Robotic Process Automation Development Guidelines and Best Practices

This document outlines development best practices and guidelines for all Blue Prism robotics projects within Capgemini Norway. Its intended use is primarily for RPA developers and technical resources. However, Business Analyst working on RPA projects should also read through it for a better understanding of how the development works.

Introductin

Introduction of document and purpose

This document is created for Blue Prism (BP) developers with the purpose of creating a standardized best practice when developing in BP.

By implementing the best practice guidelines in everyday development, we strive to see:

- Efficient knowledge and content sharing
- Foundation for scalability
- Easier and improved maintenance
- Code and configuration reusability
- Easier deliveries/releases

Please use this document as a lookup. For changes and suggestions, please contact document owner, Sina Hassani.

On some projects the customer has their own best practice development guidelines. In those cases, you as a consultant must adjust to their practices. If the situation allows for it, suggest changes or improvements to the customer.

Revisions

Author	Version	Description	Date
Sina Hassani	0.1	Document Structure	22.10.18
Sina Hassani	1.0	First draft complete	07.11.18
Sina Hassani	1.1	Minor design / resolution changes to pictures	08.11.18

Table of Contents

1. Process vs. Object studio

This Section focuses on the difference between process and object studio. It will not cover or explain in detail what they are, as this should be covered in the foundation training.

Process studio:
The process studio is where you call upon your objects and create the automation for the solution you are developing.

Here is where you handle all the business logic. Meaning, this is where you should handle exceptions and retries of the automation case.

Object studio:
In the object studio the developer focuses on developing *reusable* objects. That way other developers can use the same object in other cases where the same programs are used, or the same automation is needed.

2. Camel Case

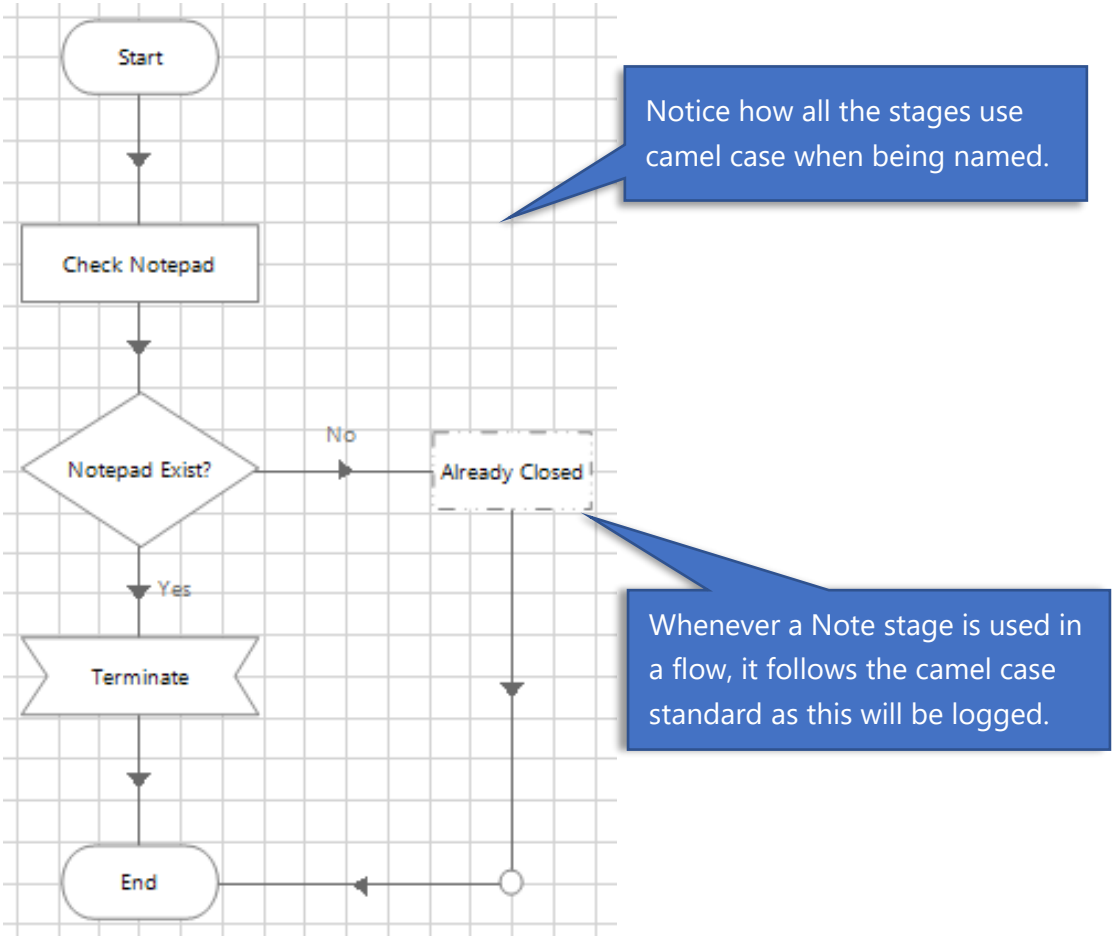
This part covers an important part of the development, that is often overlooked, camel case.

Before moving on, it is important to know how to do proper naming convention of stages in the development pane in both process and object studio. The reason this part is important, is the continuity and uniformity for other developers that might work on the same solution.

Variability and inconsistencies often lead to confusion, error and loss of time. It is therefor paramount that all developers follow the same rules and guides while developing.

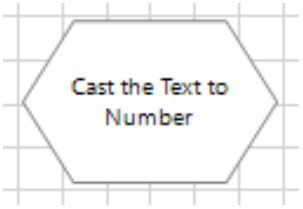
Details:
All stages (Action, Decision, Choise, Calculation, Data Item, Collection, Loop, Wait etc.) on pages (process layer), actions (object layer) and Application Modeller must be written in Camel Case (aka Upper Camel Case, Paskal Case). In short, each first letter in every word is capital.

2.1 Example



2.2 Exception

Words like “the”, “of”, “for”, “to”, “and” etc. are exceptions to the camel case standard and will remain as lower case.



Think of movie titles like “The Lord of the Rings” .

3. Flow

This section covers how the flow of a process should look when developing in process- and object studio.

When developing flow the same way across in Capgemini, it will make it easier for other developers to recognize and understand your logic and solution.

3.1 Flow Goes Down

Unless it is the process main page, a retry loop or there are multiple End stages that branches out of your flow, the flow should always go downwards in your process- and object studio.

4. Process Studio

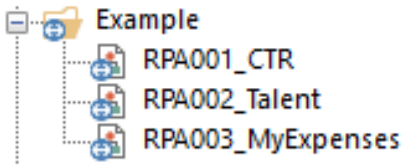
This section covers the best practices when developing in BP Process Studio.

4.1 Naming convention

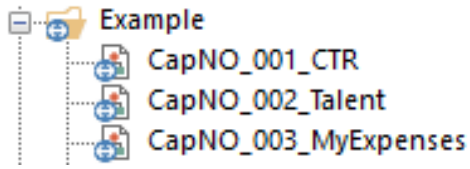
Depending on projects, the naming convention will most likely differentiate. However, the standard we follow in Capgemini is **process number** followed by **process name**.

4.1.1 Example

[Process Number]_[Process Name]



Some projects use the **Jira Epic Number** or **Country** with, or without, numbering process:
[Country]_[Process Number]_[Process Name]



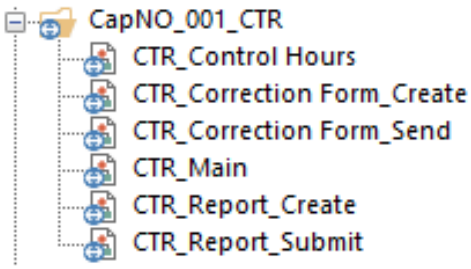
Tracking process number gives an overview of existing processes and can be of great help in both development and production.

4.1.2 Example Large Process

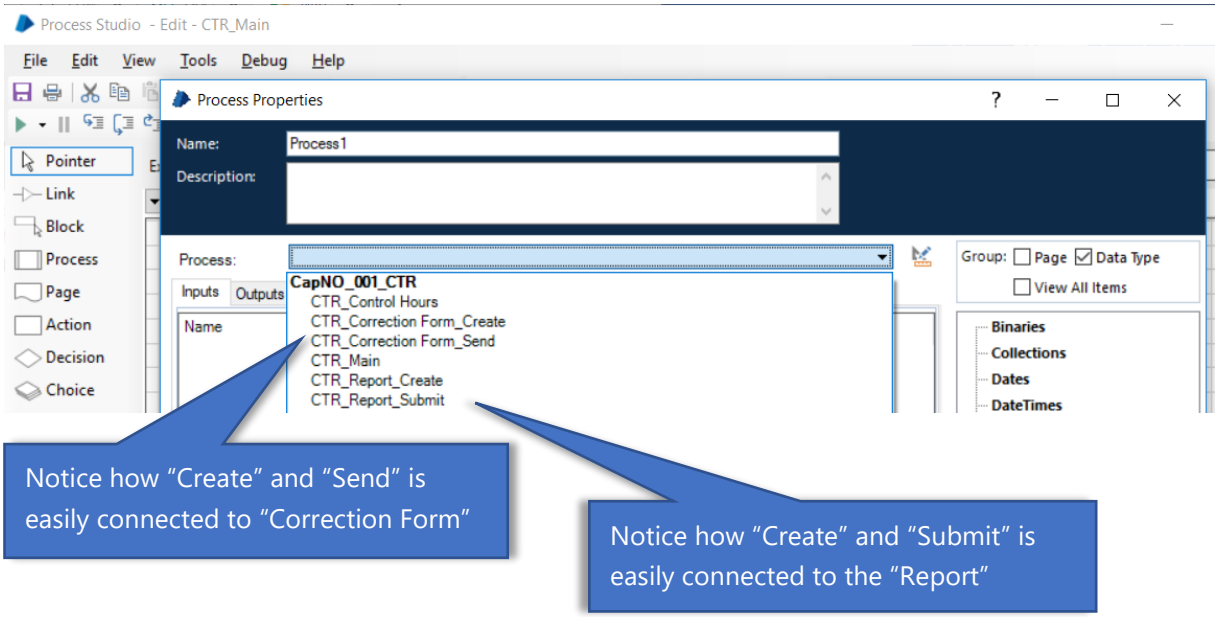
In cases with large processes, it is a good idea to split it up to a **main process** *calling* on **sub-processes** for better control. The processes should be grouped into a folder and the folder follows the naming standard given above. It is the same standard as with objects, which we will come back to later.

