**Document Title**: Supertrack Programming Standards

**Doc Version:** 1.00.1

**Published By:** Tim Reamsbottom

**Publish Date**: 15/07/2015

# Version History

*The version number corresponds with the program version number set in Automation Studio.*

|  |  |  |  |
| --- | --- | --- | --- |
| Publish Date | Version Number | Comments | Engineer Initials |
| 15/07/2015 | 1.00.0 | First Release | TR |
| 16/07/2015 | 1.00.1 | Modify request/enable interface | TR |
| 07/02/2018 | 1.00.2 | Add Standard naming conventions for local dynamic variables | TR |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |
| [Select Date] | Click here to enter text. | Click here to enter text. | Click here to enter text. |

# STRUCTURED TEXT STANDARDS

## Declaration Guidelines

#### The following rules apply to all program created in Structured Text format.

### Variable, Constant and Enumerations

|  |  |  |
| --- | --- | --- |
| Example/s | Scope of Variable | Rules |
| exampleValue,  ton10ms\_exampleTimer | Local variables and functions | Functions should include the function type in the name. |
| ExampleValue,  TON\_10ms\_ExampleTimer | Global variables and functions | Functions should include the function type in the name, written in BLOCK CAPITALS. |
| LOCAL\_EXAMPLE\_VALUE | Local Constant | For local prefix with LOCAL  First word should relate to instance where applicable  i.e. LOCAL\_FILEHANDLER\_MAX\_FILES |
| EXAMPLE\_VALUE | Global Constant | First word should relate to instance where applicable  i.e. FILEHANDLER\_MAX\_FILES.  Global Constants should be placed within a separate variable declaration file, configured with a global scope.  Pay special attention to naming conventions to avoid confusion with other constants. |
| enm\_example\_value | Local Enumerations | First word should relate to instance ie..  enm\_fileHandler\_open |
| enm\_Example\_Value | Global Enumerations | First word should relate to instance ie..  enm\_FileHandler\_Open |
|  |  |  |

### Data types

|  |  |  |
| --- | --- | --- |
| Example/s | Scope of data type | Rules |
| exampleType\_typ,  can[1].readyToLoad | Local data type | The variables that make up the data type should follow local variable declaration convention. |
| ExampleValue\_typ,  Can[1],ReadyToLoad | Global data type | The variables that make up the data type should follow global variable declaration convention. |
|  | Data Type Substructures | Sub structures should follow the rules above. If a sub structure has local scope, it should follow the local convention, If it has global scope, it should follow Global convention. |

### Inputs/Outputs

All input and output global variables (X,Y) should be mapped to local variables within each task using the following convention.

|  |  |  |
| --- | --- | --- |
| Example/s | Scope of data type | Rules |
| i\_exampleInput | Referenced input | Prefix with i\_ |
| o\_exampleOutput | Referenced output | Prefix with o\_ |

# Programming RUles

## Data transfer through sequences

|  |
| --- |
| **Incoming data**  Incoming data to a sequence is always copied in from the buffer or the preceding sequence if using the request/enable interface.  **Outgoing data.**  Data outgoing from a sequence must only be moved out if a buffer is the next data position.  If the request/enable interface is being used it is the responsibility of the proceeding sequence to move data into the next position. The sequence requesting release must ensure the data has been moved out on the EnableRel variable going false. |

## Request/Enable Sequence interfaces

|  |
| --- |
| The request/enable interface is for use between interconnecting sequences where there is no buffer.  **Conventions.**  **Incoming Bags**  The variables Request.XX[x] and Enable.XX[x] are only used for an incoming interface. The Request is passed to the incoming sequence as a “line clear” condition. **This should only be used when the input sequence is the 1\_x\_point**  **as this sequence is not able to Request Release.**  The normal procedure for an incoming sequence is to use the Request Release from the incoming sequence as the i\_bagAtBuffer condition and the EnableRelease in place of the buffer stop.  The Request should only be set by a config var “use Request”  **The Request signal once true must never be able to go false until the bag has arrived and it is safe for the pre-sequence to close the buffer stop.**  The pre-sequence responsible for delivery of the bag (Currently only the 1\_x\_Point) , and ultimately opening the buffer, should treat the request as line clear, it then watches for the state to go false, to signal that the bag has arrived. (See guidance below when using rail position confirmation switches)  **It is not good practice to use the Enable.XX[x] variable, This should be avoided.**  **Outgoing Bags**  The variables RequestRel.XX[x] and EnableRel.XX[x] are only used for an outgoing interface.  The sequence that requires release is responsible for setting RequestRel.XX[x].  The post sequence should then treat this request as a “bag at buffer” input, and set the EnableRel.XX[x] in lieu of the open buffer command. (EnableRel.VC[1] := o\_openBuffer;) .  **The EnableRel variable once true must never be able to go false until the bag has arrived and it is safe for the pre-sequence to close the stop.** |

## Rail position confirmation switches.

|  |
| --- |
| When using encoder positioning on a vertical conveyor, rear trolley removers, cross over points or any other device that has bags moving through it and a switch or proximity confirming rail position for safety purpose.  Alignment switches must not be directly referenced to drive incoming buffer stops open. In the event that the alignment switch makes/breaks a second bag can be released.   1. When using a single buffer the rail alignment state must be latched in the within the buffer open step, this latched signal is then used to open the buffer stop. The rail end stop alternatively must include a direct reference to the alignment switch. Any multiple buffers must use a request/enable with a standard point sequence. 2. When using the request/enable interface, the request signal should use the latched position confirmation the same as a buffer stop would, and this must not be set false in any condition until a bag has arrived. |

# Naming CONVENTIONS AND DYNAMIC VARIABLES

## Use of Dynamic Variables

|  |
| --- |
| Where possible all global variables that use an array should be referenced locally by a dynamic var.  For example myBag = 200, Bag[myBag] will use dynamic var bag.  See below for naming conventions |

## Dynamic Variable- local naming convention

|  |  |
| --- | --- |
| **Dynamic Variable Name** | **Replaces** |
| alarm | Alarm[mySeq] |
| bag | Bag[myBag] |
| bag\_buff | Bag[myBuffBag] |
| bag\_out | Bag[myOutBag] |
| can | CAN[myCan] |
| line\_release | Line[myReleaseLine] |