blueprism

Software Robots - the Virtual Workforce

Blue Prism Learning

PROCESS SOLUTION DESIGN

VERSION: 1.0.7

For more information please contact:

info@blueprism.com | UK: +44 (0) 870 879 3000 | US: +1 888 757 7476

www.blueprism.com

Contents

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

1. Introduction

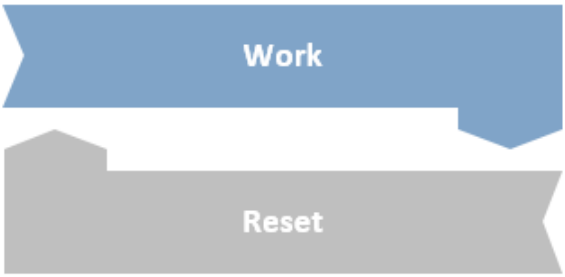
Blue Prism is a platform for creating automated processes and executing them in a secure, virtual environment. An automated process should be able to run unattended and unobserved, and this document sets out to explain how to make this principle central to a solution design.

2. Designing for Unattended Automation

A Blue Prism process needs to be much more than a series of manual steps executed automatically. It needs to be robust enough to work unattended, be able to recover from unexpected situations and work concurrently on multiple machines. This realisation is often missed when learning to design a Blue Prism solution.

After focussing on the definition of the manual process in the PDD, a common mistake is to envisage the automated process merely as a linear sequence of work steps.

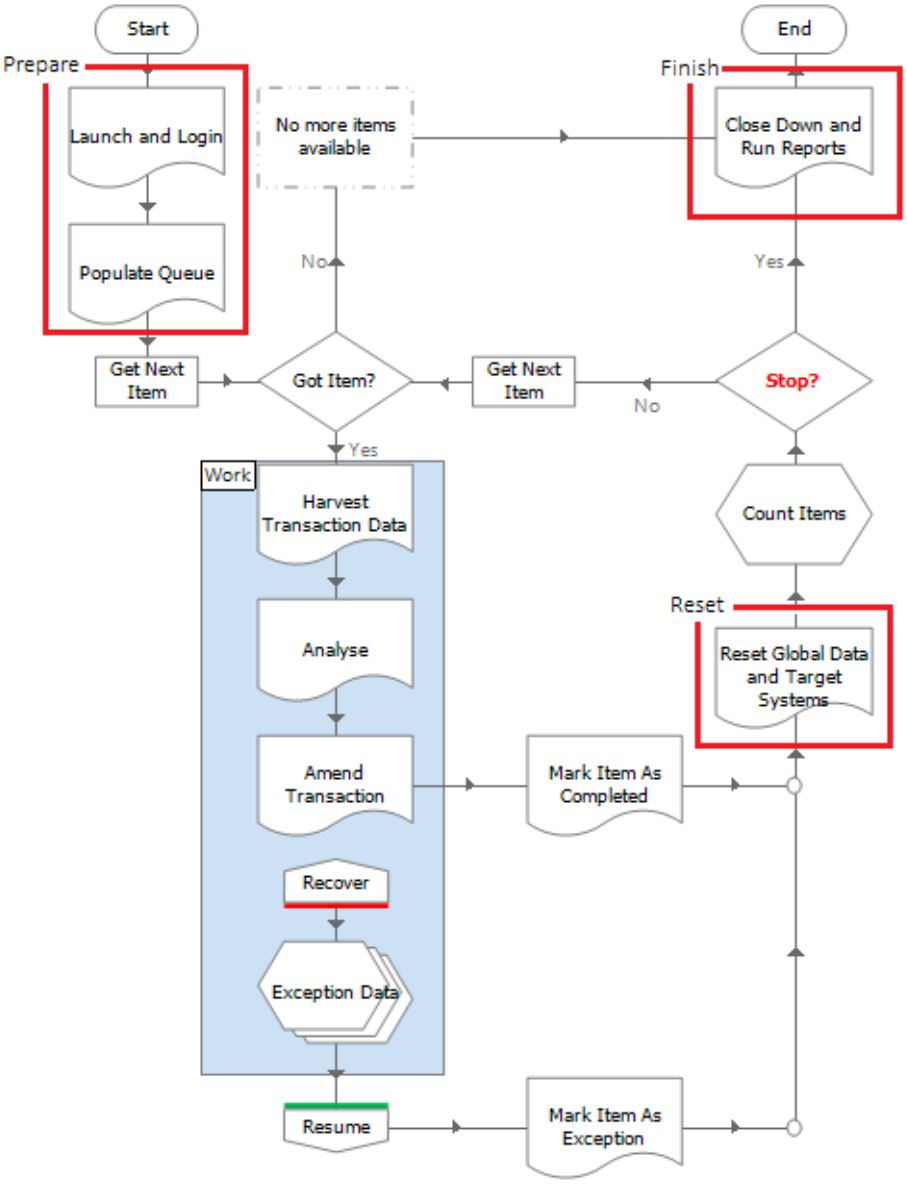
This is too simplistic because the objective of the process is to work one case after another, and after completing one case it will need to return to the start position in readiness for the next. So the process will need to maintain control over the applications and data it is using, and these reset steps may well involve navigating back to a main menu screen and resetting variables.



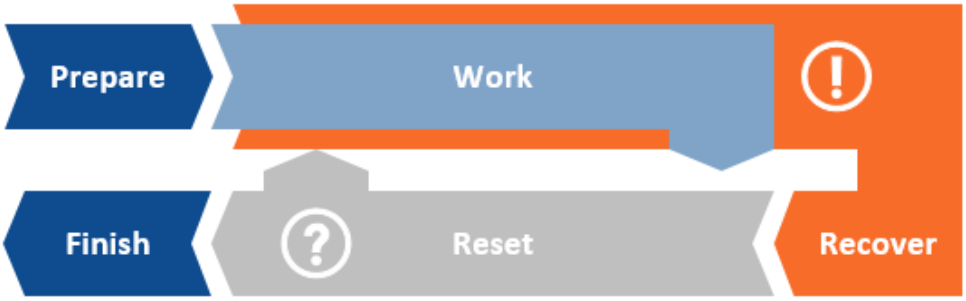
At some point the process will also need to break out of this loop, and often this is simply when there are no more cases left to work but the decision could also be made for other reasons, such as when the end of the working day is reached.