The **Main** *process* should have the following naming standard in the *folder*:
[Process Name]_Main

While the **Sub-Processes** uses a simplified naming standard:
[Process Name]_[Process Action]

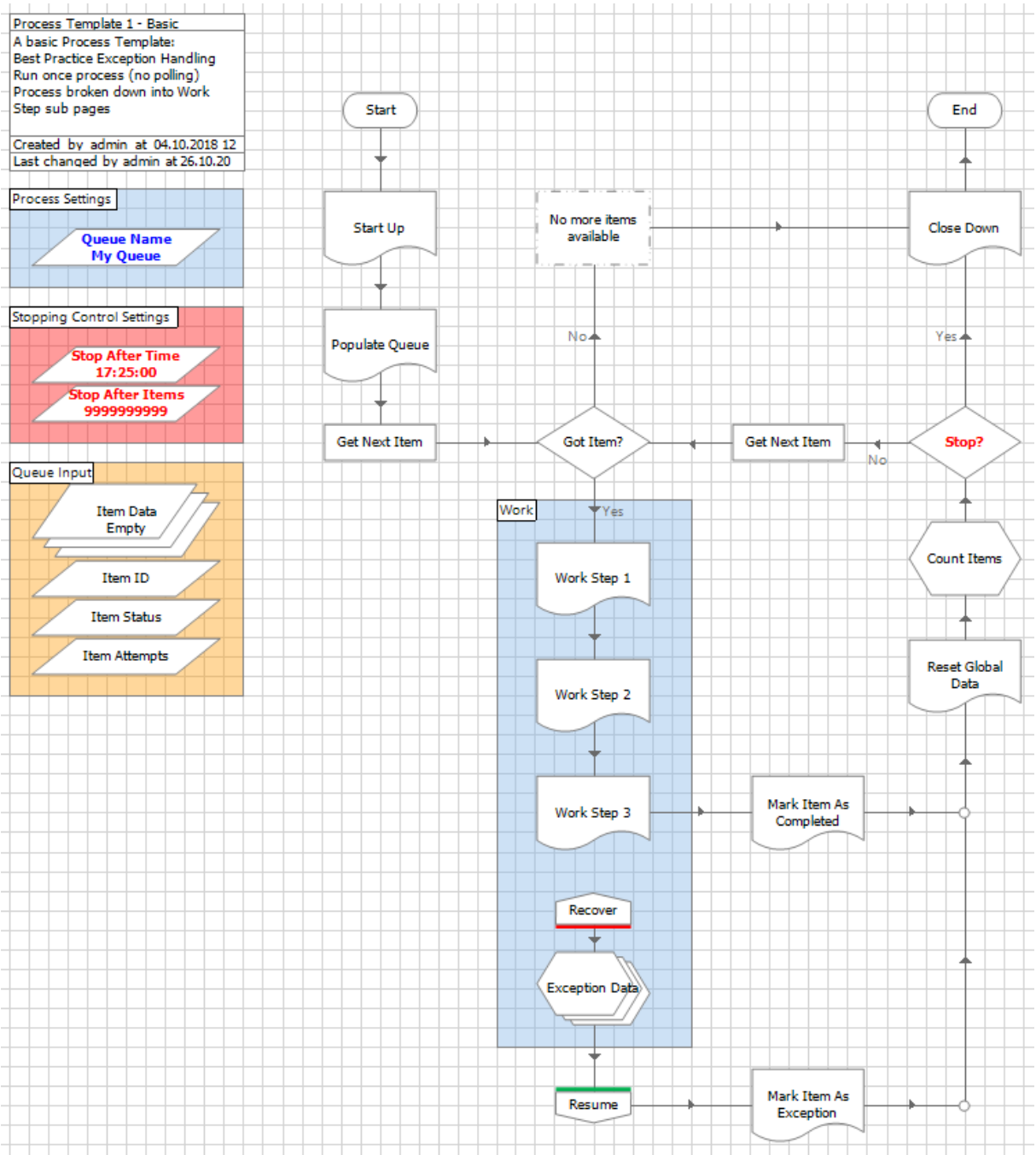


In some cases, as you can see above, the need to specify where or what the process is automating is needed before the [Process Action], so that when the main process calls upon the sub processes they will be sorted in relation to each other:



4.2 Process Main Page

Sometimes projects have their own process standards, but in the cases where they do not have a standard the Blue Prism process template should be followed. In Capgemini Norway we have our own modified template based on Blue Prisms, contact Sina Hassani for the details. For Blue Prisms original template, visit the Blue Prism Portal. Download and import it to the studio.



Main page of Blue Prism process template, basic.

4.2.1 Why Template

The template gives us a lot of "out of the box" solutions that follow the Blue Prism standards. As a developer it saves us time if we only have to modify the template after need, rather than to re-create a flow over and over again.

4.2.2 How to Use Template

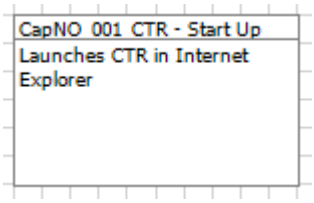
You are not meant to download and import the template every time you are creating a new process. Instead, you download the template from the Blue Prism Portal to studio once, and whenever you need to create a new process:

- Open template
- Click "**File**" and "**Save As**"
- Pick the name you want for you process, following the naming standards.

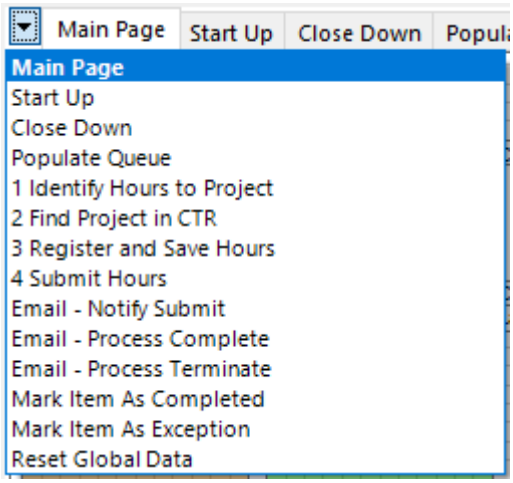
4.3 Process Pages

Keep process pages concise and in order, to make it easy to understand when first opening it. Always make sure the following is done:

- Fill out description boxes on each page

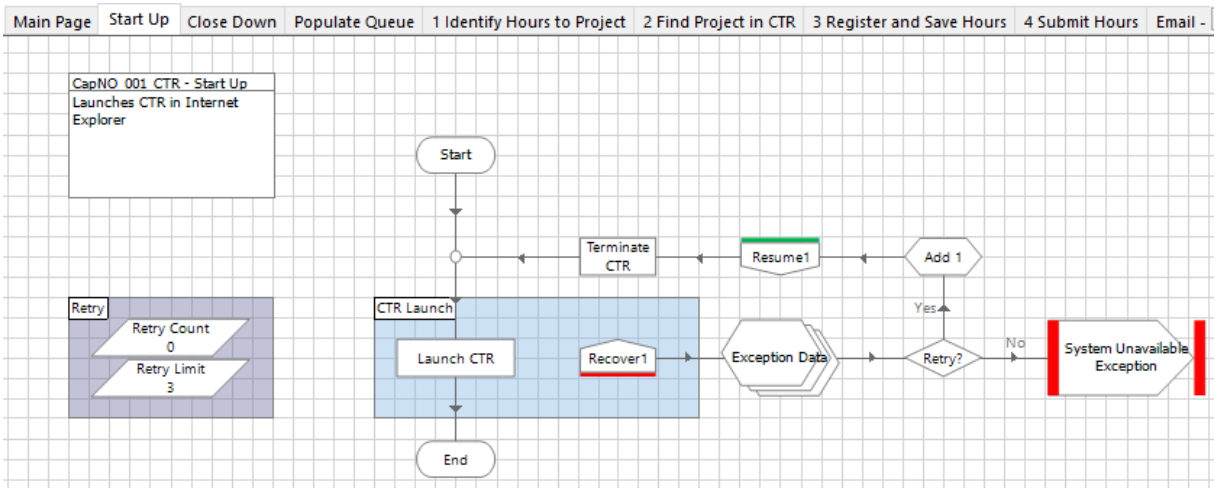


- Page names give an indication of what the page does



4.3.1 Example

Note how the "Start Up" page name is descriptive. If you start more applications, it could be an idea to name it with the application name as well: "Start Up – CTR". Also notice how the page has a descriptive description box explaining what the page does:



5. Object Studio

This section covers the best practices when developing in BP Object Studio.

*In Capgemini we focus on making objects as **reusable as possible** by having a specific naming standard and making sure no actions (pages in objects) in any objects are process specific.*

5.1 Naming Convention

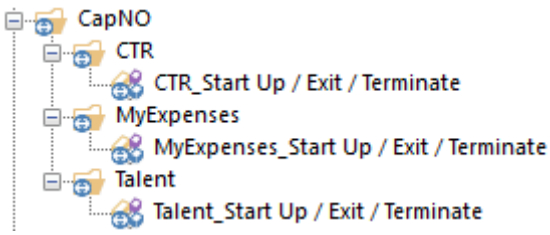
Just like process studio, object studios naming convention also varies from project to project. However, if a standard is not established at customer site, we follow the Capgemini naming standard with **Application Name** followed by **Page/Interface**.

