't assemble to form a robust construction.

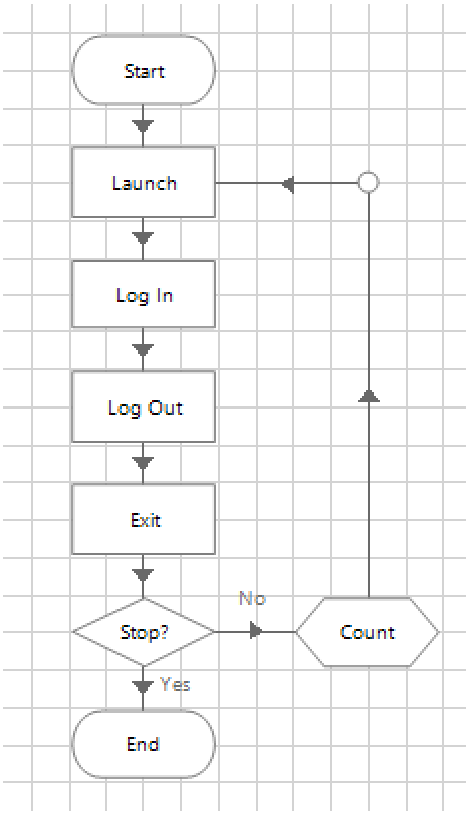
When this happens, there is a potentially large collection of components that don't fit. The process has errors in many places, and it can be difficult to know what to correct. Furthermore, the project deadline is approaching and the process does not yet work.

Testing object mechanics

As mentioned above, business objects can be thought of as the mechanical side of a Blue Prism automation. They should not contain any business logic (despite the name!) and merely act as mechanisms that enable a process to manipulate applications.

The key to creating good objects is to test in Control Room as early and as often as possible. The Studio diagram can give the impression all is well because the page appears to run well but, one should never forget that diagrams run relatively slowly, and even the maximum speed is not comparable to Control Room.

The trick is to close Object Studio regularly and create a test process to run a sequence of object pages, as shown here.

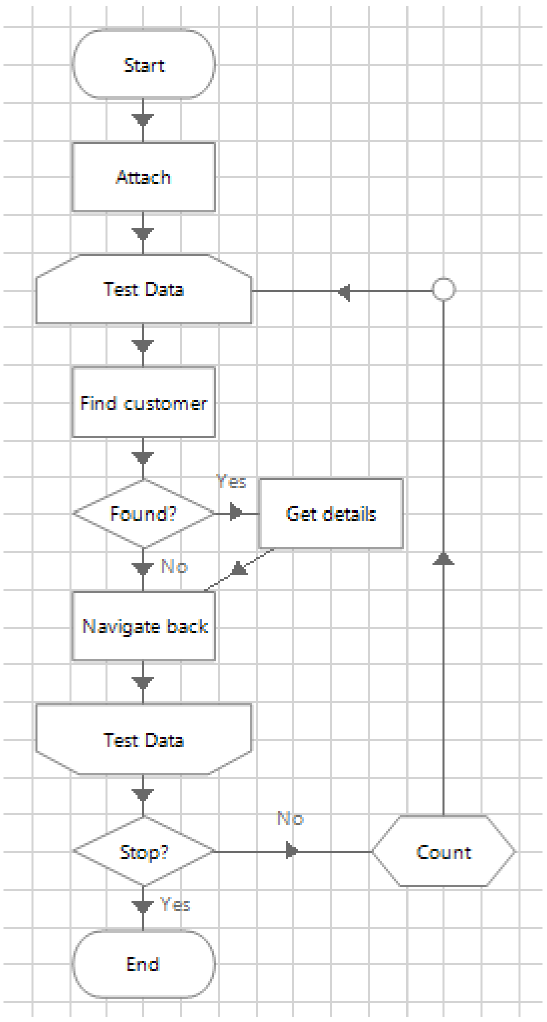


1. Run the test process as a diagram first and if it fails, go back and correct the object.
2. Create a loop that enables the process to repeat the steps.
3. Run the published process in Control Room.
4. Again, correct any problems and run again.

Once the test process is running well, the assumption can be made that these few object steps are ‘mechanically strong’. The developer can then return to creating and testing a little more object logic in the same way.

Testing objects with data

Object logic will involve a variety data – inputs, outputs, different records, screens etc. Another common pitfall when building objects is to keep using the same test data, and the risk is that the object logic will only work with this data and fail with any other. Therefore, as well as testing repeatedly, objects should be tested with as wide a variety of data as possible.



The ambition here is to prove the objects can cope with different data and to minimise the number of 'surprise' scenarios during the process build and test. Knowing that objects are already solid before assembling them into a process will bring confidence that the process is going to function and will also allow the developer to concentrate on the process logic without worrying about whether the application logic will work.

Testing objects with live data

Building objects against an application that isn't the same version as Production is almost certain to bring problems. A business object is unlikely to be able to use an application which it has not already been trained on; it cannot think or anticipate anything it has not already seen before, so expecting it to integrate with a different version of an app is asking too much.