It is also likely that there will be additional steps before and after the working loop. Examples of preparation steps are things like logging in to applications and gathering input data. The finishing steps could be something like logging out and closing down the applications, creating a report or sending an email notification.

To illustrate this consider the following process constructed using the Blue Prism Basic Template. The three areas outside of case processing (work) are clearly marked.



Another common oversight when designing a process is to assume that it will never encounter any problems and always stay on the 'happy path'. Again, this is unrealistic and recovery steps should be included to cater for the 'unhappy path' logic, as shown below in orange.



Here exception handing is employed to make the process recover from unexpected application behaviour, so that cases can be set aside for manual investigation or if necessary, reworked by the process. The recovery logic may also need the ability to restart applications for the process to continue working.

3. Solution Types

There are different types of automated solution that can be applied to a manual process. Consider this abstract process, comprised of manual steps 1 to 9.

3.1. Full Automation

The ideal scenario is when all manual steps can be automated and that the entire manual process can be replaced. Note that for simplicity the effort in manually resolving exception cases has not been illustrated here.

3.2. Partial Automation

If complete automation cannot be achieved, then it may be possible to automate a sequence of contiguous steps within the original manual process. Ideally the effort in the remaining manual steps would be offset by the savings brought by the automated steps.

Often in this kind of partial automation an adjustment to the remaining manual steps is required to enable the automation. For example, in step M1 the user may need to prepare structured data for step A2 to consume. Similarly, in step M9 the user may need to become the recipient of a handoff from A8.

3.3. Fragmented Partial Automation

If a continuous automation cannot be achieved, then maybe automation can be applied to more than one sequence of steps.

Here the links between the manual and automated steps need to be considered when weighing up the overall benefits. If the handoffs require new manual effort, then that may negate idea of automation, but if a synchronised interplay can be achieved easily, then then a fragmented automation may work.

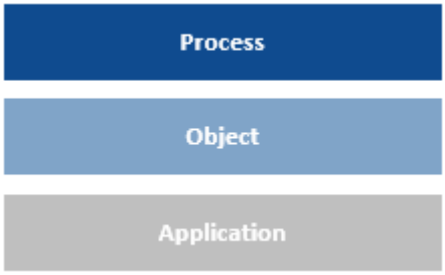
3.4. Restructured Partial Automation

To avoid a fragmentation it may also be possible to re-engineer, or re-order some of the manual steps to facilitate a continuous sequence of automation.

Here steps 1, 4, and 5 have been arranged into a manual sequence to enable 2, 3, 6, 7, and 8 to be automated in one straight-through process.

4. Solution Layers

A Blue Prism automated solution can be perceived in layers, starting with the process layer, down to the object layer and the application model, and through to the application layer.

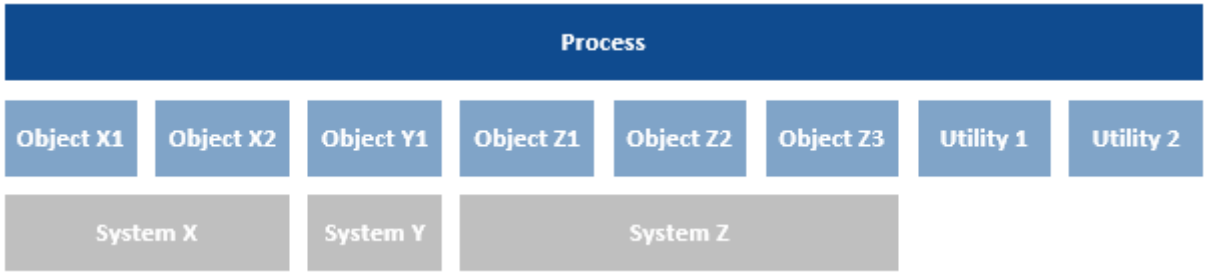


The role of the process layer and the object layer can be confusing at first, but in essence an object should be seen as a tool that provides a process with the mechanics to manipulate the application. Business logic, rules and decisions should, in general, be in the process.

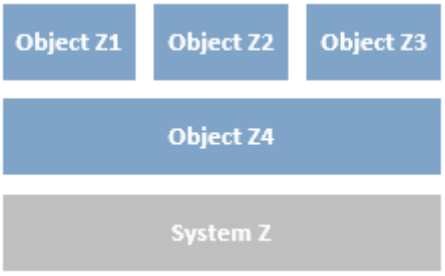
4.1. Objects

A business object is the instrument that a process uses to control an application. Ideally the object should offer a set of simple functions that the process can orchestrate into a complex sequence. By absolving the object of responsibility for business rules and decision making, the aim is to enable the objects to be reused by different processes and for an object 'library' to be built up. And as the diversity of the object library increases, the effort to deliver an automated solution should decrease.

The object layer in an automated solution is likely to comprise of more than one object, often with multiple objects used to automate different aspects of the same application. Utility objects, used to provide generic functionality unallied to any application, may also be involved.



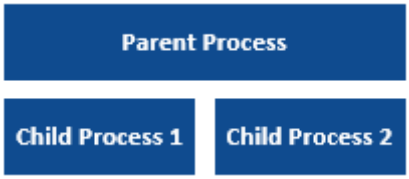
Objects can also use other objects, and in some situations such as with Surface Automation, it can be desirable to create a 'base' object to interact with the target system that other objects then employ.



Not all objects need to interact with a target system. For example you may require a rules engine that could be utilised by multiple processes. This could be developed in an object that has an action that accepts data via start-up parameters then interrogates the data before passing back a decision via an output parameter.

4.2. Sub-processes and Wrapper Objects

A process can also call another as a child, enabling layers of processes to be created.