*In cases where there are other countries working in the same environment, **country code** can come first, or the objects can be placed in a country folder.*

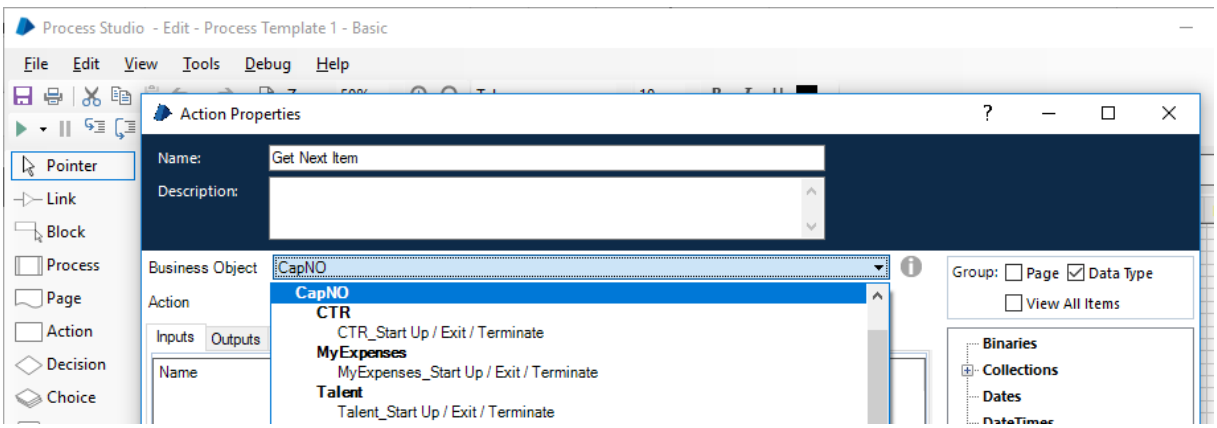
The one exception to the naming standard is when creating the **Start Up / Exit / Terminate** object that all applications that launches from an executable file has. If the application *is not launched* from an executable file, only the **Launch** part is removed.

5.1.1 Example

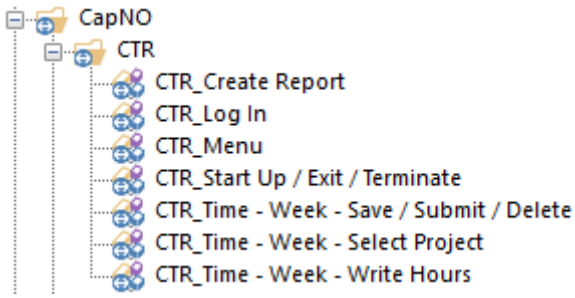
[Application Name]_Start Up / Exit / Terminate



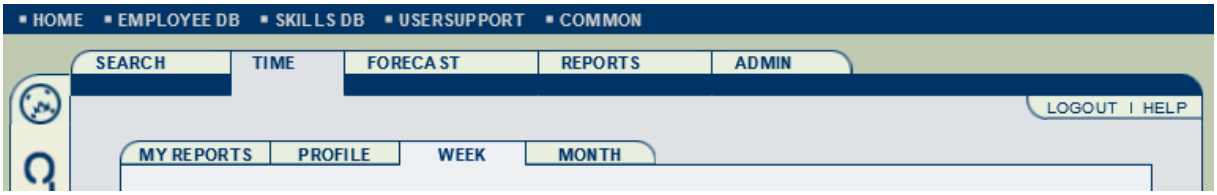
Notice how all objects are placed within the country folder "**CapNO**" and again placed within their related application folder. This makes it easier to find the objects in the process layer:



Create objects for each new page/interface of the application with the following syntax: [Application Name]_[Page/Interface]



In some cases, like above, it can be a good idea to show where you are on the page as using a simplified breadcrumb standard. CTR has a Home menu and a top menu. It also has a sub menu with tabs making it important to use breadcrumbs for others to see where you are automating.



Another option is to create folders for each menu.

5.2 Creating Objects

There are no rules on how to create objects, only that you can never have too many. In Capgemini the **standard** is to make a new object for each new page/interface in an application.

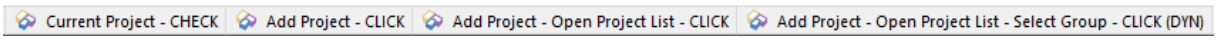
The reason is that each process consumes an entire object when it uses an action (a page from the object). In other words, if you make one big object and a process only uses one of the actions in that object, the process will still consume the entire object with all of its actions to memory, making it slow. Read more on the
***Object Design Guide** on the Blue Prism Portal documents.*

5.3 Creating Object Actions

Each page in an object is called an **action** and these have a naming convention.

5.3.1 Naming Convention

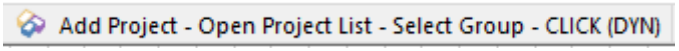
Name *Actions* in objects in a hierarchal order based on *where you are in an application*.



Notice how that when the action **does something** (CLICK, WRITE, GET, PRESS, CHECK), the action it performs is written in all caps at the end:

Click and Press are actions available when using the Win32 or AA spy action in application modeller. It is important to differentiate from them when naming your action name. Does it CLICK on the element, or does it PRESS the element?

In cases where an action uses a **dynamic value**, for example if the process sends information to the action and something is trigged based on that, this is written at the end of the action name in parentheses:

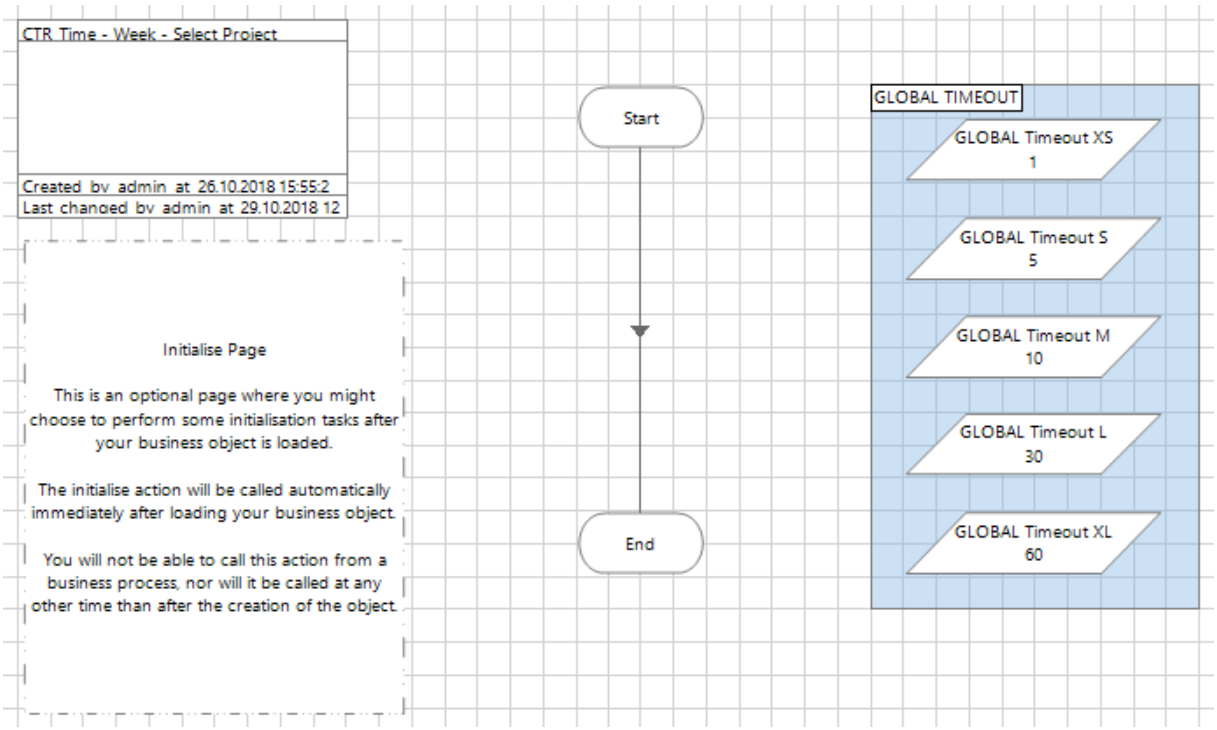


5.3.2 Initialise Action

The "Initialise" action is automatically created when creating a new object along with "Clean Up" and "Action 1".

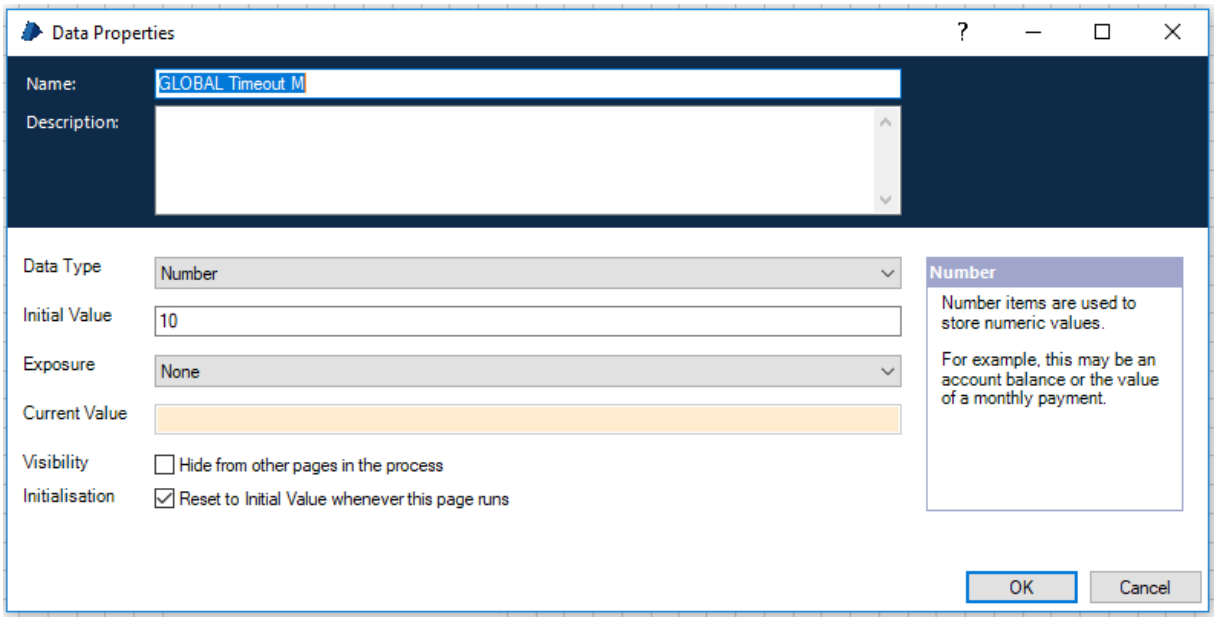
The "Initialise" stage is *always initiated* when calling upon the object. Meaning, whatever you place on the initialise page, will be executed before anything else.

