Rockwell Automation Application Content

Common Libraries

Reference Manual

Event List Sequential

raM_Opr_EventListSqntl **v2.x**

June, 2022

Rockwell Automation

Important User Information

Solid-state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://literature.rockwellautomation.com) describes some important differences between solid-state equipment and hard-wired electromechanical devices. Because of this difference, and because of the wide variety of uses for solid-state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING 	Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION 	Identifies information about practices or circumstances or death, property damage, or economic loss. Attentions avoid a hazard, and recognize the consequence.
SHOCK HAZARD 	Labels may be on or inside the equipment, that dangerous voltage may be present.
BURN HAZARD 	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Table of Contents

1 Overview

raM_Opr_EventListSqntl:

- The List Sequential instruction monitors an event queue and generate a list of events.

Use when:

- Need to generate an event queue based on selected events from a larger event queue.

Do NOT use when:

- Other mechanisms to generate a filtered event queue are preferable.

1.1 Prerequisites

- Studio 5000 - Logix Designer
  - v30.0 →
- Studio 5000 – Architect
  - v2.0 →

1.2 Functional Description

The List Sequential instruction monitors an event queue and outputs a list of events. As events are added to a queue, the List Sequential instruction monitors event data and determines if each event should be added to the list based on the enumerated configuration of Cfg_Type. Events that match the Cfg_Type value are added to the output array, events not matching will be ignored.

Output Array Data Management:
Data is added to the output array sequentially as events are processed, with the most recent event placed at array location [0] and oldest event at array location [n]. As events are added to the output data array, a count of one is assigned to each Event. The output data array created when using the List Sequential instruction can be sized to the users' required length and is independent of the source queue length.

1.2.1 Event Instructions General Overview

Event instructions facilitate creation, transportation, monitoring and interpretation of events. A user-generated queue can be created (i.e. not bound to single object specific datatype) as well as the handling of string lengths for various event messages.

1.2.2 Event Instructions functions

Creation:

- Create
  a. Assign Type, ID, Category, Action and Value to a Message and apply a timestamp
  b. Event data is stored in an event queue – a variable length array of event members

Transportation:

- Transfer
  a. Move event data into an event data structure of **different** member datatype
  b. Message is prefixed for contextualization
  c. Original timestamp is unaltered
- Forward
  a. Move event data into an event data structure of the **same** member datatype
  b. Message is prefixed for contextualization
  c. Original timestamp is unaltered

Interpretation:

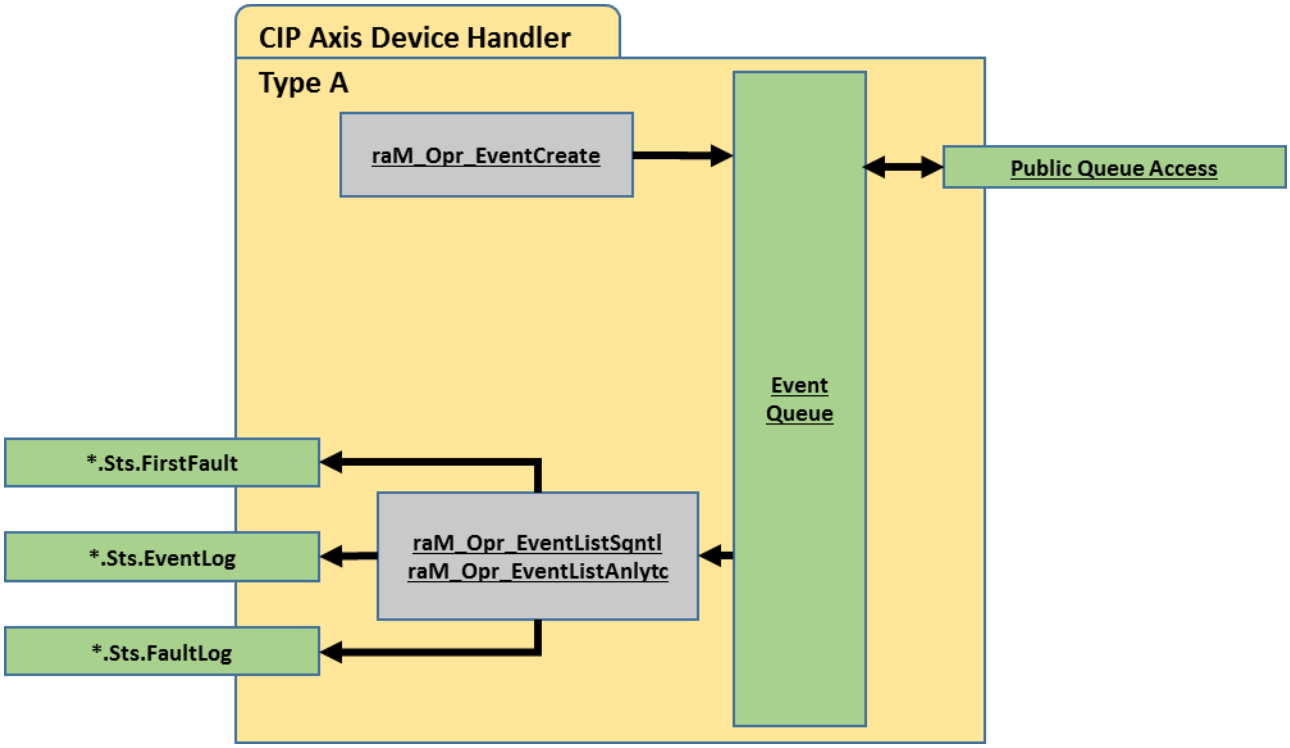
- Watch
  a. Monitoring an event queue to identify an event's location within the queue
- List Sequential
  a. Monitoring an event queue, add events to an output array as they match search criteria
  b. Events are added to the output array sequentially
- List Analytic
  a. Monitoring an event queue, add events to an output array as they match search criteria
  b. Events are added to the output array sequentially providing they are not already in the array based on message. If the message is in the array, a count is added and it is moved to the top of the list with the timestamp of the most recent occurrence displayed.

Event creation can be included at various levels of an application – either at the device level, such as the CIP Axis Device Handler, or at the user level as application code is created.

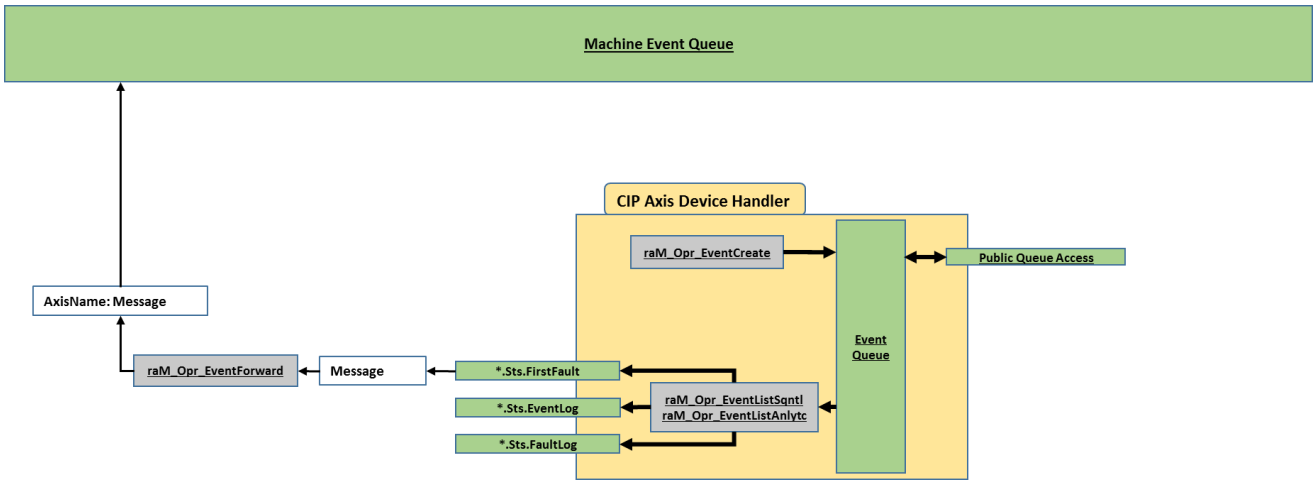
1.2.3 Implementation

Device Events:

Embedded into a device handler: the creation, queuing, and processing of events can be encapsulated and presented to consumers of event information. For example, the CIP Axis Device Handler.

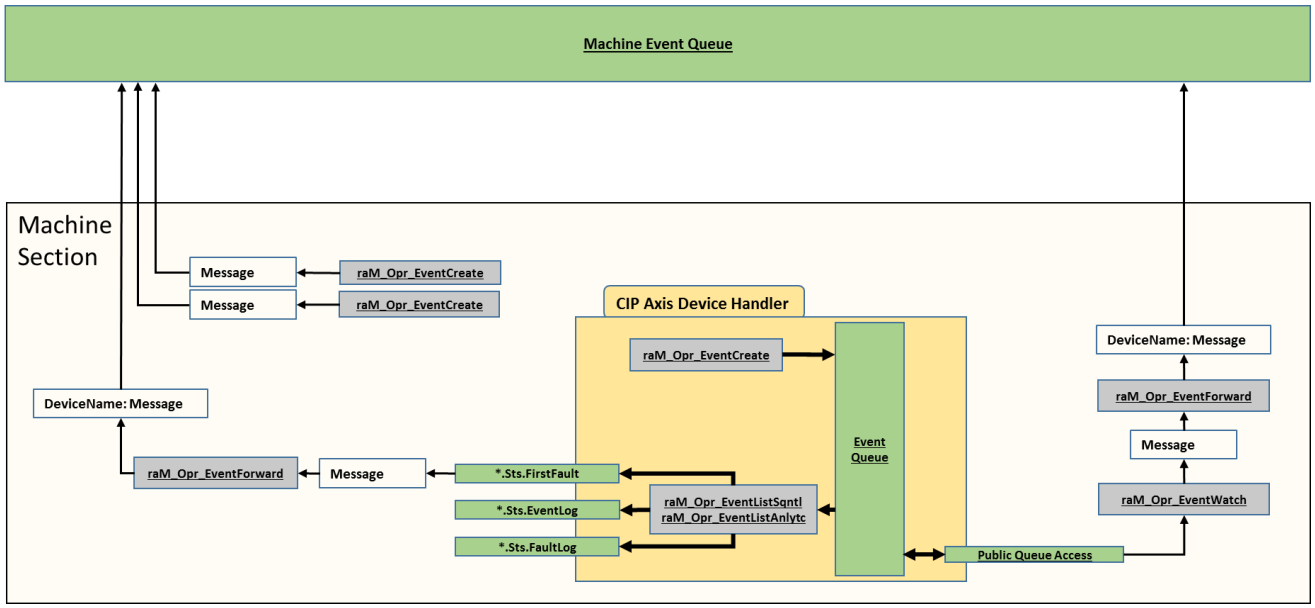


Device events can be contextualized by a consumer and moved into a machine-level event queue.