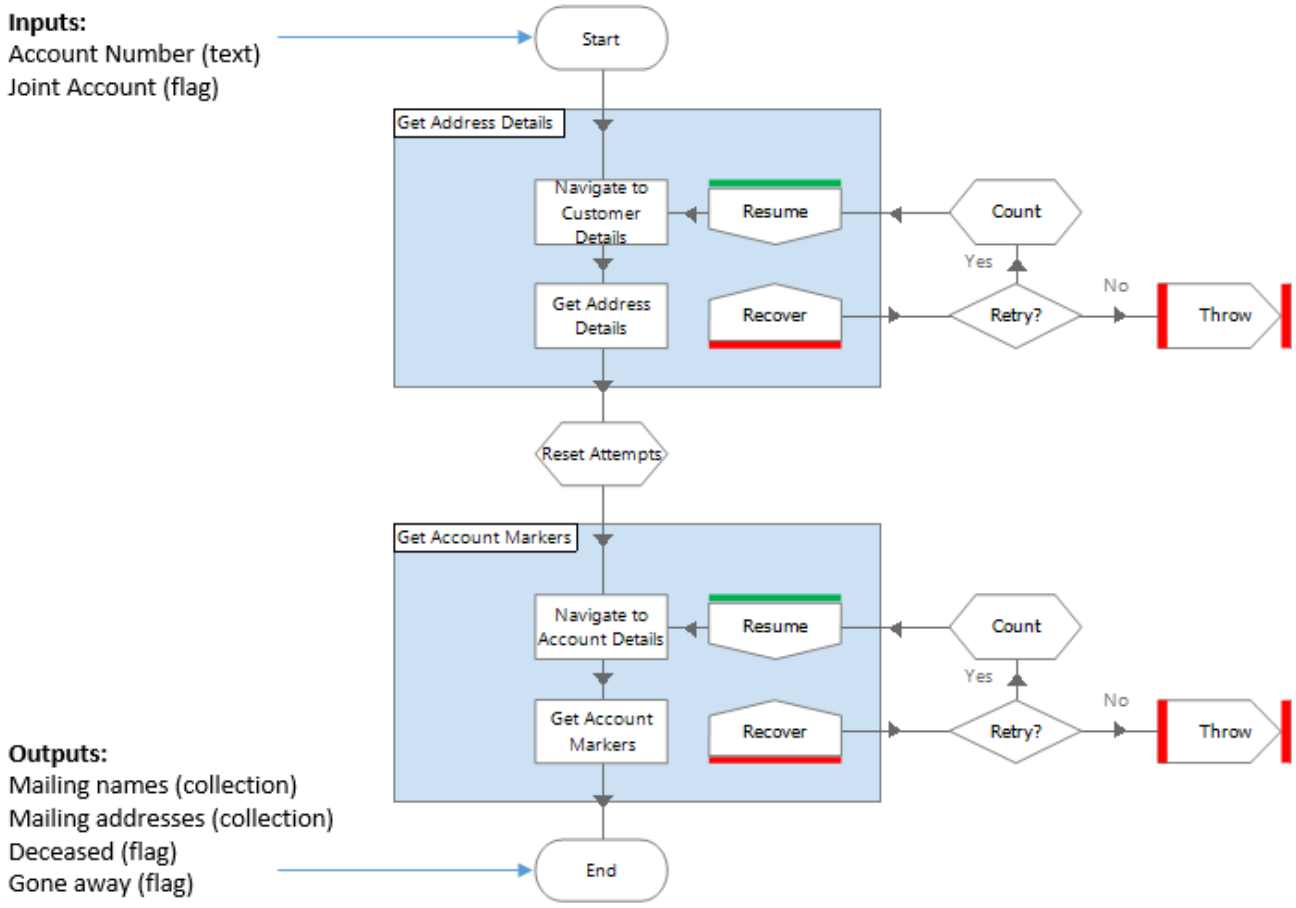


However, the way in which Blue Prism manages its memory needs to be considered carefully when thinking about using a child process. Unlike an object, a child process is not preserved in memory by the parent; once the child process has ended, the parent process releases the memory for the .Net Garbage Collector to reclaim. Because of this, if the child process is being used repeatedly by the parent, it is possible for unwanted memory leaks to build up if memory is being consumed by the child process (and its objects) faster than the Garbage Collector can clean up.

It is therefore recommended not to use a child process where it will be called repeatedly, e.g. in the case working loop. As an alternative use objects to wrap other object's actions.

An example might be if you have a common sequence of steps when harvesting customer mailing address details before sending a letter. Instead of replicating this in each process that orders customer letters you could wrap all the actions together in a standalone object that could be called by any process.

The action below is from an object that does ***not*** interface with any target systems. However it wraps four actions from three different objects that do.



5. Design Basics

5.1. Recoverability

In simple terms, recoverability is the capacity of the solution to handle problems and return to normality. It is naïve to assume that the happy path is the only possibility and provision should always be made to attempt recovery from unexpected situations.

System Recovery

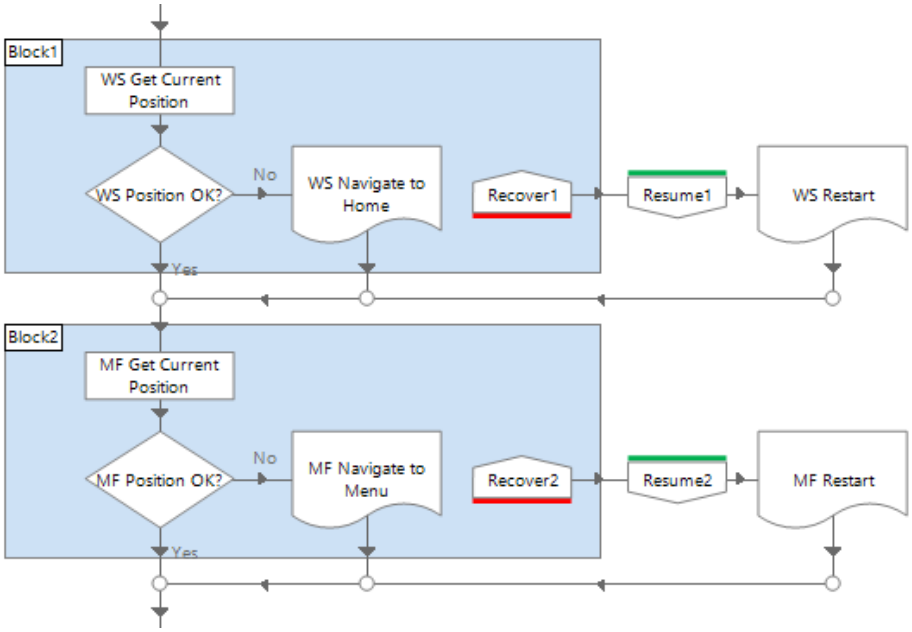
It is vital that an automated solution maintains control over its applications at all times. When all goes well, this is relatively straight forward and the only thing to make sure of is that the applications are back at the starting position before working the next case.