Typically, you **always place** the GLOBAL TIMEOUTS on the "Initialise" page (Se picture below). The timeouts are used in wait stages and other stages that requires time.



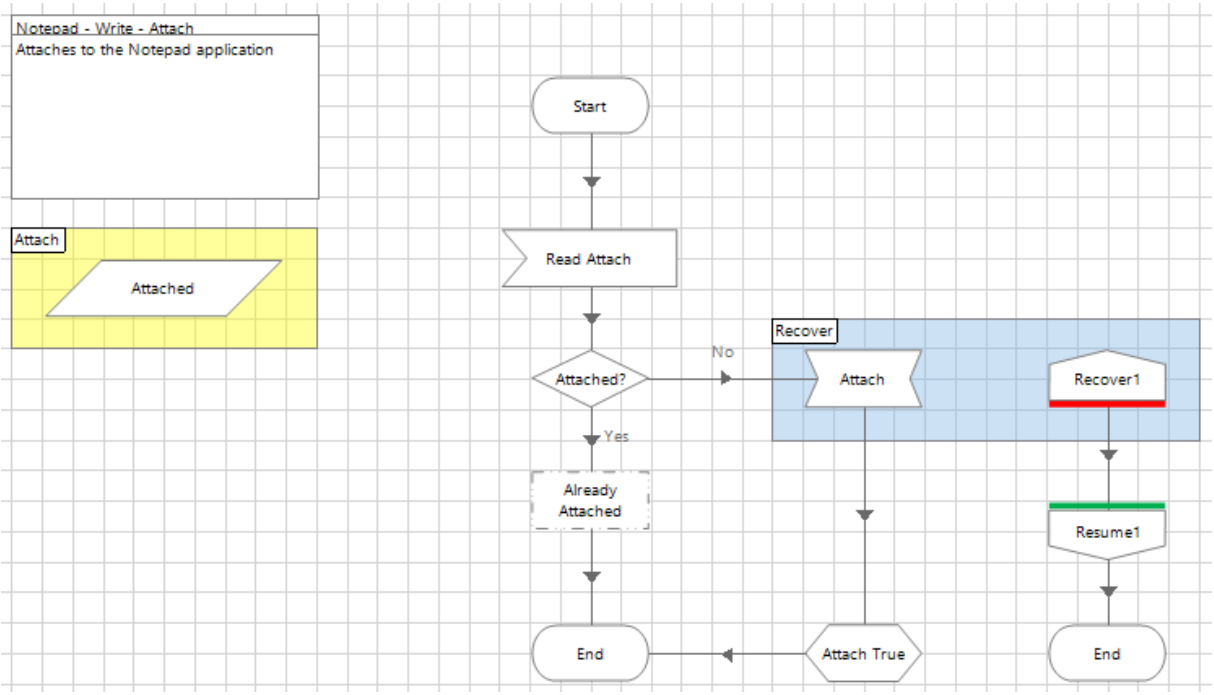
The initialise stage **should look as shown above**, however you can add more Data Items with different number of seconds where that is needed.

Each data item must be visible for other actions. This is done by *unchecking* "Hide form other pages in process":

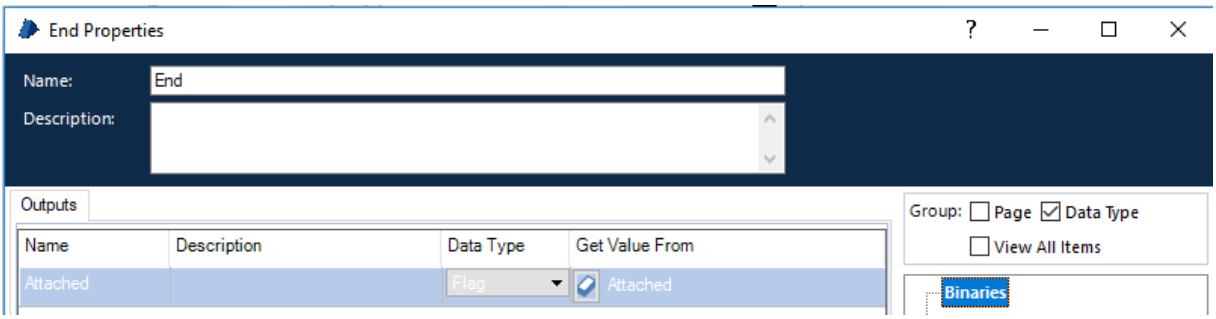


5.3.3 Attach Action

The Attach action should always be included in each object, unless the project or process dictates otherwise, and should always look like the flow shown below:



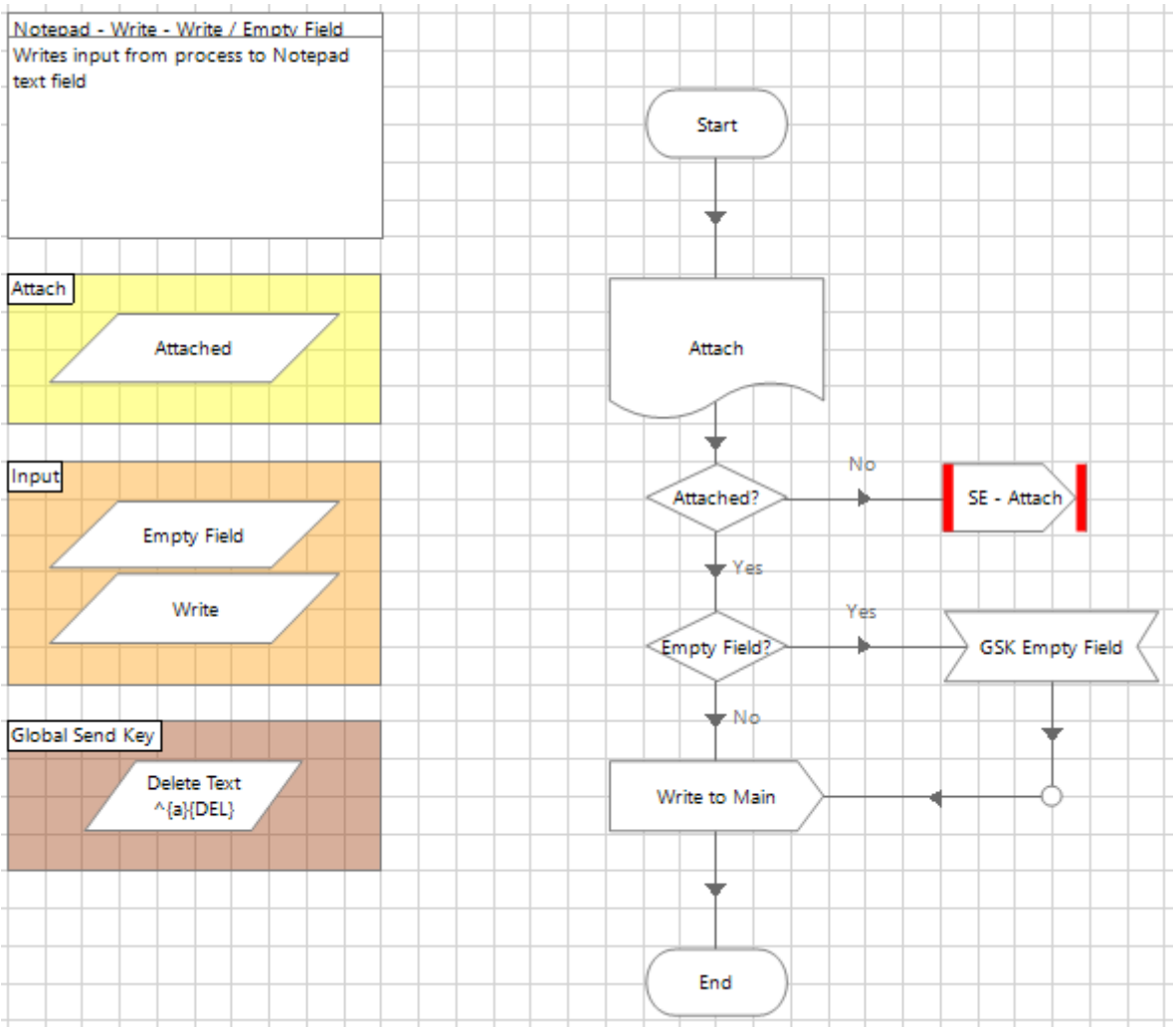
The Attach action gives the "Attached" data item as a Flag/Boolean (True/False) output element on the "End" stage:



5.3.4 Attach Action - Calling

All other actions in the object should then call upon the Attach action *before* executing other actions. The Attach action is the action that *attaches to the application* being automated. So, for each time we invoke an action from the object, the first thing it does is to attach to the application (After doing whatever is on the Initialise action).