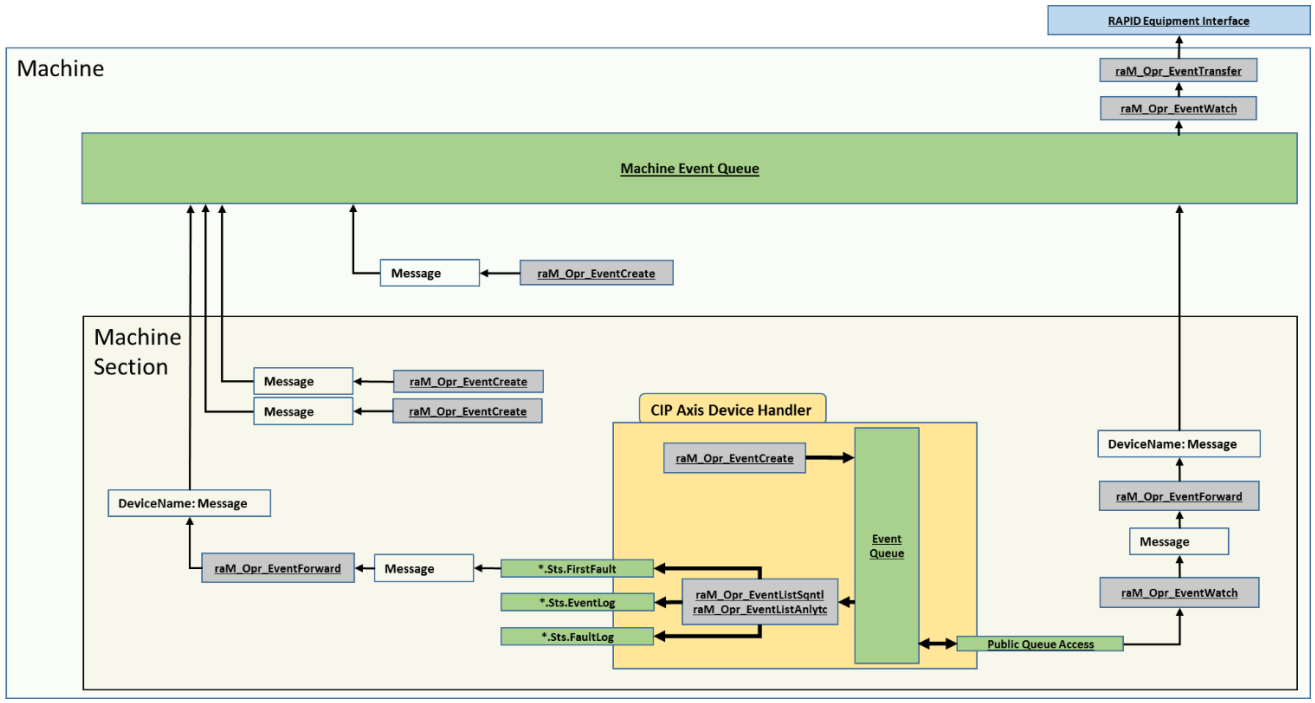
Equipment Events:

The creation of events can be applied as developers create application specific modules, such as equipment modules, by creating equipment specific events. At the same time, embedded events can be forwarded either from a single event output such as the device handler first fault event output or by monitoring the queue directly for specific user-determined events.



Machine Events:

Additionally, Event creation can occur at the machine or unit level, with a machine-level event queue being monitored as well if the user chooses. Lastly, events can be moved to dissimilar datatypes by using the Transfer instruction as shown in the case of moving machine event data into the Rapid Equipment Interface data structure as not all members align.



1.2.4 Event Data

The suite of event instructions dependents on two distinct data structures.

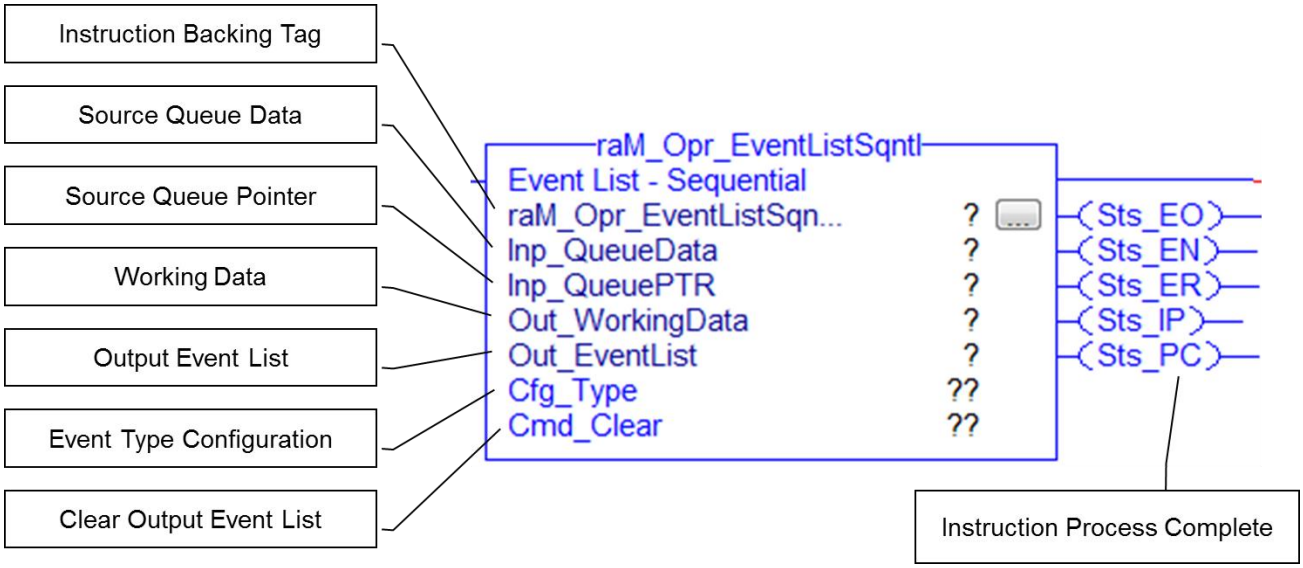
One data structure consists of the event data members. The other facilitates the list instructions as live queue data is processed and output event lists are generated.

1.2.4.1 raM_UDT_Opr_EventCreate_Members

Member	Description	Data Type
Type	Event 'Type' Enumeration. Configuration: -1 = All Event Types 0 = Not Used 1 = Notification 2 = Warning 3 = Fault 4..n = User Defined	DINT
ID	Event Numeric Identification	DINT
Category	Event Category	DINT
Action	Event Action	DINT
Value	Event Value	DINT
Message	Event Message String – Data	STRING
EventTime_L	Event Time Stamp - LINT	LINT
EventTime_D	Event Time Stamp - DINT	DINT[7]

1.2.4.2 raM_UDT_Opr_EventList_Members

Member	Description	Data Type
Event	Event Members	raM_UDT_Opr_EventCreate_Members
Count	Event Count	DINT



General Status Bit Behavior:

Note: Status bit not shown on the output side of the instruction are not used and will not exist in the instruction backing tag.

Status Bit	Description / Behavior
*.Sts_EO	• Enable Out indicated the status of the output line of the instruction. • If false (logically LO) any instruction on the ladder rung between the instruction and the neutral rail will not be energized. • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.
*.Sts_EN	• The rung-in condition of the ladder rung is true and the instruction is being evaluated. • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.
*.Sts_ER	• If the instruction experiences an internal error, the *. Sts_ER bit will be set. Error codes / Extended codes can be found by monitoring the backing tag *.Sts_ERR / *.Sts_EXERR members respectively. • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.
*.Sts_DN	• Used when the execution of the instruction completes within a single scan. • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.
*.Sts_IP	• Used to identify the instruction is in the process • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.
*.Sts_PC	• Used when the execution of the instruction requires more than a single scan to complete, and indicates the 'process' carried out by the instruction has successfully completed. • If the instruction is removed from ladder scan either in a conditional subroutine, MCR zone, JMP/LBL etc., the bit will remain in its last evaluated state.

1.3 Execution

- Level

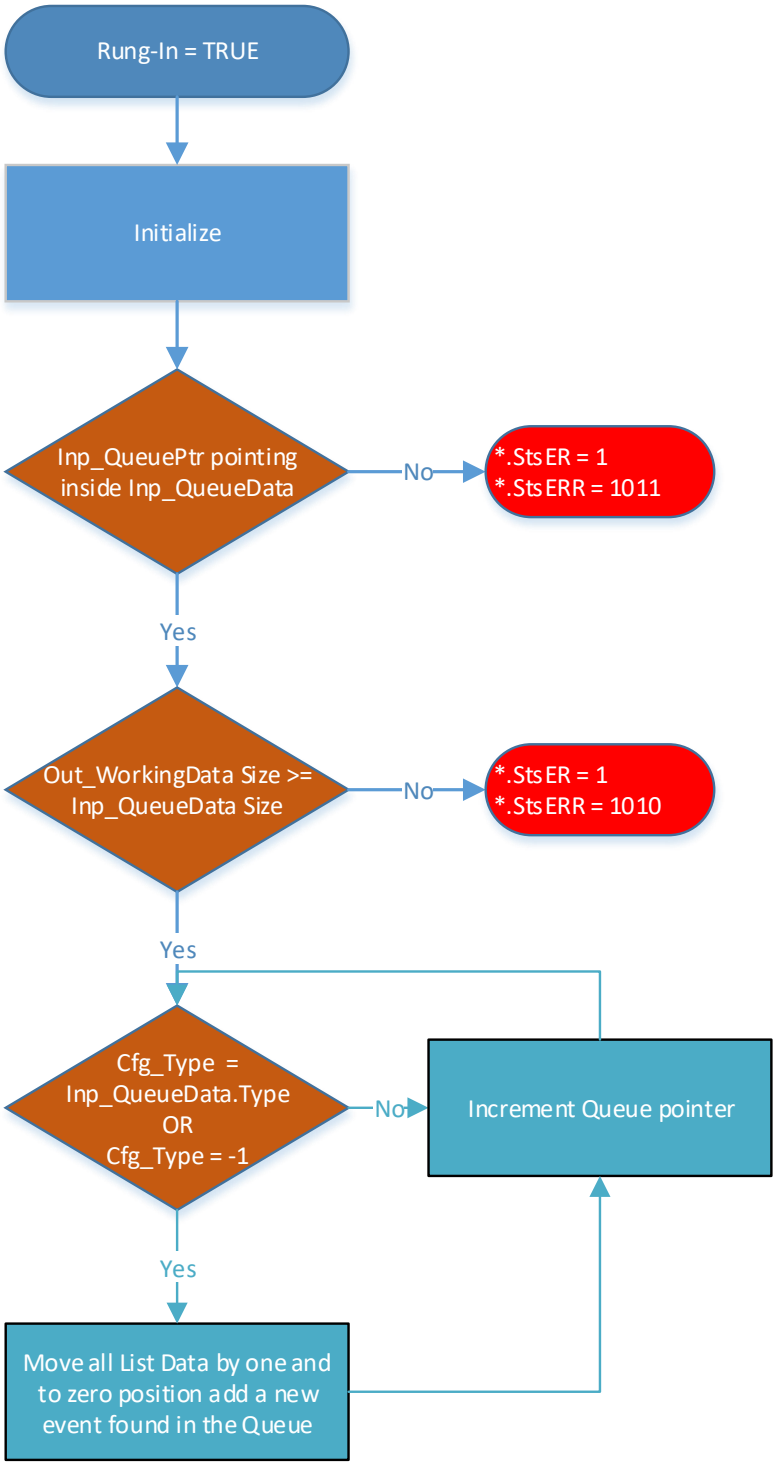
1.3.1 Overview

Rung in condition transition response:

- False → True
  - *.Sts_EO = 1
  - *.Sts_EN = 1
  - *.Sts_ER = 0
  - *.Sts_IP = 1
  - *.Sts_PC = 0
    - IF: Positive Event Match
      - THEN: *.Sts_PC = 1 and *.Sts_IP = 0
    - IF: Instruction Error
      - THEN: *.Sts_ER =1 and *.Sts_IP = 0