The difficulty comes when applications aren't behaving as expected and control has been lost. Typically this will manifest itself as a 'time out' or a 'failed to find element' error from an object. The resulting exception can be managed with exception handling, but the applications must still be brought back under control if the process is to continue.

On some applications, this can be easy – for example on a web application it might just be a case of navigating directly to the home URL, or on a mainframe there might be a universal keystroke to reverse from any position. Sometimes though, navigating back is not that simple, and it may be necessary to restart the application. Again, this might be painless and the application can just be terminated and relaunched. Some applications however do not like to be treated roughly and require that the prescribed log out procedure is rigidly followed.

Each application must be carefully assessed in anticipation of including a recoverability section in the solution. And although system recovery is not explicitly captured in the Solution Design Document or Process Design Instruction, the developer will need to appreciate the concept when configuring robust solutions.

Below is an example of recovery logic used to make two applications (Web System and Mainframe) ready, prior to working the next case. The current position of the application is determined, and if it isn't as expected, or the attempt fails, the logic will try to recover. The 'navigate' page will make several attempts before throwing an exception and triggering a restart. The 'restart' page will also make several attempts, but any exception it throws is deliberately left unhandled in order to cause a termination. Although it would be undesirable, a termination here is necessary to show that control has been lost and the processing cannot continue without attention.



Case Recovery

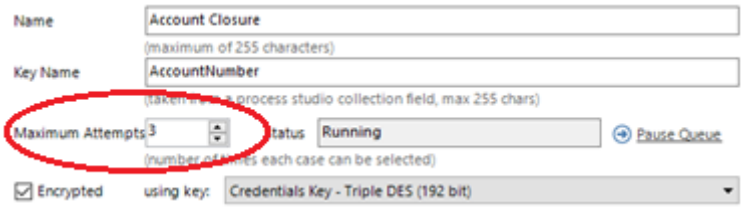
Case recovery is the ability to resolve a case after a process restart. For example, if an external issue causes Blue Prism to fail completely while the process is midway through a case, what will happen to that case? Is it enough to simply say that it will be handed over to a manual team as an exception? Or, should the process have the ability to continue the case from the previous position, and if so, how will that be achieved?

The current state of the case can be recorded by updating the queue item as the case moves forward. Then in the event of a sudden shock, when the case is picked up again, either by a Blue Prism process or a manual team, its previous level of progress will be known.

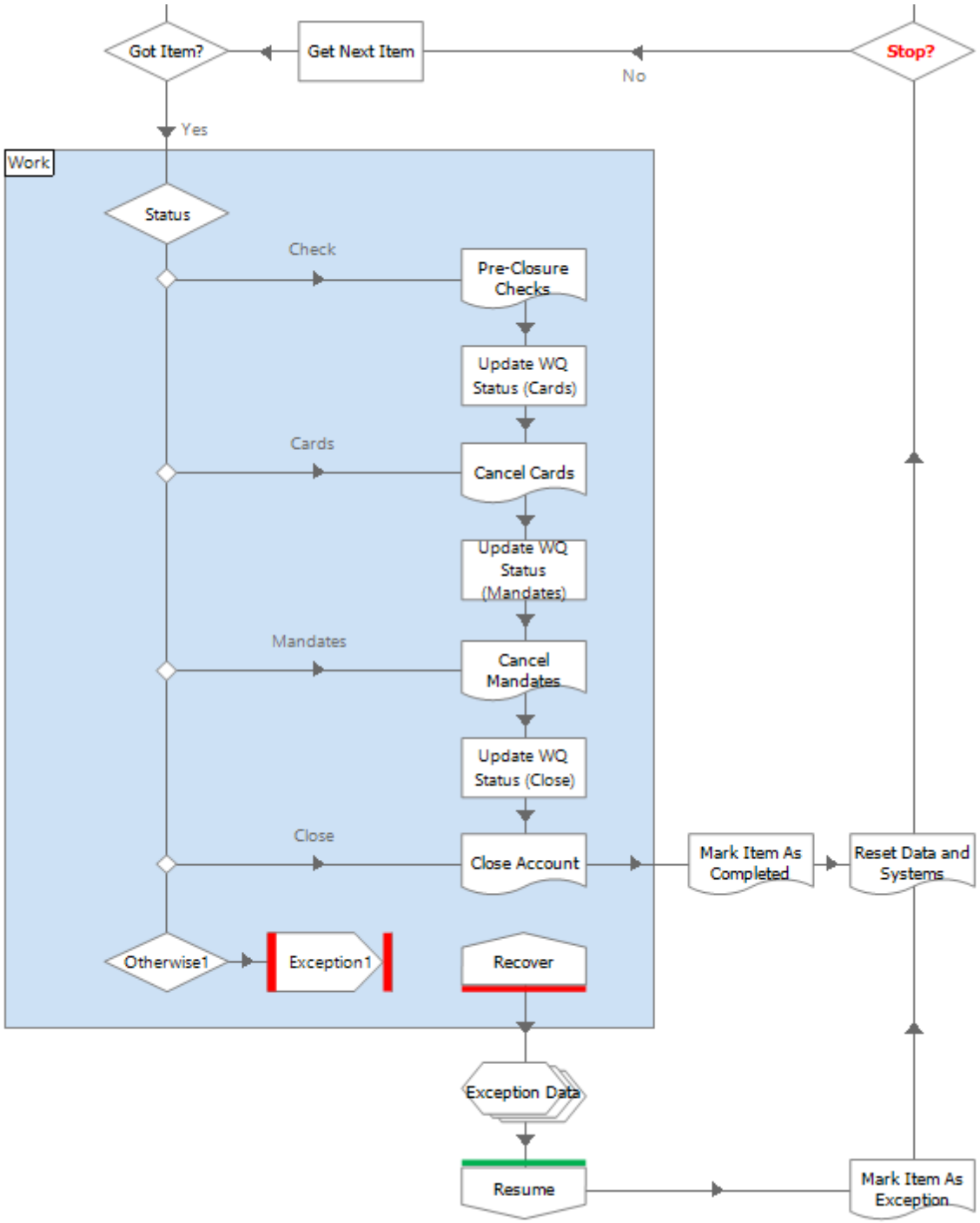
In the abstract below, the state of the case is recorded at each step until a problem occurs after step 3.