Using unrealistic test data is also a potential problem. Even if the test application is the same version as Production, if the data is old and invalid, the application is likely to behave differently. The concept of using Live apps and data can be unthinkable to many new RPA clients, but like a trainee human workforce, at some point the robots must be allowed into the real world, albeit under supervision.

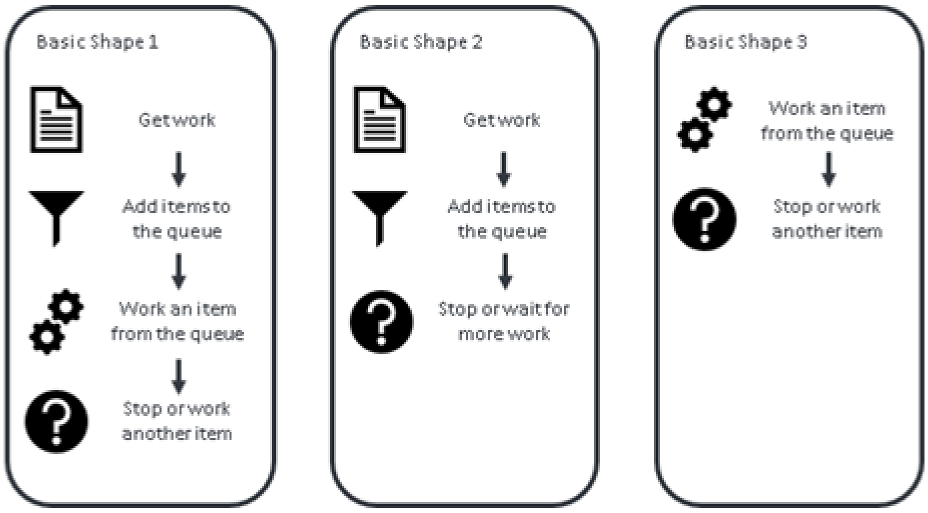
For a fuller explanation, see 'Introducing Your Process To Live Data' document on the Blue Prism Portal and 'Delivery Roadmap' or the Lifecycle Orientation training course.

Process Building

Fundamentals

The number one requirement when first creating a Production process is to use a template. A template saves time, includes Best Practice and propagates a common design. Example templates can be found on the Portal and delivery teams are encouraged to create their own versions.

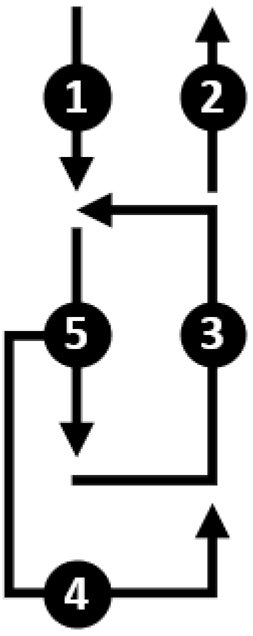
Having more than one template is not uncommon - although all processes have similarities, there are a handful of common RPA solution ‘shapes’ and having a template for each would make sense. For example, many Blue Prism processes follow one of these generic types.


Basic Shape 1
Get work
Add items to the queue
Work an item from the queue
Stop or work another item
Basic Shape 2
Get work
Add items to the queue
Stop or wait for more work
Basic Shape 3
Work an item from the queue
Stop or work another item

Another key requirement is to use a work queue. Very occasionally there may be a valid reason for not using a queue, but this is quite rare. As mentioned previously, this document assumes the reader has completed the Foundation Training course and is already familiar with the guidance on work queues available on the Portal.

The Five Part Pattern

Almost any Blue Prism process intended to execute repetitive tasks can conceptually be reduced to a *Five Part Pattern*, as shown below.



Obviously, repetition implies a loop of some kind and so we can perceive the parts of the pattern as follows.

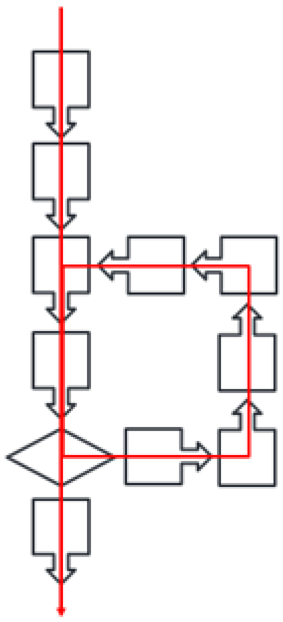
- Part 1 defines the start of the process, before the looping has begun.
- Part 2 covers the steps after leaving the loop, when the process comes to an end.
- Part 3 is the logic required to complete the loop before starting the next task.
- We expect to encounter problems, so Part 4 is the recovery logic, bringing the process back to the happy path.
- Finally, Part 5 is the actual 'work task' logic where again, we anticipate that some tasks will result in exceptions and follow the unhappy path.