In the example below, we automate writing to Notepad. Before we write anything, we drag the "Attach" action page in and check that Blue Prism is attached to the Notepad application:

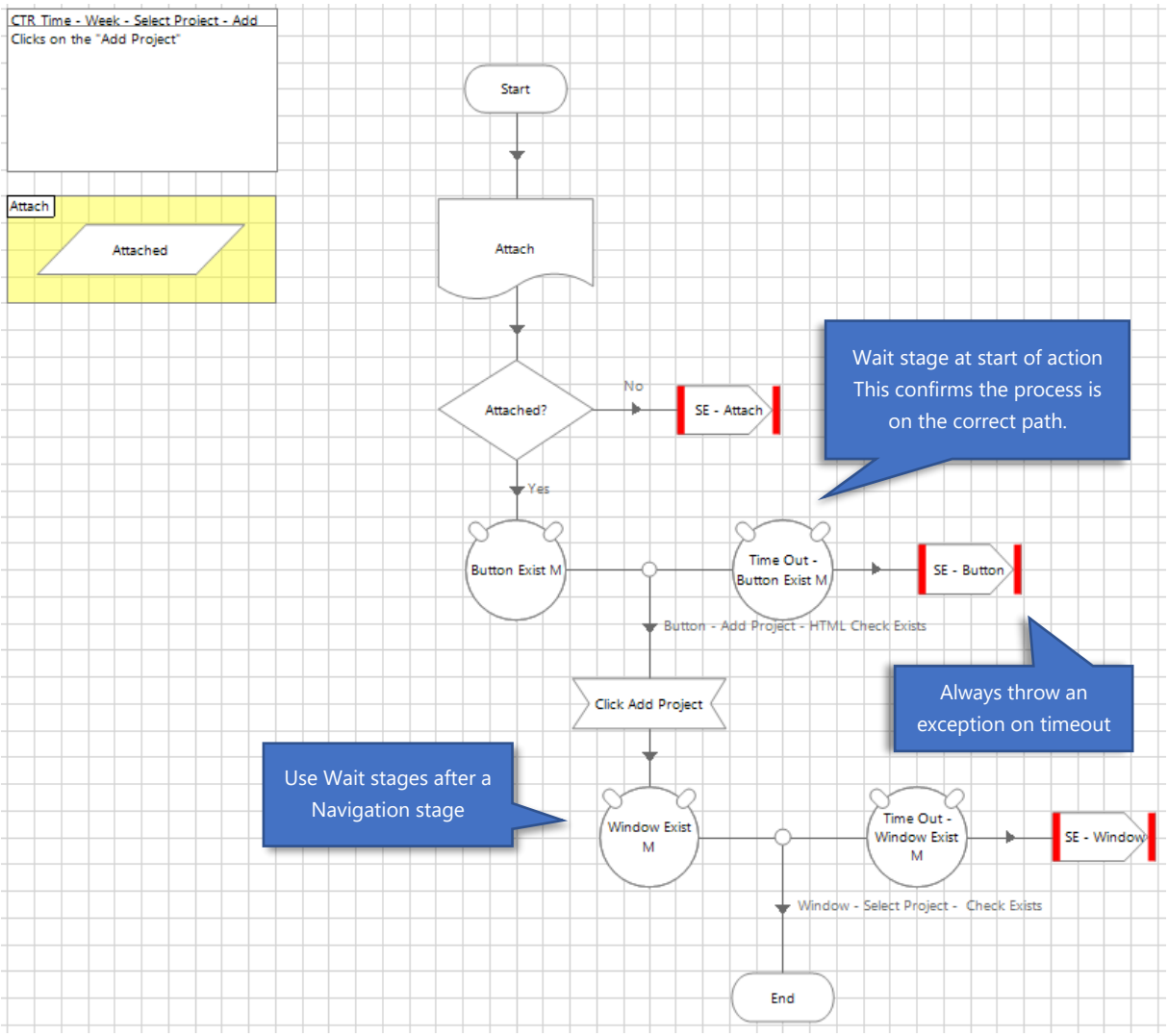


5.4 Wait Stages

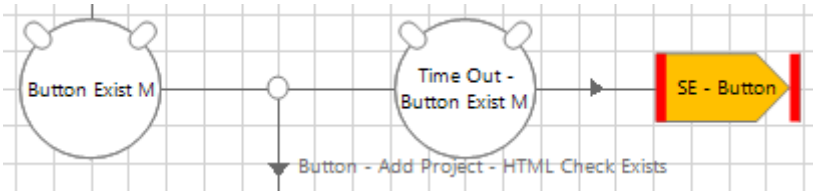
Always use Wait stages at the start of each action and after Navigate stages (or any change that causes the screen/interface to update/change).

Since Wait stages only waits for a specific element to exist or disappear, it absorbs any latency and ensures the process runs at its fastest.

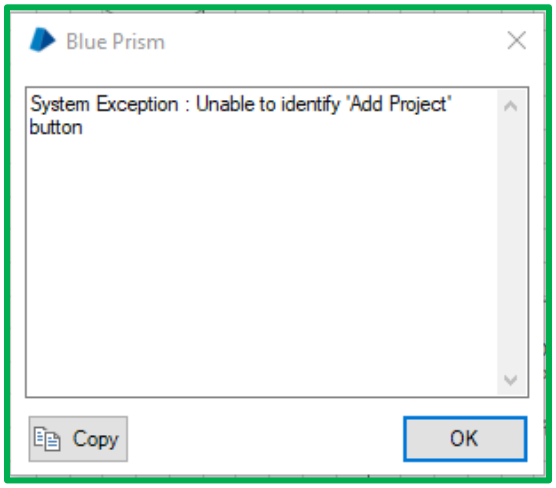
In the example below there is no need to wait 10 seconds if the system is available after 1:



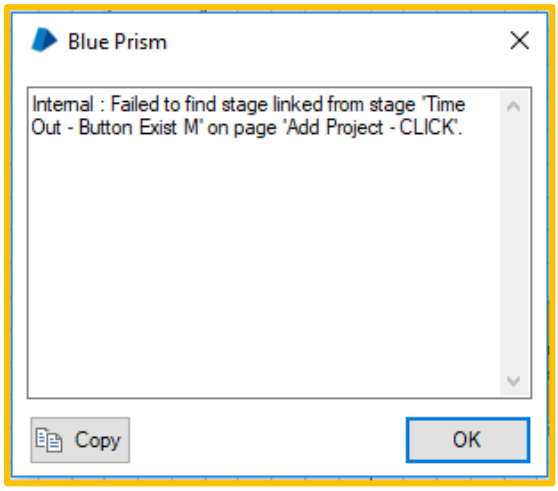
Remember to always attach an **Exception** stage to the Wait stages timeout:



This way you can control the exception message you get from the object or process instead of getting an "Internal" error from Blue Prism.



Notice how we in this example have defined it as a "System Exception" in type and that I have added my own exception detail "Unable to identify 'Add Project' button" so that we know exactly where we went wrong and can add exception handling accordingly.

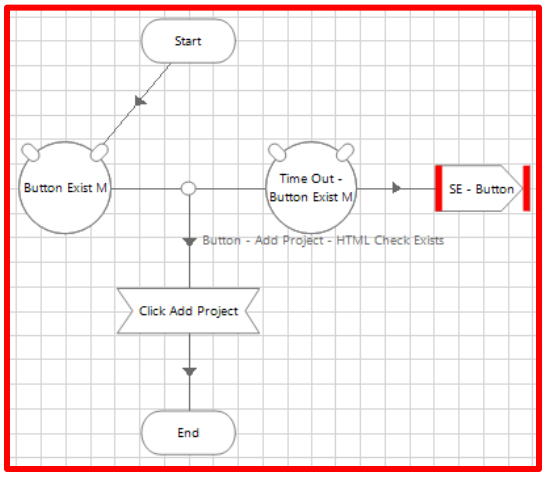


Internal exception types usually mean you have done something wrong in your flow, and can also be identified with an error on your "tool" tab

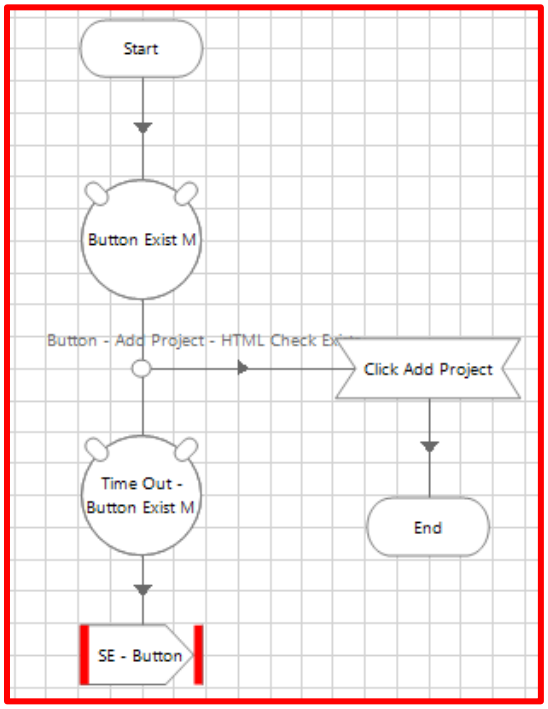
5.4.1 Wait Stage Use

Whenever you use a Wait Stage, it tells the user that something has changed or is about to change, and we are waiting for it to happen. Therefore, it is logical for the flow to have a small change in its flow downwards indicating a change, like shown above.

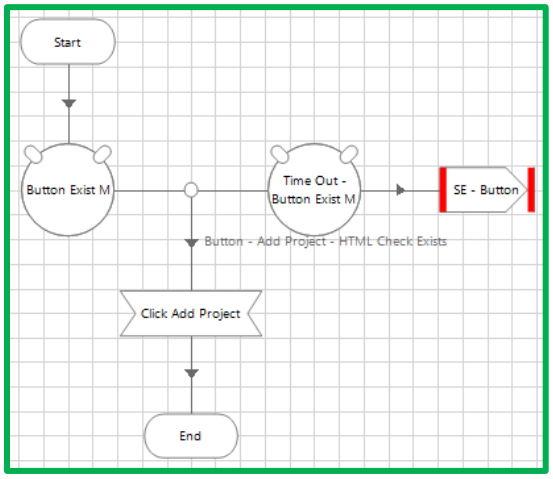
Below you can see simplified examples of wrong and right ways of using Wait Stages:



Do not drag the flow to the side to connect to a wait stage, as this blocks potential loops on the left side, and does not look good. Although it allows for a downward flow, it makes the flow harder to read.



Do not use the standard position Blue Prism offers when dragging in the Wait Stage. When developing bigger processes or more complex objects, this way of using wait stages makes it hard to develop a downward flow.