When the case is reworked it can be started at step 4.

To illustrate this consider the following example. The Account Closure work queue has been configured to allow up to three attempts per case. Blue Prism uses this property to clone a queue item when it is marked as an exception, creating a new pending item.



The Account Closure process below has been designed to use the queue item Status property to track case progress. There are four steps to the process – Check, Cards, Mandates and Close, and following each successful step the status is updated. This ensures that any retry case presented to the process will navigate to the correct next step. Additionally, when exceptions are sent for manual review, the status value will indicate to the manual team where the case was when it failed.



Database Recovery

How a Production database would be recovered may need to be considered as part of the solution design. Where a Production database is not replicated or mirrored in real time, it is possible that a back-up will need to be restored in order to recover from a disaster. If this was ever to happen, then cases that were worked after the backup was taken could appear as unworked in the queue, and the effect of the solution attempting to rework these cases would need to be considered.

Consider an example where 1000 cases have been loaded to a work queue and midway through processing there is a critical database failure. A backup service provides incremental backups every four hours, but when the latest back up is restored, all cases are shown as pending. Simply restarting the process would rework the 500 already worked cases.

Mitigating the effect of restoring an aged production database is best achieved during the design phase. Depending on the process, there may be some natural defence in the target application. For example, with a process for cancelling customer contracts, if the target application did not allow an already cancelled contract to be cancelled again, the process could be designed to simply mark such a case as completed.

However, in the example of a financial process that transfers funds, then reprocessing cases could disastrously make duplicate payments. But if the process was designed to look for and create 'footprints', for example by applying a recognisable note to the account at key stages, then it could check to see if a footprint exists and avoid repeating a critical step.

5.2. Scalability

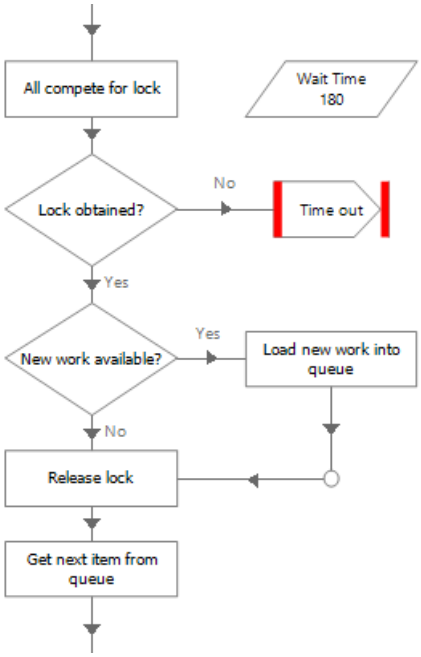
A common oversight when designing an automated process is to forget that the process may need to run concurrently on multiple machines. Although the Blue Prism work queue prevents multiple process instances accessing the same queue item, the steps outside the case working loop may need special attention if the process is to run on more than one machine.

For example, take the common scenario where a process is to start by consuming a daily input file and loading a work queue. If three instances of the process are running, what will happen? Will they all try to read the file and load the queue in triplicate? Or suppose a report is to be created at the end of the day – how do you avoid duplicate reports from being created?

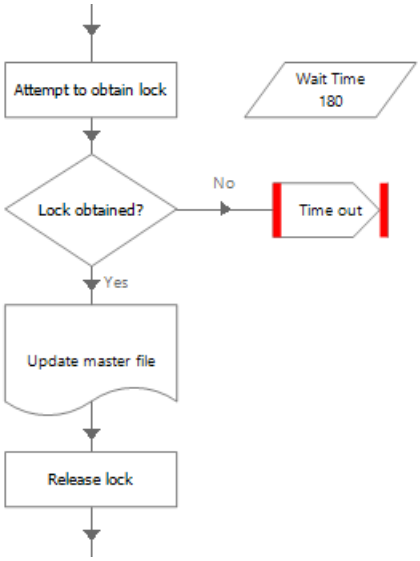
Environment Locks

Environment Locks are a Blue Prism feature that enable a 'permission' to be shared between processes (and objects).

Process instances can be made to compete for permission to ensure that only one instance performs a particular step. For example, imagine that the first step of the process is to read a data source and load the work queue. Three instances of the process start at the same time and all compete for possession of an environment lock. There can be only one winner and that instance loads the queue, while the losers wait. Once the lock has been released, the losers compete for it again.



Another requirement might be to control the number of instances capable performing a particular step simultaneously. For example, imagine a process that updates a file that can cannot be opened by more than one user. An environment lock is used to ensure 'one at a time' access to the file and make instances of the process wait their turn to update the file.



5.3. Reusability

Reusability is a key tenet of the Blue Prism RPA delivery methodology. By carefully designing objects, wrapper objects and sub-processes, a library of reusable logic can be built up and delivery effort can be reduced and maintenance overheads minimised.

Objects should be free of 'business process logic', and object pages should be small, executing simple, mechanical steps. The function of an object is to provide processes with the tools to manipulate an application, and the more generic the object logic is, the more likely it can be reused by different processes.

For example, if the first step of a business process is 'Open MediSys and get patient details', then the first instinct may be to create an object page that does just that. However, if another process began with 'Open MediSys and get clinician appointments', then the existing MediSys logic could not be reused without changing (and retesting) the object.

A better tactic would be to design a series of pages that each performed a basic step, eg, 'Launch', 'Log In', 'Find Patient' and 'Read Patient Details'. That way, the second process could take advantage of some of the logic created for the first process, eg 'Launch' and 'Log in'. So by increasing the granularity of object functionality and having pages that perform 'atomic' actions (like Read, Write or Navigate) on one screen of the application, the more reusable the object is likely to be.