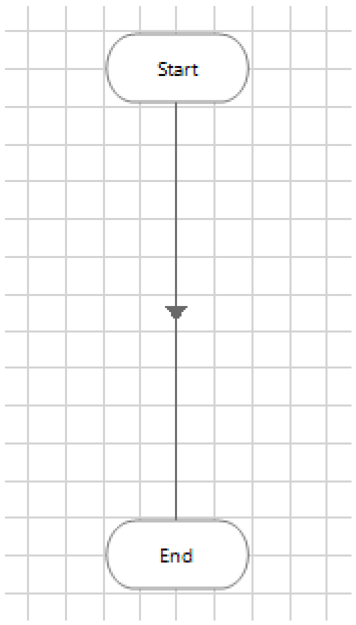
Hopefully the reader will recognise this pattern in the template diagrams available on the Portal. Visualising an automated process in this way can help shape the solution design, dictate the build sequence and inform the test approach. And by starting with a template, the construction of the process can be broken down into phases.

Phase 1 – Template structure

The objective of this phase is to create a skeleton process with a complete 'end to end' path.

1. Take a copy of a template.
2. Rename the main 'work item' pages to suit the major steps of the process. Using recognisable names will help SMEs understand the diagram.
3. Add any extra 'work item' pages as required.
4. Ensure the path from the Start and End stages of each page is complete, and if necessary simply link the Start of an empty page to the End.

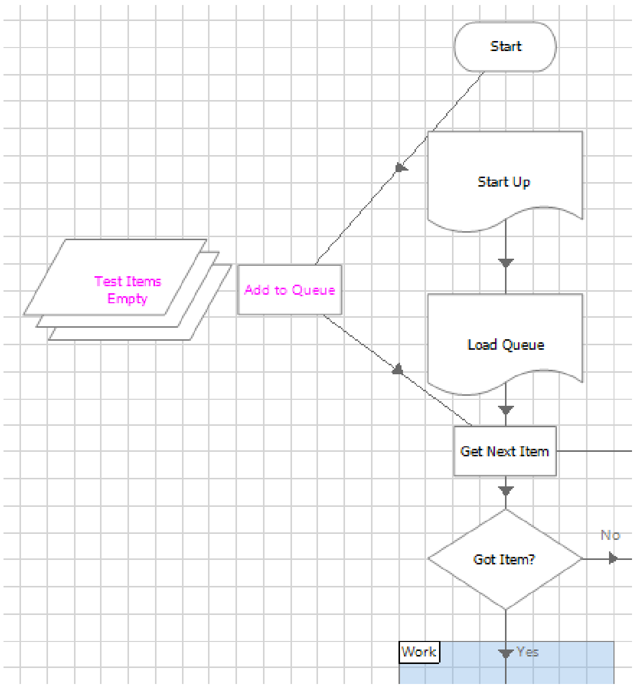




Phase 2 – Work queue

This phase is to prove that the diagram flow is intact and that the process can work through queue items and apply results.

1. Create a queue with Item Key named as the key field.
2. Create a collection called Test Items on the Main page of the process. Add a column called Item Key and populate a few rows on the Initial Values tab with fake data.
3. Join an Add to Queue action between Start and Get Next Item. This will be a temporary step that will be deleted later, so just add it anywhere on the Main page.



4. Step through the process, checking that the data is added to the queue, Get Next Item works as expected, all the empty pages can be stepped through and Mark Completed updates the queue with results.

This activity may seem insignificant because most pages are empty. But if this step is missed, there is a real risk the process only makes it's first 'end to end' run near the end of the development period, when time is running out to fix any problems.

Phase 3 – Item data structure

The idea with this step is to focus on the queue and to simulate how a queue item will move through the diagram.

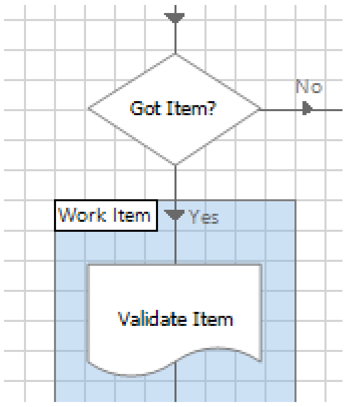
1. Add the additional columns to Test Data collection, as directed by the design document.
2. Add more fake data to the Initial Values of this collection.
3. Step through the process and check that Get Next Item still works and there are no spelling or datatype mistakes in the collection field names.
4. Add any other data items that will be used after getting an item from the queue and check there are no spelling, value, datatype or exposure mistakes (local/global).
5. Create any necessary Environment Variables and check that they work and the corresponding data items and have the right exposure.
6. If the design mandates the use of tags, statuses and the like to manage queue items and control the output of Get Next Item, then rehearse these movements by stepping through the process.
7. If there are any Tag Item or Update Status steps required at strategic points, then add them in. Check that the desired effect is visible in Control Room.
8. If the solution defers or unlocks items, uses more than one queue, spans different days or has multiple phases, then practice and perfect the item 'journey' while the process is still in this 'data only' phase.