- True → False
  - *.Sts_EO = 0
  - *.Sts_EN = 0
  - *.Sts_IP = 0

1.3.2 Execution Table



2 Instruction

2.1 Footprint

Characteristic	Description	Value	Unit
Definition	Estimated memory required to store the object definition, including all dependents. UDT arrays with user configurable sizes are not included in the estimation.	16.0	kB
Instance	Estimated memory required per object instantiated. This includes the object instance and all datatypes required to verify the project. In the case of user configurable arrays, an application relevant array length will be used for estimation.	1.0	kB
Execution L7x	Estimated execution time / scan footprint evaluated in 1756-L7x PAC	30	us
Execution L7x	Additional estimated execution time for one event / scan, footprint evaluated in 1756-L7x PAC.	170	us

2.2 Input Data

Input	Function / Description	Data Type
Inp_QueueData	Event Queue Data Array (size is user defined, min 2)	raM_UDT_Opr_EventCreate_Members[2]
Inp_QueuePTR	Event Queue Data Array Pointer	DINT
Cfg_Type	Event Type Configuration -1 = All 0 = None 1...n+1 = Exact Match	DINT
Cmd_Clear	Clear Event List Array	BOOL

2.3 Output Data

Output	Function / Description	Data Type
raM_Opr_EventListSqntl	Object Identifier	BOOL
Sts_ERR	Instruction Error Code	DINT
Sts_EXERR	Instruction Extended Error Code	DINT
Out_WorkingData	'Working' Data Buffer	raM_UDT_Opr_EventCreate_Members[2]
Out_EventList	Output Event List Array	raM_UDT_Opr_EventList_Members[2]

2.4 Error Codes

Sts_ERR	Description
1010	New message length including prefix is greater than destination message length.
1011	Invalid queue pointer (outside array).

Sts_EXERR	Description
< Number >	If a native instruction error occurs internally, the value of the instruction *.ERR DINT will be placed in Sts_EXERR.

3 Application Code Manager

3.1 Definition Object: raM_Opr_EventSqntl

This object contains the AOI definition and used as linked library to implement object. This gives flexibility to choose to instantiate only definition and create custom implement code. User may also create their own implement library and link with this definition library object.

3.2 Implement Object: raM_LD_EventListSqntl

Implement Language: Ladder Diagram

Parameter Name	Default Value	Instance Name	Definition	Description
ObjectName	raM_LD_EventListSqntl			Object Name
RoutineName	{ObjectName}	{RoutineName}	Routine	Name of the routine where the object will be placed
TagName	_{ObjectName}	{TagName}	Local Tag	Instruction Backing Tag
EventListName	_EventList	{EventListName}	Local Tag	Event List Tag Name
EventListSize	10	--	Int	Event list array size
HMIOutListNr	4	--	Int	Unique number to identify the event list on hmi 4-31 available (only visible when in queue library HMIOutputList Size <>'None')

Input Interface

Interface Name	Linked Library	Revision
raC_Itf_CtrlEventQueue	raM_Opr_CtrlEvtQueue	1.0

Interface Members

Member Name	Description
ProgramName	Program name where Queue Object resides
TagName	Name of Queue Tag
TaskName	Task name where Queue Object resides
QueueSize	Queue size
HMIOutputListSize	Number of entries in HMI output list

Linked Library

Link Name	Catalog Number	Revision	Solution	Category
raM_Opr_Event	raM_Opr_Event	>=2.0	(RA-LIB) Machine	Event
raM_Opr_CtrlEvtQueue	raM_Opr_CtrlEvtQueue	1.x	(RA-LIB) Machine	Event

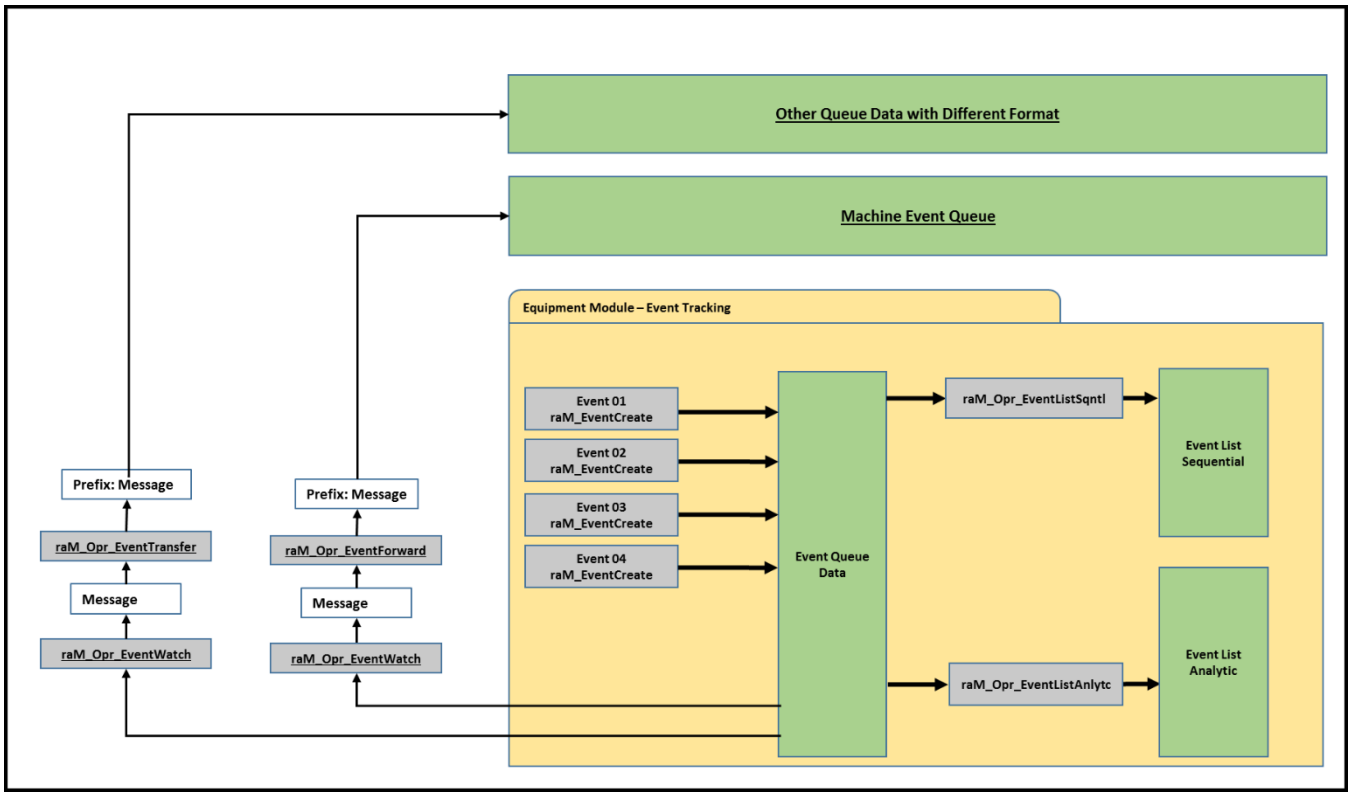
3.3 Attachments

Name	Description	File Name	Extraction path
V2_{LibraryName}	Reference Manual	RM-{LibraryName}.pdf	{ProjectName}\Documentation

4 Application

4.1 Event Instructions Routine Example

This example shows how to create a routine to perform the functions illustrated in the diagram below.



4.1.1 Message Strings

The messages used is this example are prepopulated in the Message String Tag.
There are four different messages.

Message_STRING	{...}
Message_STRING[0]	'Event 01 - Timer is greater than 100ms'
Message_STRING[1]	'Event 02 - Timer is greater than 200ms'
Message_STRING[2]	'Event 03 - Timer is greater than 300ms'
Message_STRING[3]	'Event 04 - Timer is greater than 400ms'

Each Message String will correspond to a different event:

Event Number	Event Message
01	Event 01 – Timer is greater than 100ms
02	Event 02 – Timer is greater than 200ms
03	Event 03 – Timer is greater than 300ms
04	Event 04 – Timer is greater than 400ms

4.1.2 Event Queue and Event List

This example uses the terminology Event Queue and Event List.

An Event Queue represents raw data information, events are listed in the order in which they entered the array, first event is at the top of the array (array[0] has the first event occurrence). This queue operates as a circular buffer for events, continuously overwriting its contents as new events are created.

An Event List represents the data after the execution of an event list instruction, data is organized with latest event first (array[0] has the latest event occurrence).

Queue Data Types:

Event Queue tag data type: raM_UDT_Opr_EventCreate_Members[8].
Machine Event Queue tag data type: raM_UDT_Opr_EventCreate_Members[8].

Event List Data Types:

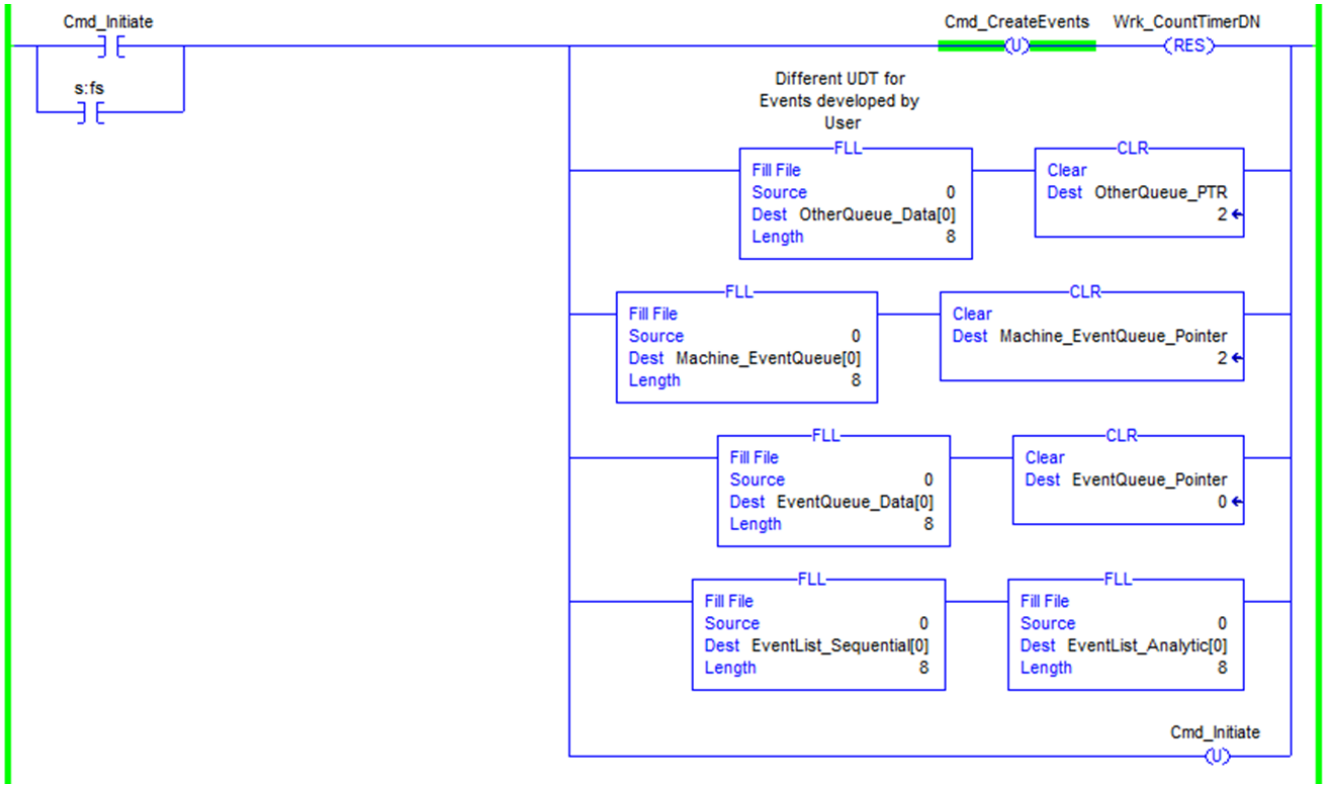
Event List sequential tag data type: raM_UDT_Opr_EventList_Members[8].
Event List analytic tag data type: raM_UDT_Opr_EventList_Members[8].