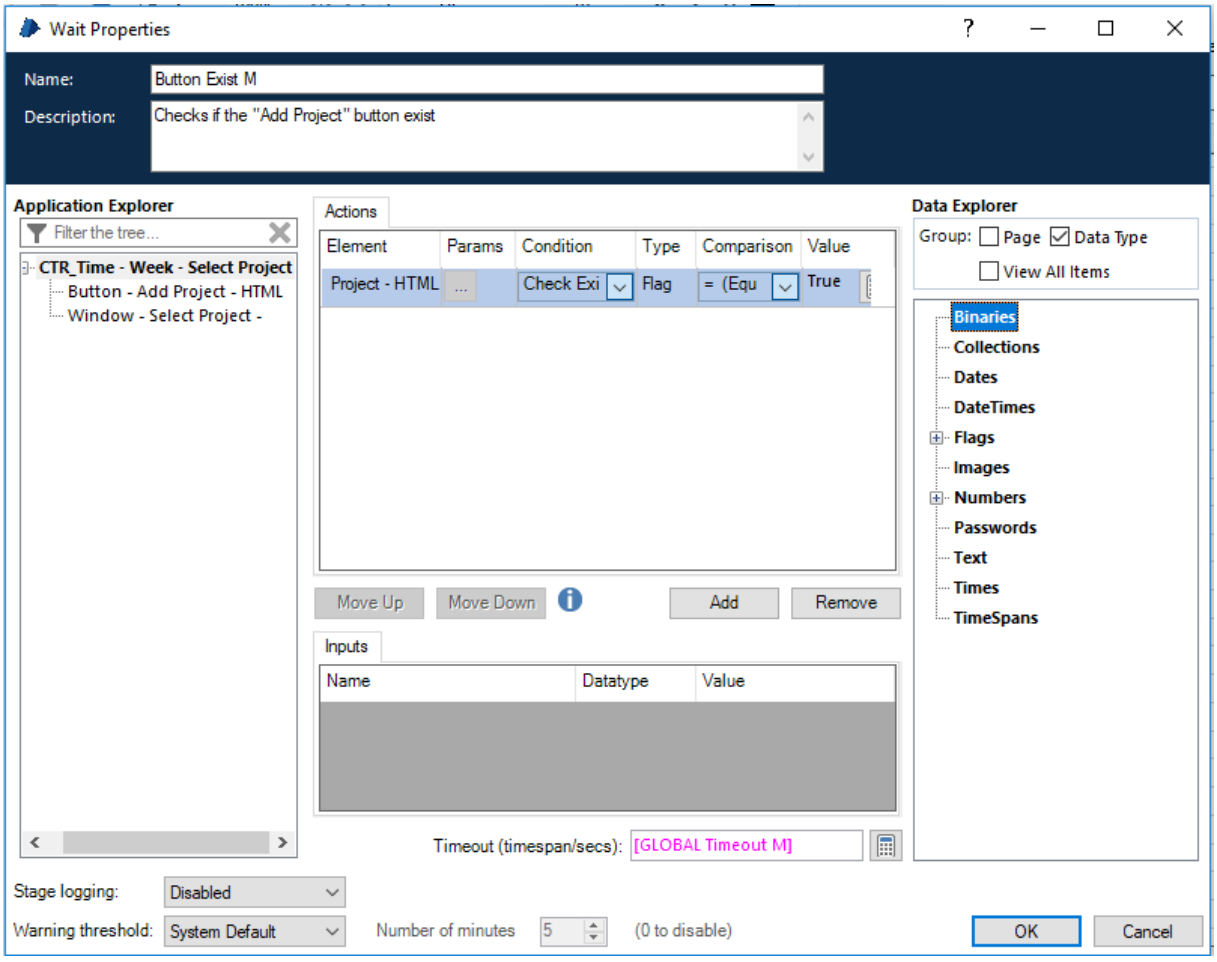


The right way to do it would be to keep the straight lines and let the process move slightly to the right, to indicate a change as you are waiting for something.

This also leaves better room for retries and in general better flow handling.

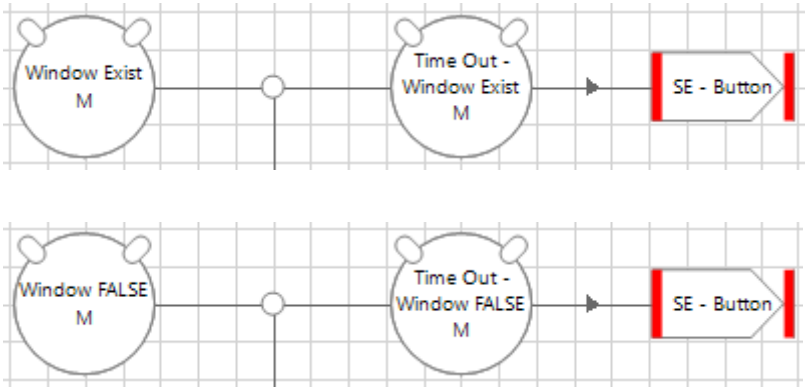
When using Wait stages, refrain from giving them long names. Simply give it the *element description* name from the Application Modeller: Window, Button, Field, Text etc.

Always include **size of** the Wait stage: **XS**, **S**, **M**, **L**, **XL**. These are the ones we create on the Initialise Action page and use in the Wait stage in the "Timeout (timespan/secs)" input field:

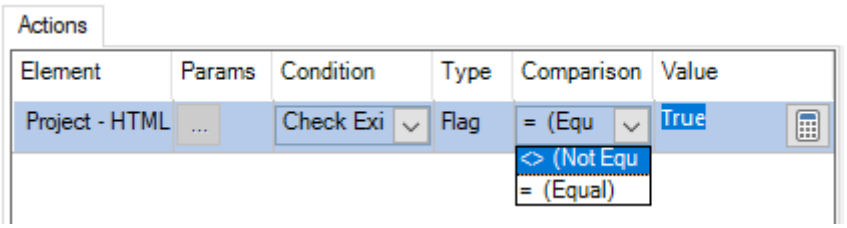


Remember to give a proper description for the wait stage, describing what we are waiting for:

Include if you are checking if an element ***Exist*** or is ***False*** (Does not exist):

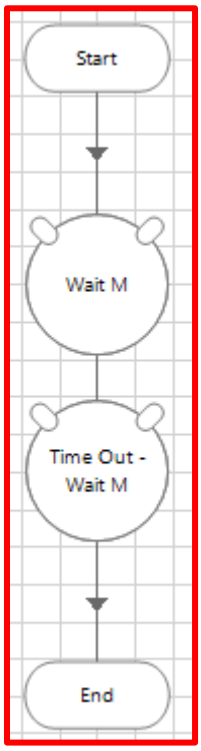


When false, the "false" is written in all caps like shown above. When choosing to wait for an element to appear or disappear, change the "Comparison" drop down to "< > *(Not Equal)*" to indicate FALSE (Waiting for something to disappear), and "= *(Equal)*" to indicate true (Waiting for something to appear):



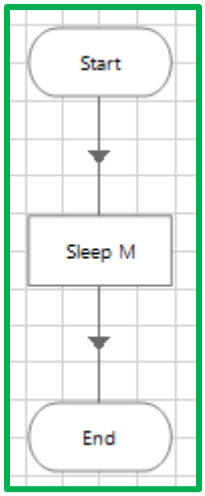
5.5 Sleep Stage

In some situations, elements are identified in the background before they are shown on the display, and this can make the flow crash. Other times the process simply moves too fast for the process and causes a crash. In those situations, **do not use empty** Wait stages and wait for them to timeout:



This not only takes extra space and looks bad and is also a misuse of a Wait stage.

In cases where the process moves to fast or we simply need it to take a "pause", we use **Sleep** stages.



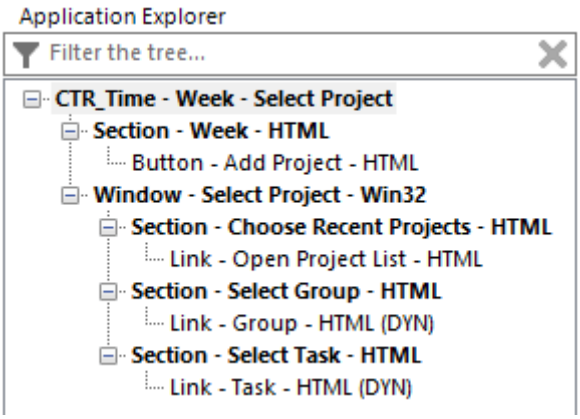
To get the Sleep action, **import** the "*BPA Object – Utility – General.xml*" located in Blue Prisms own internal files:

C:\Program Files\Blue Prism Limited\Blue Prism Automate\VBO

The input it takes is the number of seconds to Sleep before continuing the process.

5.6 Application Modeller

Application modeller is a part of the Object studio and is created with a hierarchical order. That means that the first window/interface-element is placed at the top and as the user navigates through the application, we place new elements under it:



Following a hierarchical order makes for an easier understanding of one's Application Modeller.

5.6.1 Naming Convention

Camel Case is used when naming elements in the Application Modeller.

Syntax for elements in the Application Modeller:

[Type] – [Name] – [Spy Mode] (DYN)

*(DYN) is only used if the element has a dynamic attribute

[Type]

Set the type of element in order to differentiate between types such as Button, Window, Field, Value etc., as many elements might have the same obvious <NAME>. To see explanations of different types, please see the Type Table below.

[Spy Mode]

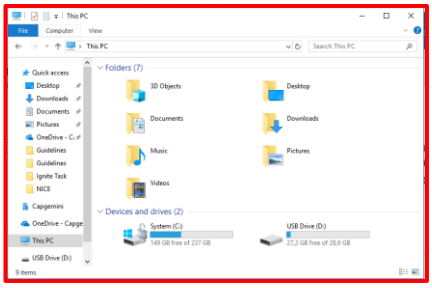
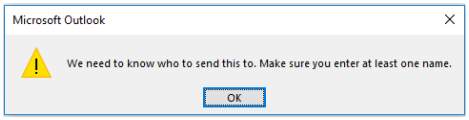
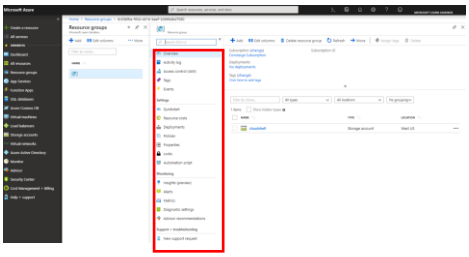
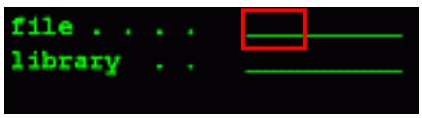
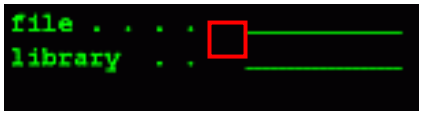
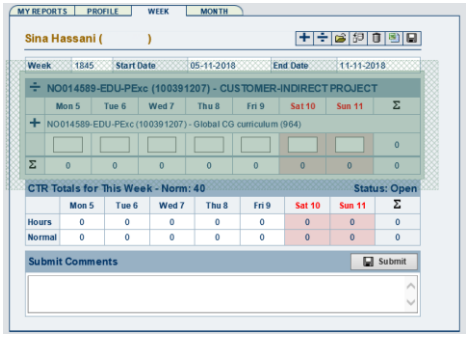
This helps give a full understanding of the element. Add the mode in which the element was spied (Win32, AA, UIA, HTML).