Again, the process is still not yet doing anything significant, other than proving that the queue data and logic is working. Perfecting this aspect of the process is very much easier to do when the process is in skeleton form, without the distraction of other business objects and application logic.

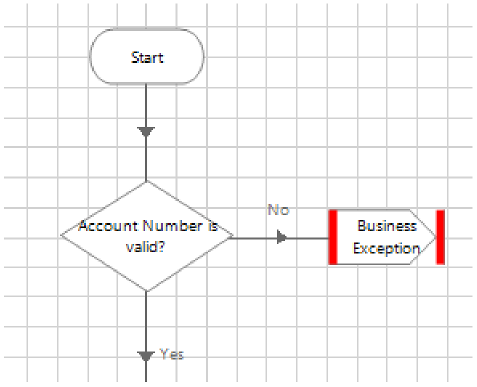
Phase 4 – Data validation

This phase is to prove that the data in the queue item collection can be validated and exception handling will update the queue correctly.

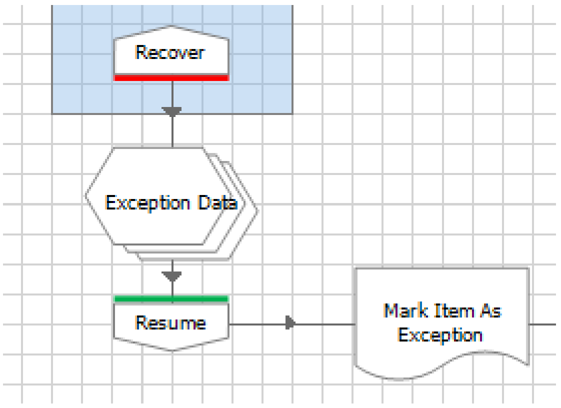
1. If it does not already exist, add a Validate Item page immediately after getting the queue item.



2. Add decision stages to the Validate Item page that check the collection output from Get Next Item. For example, `Len([Item Data.Account Number])=8`



3. Throw exception stages when the validation checks are not met.
4. Change the data in the Test Data collection to include 'bad' data that tests every validation decision.
5. Step through the process to prove that the exceptions thrown by the validation logic are correctly handled and the queue is updated.



6. Check the exception detail recorded in the queue is correct and meaningful. Remember these are the results that will be used in UAT and ultimately be sent back to the Business from the Production environment.
7. Make a list of all the validation exceptions – this will help in testing later on.

Phase 5 – Load queue

The assumption here is that the process is getting input data from a file. Evidently there are many other possible data sources and input mechanisms, and the reader may have to interpret this phase differently to suit their needs.

1. Disconnect the temporary Add to Queue step and reconnect the Start stage as it was originally.
2. Add the logic necessary to populate the queue.
3. Prepare some dummy input data and step through the process to confirm the logic.
4. Assuming the logic works, run from Control Room.
5. Confirm that items are added to the queue as expected and results are applied.
6. Introduce invalid data to the input file and confirm that the Validate Item page is able to detect the problems and raise exceptions accordingly. Confirm that the queue items are correctly marked as exceptions.
7. Add logic to handle an invalid input file, and check that it works in Control Room.
8. Add logic to avoid reading the same file twice, and check that it works in Control Room.
9. Add logic to detect duplicate cases, and check that it works in Control Room.

At this point you should have a rudimentary process that technically works. The process doesn't really do anything apart from move data around but it has a structure, a queue, exception handling and runs 'end to end'.

When developing any process, it is strongly recommended to reach this position, where early in the build period (and for an experienced developer this can be before the end of the first day), the process can run in Control Room from start to finish. Even though the process is still useless in practical terms, the fact that it runs without breaking down is extremely valuable.

By contrast, processes that are not developed in this way risk staying in 'diagram mode' and never running in Control Room until late in the development period. And if the process won't run because of spelling mistakes, syntax errors or worse, incorrect logical flow or queue data structure, then the delivery team will start to feel the pressure as they struggle to make (potentially fundamental) changes at the last moment.

Phase 6 – Outputs and MI

Every design should describe how exception cases will be communicated back to the Business. The design may require that the process creates event-based outputs or notifications (e.g. process termination) and MI reports. If this is the case, then consider adding in the output logic while the process is in this simplified, 'dummy data' state.

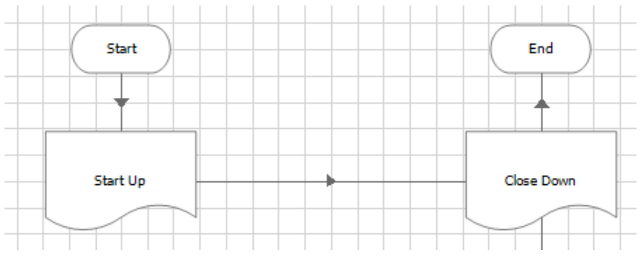
Even if an MI mechanism already exists (e.g. another reporting process), check that it works with your new process and queue. If your process is to issue notifications or alerts, check that they are working. As always, test the new logic from Control Room and never assume Studio testing is enough.