OtherQueue_Data Tag Data Type is UDT_Events[20].
OtherQueue_Data demonstrates the fact that user can transfer an event to a different format.

⊞ EventList_Analytic	raM_UDT_Opr_EventList_Members[8]
⊞ EventList_Sequential	raM_UDT_Opr_EventList_Members[8]
⊞ EventQueue_Data	raM_UDT_Opr_EventCreate_Members[8]
⊞ EventReference	raM_UDT_Opr_EventCreate_Members
⊞ Machine_EventQueue	raM_UDT_Opr_EventCreate_Members[8]
⊟ OtherQueue_Data	UDT_Events[20]
⊟ OtherQueue_Data[0]	UDT_Events
⊞ OtherQueue_Data[0].Type	DINT
⊞ OtherQueue_Data[0].ID	DINT
⊞ OtherQueue_Data[0].Category	DINT
⊞ OtherQueue_Data[0].Action	DINT
⊞ OtherQueue_Data[0].Value	DINT
⊞ OtherQueue_Data[0].Message	STRING
OtherQueue_Data[0].EventTime_L	LINT
⊞ OtherQueue_Data[0].EventTime_D	DINT[7]
OtherQueue_Data[0].UserDefined01	REAL
⊞ OtherQueue_Data[0].UserDefined02	DINT

4.1.3 Initialization Rung

Initialization clears “Event Queue Data”, “Event List Sequential”, “Event List Analytic”, “Machine Event Queue” and “Other Queue Data”.



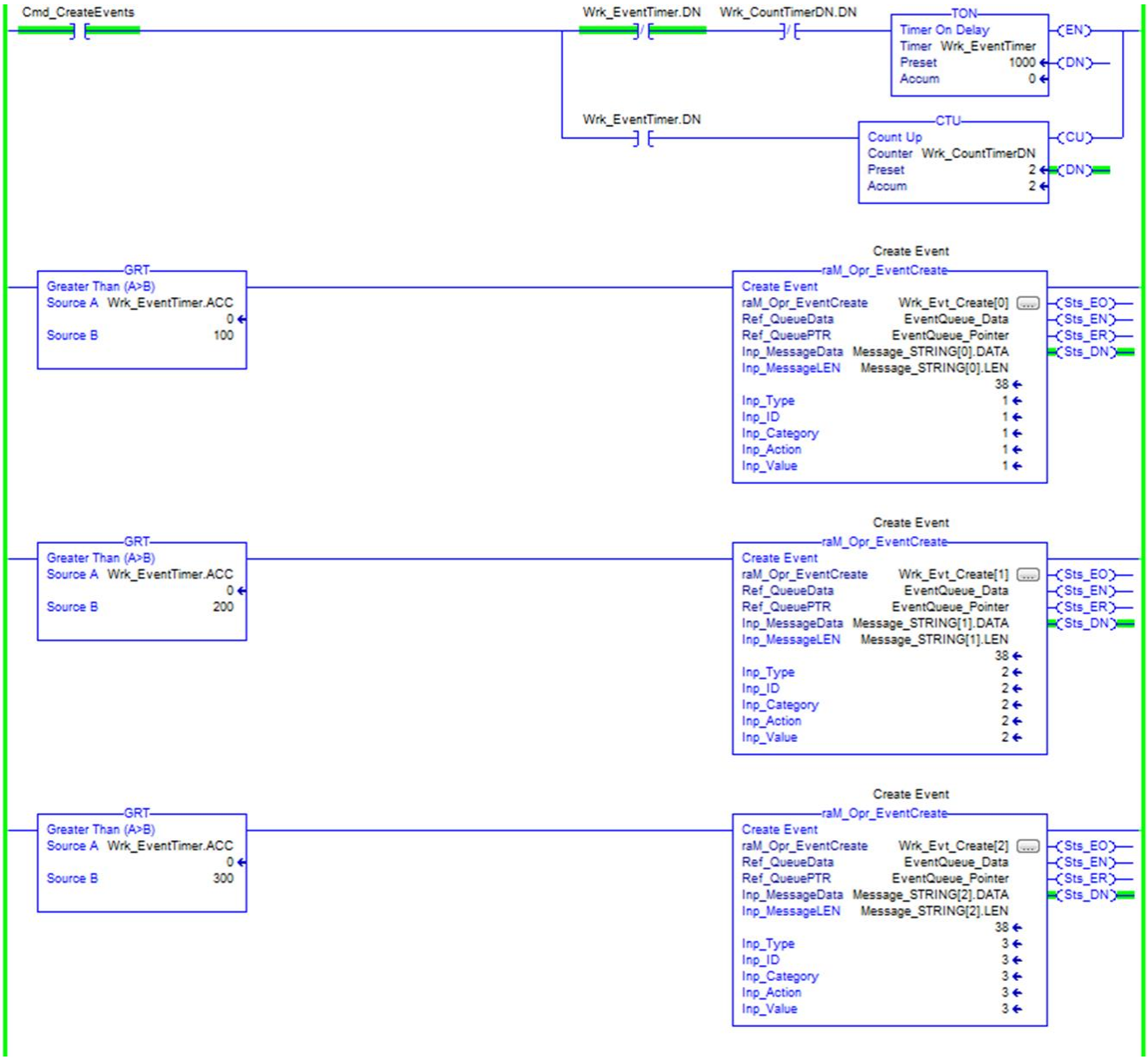
4.1.4 Create Events

The logic in the Create Events section shows how to create events in the Event Queue Data using the Event Create Instruction.

Event Queue Data

Event Creation Order	Event Queue Data Sequence
01	Event 01 (first occurrence)
02	Event 02 (first occurrence)
03	Event 03 (first occurrence)
04	Event 04 (first occurrence)
05	Event 01 (second occurrence)
06	Event 02 (second occurrence)
07	Event 03 (second occurrence)
08	Event 04 (second occurrence)

Events Creation Logic Section



Status of "Event Queue Data" after execution of the events creation logic

Name	Value
EventQueue_Data	{...}
EventQueue_Data[0]	{...}
EventQueue_Data[0].Type	1
EventQueue_Data[0].ID	1
EventQueue_Data[0].Category	1
EventQueue_Data[0].Action	1
EventQueue_Data[0].Value	1
EventQueue_Data[0].Message	'Event 01 - Timer is greater than 100ms'
EventQueue_Data[0].EventTime_L	DT#2016-10-03-14:13:07.116_676(UTC-06:00)
EventQueue_Data[0].EventTime_D	{...}
EventQueue_Data[1]	{...}
EventQueue_Data[1].Type	2
EventQueue_Data[1].ID	2
EventQueue_Data[1].Category	2
EventQueue_Data[1].Action	2
EventQueue_Data[1].Value	2
EventQueue_Data[1].Message	'Event 02 - Timer is greater than 200ms'
EventQueue_Data[1].EventTime_L	DT#2016-10-03-14:13:07.216_604(UTC-06:00)
EventQueue_Data[1].EventTime_D	{...}
EventQueue_Data[2]	{...}
EventQueue_Data[2].Type	3
EventQueue_Data[2].ID	3
EventQueue_Data[2].Category	3
EventQueue_Data[2].Action	3
EventQueue_Data[2].Value	3
EventQueue_Data[2].Message	'Event 03 - Timer is greater than 300ms'
EventQueue_Data[2].EventTime_L	DT#2016-10-03-14:13:07.316_565(UTC-06:00)
EventQueue_Data[2].EventTime_D	{...}
EventQueue_Data[3]	{...}
EventQueue_Data[3].Type	4
EventQueue_Data[3].ID	4
EventQueue_Data[3].Category	4
EventQueue_Data[3].Action	4
EventQueue_Data[3].Value	4
EventQueue_Data[3].Message	'Event 04 - Timer is greater than 400ms'
EventQueue_Data[3].EventTime_L	DT#2016-10-03-14:13:07.416_726(UTC-06:00)
EventQueue_Data[3].EventTime_D	{...}

Name	Value
EventQueue_Data[4]	{...}
EventQueue_Data[4].Type	1
EventQueue_Data[4].ID	1
EventQueue_Data[4].Category	1
EventQueue_Data[4].Action	1
EventQueue_Data[4].Value	1
EventQueue_Data[4].Message	'Event 01 - Timer is greater than 100ms'
EventQueue_Data[4].EventTime_L	DT#2016-10-03-14:13:08.116_579(UTC-06:00)
EventQueue_Data[4].EventTime_D	{...}
EventQueue_Data[5]	{...}
EventQueue_Data[5].Type	2
EventQueue_Data[5].ID	2
EventQueue_Data[5].Category	2
EventQueue_Data[5].Action	2
EventQueue_Data[5].Value	2
EventQueue_Data[5].Message	'Event 02 - Timer is greater than 200ms'
EventQueue_Data[5].EventTime_L	DT#2016-10-03-14:13:08.216_611(UTC-06:00)
EventQueue_Data[5].EventTime_D	{...}
EventQueue_Data[6]	{...}
EventQueue_Data[6].Type	3
EventQueue_Data[6].ID	3
EventQueue_Data[6].Category	3
EventQueue_Data[6].Action	3
EventQueue_Data[6].Value	3
EventQueue_Data[6].Message	'Event 03 - Timer is greater than 300ms'
EventQueue_Data[6].EventTime_L	DT#2016-10-03-14:13:08.316_711(UTC-06:00)
EventQueue_Data[6].EventTime_D	{...}
EventQueue_Data[7]	{...}
EventQueue_Data[7].Type	4
EventQueue_Data[7].ID	4
EventQueue_Data[7].Category	4
EventQueue_Data[7].Action	4
EventQueue_Data[7].Value	4
EventQueue_Data[7].Message	'Event 04 - Timer is greater than 400ms'
EventQueue_Data[7].EventTime_L	DT#2016-10-03-14:13:08.416_705(UTC-06:00)
EventQueue_Data[7].EventTime_D	{...}

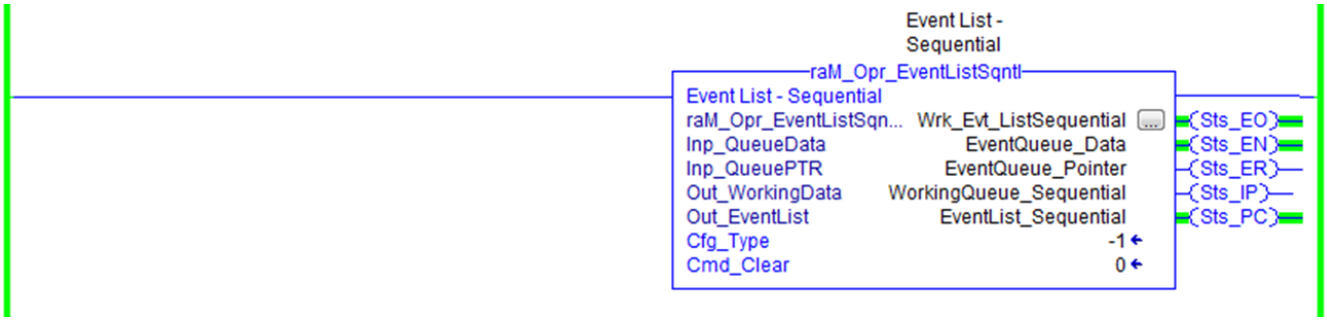
4.1.5 Sequential Event List

Event List Sequential Instruction will list events in sequential order (last event first).