(DYN)

The DYN value shows that there is a Dynamic attribute in the attribute list. It is only added if there is a dynamic attribute, if not then this part is not necessary.

5.6.2 Type Table

Below you can see a description of the different spy types to use in Application Modeller.

Element Type Overview:		
[TYPE]	**Description**	**Example**
Window	An applications window (The entire frame). Usually this is spied in Win32 mode, but depending on the process use it is not limited to it.	
Popup	These are usually special windows with warnings and should only be used as a *child* to another element (Window for example).	
Section / Window Part	A section, or window part, is simply a part of a window which can be spied.	
Area	Usually limited to a fixed part of a mainframe window/grid, that can be used for both reading and writing. It is an area connected to a fixed area.	
Dynamic Area	Usually limited to a mainframe grid area with one or more dynamic coordinates defined in its attributes.	
Dynamic Region	An area defined with *Region Mode* (Application Modeller) that has dynamic attributes for *StartX*, *StartY*, *EndX* and *EndY*.	

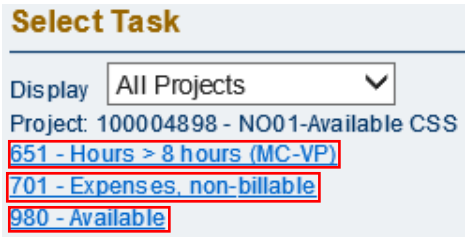
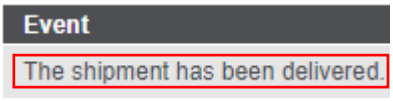
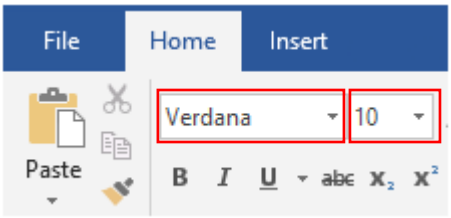
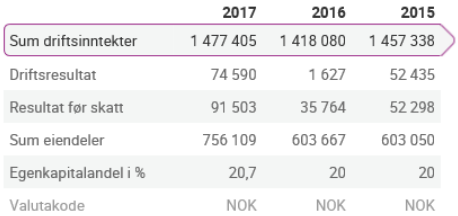
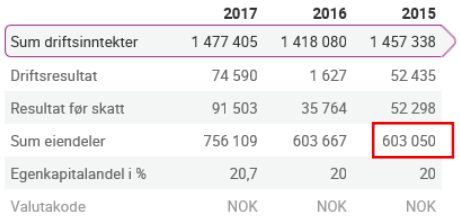
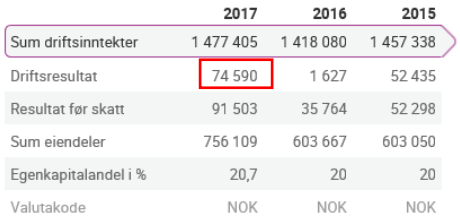
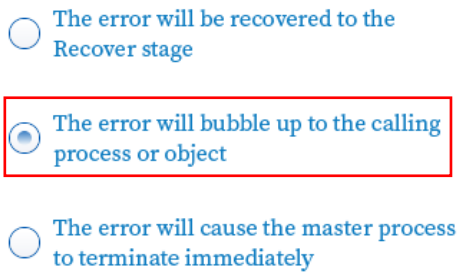
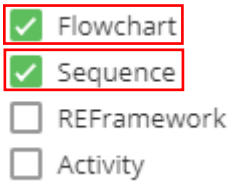
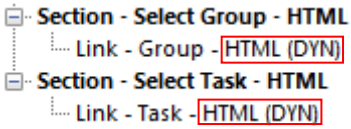
Button	A button is a button. It should only be labelled as button if it has an interactivity through the element. If not, it is most likely a label.	
Link	A link is almost like a button, but without the button "interface". When clicked or pressed it takes you to another interface.	
Label	A label is a fixed component with no interactivity, used for image recognition in screenshots or for OCR. This is especially used to find: Fields in surface automation and reading dynamic texts in surface automation.	
Field	Fields are input boxes in which text or numbers can be written. Fields can be further defined by type of field: Text Field, Number Field etc.	
Value	A value is either text (Sent, Delivered, In Holding etc.) or numbers, usually a result of something, and cannot be edited.	
Dropdown	See example. Dropdown should not require further explanation.	
Table	A table that can be read with a "Read" stage in Blue Prism giving you a collection of the table as output.	

Table Cell	A single-cell reference in a table. Should not be mixed with a "Value" type and should be placed *under* a "Table" type element in application modeller hierarchically.	
Dynamic Cell	Same as a "Table Cell" type, only with a dynamic attribute in the application modeller.	
Radio Button	A radio button that allows interactivity.	
Check Box	A check box that allows interactivity.	

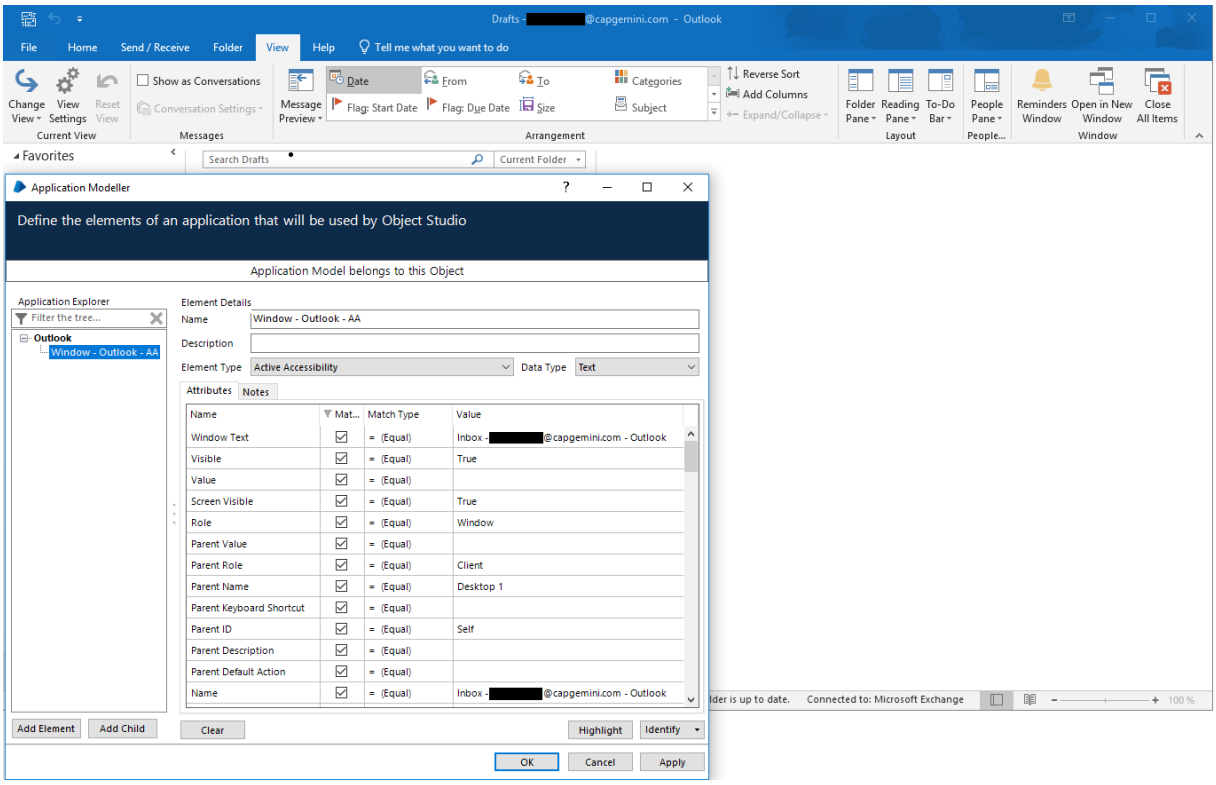
5.6.3 Spy Mode

Always include the spy mode you used (Win32, AA, UIA, HTML). If an element is dynamic, include that in parenthesis at the end of the element name like shown below:



When spying an element, do not rely on the default attributes checked for matching the element spied. Check and uncheck so that you have attributes that are both generic and descriptive enough to identify the element.