At this point the process can load new data into the queue, simulate working through it and then output results. The basic skeleton process is now ready to be fleshed out with objects.

Phase 7 – Start up and close down

Only now should the application logic - the business objects - be introduced into the process skeleton.

1. Connect the Start Up page directly to the Close Down page.



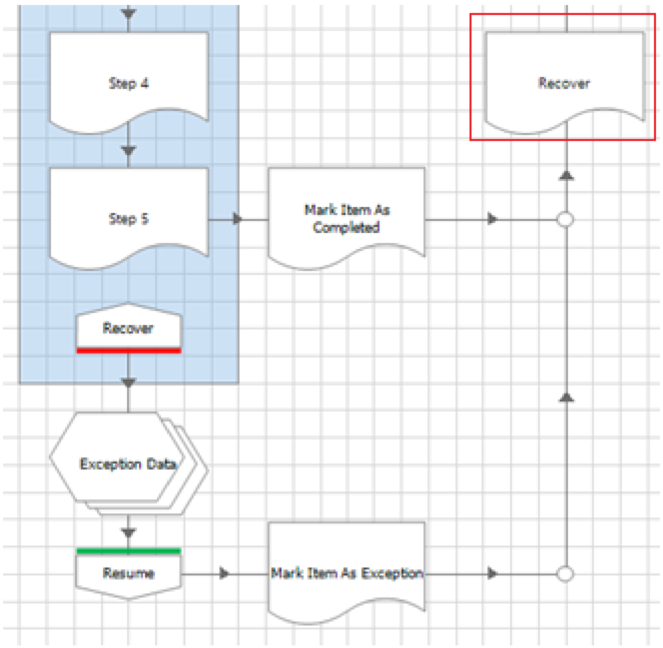
2. Create any credentials necessary to access the first target application.
3. Add logic to the Start Up page to log into the application.
4. Add logic to the Close Down page to exit from the application.
5. Step through the process to check the logic works (which it should because the object mechanics have already been well tested).
6. Run the process in Control Room.
7. Repeat the previous steps for the next target application.
8. Consider the 'unhappy path', e.g. what could happen when a credential has expired or a password is invalid.

So now at this point the process can launch and close all the applications it needs, and the process runs in Control Room without any issues.

Phase 8 – Recovery

The purpose of this phase is to prove that the process has complete control over its applications and can recover and restart them at will.

A critical aspect of process design is the ability to recover from unexpected scenarios. Enterprise RPA means 'unattended' automations running in a 'lights out' environment. This is usually achieved by having a Recover page somewhere between the end of the current case and the start of the next. The idea of the Recover page is to clean up and prepare the applications in anticipation of the next queue item.



Under normal circumstances (i.e. the happy path), this is likely to be some sort of navigation back to the ‘ready’ position or home page. But on the unhappy path, this may be more complex because the navigation could start from various exception positions and may necessitate a full restart of the applications.

To test recovery logic, force the applications into awkward positions, either by deliberately generating exceptions or by manually moving the application into a position the process is not expecting, to force a wait stage time out.

Run in Control Room to prove that the process can escape from the unhappy path. If necessary, the process should be able to close an application down, re-start it and navigate to the ‘ready’ position.

Phase 9 – Case recovery

As well as the ability to recover from unexpected application behaviour, thought should begiven as to how a process will behave in the event of a ‘shock’, such as a sudden network outage or power failure.

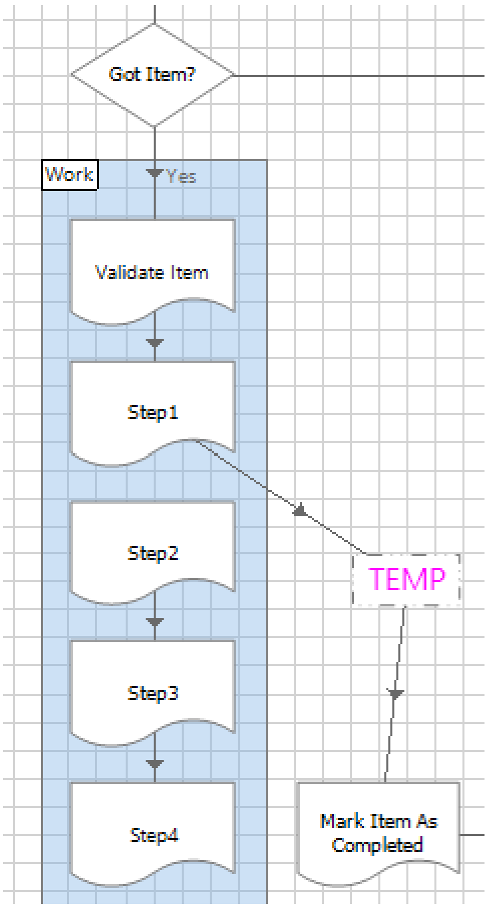
To some extent Blue Prism will handle its queue items by applying the ‘Automatically set exception at clean up’ exception, but what happens to this item should be considered.