Sequential Event List

Event List Order (last event first)	Sequential Event List Sequence	Count
01	Event 04 (second occurrence)	1
02	Event 03 (second occurrence)	1
03	Event 02 (second occurrence)	1
04	Event 01 (second occurrence)	1
05	Event 04 (first occurrence)	1
06	Event 03 (first occurrence)	1
07	Event 02 (first occurrence)	1
08	Event 01 (first occurrence)	1

Event List Sequential Instruction



Status of "Event List Sequential" after execution of Event List Sequential Instruction

Name	Value
EvenList_Sequential[0]	{...}
EvenList_Sequential[0].Event	{...}
EvenList_Sequential[0].Event.Type	4
EvenList_Sequential[0].Event.ID	4
EvenList_Sequential[0].Event.Category	4
EvenList_Sequential[0].Event.Action	4
EvenList_Sequential[0].Event.Value	4
EvenList_Sequential[0].Event.Message	'Event 04 - Timer is greater than 400ms'
EvenList_Sequential[0].Event.EventTime_L	DT#2016-10-03-14:13:08.416_705(UTC-06:00)
EvenList_Sequential[0].Event.EventTime_D	{...}
EvenList_Sequential[0].Count	1
EvenList_Sequential[1]	{...}
EvenList_Sequential[1].Event	{...}
EvenList_Sequential[1].Event.Type	3
EvenList_Sequential[1].Event.ID	3
EvenList_Sequential[1].Event.Category	3
EvenList_Sequential[1].Event.Action	3
EvenList_Sequential[1].Event.Value	3
EvenList_Sequential[1].Event.Message	'Event 03 - Timer is greater than 300ms'
EvenList_Sequential[1].Event.EventTime_L	DT#2016-10-03-14:13:08.316_711(UTC-06:00)
EvenList_Sequential[1].Event.EventTime_D	{...}
EvenList_Sequential[1].Count	1
EvenList_Sequential[2]	{...}
EvenList_Sequential[2].Event	{...}
EvenList_Sequential[2].Event.Type	2
EvenList_Sequential[2].Event.ID	2
EvenList_Sequential[2].Event.Category	2
EvenList_Sequential[2].Event.Action	2
EvenList_Sequential[2].Event.Value	2
EvenList_Sequential[2].Event.Message	'Event 02 - Timer is greater than 200ms'
EvenList_Sequential[2].Event.EventTime_L	DT#2016-10-03-14:13:08.216_611(UTC-06:00)
EvenList_Sequential[2].Event.EventTime_D	{...}
EvenList_Sequential[2].Count	1
EvenList_Sequential[3]	{...}
EvenList_Sequential[3].Event	{...}
EvenList_Sequential[3].Event.Type	1
EvenList_Sequential[3].Event.ID	1
EvenList_Sequential[3].Event.Category	1
EvenList_Sequential[3].Event.Action	1
EvenList_Sequential[3].Event.Value	1
EvenList_Sequential[3].Event.Message	'Event 01 - Timer is greater than 100ms'
EvenList_Sequential[3].Event.EventTime_L	DT#2016-10-03-14:13:08.116_579(UTC-06:00)
EvenList_Sequential[3].Event.EventTime_D	{...}
EvenList_Sequential[3].Count	1

Name	Value
EvenList_Sequential[4]	{...}
EvenList_Sequential[4].Event	{...}
EvenList_Sequential[4].Event.Type	4
EvenList_Sequential[4].Event.ID	4
EvenList_Sequential[4].Event.Category	4
EvenList_Sequential[4].Event.Action	4
EvenList_Sequential[4].Event.Value	4
EvenList_Sequential[4].Event.Message	'Event 04 - Timer is greater than 400ms'
EvenList_Sequential[4].Event.EventTime_L	DT#2016-10-03-14:13:07.416_726(UTC-06:00)
EvenList_Sequential[4].Event.EventTime_D	{...}
EvenList_Sequential[4].Count	1
EvenList_Sequential[5]	{...}
EvenList_Sequential[5].Event	{...}
EvenList_Sequential[5].Event.Type	3
EvenList_Sequential[5].Event.ID	3
EvenList_Sequential[5].Event.Category	3
EvenList_Sequential[5].Event.Action	3
EvenList_Sequential[5].Event.Value	3
EvenList_Sequential[5].Event.Message	'Event 03 - Timer is greater than 300ms'
EvenList_Sequential[5].Event.EventTime_L	DT#2016-10-03-14:13:07.316_565(UTC-06:00)
EvenList_Sequential[5].Event.EventTime_D	{...}
EvenList_Sequential[5].Count	1
EvenList_Sequential[6]	{...}
EvenList_Sequential[6].Event	{...}
EvenList_Sequential[6].Event.Type	2
EvenList_Sequential[6].Event.ID	2
EvenList_Sequential[6].Event.Category	2
EvenList_Sequential[6].Event.Action	2
EvenList_Sequential[6].Event.Value	2
EvenList_Sequential[6].Event.Message	'Event 02 - Timer is greater than 200ms'
EvenList_Sequential[6].Event.EventTime_L	DT#2016-10-03-14:13:07.216_604(UTC-06:00)
EvenList_Sequential[6].Event.EventTime_D	{...}
EvenList_Sequential[6].Count	1
EvenList_Sequential[7]	{...}
EvenList_Sequential[7].Event	{...}
EvenList_Sequential[7].Event.Type	1
EvenList_Sequential[7].Event.ID	1
EvenList_Sequential[7].Event.Category	1
EvenList_Sequential[7].Event.Action	1
EvenList_Sequential[7].Event.Value	1
EvenList_Sequential[7].Event.Message	'Event 01 - Timer is greater than 100ms'
EvenList_Sequential[7].Event.EventTime_L	DT#2016-10-03-14:13:07.116_676(UTC-06:00)
EvenList_Sequential[7].Event.EventTime_D	{...}
EvenList_Sequential[7].Count	1

4.1.6 Analytical Event List

Event List Analytical Instruction will list events in sequential order (latest event first) with a count of the event occurrences (no repetition of events).

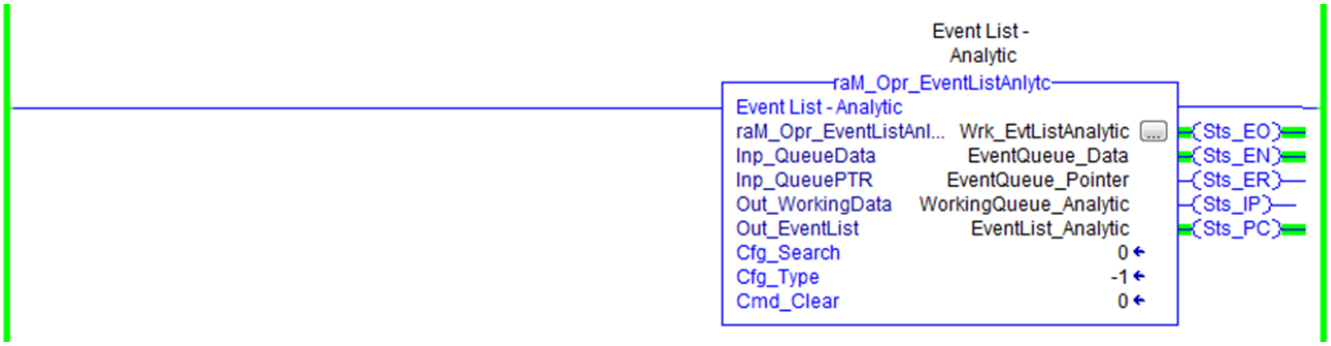
Analytical Event List

Event List Order (last event is first)	Analytical Event List Sequence	Count
01	Event 04 (second occurrence time stamp)	2
02	Event 03 (second occurrence time stamp)	2
03	Event 02 (second occurrence time stamp)	2
04	Event 01 (second occurrence time stamp)	2
05		0
06		0
07		0
08		0

Event List Analytic Instruction lists the events according to the search and type configuration

Input	Value
Cfg_Search	Search method for Existing Event Entries 0 = Message
Cfg_Type	Event Type Configuration -1 = All

Event List Analytic Instruction



Status of "Event List Analytic" after execution of Event List Analytic Instruction

Name	Value
EventList_Analytic[0]	{...}
EventList_Analytic[0].Event	{...}
EventList_Analytic[0].Event.Type	4
EventList_Analytic[0].Event.ID	4
EventList_Analytic[0].Event.Category	4
EventList_Analytic[0].Event.Action	4
EventList_Analytic[0].Event.Value	4
EventList_Analytic[0].Event.Message	'Event 04 - Timer is greater than 400ms'
EventList_Analytic[0].Event.EventTime_L	DT#2016-10-03-14:13:08.416_705(UTC-06:00)
EventList_Analytic[0].Event.EventTime_D	{...}
EventList_Analytic[0].Count	2
EventList_Analytic[1]	{...}
EventList_Analytic[1].Event	{...}
EventList_Analytic[1].Event.Type	3
EventList_Analytic[1].Event.ID	3
EventList_Analytic[1].Event.Category	3
EventList_Analytic[1].Event.Action	3
EventList_Analytic[1].Event.Value	3
EventList_Analytic[1].Event.Message	'Event 03 - Timer is greater than 300ms'
EventList_Analytic[1].Event.EventTime_L	DT#2016-10-03-14:13:08.316_711(UTC-06:00)
EventList_Analytic[1].Event.EventTime_D	{...}
EventList_Analytic[1].Count	2
EventList_Analytic[2]	{...}
EventList_Analytic[2].Event	{...}
EventList_Analytic[2].Event.Type	2
EventList_Analytic[2].Event.ID	2
EventList_Analytic[2].Event.Category	2
EventList_Analytic[2].Event.Action	2
EventList_Analytic[2].Event.Value	2
EventList_Analytic[2].Event.Message	'Event 02 - Timer is greater than 200ms'
EventList_Analytic[2].Event.EventTime_L	DT#2016-10-03-14:13:08.216_611(UTC-06:00)
EventList_Analytic[2].Event.EventTime_D	{...}
EventList_Analytic[2].Count	2
EventList_Analytic[3]	{...}
EventList_Analytic[3].Event	{...}
EventList_Analytic[3].Event.Type	1
EventList_Analytic[3].Event.ID	1
EventList_Analytic[3].Event.Category	1
EventList_Analytic[3].Event.Action	1
EventList_Analytic[3].Event.Value	1
EventList_Analytic[3].Event.Message	'Event 01 - Timer is greater than 100ms'
EventList_Analytic[3].Event.EventTime_L	DT#2016-10-03-14:13:08.116_579(UTC-06:00)
EventList_Analytic[3].Event.EventTime_D	{...}
EventList_Analytic[3].Count	2