In general large objects should also be avoided. Although an object with many pages will function, it can bring inefficiencies: more network bandwidth will be required to transport a large object across the network; users will commit the whole object in to PC memory regardless of how fleetingly they use it; only one developer can work on the object at any one time; and as dependencies on a single object grow, the impact of a fault in the object affecting multiple processes increases.

To avoid these problems, it is recommended that application integration logic is separated into a series of small objects rather than concentrated into a single large object. This approach offers multiple advantages: lightweight objects consume less bandwidth, memory and disk space; a group of objects make it easier for application integration to be developed by a team; and the potential effect of a corrupted object is minimised.

For further details please review the following where sample Object Design Instructions can be found:

- Solution design example documents
- Blue Prism Delivery Roadmap within Lifecycle Orientation

6. Case Management

6.1. Reset

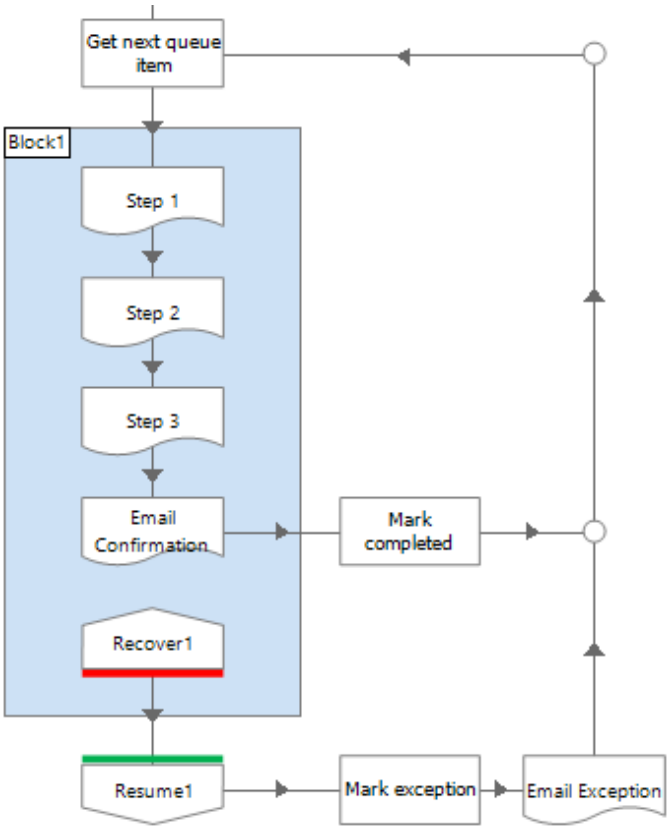
Almost every process will contain a main loop where one case is worked after another. Data items will be reused and therefore care must be taken to ensure that values from a previous case are not inadvertently used in the next case.

As mentioned above, a process keeps objects in memory during its lifespan, so it is possible that data values within the object layer will persist. This can be desirable if the data is not case-specific, but it is important to avoid old case data affecting the current case.

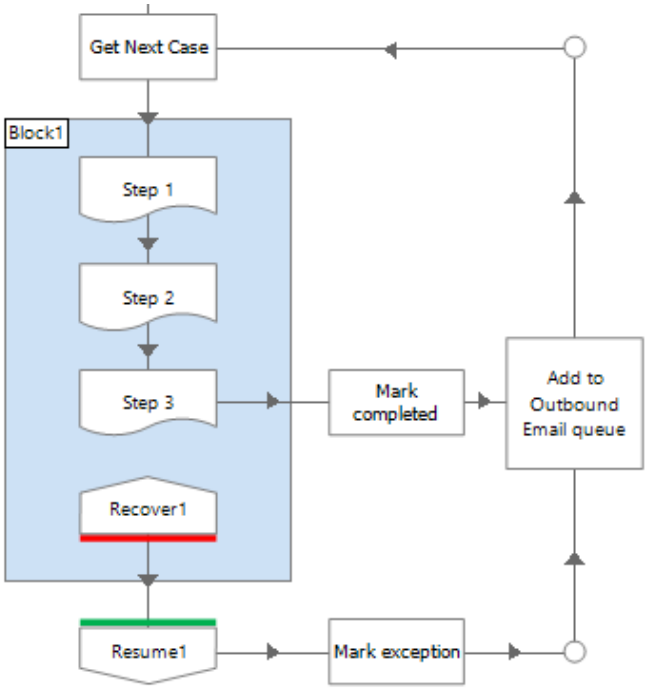
6.2. Resilience

Like the ability to recover a case after a restart, the design of a solution can improve resilience and reduce the number of exception cases. Multiple queues can be used to subdivide a case into a series of tasks. For example, if it is imperative to communicate a result at the end of each case, maybe by email or web service, it may be better to consider this task as a separate case in a secondary queue. That way, if necessary the task can be completed at a later time, or delegated to another machine.

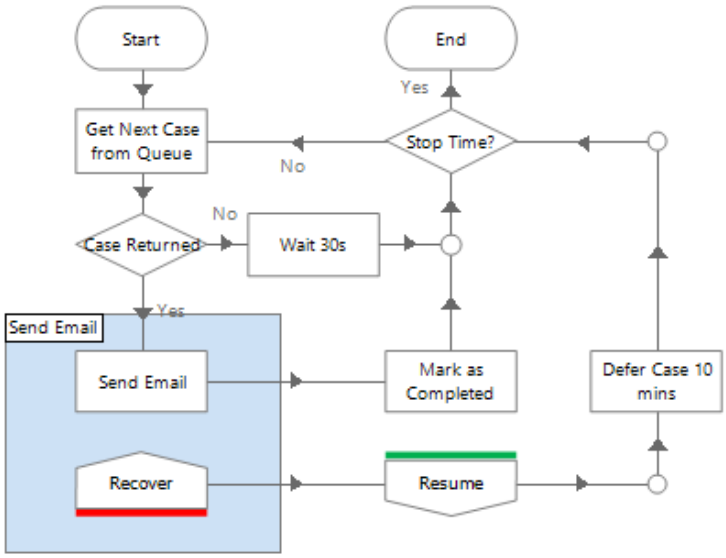
In this (deliberately simplified) example, the process works a case and emails the result. Here, if there is an exception thrown when emailing a confirmation because for example the mail server is unavailable, the case will be marked as an exception and sent for manual review. This is unnecessary as the case has essentially been completed.