For example, if a financial transaction was being submitted when the outage struck, would it be better to rework the case (and risk transacting twice) or send the case for manual referral (and risk missing the transaction). Not every process requires such careful design but depending on the business context, it may be necessary to think carefully about such possibilities, however unlikely.

Phase 10 – First work step

Only here, at the tenth phase of development, do we introduce the first step of the actual 'work case' part of the process.

1. Connect the first 'work' page reference directly to the Mark Completed step. If necessary, use an inert stage like a Note or Anchor to create the link.



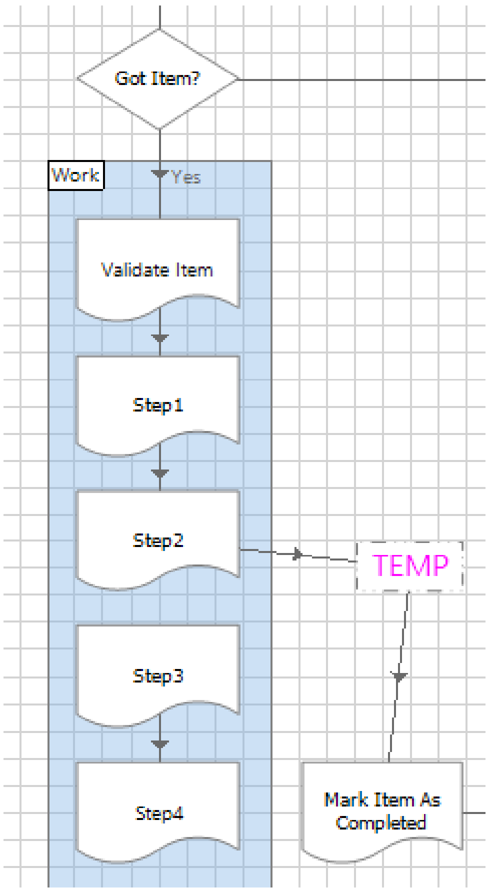
2. Add logic to the sub-page and step through to check that it works (which it should, again because the objects have already been well tested).
3. Run the process in Control Room. More than once. With a variety of data. Good data and bad data.

If there are any issues, then they are very likely to be on this newest page – you should not need to look very far for the source of the problem.

Phase 11 – Next work step

This phase continues the idea of gradually adding more detail to the 'work' section of the Main page and regularly using Control Room to prove that the process still works.

1. Reconnect the first 'work' page to the second page.



2. Connect the second page directly to the Mark Completed step.
3. Add logic to the second page and step through to check that it works.
4. Run the process in Control Room. Multiple times. With different data.

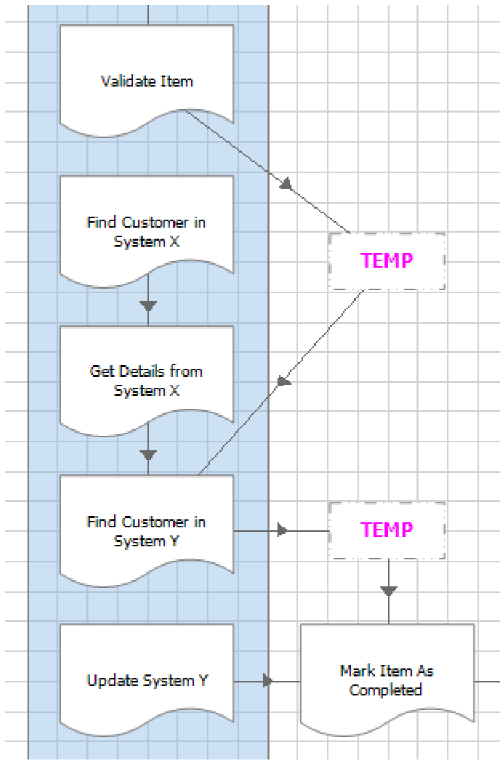
As before, if there are problems then they are very likely to be on this latest page.

Phase 12 – More work steps

A consequence of developing incrementally like this is that it takes longer and longer to arrive at the later pages of the process. And this brings the possibility that the later pages are not as well tested as the earlier pages.

As the sequence lengthens, the time taken to reach these steps increases and possibly this will slow the development progress. Therefore, where possible, ‘short circuit’ the logic and jump directly to these later pages. Note that this isn’t always logically possible, depending on the business process.

As ever, run the process in Control Room. Multiple times. With different data. And remember, if there are problems then they are very likely to be on this latest page.