Name	Value
EventList_Analytic[4]	{...}
EventList_Analytic[4].Event	{...}
EventList_Analytic[4].Event.Type	0
EventList_Analytic[4].Event.ID	0
EventList_Analytic[4].Event.Category	0
EventList_Analytic[4].Event.Action	0
EventList_Analytic[4].Event.Value	0
EventList_Analytic[4].Event.Message	''
EventList_Analytic[4].Event.EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
EventList_Analytic[4].Event.EventTime_D	{...}
EventList_Analytic[4].Count	0
EventList_Analytic[5]	{...}
EventList_Analytic[5].Event	{...}
EventList_Analytic[5].Event.Type	0
EventList_Analytic[5].Event.ID	0
EventList_Analytic[5].Event.Category	0
EventList_Analytic[5].Event.Action	0
EventList_Analytic[5].Event.Value	0
EventList_Analytic[5].Event.Message	''
EventList_Analytic[5].Event.EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
EventList_Analytic[5].Event.EventTime_D	{...}
EventList_Analytic[5].Count	0
EventList_Analytic[6]	{...}
EventList_Analytic[6].Event	{...}
EventList_Analytic[6].Event.Type	0
EventList_Analytic[6].Event.ID	0
EventList_Analytic[6].Event.Category	0
EventList_Analytic[6].Event.Action	0
EventList_Analytic[6].Event.Value	0
EventList_Analytic[6].Event.Message	''
EventList_Analytic[6].Event.EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
EventList_Analytic[6].Event.EventTime_D	{...}
EventList_Analytic[6].Count	0
EventList_Analytic[7]	{...}
EventList_Analytic[7].Event	{...}
EventList_Analytic[7].Event.Type	0
EventList_Analytic[7].Event.ID	0
EventList_Analytic[7].Event.Category	0
EventList_Analytic[7].Event.Action	0
EventList_Analytic[7].Event.Value	0
EventList_Analytic[7].Event.Message	''
EventList_Analytic[7].Event.EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
EventList_Analytic[7].Event.EventTime_D	{...}
EventList_Analytic[7].Count	0

4.1.7 Watch Event 03

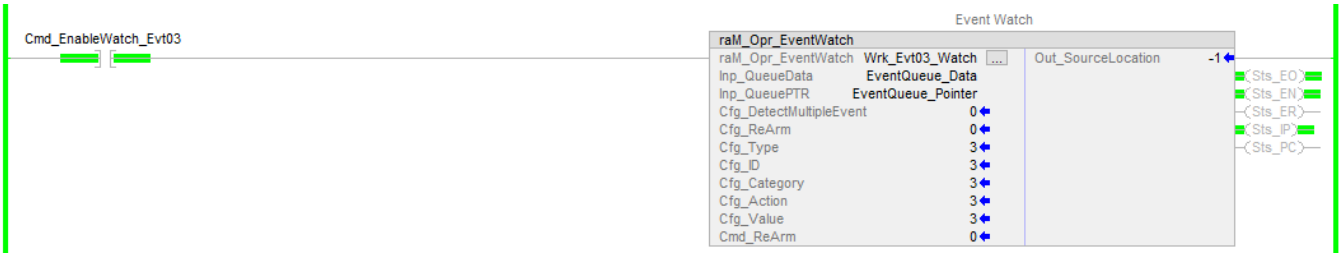
Watch Event Instruction monitors the Event Queue Data. If instruction sees an event that matches the configuration, it will capture the location of the event in the Event Queue Data (Out_SourceLocation).

Watch Event 03 will capture an event that exactly matches its configuration.
In this case, Event 03 matches the configuration.

Watch Event 03 Instruction Configuration

Cfg_Type = 3
Cfg_ID = 3
Cfg_Category = 3
Cfg_Action = 3
Cfg_Value = 3

Watch Event 03 Rung



4.1.8 Machine Event Queue

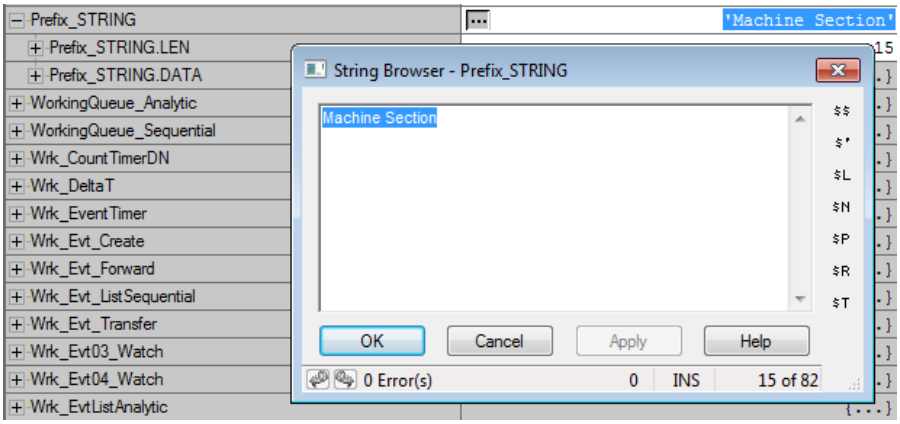
Event List Forward Instruction will forward Event 03 to the Machine Event Queue adding a Prefix.

Machine Event Queue receives only Event 03 and displays the events in the queue in the order that they were captured (first occurrence is the first on the queue).

Machine Event Queue

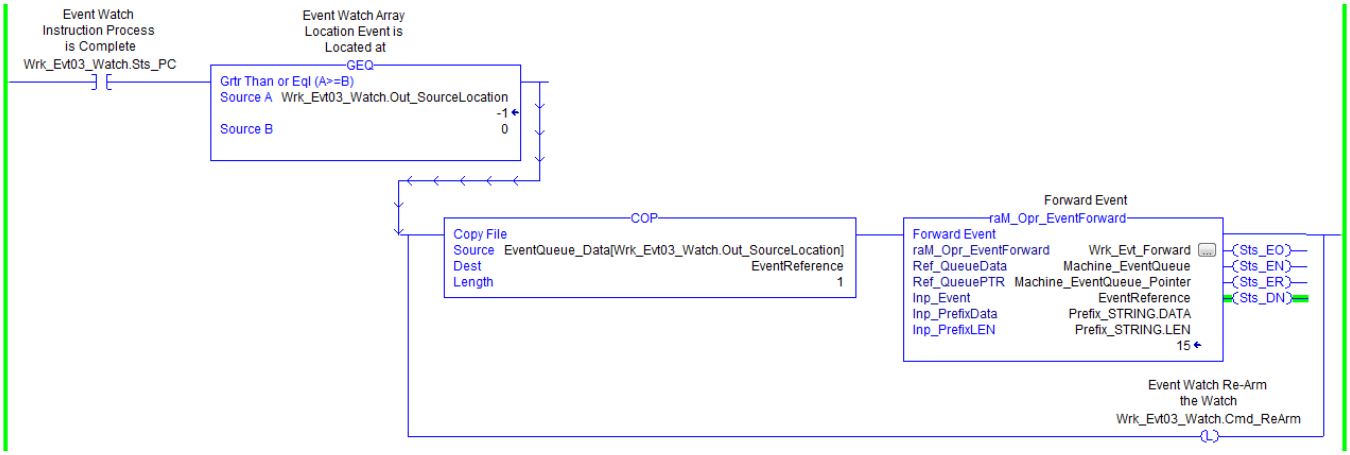
Event List Order (last event is first)	Machine Event Queue Sequence
01	Event 03 (first occurrence time stamp)
02	Event 03 (second occurrence time stamp)
03	
04	
05	
06	
07	
08	

The prefix used in this example (Machine Section) is prepopulated in the Prefix STRING Tag.



Events Forward to Machine Event Queue Logic Section

Note: The GEQ instruction avoids a negative number (-1) to trigger Event Forward Instruction, thus preventing an invalid array location lookup and causing a controller fault.



Status of "Machine Event Queue" after execution of forward event instruction.

Machine_EventQueue	{...}
Machine_EventQueue[0]	{...}
Machine_EventQueue[0].Type	3
Machine_EventQueue[0].ID	3
Machine_EventQueue[0].Category	3
Machine_EventQueue[0].Action	3
Machine_EventQueue[0].Value	3
Machine_EventQueue[0].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
Machine_EventQueue[0].EventTime_L	DT#2016-10-03-14:13:07.316_565(UTC-06:00)
Machine_EventQueue[0].EventTime_D	{...}
Machine_EventQueue[1]	{...}
Machine_EventQueue[1].Type	3
Machine_EventQueue[1].ID	3
Machine_EventQueue[1].Category	3
Machine_EventQueue[1].Action	3
Machine_EventQueue[1].Value	3
Machine_EventQueue[1].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
Machine_EventQueue[1].EventTime_L	DT#2016-10-03-14:13:08.316_711(UTC-06:00)
Machine_EventQueue[1].EventTime_D	{...}
Machine_EventQueue[2]	{...}
Machine_EventQueue[2].Type	0
Machine_EventQueue[2].ID	0
Machine_EventQueue[2].Category	0
Machine_EventQueue[2].Action	0
Machine_EventQueue[2].Value	0
Machine_EventQueue[2].Message	''
Machine_EventQueue[2].EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
Machine_EventQueue[2].EventTime_D	{...}
Machine_EventQueue[3]	{...}
Machine_EventQueue[4]	{...}
Machine_EventQueue[5]	{...}
Machine_EventQueue[6]	{...}
Machine_EventQueue[7]	{...}

4.1.9 Watch Event 04

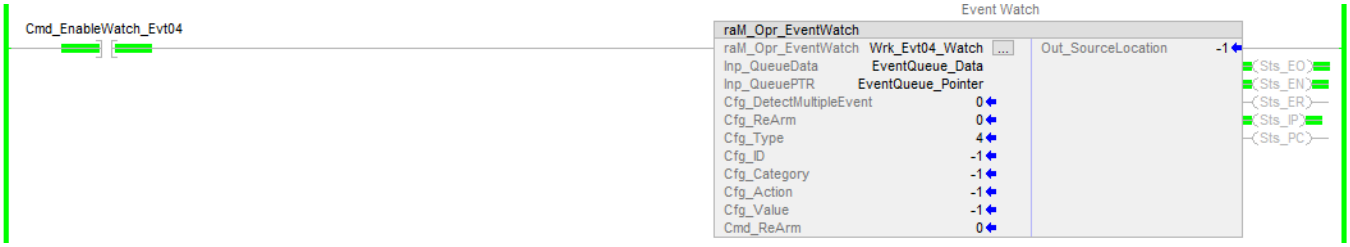
Watch Event Instruction monitors the Event Queue Data. If instruction sees an event that matches the configuration, it will capture the location of the event in the Event Queue Data (Out_SourceLocation).

Watch Event 04 will capture an event that matches has a Cfg_Type equals 4.
In this case, Event 04 matches the configuration.

Watch Event 04 Instruction Configuration

Cfg_Type = 4
Cfg_ID = -1
Cfg_Category = -1
Cfg_Action = -1
Cfg_Value = -1

Watch Event 04 Rung



4.1.10 Other Queue Data

Event List Transfer Instruction will transfer Event 04 to the Other Queue Data, adding a Prefix.

The prefix used in this example (Machine Section) is prepopulated in the Prefix_STRING tag.

OtherQueue_Data tag datatype is UDT_Events[20].
OtherQueue_Data demonstrates the fact that users can transfer an event to a different format.

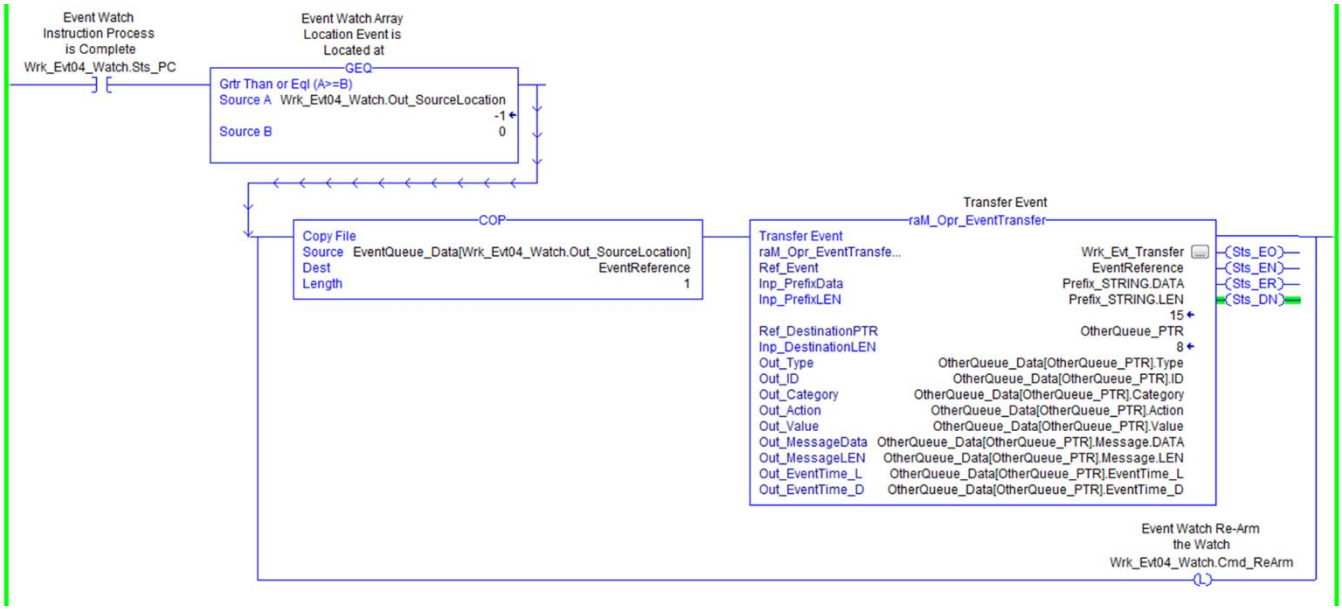
OtherQueue_Data receives only Event 04 and displays the events in the queue in the order that they were captured (first occurrence is the first on the queue).

Other Queue Data

Event List Order (last event is first)	Other Queue Data Sequence
01	Event 04 (first occurrence time stamp)
02	Event 04 (second occurrence time stamp)
03	
04	
05	
06	
07	
08	

Events Transfer to Other Queue Data Logic Section

Note: The GEQ instruction avoids a negative number (-1) to trigger Event Forward Instruction, thus preventing an invalid array location lookup and causing a controller fault.

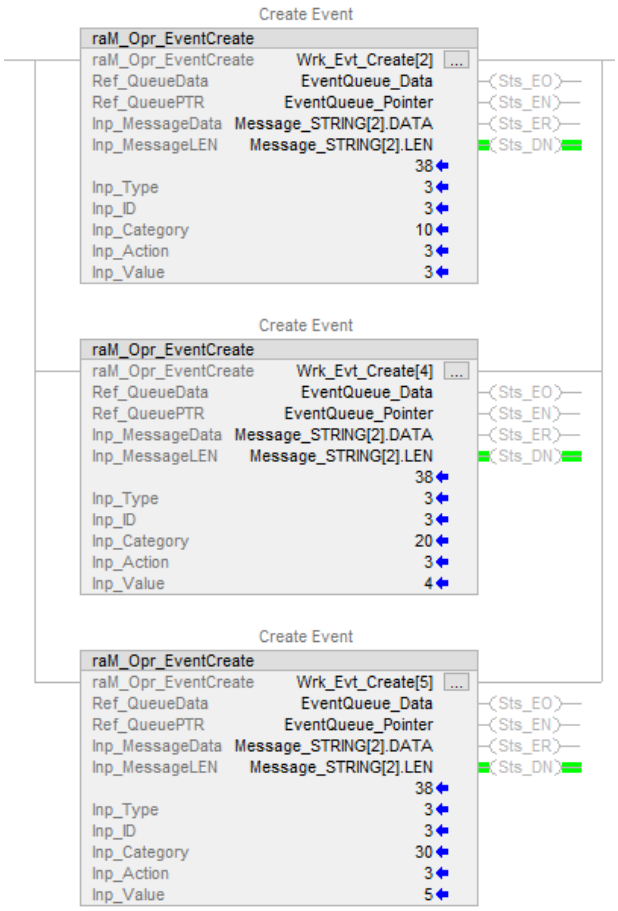


Status of "Other Queue Data" after execution of transfer instruction

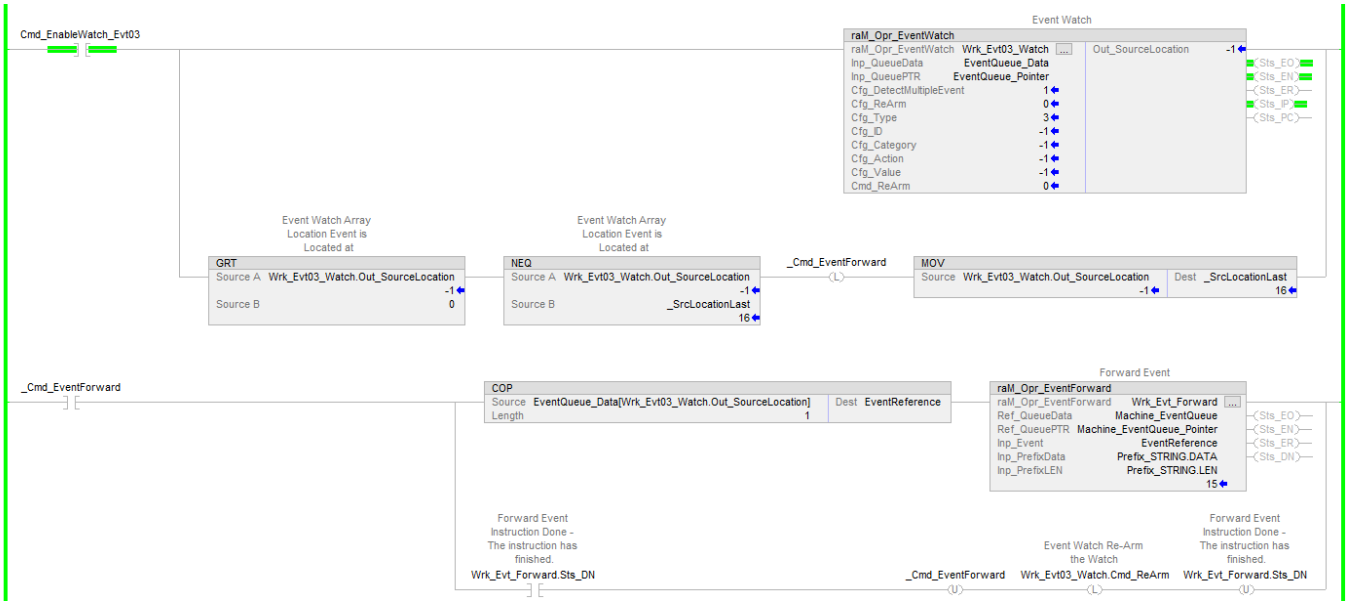
⊟ OtherQueue_Data[0]	{...}
⊞ OtherQueue_Data[0].Type	4
⊞ OtherQueue_Data[0].ID	4
⊞ OtherQueue_Data[0].Category	4
⊞ OtherQueue_Data[0].Action	4
⊞ OtherQueue_Data[0].Value	4
⊞ OtherQueue_Data[0].Message	'Machine Section: Event 04 - Timer is greater than 400ms'
OtherQueue_Data[0].EventTime_L	DT#2016-10-03-14:13:07.416_726(UTC-06:00)
⊞ OtherQueue_Data[0].EventTime_D	{...}
OtherQueue_Data[0].UserDefined01	0.0
⊞ OtherQueue_Data[0].UserDefined02	0
⊟ OtherQueue_Data[1]	{...}
⊞ OtherQueue_Data[1].Type	4
⊞ OtherQueue_Data[1].ID	4
⊞ OtherQueue_Data[1].Category	4
⊞ OtherQueue_Data[1].Action	4
⊞ OtherQueue_Data[1].Value	4
⊞ OtherQueue_Data[1].Message	'Machine Section: Event 04 - Timer is greater than 400ms'
OtherQueue_Data[1].EventTime_L	DT#2016-10-03-14:13:08.416_705(UTC-06:00)
⊞ OtherQueue_Data[1].EventTime_D	{...}
OtherQueue_Data[1].UserDefined01	0.0
⊞ OtherQueue_Data[1].UserDefined02	0
⊟ OtherQueue_Data[2]	{...}
⊞ OtherQueue_Data[2].Type	0
⊞ OtherQueue_Data[2].ID	0
⊞ OtherQueue_Data[2].Category	0
⊞ OtherQueue_Data[2].Action	0
⊞ OtherQueue_Data[2].Value	0
⊞ OtherQueue_Data[2].Message	''
OtherQueue_Data[2].EventTime_L	DT#1969-12-31-18:00:00.000_000(UTC-06:00)
⊞ OtherQueue_Data[2].EventTime_D	{...}
OtherQueue_Data[2].UserDefined01	0.0
⊞ OtherQueue_Data[2].UserDefined02	0
⊞ OtherQueue_Data[3]	{...}
⊞ OtherQueue_Data[4]	{...}
⊞ OtherQueue_Data[5]	{...}
⊞ OtherQueue_Data[6]	{...}
⊞ OtherQueue_Data[7]	{...}
⊞ OtherQueue_Data[8]	{...}

4.2 Event Watch – Detect Multiple Event

Here is an example to demonstrate the function of Cfg_DetectMultipleEvent parameter. The below rung, when triggered, creates multiple events in a single scan.



For this example, the Event Watch instruction is used along with EventForward instruction to detect an event and then forward the event to another Queue using EventForward instruction. The 3 EventCreate instructions create event with Type as 3 which is compatible with the EventWatch configuration.



Event03_Watch will capture an event that exactly matches its configuration.
In this case, Event 03 matches the configuration.

Watch Event 03 Instruction Configuration

Cfg_Type = 3
Cfg_ID = -1
Cfg_Category = -1
Cfg_Action = -1
Cfg_Value = -1

4.2.1 Cfg_DetectMultipleEvent = 0

Below is sequence of events

1. The EventQueue_Pointer is at location 2 and EventWatch instruction is at 2
2. EventCreate instructions are triggered and 3 events are created in the EventQueue. The EventQueue_Pointer is moved to location 5
3. Below is the EventQueue_Data which shows the 3 events created by the EventCreate instructions.

◢ EventQueue_Data[2]	{...}
▷ EventQueue_Data[2].Type	3
▷ EventQueue_Data[2].ID	3
▷ EventQueue_Data[2].Category	10
▷ EventQueue_Data[2].Action	3
▷ EventQueue_Data[2].Value	3
▷ EventQueue_Data[2].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[2].EventTime_L	DT#2021-10-13-06:48:24.153_816(UTC-05:00)
▷ EventQueue_Data[2].EventTime_D	{...}
◢ EventQueue_Data[3]	{...}
▷ EventQueue_Data[3].Type	3
▷ EventQueue_Data[3].ID	3
▷ EventQueue_Data[3].Category	20
▷ EventQueue_Data[3].Action	3
▷ EventQueue_Data[3].Value	4
▷ EventQueue_Data[3].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[3].EventTime_L	DT#2021-10-13-06:48:24.153_842(UTC-05:00)
▷ EventQueue_Data[3].EventTime_D	{...}
◢ EventQueue_Data[4]	{...}
▷ EventQueue_Data[4].Type	3
▷ EventQueue_Data[4].ID	3
▷ EventQueue_Data[4].Category	30
▷ EventQueue_Data[4].Action	3
▷ EventQueue_Data[4].Value	5
▷ EventQueue_Data[4].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[4].EventTime_L	DT#2021-10-13-06:48:24.153_861(UTC-05:00)

4. When Cfg_DetectMultipleEvent is 0, and EventWatch instruction rung is set True, the EventWatch instruction will detect the first event, which is in EventQueue_Data[2]. Once the event is detected, the EventWatch instruction will give its location to Out_SourceLocation. The same is used by EventForward to forward the event to Machine_EventQueue.
5. When ReArmed, the EventWatch instruction will move its pointer to the location where the EventQueue_Pointer is. This means that the EventWatch instruction skips rest of the events and moves to location 5.

As shown below the EventWatch instruction only detected the first event and the same is forwarded to Machine_EventQueue using EventForward instruction.

◢ Machine_EventQueue	{...}
◢ Machine_EventQueue[0]	{...}
▷ Machine_EventQueue[0].Type	3
▷ Machine_EventQueue[0].ID	3
▷ Machine_EventQueue[0].Category	10
▷ Machine_EventQueue[0].Action	3
▷ Machine_EventQueue[0].Value	3
▷ Machine_EventQueue[0].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
▷ Machine_EventQueue[0].EventTime_L	DT#2021-10-13-06:48:24.153_816(UTC-05:00)
▷ Machine_EventQueue[0].EventTime_D	{...}

4.2.2 Cfg_DetectMultipleEvent = 1

Below is sequence of events

1. The EventQueue_Pointer is at location 2 and EventWatch instruction is at 2
2. EventCreate instructions are triggered and 3 events are created in the EventQueue. The EventQueue_Pointer is moved to location 5
3. Below is the EventQueue_Data which shows the 3 events created by the EventCreate instructions.

◢ EventQueue_Data[2]	{...}
▷ EventQueue_Data[2].Type	3
▷ EventQueue_Data[2].ID	3
▷ EventQueue_Data[2].Category	10
▷ EventQueue_Data[2].Action	3
▷ EventQueue_Data[2].Value	3
▷ EventQueue_Data[2].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[2].EventTime_L	DT#2021-10-13-07:01:34.225_906(UTC-05:00)
▷ EventQueue_Data[2].EventTime_D	{...}
◢ EventQueue_Data[3]	{...}
▷ EventQueue_Data[3].Type	3
▷ EventQueue_Data[3].ID	3
▷ EventQueue_Data[3].Category	20
▷ EventQueue_Data[3].Action	3
▷ EventQueue_Data[3].Value	4
▷ EventQueue_Data[3].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[3].EventTime_L	DT#2021-10-13-07:01:34.225_930(UTC-05:00)
▷ EventQueue_Data[3].EventTime_D	{...}
◢ EventQueue_Data[4]	{...}
▷ EventQueue_Data[4].Type	3
▷ EventQueue_Data[4].ID	3
▷ EventQueue_Data[4].Category	30
▷ EventQueue_Data[4].Action	3
▷ EventQueue_Data[4].Value	5
▷ EventQueue_Data[4].Message	'Event 03 - Timer is greater than 300ms'
▷ EventQueue_Data[4].EventTime_L	DT#2021-10-13-07:01:34.225_949(UTC-05:00)

4. When Cfg_DetectMultipleEvent is 1, the EventWatch instruction will detect the first event, which is in EventQueue_Data[2]. Once the event is detected, the EventWatch instruction will give its location to Out_SourceLocation. Now EventWatch instruction is at location 2, while EventQueue_Pointer is at 4. The detected event is forwarded to Machine_EventQueue using EventForward instruction.
5. When ReArmed, the EventWatch instruction will detect the event at location 3 and since its compatible with the configuration, this event location is also given to Out_SourceLocation. EventWatch instruction is at location 3. The detected event is forwarded to Machine_EventQueue using EventForward instruction.
6. When ReArmed again, the EventWatch instruction will detect the event at location 4 and since its compatible with the configuration, this event location is also given to Out_SourceLocation. EventWatch instruction is at location 4. The detected event is forwarded to Machine_EventQueue using EventForward instruction.
7. When ReArmed, the EventWatch instruction moves to location 5 and stays in Sts_IP until next event is detected.

As shown below the EventWatch instruction is now able to detect all the event location as per configuration. The same is moved to next Queue (Machine_EventQueue) using EventForward instruction.

◢ Machine_EventQueue	{...}
◢ Machine_EventQueue[0]	{...}
▷ Machine_EventQueue[0].Type	3
▷ Machine_EventQueue[0].ID	3
▷ Machine_EventQueue[0].Category	10
▷ Machine_EventQueue[0].Action	3
▷ Machine_EventQueue[0].Value	3
▷ Machine_EventQueue[0].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
▷ Machine_EventQueue[0].EventTime_L	DT#2021-10-13-07:01:34.225_906(UTC-05:00)
▷ Machine_EventQueue[0].EventTime_D	{...}
◢ Machine_EventQueue[1]	{...}
▷ Machine_EventQueue[1].Type	3
▷ Machine_EventQueue[1].ID	3
▷ Machine_EventQueue[1].Category	20
▷ Machine_EventQueue[1].Action	3
▷ Machine_EventQueue[1].Value	4
▷ Machine_EventQueue[1].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
▷ Machine_EventQueue[1].EventTime_L	DT#2021-10-13-07:01:34.225_930(UTC-05:00)
▷ Machine_EventQueue[1].EventTime_D	{...}
◢ Machine_EventQueue[2]	{...}
▷ Machine_EventQueue[2].Type	3
▷ Machine_EventQueue[2].ID	3
▷ Machine_EventQueue[2].Category	30
▷ Machine_EventQueue[2].Action	3
▷ Machine_EventQueue[2].Value	5
▷ Machine_EventQueue[2].Message	'Machine Section: Event 03 - Timer is greater than 300ms'
▷ Machine_EventQueue[2].EventTime_L	DT#2021-10-13-07:01:34.225_949(UTC-05:00)

5 Appendix

General

This document provides a programmer with details on this instruction for a Logix-based controller, its Application Code Manager library content, and visualization content, if applicable. This document assumes that the programmer is already familiar with how the Logix-based controller stores and processes data.

Novice programmers should read all the details about an instruction before using the instruction. Experienced programmers can refer to the instruction information to verify details.

IMPORTANT This object includes a Logix Designer Asset for use with Version 30 or later of Studio 5000 Logix Designer.

Common Information for All Instructions

Rockwell Automation Application Content may contain many common attributes or objects. Refer to the following reference materials for more information:

- Foundations of Modular Programming, **IA-RM001C-EN-P**

Conventions and Related Terms

Data - Set and Clear

This manual uses set and clear to define the status of bits (Booleans) and values (non-Booleans):

This Term:	Means:
Set	The bit is set to 1 (ON) A value is set to any non-zero number
Clear	The bit is cleared to 0 (OFF) All the bits in a value are cleared to 0

Signal Processing - Edge and Level

This manual uses Edge and Level to describe how bit (BOOL) Commands, Settings, Configurations and Inputs to this instruction are sent by other logic and processed by this instruction.

Send/Receive Method:	Description:
Edge	• Action is triggered by "rising edge" transition of input (0-1) • Separate inputs are provided for complementary functions (such as "enable" and "disable") • Sending logic SETS the bit (writes a 1) to initiate the action; this instruction CLEARS the bit (to 0) immediately, then acts on the request if possible • LD: use conditioned OTL (Latch) to send • ST: use conditional assignment [if (condition) then bit:=1;] to send • FBD: OREF writes a 1 or 0 every scan, should use Level, not Edge Edge triggering allows multiple senders per Command, Setting, Configuration or Input (many-to-one relationship)
Level	• Action ("enable") is triggered by input being at a level (in a state, usually 1) • Opposite action ("disable") is triggered by input being in opposite state (0) • Sending logic SETS the bit (writes a 1) or CLEARS the bit (writes a 0); this instruction does not change the bit • LD: use OTE (Energize) to send • ST: use unconditional assignment [bit:= expression_resulting_in_1_or_0;] or "if-then-else" logic [if (condition) then bit:= 1; else bit:= 0;] • FBD: use OREF to the input bit Level triggering allows only one sender can drive each Level

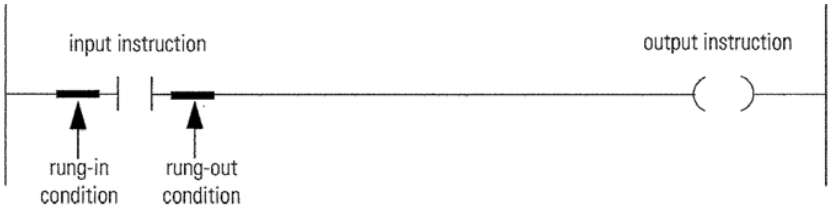
Instruction Execution - Edge and Continuous

This manual uses Edge and Continuous to describe how an instruction is designed to be executed.

Method:	Description:
Edge	• Instruction Action is triggered by "rising edge" transition of the rung-in-condition
Continuous	• Instruction Action is triggered by input being at a level (in a state, usually 1) • Opposite action is triggered by input being in opposite state (0) • Instructions designed for continuous execution should typically be used on rungs without input conditions present allowing the instruction to be continuously scanned

Relay Ladder Rung Condition

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-in condition). Based on the rung-in condition and the instruction, the controller sets the rung condition following the instruction (rung-out condition), which in turn, affects any subsequent instruction.



If the rung-in condition to an input instruction is true, the controller evaluates the instruction and sets the rung-out condition based on the results of the instruction. If the instruction evaluates to true, the rung-out condition is true; if the instruction evaluates to false, the rung-out condition is false.

IMPORTANT

The rung-in condition is reflected in the EnableIn parameter and determines how the system performs each Add-On Instruction. If the EnableIn signal is TRUE, the system performs the instruction's main logic routine. Conversely, if the EnableIn signal is FALSE, the system performs the instruction's EnableInFalse routine.

The instruction's main logic routine sets/clears the EnableOut parameter, which then determines the rung-out condition. The EnableInFalse routine cannot set the EnableOut parameter. If the rung-in condition is FALSE, then the EnableOut parameter and the rung-out condition will also be FALSE.

Pre-scan

On transition into RUN, the controller performs a pre-scan before the first scan. Pre-scan is a special scan of all routines in the controller. The controller scans all main routines and subroutines during pre-scan, but ignores jumps that could skip the execution of instructions. The controller performs all FOR loops and subroutine calls. If a subroutine is called more than once, it is performed each time it is called. The controller uses pre-scan of relay ladder instructions to reset non-retentive I/O and internal values.

During pre-scan, input values are not current and outputs are not written. The following conditions generate pre-scan:

- Transition from Program to Run mode.
- Automatically enter Run mode from a power-up condition.

Pre-scan does not occur for a program when:

- Program becomes scheduled while the controller is running.
- Program is unscheduled when the controller enters Run mode.

IMPORTANT

The Pre-scan process performs the Process Add-On Instruction's logic routine as FALSE and then performs its Pre-scan routine as TRUE.