But in this example below, the process does not send email and instead adds messages to a work queue for another process to send.



A separate 'Send Email' process works in parallel to the other process. This process is able to send email on behalf of any other process that adds items to its queue. If there is ever any failure sending an email then the case is simply deferred and retried later. If there is a critical issue with email gateways then once they are fixed the emails will be sent.



7. Workload Management

7.1. Accountability

Almost every automated solution will consume some sort of workload, whether reading a file, polling an inbox, a web service etc. And inevitably this data will be loaded into a Blue Prism work queue, always for audit and MI, but also as a means of safely distributing the work between multiple instances of a process.

To this end, thought must be given to designing a solution that is able to account for all work, that no cases are lost, none are duplicated and all results can be substantiated. Once implemented, the solution will become part of the wider Operation, so it must be designed to not only reliably consume work provided to it, but to also report trustworthy results back to the Business. As a minimum this communication should be for exception cases and referrals, but may also need to include details of completed cases too.

To achieve the required level of accountability it may be necessary to update the input data information gathered while working each case. This could be something as simple as setting the queue item status field, or applying a tag, but could also involve writing data into the queue item data collection.

7.2. Balance

It is not uncommon that an automated solution will be required to get its work from another application, such as a workflow tool or database. It may also be required to return to the source and update it with the results of its labours. Clearly care must be taken by the designer to ensure that the data source and the automated solution are kept in balance, and that no cases go unworked, duplications cannot occur and results do not go unreported.

Some designs may necessitate the use of multiple Blue Prism work queues, so again, consideration must be given to ensure that equilibrium is maintained.

7.3. Overload

If it is possible that spikes in workload can occur and/or SLAs are tight, then there may be justification in designing a mechanism to detect and notify of an impending problem. For example, by multiplying the number of pending items with the average case time it is possible to compare a completion time forecast with an SLA.

Equally, an unusually light or even non-existent workload might also warrant an automated alert or trigger an alternative course of action.

8. Data Management

8.1. Preservation

The functional requirements should indicate how long solution data is to be preserved, but it is important to realise that some types of data will persist indefinitely unless some sort of control is implemented. Session logs can be archived using System Manager and an Archiving Policy should be in place to limit the growth of log data.

Work queue data however will remain in the database unless an automated mechanism is created. This may be in the form a single 'Manage All Queues' process, or the functionality of the 'Internal – Work Queues' object could be used to enable each process to maintain its own queue. Commonly 'maintenance' is simply a case of deleting aged items that no longer serve any purpose, either as part of the workload, MI or audit.

Similarly, the longevity of any input and output files must be addressed and an automated control procedure should be considered.

8.2. Security

Some scenarios may require handling of sensitive data and this may influence the design of the solution. At a basic level, encryption should be enabled on a work queue and care taken not to expose key data in Control Room or in the session logs. More seriously, perhaps there is a security policy prescribing key data from being stored anywhere in the solution. Such a requirement may compel the solution to be designed so that the data is entirely transitory and never persisted in queues, in logs or anywhere else. A credit card number for example, may need be fetched 'just in time', with great care taken to ensure no trace is left once the case has been worked.

9. Efficiency

9.1. Integration Efficiency

Often there are alternative application integration options and the pros and cons of different techniques should be weighed up. Perhaps an application has an API or web service that could be used instead of modelling the user interface. Or perhaps a file can be read directly, without opening it as a user would.

The natural tendency is to faithfully replicate the manual steps but there may be instances where Blue Prism could take advantage of a mechanism not available to a human. At a basic level, a robot is able to hold more information in memory, so where a person may need to move back and forth to harvest information from two applications, a robot can read everything it needs in one go.

Fundamentally the automated solution does need to follow the 'as is' manual process, but there may be scope to increase efficiency where the robot is more capable than the human.

9.2. Exception Efficiency

An automated solution should anticipate cases that cannot be resolved automatically, either for technical reasons or because they are out of scope. The design should also consider whether the manual resolution of these cases could be assisted in some way. For example, maybe the queue item Status field could be used to indicate different exception categories. Or, if system data read while working a case was regularly saved back into the queue, this additional information could be included in an exception report to aid manual workers by eliminating the need for them to traverse the systems to re-harvest the case data.

There may even be situations where it would be worth continuing working *after* the exception occurred, perhaps by fetching data to minimise the subsequent manual referral effort, or maybe even to continue working the case before ultimately marking it as an exception.

Where parent/child (i.e. nested) cases are used, it could be more efficient to continue to work child cases after an exception, the rationale being that it is better to mark the parent as an exception with 90% of its children worked, rather than stop immediately.

9.3. License Utilisation

License usage should also be considered when designing a solution comprised of multiple processes. Separate processes may be the optimum configuration, but where licenses are at a premium, thought should be given as to whether a single process (perhaps making use of sub-processes, environment locks or multiple queues) could be used.

10. Notification

The client's requirements may stipulate that certain events should be alerted, and as such the design should make provision for when, where and how these notifications are to be issued.

For example, the Control team may want to be notified of key events like a process termination. Perhaps a process completion should be alerted to the Operation so that they can pick up from where the automation finished. Or, more seriously, maybe an SLA breach needs to be alerted to trigger a contingency plan.

Whatever the reason, how the solution is to communicate needs to be considered as part of the design. Email is typically the preferred method but SNMP messages to a helpdesk could also be used. Alternatively, notifications could be issued as updates to another application such as a workflow tool or MI database.

11. Design Procedure

The delivery of a Blue Prism solution into a production environment follows a standard path of Define- Design-Build-Test-Implement, familiar to many IT projects. Blue Prism offers a proven design methodology with accompanying document templates, and trainees are introduced to these as part of the Lifecycle Orientation module. These templates are not prescriptive however, and some clients are more comfortable adapting their own methodology and documentation to work for RPA. It is the purpose of the documentation that is important, the documents themselves are merely vehicles - for defining requirements, exposing detail and for reaching agreement.

11.1. 'As is' Definition and Requirements

As mentioned elsewhere in this document, the Process Definition Document (PDD) is used to describe the 'as is' manual process. This information may already exist in Statements of Procedure (SOP) or other process documentation, but it should be in sufficient detail that an unthinking robot could be instructed to work the process.

In tandem with the definition of the manual process, the process owner's wishes for the automated solution should be documented. The Functional Requirements Questionnaire (FRQ) isn't used to describe how the automated solution will work cases, but rather it is a means of interviewing the process owner to extract high-level requirements of how they would like the solution to operate. For example, should it run during business hours Monday to Friday, or to some other schedule, how should exception be communicated?

11.2. 'To be' Design

With the PDD and the FRQ complete, the designer can translate the information into Solution design Document (SDD). We have found that separating the 'as is' and the 'to be' into different documents to be a great defence against misunderstandings and omissions. As with the PDD, walking through the SDD in a workshop scenario is also a great way of not only explaining the proposal but also of exposing any shortcomings.

The SDD also gives the Development Lead the chance to assess the quality of the proposed solution and check that full use of any existing logic (i.e. an object library) is being used and valuable project time will not be wasted. With the design signed off, development can be initiated via the Process Design Instruction (PDI) and Object Design Instruction (ODI). Both these documents are a means of the designer informing the developers how and what to build.

11.3. Shock Proof Design

A design should be tested by theoretical 'shock testing'. What is meant by this is to imagine how the solution will respond to a sudden blow, for example a network outage.

Imagine that a process is midway through a case and all network communication is lost, database connectivity is lost, and the process terminates. When the network is restored, and the process is restarted, what will happen? What will happen to the previous case? If the process reworks it, will it duplicate steps or pick up where it left off? If applications have been left open, can the process deal with that? As well as a sudden event occurring mid-case, what if it happened while loading the queue, would duplicates be added or cases lost once the shock had abated?

Although such imaginings may seem pessimistic, they may well be necessary if the consequences of a malfunctioning solution are serious enough.

11.4. Application Assessment

Whenever a new target system comes into scope it is always worth while performing an application assessment. This simply requires running the Application Modeller spy over the elements of the new system to be automated to see how much can be identified and how easily. Typically for Windows, browser and Java applications this activity requires a developer to determine the most appropriate technique for interfacing with each element type (combo box, check box, table, lists etc.). This will help estimate development effort and provide initial approximations of case handling times.

11.5. Project Types

Proof of Concept

The aim of a POC should be to demonstrate the potential of automation rather than to produce a fully autonomous solution capable of unattended automation. To that end the design should focus more on working the in-scope cases and less on the features required of a Production solution, such as scalability, application control, exception handling, notifications and MI.

Evidently the solution needs to work, but it does not need to be a full-strength Production piece and most likely will only be run for a limited period often in attended mode for purposes of demonstration. The design should take into account what can realistically be delivered in the time available and remember what the overall purpose of the project is. Following any proof on concept a design review must take place, final design documented and further development incorporated to ensure the process is production-strength and ready for formal testing.

Pilot

A pilot is a process that is built to run temporarily against production cases. Similar to a POC, the design of a pilot project should reflect the aim of the project. The objective of a pilot may not be to produce a full-scale solution, and so the agreed scope of what a solution will and will not do should be covered in the design.

If the pilot is based on an earlier POC then care must be taken to revisit the original design. As mentioned above, for expediency a POC is likely to have limited the robustness of the solution and any subsequent pilot will need to make improvements.

11.6. Design Authority

The principle activities of a Design Authority are to define and govern:

- The design process
- The design documentation to be used
- The design review process

The design authority must decide on how common technologies should be used such as:

- Email – email clients or SMTP?
- Workflow systems – how will Blue Prism utilise existing workflow systems to interact with manual users?
- Third party code – how should this be wrapped and exposed to Blue Prism.
- Web services

The design authority must define design and development conventions e.g. naming conventions for processes, objects, work queues, environment variables, data items etc., prescribe logging policy and data storage policy and be the custodians of development best practice.

Appendix

12. Design Review Checklist

12.1. Solution

- Has the 'as is' manual process been defined and documented with enough detail to design an automated solution?
- Are the client's requirements documented and agreed?
- Are the requirements met by this design?
- Does the design enable the client to understand and sign off the 'to be' automated solution?

12.2. Process

- What logic exists outside the 'case working' phase?
    - Is there need of a 'Preparation' phase?
    - Is there need of a 'Finalisation' phase?
    - Is there need of a 'Reset' phase?
    - Is there need of a 'Recover' phase?
- Does the process use a Blue Prism work queue?
    - If not, why?
- If the process runs on multiple machines in parallel, what will be the effect?
    - Are there any steps that must not be executed concurrently by multiple machines?
    - Are there any steps that must only be executed once, regardless of how many machines are running?
- Where are the passwords used by the process?
    - Are they kept in the credential store?
    - Are any hardcoded in any diagrams?
- When will the passwords expire?
    - How will password changes be managed?
- How will the process stop?
    - When the queue is empty?
    - At a specific time?
- Is it possible that after an exception applications will not be in an ideal state to work the next case?
    - How will that situation be rectified?
    - Will it be necessary to restart an application?
- How will exceptions be managed?

    - Are queue retries going to be used?
    - How will exceptions be sent to the Business?
    - How will exception rates be monitored?
- Do queue results need to be replicated in another Blue Prism queue, workflow application, database or file?
    - How and when will that be done?
    - Is it possible the two sides will become unbalanced?
- Is the process required to send notifications?
    - How and when will this be done?
- Are sub-processes in use?
    - Is there any risk of memory leak?

12.3. Objects

- Have the objects required for this solution been itemised?
    - Is it clear which objects already exist and which objects need to be created?
    - Is a checkpoint in place to review the design before development work starts?
- Would it be possible to reuse the objects created for this solution?
    - If not, why?
- Is there any 'business process logic' in the object layer that would negate reuse?
    - Should this logic be in the process layer
- Other than System Exceptions, do any objects throw exceptions that require special treatment by the process?
    - Is this made clear to the user of the object?
    - Would an output parameter be more explicit?
- Are there any overly complex pages that could be broken up?
    - For ease of use
    - For ease of reuse
    - For more effective testing
    - For increased efficiency
    - For better security
- Do the names of objects and pages give the process developer a good idea of their purpose?
    - Are any names meaningless or vague?