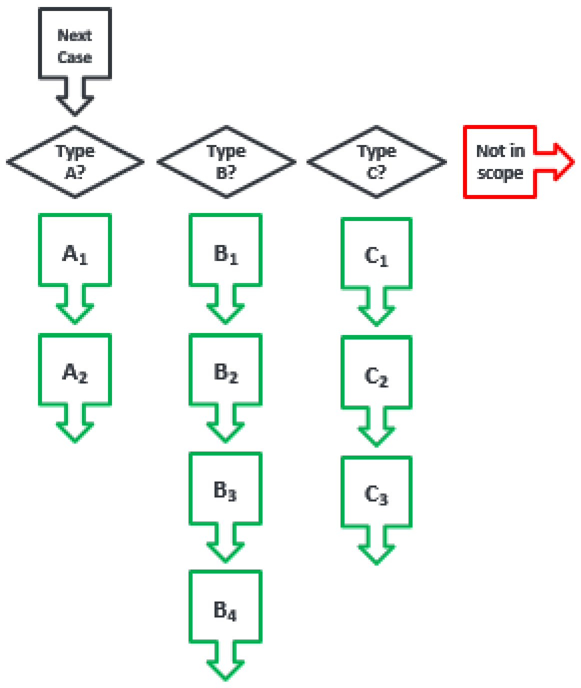

Validate Item
Find Customer in System X
TEMP
Get Details from System X
Find Customer in System Y
TEMP
Update System Y
Mark Item As Completed

Evidently the example above is simplified for the sake of clarity, and in the real world processes can be more complex. But even so, the same principle applies – build and test incrementally in order to ‘fail fast’ and fix problems early rather than find them when time is running out.

Phase 13 – Scenario testing

We are all aware that an RPA solution is a predefined logical sequence. There is no brain, no judgment, no adapting, no learning. Maybe in the future AI and ML could change this but for now, that topic is not within the scope of this document.

Knowing this, the project's functional requirements and solution design should define the automation scope - what the process will do and, importantly, what it will not do. Adopting a 'scenario mindset' throughout the delivery phase and visualising a solution as a set of scenarios rather than as an automated replica of the manual process, helps focus minds on what the scope will and will not cover.



This in turn adds precision to the build specification and defines the testing and acceptance criteria. With this in mind, the developer should try to concentrate on one scenario at a time. Where possible, input data should be rigged so that only one type of case is loaded and worked. Or maybe the process diagram can be temporarily changed to focus on a particular scenario.

Throughout delivery, there should be a record of which scenarios have been developed, worked, tested and accepted. Unfortunately, this is often overlooked and processes go to Production too soon, resulting in a prolonged and painful go-live where the automation is beset by 'surprise' problems.

To repeat, RPA automation logic can only be predefined.

Process Debugging

An obvious key to debugging a process is the ability to locate the source of the problem.

As any Blue Prism developer should know, the work queue provides a high-level view and the session log provides a low-level view of exceptions. With this in mind, effort should be made to ensure the information recorded in the queue and logs is useful. In particular, exception types should be carefully chosen, and exception details should be accurate.

We all know Copy and Paste is a great time-saver during development, but care must be taken to avoid duplicating exception information. A 'narrative' of a queue item's progress through a process can also be recorded to assist investigation into problems. The Status, Tags and Item Data collection can all be updated with trace information.

Recall however that Best Practice mandates that Tags must not be used to carry case-specific data (such as a Client ID or address) and should only be used for the categorisation of queue items. Excessive numbers of unique Tags have been known to put undue strain on the database.

Similarly, the logging level in a diagram can and should be adjusted. Note stages can be useful for recording a 'marker' in a session log. Screenshots, both as temporary aides or as part of exception handling, can also be helpful when it's difficult to witness a problem.

When problems do arise, it can be useful to have the ability to identify, isolate and replicate the type of input/data/case/scenario/time/whatever that is causing the issue. And with practice, a developer can learn to prepare a process diagram in such a way as to make this easier.

Object Debugging

When the 'build and test' strategy described earlier in the document has been used to create business objects, problems with application integration logic should be minimal – the objects have been well tested via Control Room, and any problems are likely to have been exposed before the objects were introduced into the process. Experience has shown that users who do have such problems are likely to be the ones who have not used this recommended approach to creating business objects.

Retaining the 'test rig' processes used to first run objects in Control Room can also be useful in isolating any future problems. Using a full-scale process to test a particular instance of object performance can be difficult and having these lightweight test processes to hand can assist the investigation effort.

That said, assets build up quickly in a busy development environment, so from a system maintenance perspective, it may be useful to remove test rigs to a file storage (as XML), rather than let the development database bulge to an extent where the performance of the BP client UI is impacted.

Summary

Objects

- Avoid designing single large objects with many pages.
  - Use a series of mini objects instead of one mega object.
  - Keep the purpose of each object page simple and functional.
- Develop and test objects at the same time.
  - Stop every few pages and run a test process.
  - Run tests via Control Room, not just in Studio.
- Think about the data used to test objects.
  - Prove consistency by repeating tests with the same data.
  - Prove reliability by testing with different data.
  - Prove resilience by testing with bad data as well as good.
- Strong objects minimise the chance of ‘mechanical’ errors complicating the process build phase.

Processes

- Use a process template.
- Use a queue.
- Start by making a ‘data only’ skeleton that is able to run.
- Add fake data to the queue to begin with.
- Create validation logic to mark bad work items as exceptions.
- Create the logic to produce output.
- Enable to process to control (launch, close and restart) the applications.
- Develop one ‘work case’ page of the process at a time.
- Perceive the process as a set of scenarios, both positive and negative.
- Keep track of which scenarios have been built and tested. Pay attention to any that have not been seen.

Remember that ultimately the only test that counts is Control Room.

Appendix

Build Sequence Checklists

These lists can be used as aides to follow the build sequence recommended by this guide. An experienced developer will be able to use this technique without such detailed guidance but for those new to Blue Prism, these ‘recipes’ may prove useful.

Basic Process Operations

Objectives

- Start from a template diagram to avoid wasting time starting from scratch
- Create a skeleton process that can run
- Create a page structure that can be fleshed out later
- Prove the basic ‘queue item’ journey

Test	Description	Run in Studio	Run in Control Room
1	Process copied from template		
2	Empty page ‘stubs’ created	☐	☐
3	Test items (with Key Field only) added to work queue	☐	☐
4	Run from *Get Next Item* directly to *Mark Completed*	☐	☐
5	Run from *Get Next Item* directly to *Mark Exception*	☐	☐
6	Run from *Get Next Item* directly to *Defer/Unlock* (if required)	☐	☐

Basic Queue Operations

Objectives

- Confirm basic data flow around the process
- Confirm touch points during the lifespan of a queue item
- NB - do not read any external data yet

Test	Description	Run in Studio	Run in Control Room
1	Fields added to Item Data collection	☐	☐
2	Queue populated with fake data	☐	☐
3	Get Next Item (outputs populated as expected)	☐	☐
4	Update item status works as expected (if required)	☐	☐
5	Update item tags works as expected (if required)	☐	☐
6	Update item data works as expected (if required)	☐	☐

7	Mark Exception works as expected	☐	☐
8	Mark Complete works as expected	☐	☐
9	Defer item works as expected (if required)	☐	☐

Basic Exception Handling

Objectives

- Prove exception mechanisms while working a queue item
- Prove exception mechanisms while not working a queue item
- Prove queue retry mechnism (if required)

Test	Description	Run in Studio	Run in Control Room
1	Recover exception while working item	☐	☐
2	Handle exception while working item	☐	☐
3	Record exception while working item	☐	☐
4	Recover exception while not working item	☐	☐
5	Handle exception while not working item	☐	☐
6	Generate queue item retry (if required)	☐	☐
7	Work queue item retry (if required)	☐	☐
8	Handle maximum limit of queue item retries (if required)	☐	☐

Process Data

Objectives

- Finalise queue item data structure
- Eliminate basic spelling, exposure and data type mistakes
- Confirm operation of credentials, variables and environment locks

Test	Description	Run in Studio	Run in Control Room
1	Confirm queue item data collection structure	☐	☐
2	Prepare any page inputs and outputs	☐	☐
3	Prepare data items for object inputs and outputs	☐	☐
4	Confirm operation of global variables	☐	☐
5	Confirm operation of session variables	☐	☐

6	Confirm operation of environment variables	☐	☐
7	Confirm operation of credentials	☐	☐
8	Confirm operation of environment locks	☐	☐

Process Notification

Objectives

- Confirm operation of any notifcations required by the solution design
- Note that not all (or even any) of these may be within the scope of the project

Test	Description	Run in Process Studio	Run in Control Room
1	Notify process complete	☐	☐
2	Notify process termination	☐	☐
3	Notify special case type	☐	☐
4	Notify SLA breach prediction	☐	☐
5	Notify SLA breach event	☐	☐
6	Notify queue volume	☐	☐
7	Notify ‘all work complete’ prediction	☐	☐
8	Notify ‘all work complete’ event	☐	☐
9	Notify log on failure due to expired password/locked account etc.	☐	☐

Process MI

Objectives

- Confirm operation of any MI reporting required by the solution design
- Not that not all (or even any) of these may be within the scope of the project

Test	Description	Run in Process Studio	Run in Control Room
1	How – confirm operation of MI retrieval method	☐	☐
2	When – confirm timing and frequency of MI	☐	☐
3	What – confirm accuracy of MI content	☐	☐
4	Where – confirm arrival of MI at destination	☐	☐

Process Build Checkpoint

After following the steps in the checklists above, the process should be in a usable state, where although it has no practical use yet, it has structure and can run in Control Room. The process is obviously far from finished but reaching this stage leaves the developer free to concentrate on the next phase – introducing target applications and developing the actual business process logic.

Business Object – Mechanical Tests

Objectives

- Incremental build and test of object pages
- Prove object effectiveness at Control Room speed
- Fail fast – find and fix errors early in the delivery phase
- Minimise application integration problems during process build

Object	Customer System – Basic Actions		
Action/Page	**Run in Object Studio**	**Run in Process Studio**	**Run in Control Room**
Launch	☐	☐	☐
Log In	☐	☐	☐
Log Out	☐	☐	☐
Exit	☐	☐	☐
Attach	☐	☐	☐