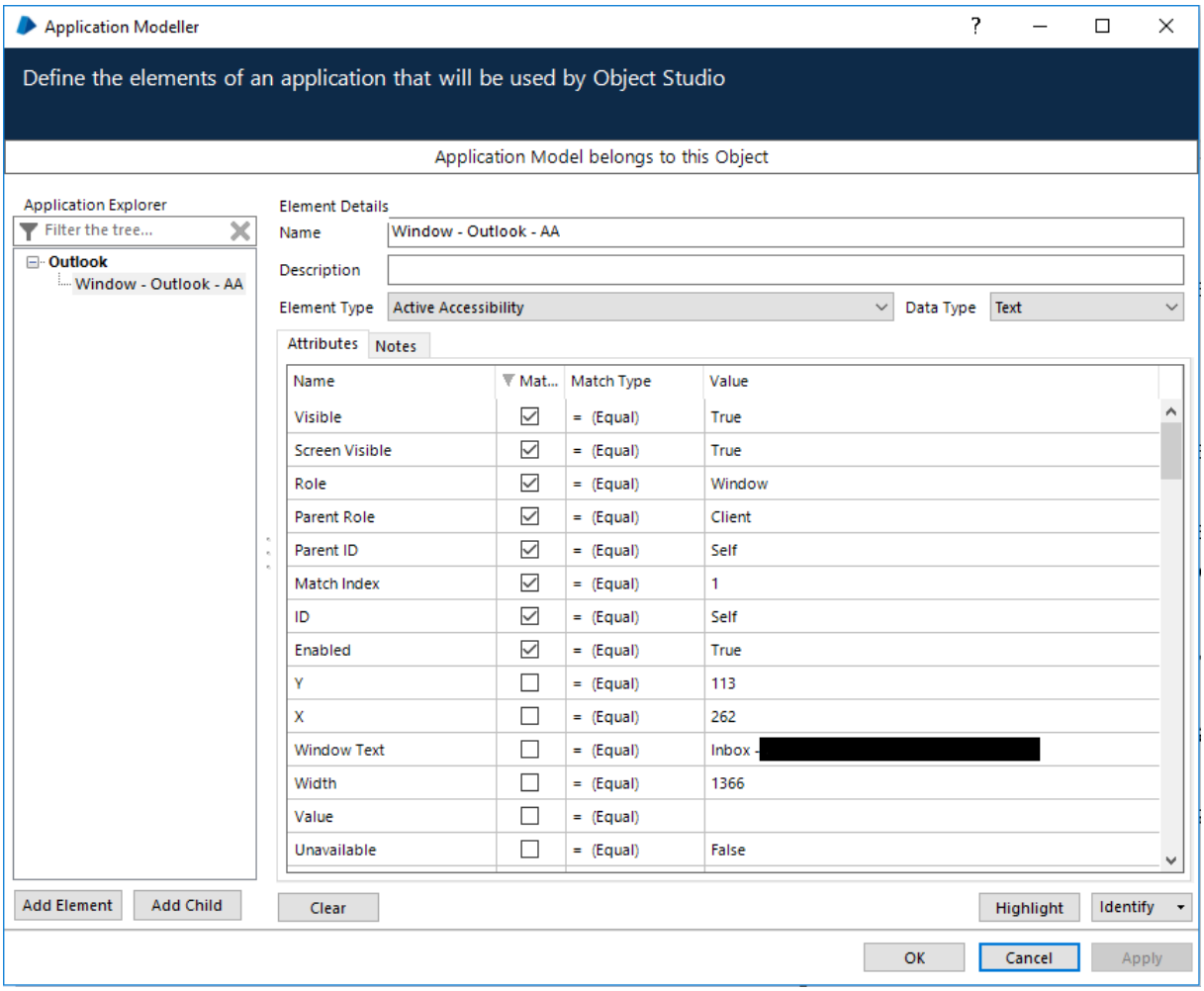
For example, if you spy the main window in Outlook you will, by default, have the header name in Outlook checked in the attributes “Name” and “Description, which is the current users email address. This does not make the object generic, nor is it reusable for others, unless we change the attributes.



Below you can see an example of Attributes that could be checked or unchecked to make the element generic and reusable:

- Uncheck all attributes with empty elements
- Uncheck all attributes with hard coded values like email addresses (ola.normann@...)
- Uncheck all attributes with names from the program that is in a specific language (For example if your Outlook is in Norwegian and there are attributes with Norwegian languages, this will most likely be different for a person in Sweden, England etc.)
- Uncheck all generic ID’s like ASP.NET
- Uncheck “Element Count” attribute, as this almost always changes
- Check “Match Index” attribute
    - If that slows the spying down, try checking “Match Reverse” as well

Following the points listed above and then testing a bit, leads to the following attributes match for an instant hit/spy of the Outlook window:



Remember that there is no "One solution fit all" applications when it come to the Application Modeller. You must test and see what makes the element generic and what can help spy it (fast).

It is **highly recommended** to ask your colleagues for tips and tricks when it comes to attributes and the differences in spy modes. You can also look at the foundation training material or contact Blue Prism directly.

5.6.4 Copying Application Modeller

Open the first Application Modeller (1) you wish to copy elements from, right click and copy. Open the second Application Modeller (2) you wish to paste to, and paste it in an empty *child* element (You cannot paste to top layer element). Click OK in the second Application Modeller (2) so that it closes, and only then do you close (OK or Cancel) the first Application Modeller you copied from.

Do it any other way and you risk of getting many, many, popup messages notifying you of an error.

6. Exception Handling

The Exception Handling part will be covered in detail on an updated version of this document. As of 07.11.2018 it is a big part of many of the certifications from Blue Prism, and it is therefor highly recommended to read the Exception Handling Guide on the Blue Prism Portal documents.

The benefits of a good exception handling are that your solution will be resilient and less prone to errors. Key things to take away from exception handling is where to do what.

It is important to know what should be done on process layer and what should be done on object layer. A rule of thumb is that **Process Layer** handles the logic of the flow, the retries and the handling of exceptions, while the **Object Layer** only *throws* exceptions and does not handle or do any retries in the objects.

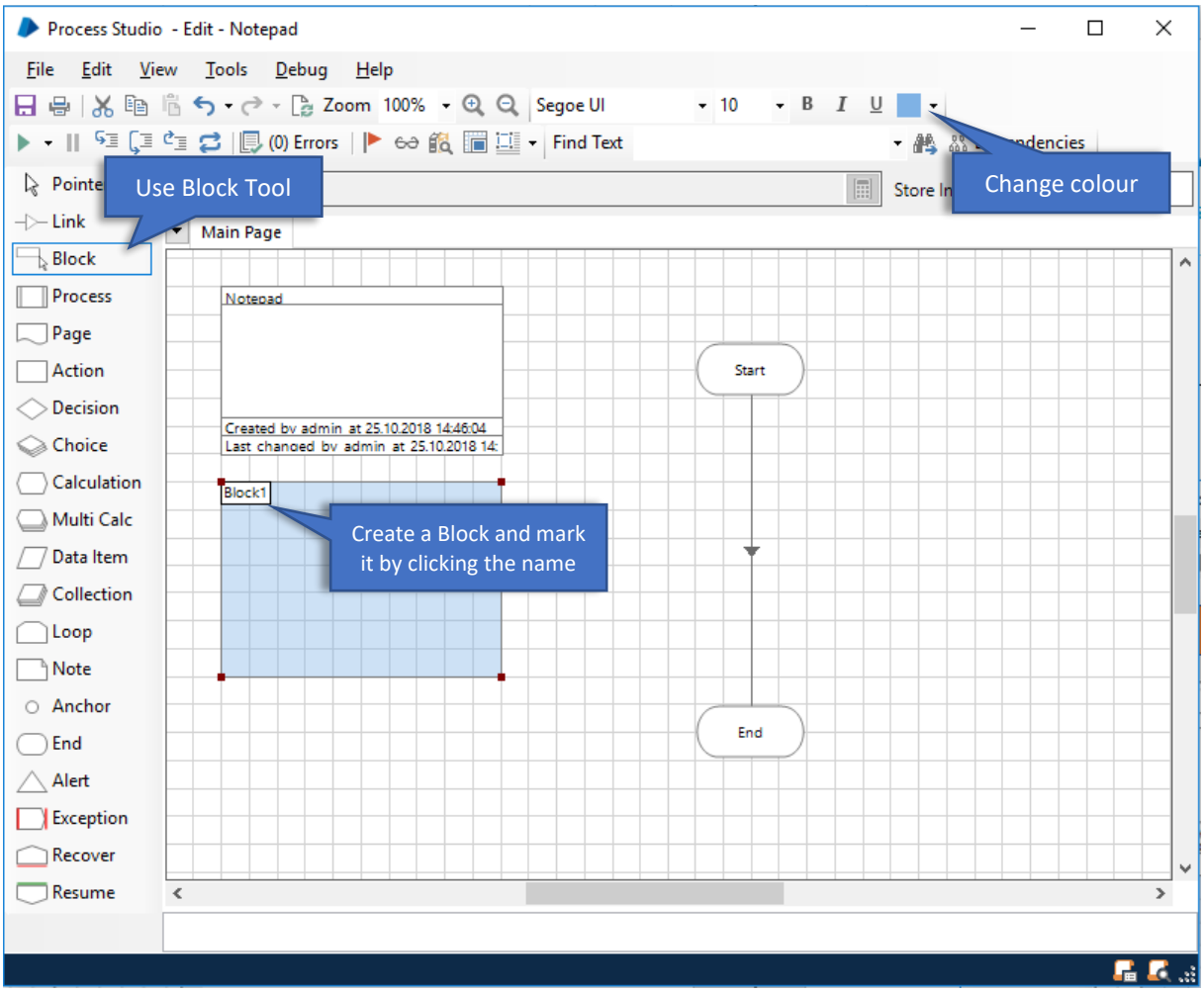
Once again, exception handling is an important part of Blue Prism, and while we have not filled this part of the best practices yet, we strongly recommend for you to read the exception handling guide.

7. Block Use

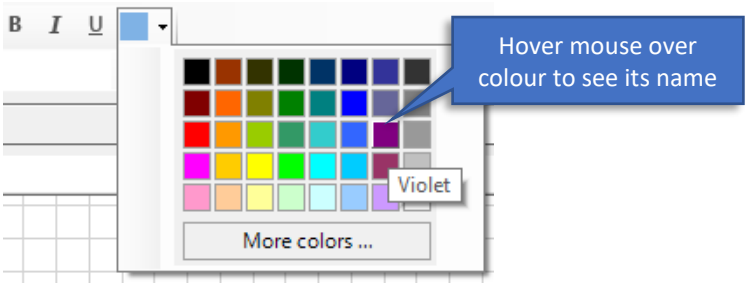
Structuring your blocks will make it easier for yourself and other developers to get a quick overview of the different pages, actions and elements in your process- and object studio.

7.1 Colours

Changing colour is done by creating a Block, marking it and then changing colour in the top menu like the example below:



Below is the list of colours to use when creating blocks. When opening the dropdown colour panel, holding the mouse over a colour will show the name:



Block	Definition	Colour
Attach	The Attach data item	Yellow
Input	Inputs to pages	Light Orange
Output	Outputs from pages	Bright Green
File Management	Files to handle	Sea Green
Path / Shortcut / URL	Paths to files and websites etc.	Plum
Credentials	Username and password	Pink
Retry	Count and limit	Blue Grey
Numbers	Handling of numbers, usually dynamic numbers	Sky Blue
Environmental	Environmental items	Violet
Static	Never changing items	Dark Blue
Send Keys	Global Send Key items	Brown
Dynamic	Dynamic items	*Light Blue*
Not In Use	When developing a part or keeping a part for later use	*Red*

It is recommended to add empty blocks on the main page of the process, so to easily copy to other pages and object layer. See below: