MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Release 4, Version 1.0

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 1 - Overview

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

This set of Users Guide documents describes the implementation of Mitsubishi OEM PackML Implementation Templates[1] and steps on how to use the Templates to implement packaging machine control programs by OEM users. Using the Mitsubishi PackML templates enables OEMs to implement packaging machine control programs that satisfy the OMAC PackML standard and align with the OMAC PackML Implementation Guide with much reduced effort.

The main functions of the Mitsubishi PackML templates are to (1) handle PackML state and mode transitions, (2) accumulate machine execution time in each valid mode and state, and (3) process events (e.g. alarms and warnings) of machine operations. However, the Mitsubishi PackML templates are NOT intended to be used without modifications or enhancements with machine control PLC, motion and HMI programs. For example, different PLC, motion controller, and GOT types that are used in an actual OEM machine will require PLC, motion controllers, and GOT setup parameters to be adjusted accordingly.

These templates depend on PackML commands and status from PLC, motion and HMI programs to properly perform machine mode and state transitions at the unit machine level per ISA-88 definition. Thus it is OEM's responsibility to supply the proper commands and state status from their machine control programs to the Mitsubishi PackML templates in order for the PackML machine modes and states to function properly. Event handling function blocks are included in the Templates to enable easier and more consistent machine event handling.

The details on how the machine control programs should be integrated in the Mitsubishi PackML templates are described in this document.

2 PackML Template System Architecture

The PackML templates are designed to run on a system with the minimum of a R08CPU and a GT27 HMI. The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a R08CPU and the GOT is a GT27 with the resolution of 800 x 600.

Because of the large number of tags required to support the PackTags specification, an extended memory card may be required to be installed in the R08CPU.

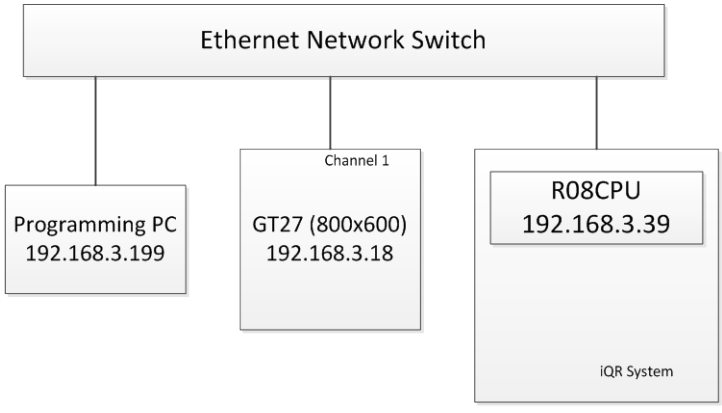


Figure 1 – Mitsubishi PackML GXW3 Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT and the PLC CPU using the Ethernet connections to download screen information and PLC project.

The configurations of these components are described in more details in other parts of the Mitsubishi PackML Implementation Users Guide.

[1] Also referred to as *Mitsubishi PackML Templates* or *PackML Templates*, or simply *Templates* in this document.

3 Mitsubishi PackML Template Key Components

The Mitsubishi PackML Template consists of the following key components that an OEM can use **directly without modifications**:

1. All PackTags defined and allocated to specific PLC registers
2. PackML_ModeStateManager Function Block
3. PackML_ModeStateTimes Function Block
4. Event Handling Function Blocks
   - CM_Event Function Block
   - Event_Manager Function Block
   - Event_Summation Function Block
   - Event_Sort Function Block

The PackTags, PackML Core function blocks, and Event Handling Function Blocks are developed in GX Works 3 and provided as integral parts of the PackML Template GX Works 3 program in the iQ Works Workspace.

The description and implementation of PackTags are described in *Mitsubishi PackML Implementation Users Guide – Part 3 PackTags Design Document*. The core PackML function blocks are described in *Mitsubishi PackML Implementation Users Guide – Part 4 PackML Core Function Block* document. The Event Handling Function Blocks are described in *Mitsubishi PackML Implementation Users Guide – Part 5 Event Handling Function Block* document.

4 Mitsubishi PackML Template Program Structure

The Mitsubishi PackML Template program utilizes the key components described in the above section and are organized following the OMAC Users Group PackML Implementation Guide and the ISA-88 Make2Pack modular structure as shown in Figure 2 below. All routines of the PackML template program are developed in GX Works 3 and function blocks are used extensively. The PLC languages used in this template are mainly Function Block Diagram (FBD) and Structure Text (ST).

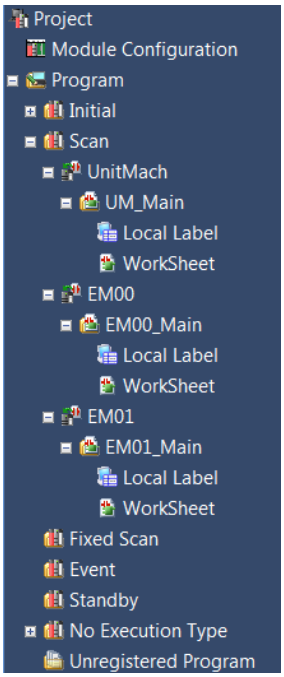


Figure 2 – Mitsubishi PackML Template Program Structure

Each Equipment Module Worksheet contains all the Control Modules within the Equipment Module and each Control Module is encapsulated into a function block itself.

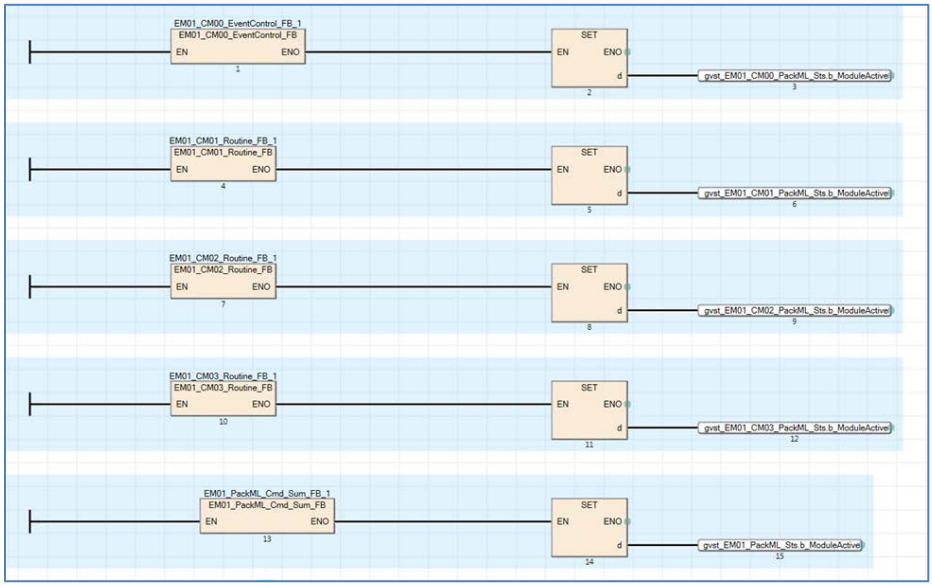


Figure 3 – Example Breakdown of an Equipment Module

As an example, Equipment Module 00 (EM00) is further divided into Control Modules as shown in Figure 3 above. In this case, Control Modules 00 through 03 are encapsulated in individual function blocks. In the GX Works3 project, all control modules for an Equipment Module are organized into a folder named EMxx_Subroutines as shown in the figure below:

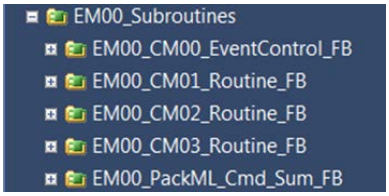


Figure 4 – Control Modules of Equipment Module 00

Each control module is implemented as a function block for easy project and code organization. When additional control modules are needed, they can be implemented as function blocks first and then inserted into the Equipment Module Worksheet.

In contrast to the Key Components, the Template program structure is intended to be modified to reflect the actual packaging machine that is being developed. One key objective of the Mitsubishi Template Implementation program is to demonstrate how a packaging machine program can be laid out and created. It is never intended to be used as is.

As shown Figure 2, the Mitsubishi PackML Template program is designed to represent a packaging machine (Referred to as Unit_Machine) consists of two equipment modules (EM00 and EM01). Each equipment module consists of four Control Modules (CM00 to CM03) for machine operations and a Control Module to integrate appropriate PackML commands and status for each Equipment Module from its control modules CM00 to CM03. An OEM has the flexibility to add or delete equipment modules and control modules to match the actual machine that is being built.

Release 4 of the Mitsubishi PackML Implementation template does include the function blocks and example codes for handling events (e.g. alarms and warnings) of the machine. An OEM needs to develop and incorporate machine control routines to handle events and utilize the Template to help aggregate and manage the event lists. The details of event handling are listed in Part 5 of the Users Guide.

5 High Level OEM Implementation Steps

High level steps of tailoring the Mitsubishi PackML Templates to an actual packaging machine are described in this section:

1. Install the latest version of the Mitsubishi iQ Works on the programming computer.
2. Establish the Ethernet communication among the iQ PLC system, the GOT, and the programming computer by configuring proper IP addresses and communication parameters of each device.
3. Analyze the Unit Machine design and divide the machine into proper equipment modules.
4. Define and allocate control functions into proper modular code and assign them to various control modules.
5. Follow the Mitsubishi PackML Template program structure and add or subtract equipment and control modules as appropriate. For example, one may add additional Equipment Modules EM02 and EM03 (by cutting, pasting, and modifying the labels and names using one of the existing module in the template) or delete control modules EM00_CM03 if it is not needed.
    a. The routine names such as "EM00_CM01_Routines" can be modified to "Load_HMI" for example to better reflect the actual purpose of the module which performs "Load Station Operator interface" functions.
6. Develop machine PLC code and assign them in proper modules using iQ Works and GX Works 3.
7. Develop the GOT and motion control programs using iQ Works and GT Designer 3 and MT Works 2 respectively
8. Load programs in PLC, motion control and GOT.

6 Parts of the PackML Implementation Users Guide

The Mitsubishi PackML Implementation Users Guide consists of 6 documents:

Documents	Descriptions
Part 1 - Overview	Overview of the Mitsubishi PackML Template package and program structure
Part 2 – MELSOFT Navigator	Descriptions of configuring the PackML Template System using iQ Works MELSOFT Navigator
Part 3 – PackTags	Design details of implementing PackTags in the PackML Templates
Part 4 – PackML Function Blocks	Design details and PLC code of the core PackML function blocks
Part 5 – Event Handling FBs	Design details and PLC code of event handling function blocks
Part 6 – Program Structure	Design details on the structure of the OEM Machine program following the OMAC Implementation guide and the Make2Pack modularization. Description on the initialization of PackML states, aggregation of PackML status and commands through various equipment and control modules, and steps to modify the aggregation of PackML status and commands when equipment modules and control modules are added or removed.
Part 7 – GOT Screens	Description of GOT sample screens to display PackML current mode and state and also the accumulated time for each mode and state.

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 2 - MELSOFT Navigator Configuration

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

This document describes the steps of configuring MELSOFT Navigator within the iQ Works software to establish the PackML Template System.

The PackML template system consists of a R08CPU and a GT27 HMI. The system architecture used to create the Mitsubishi PackML templates is shown in the following block diagram. The PLC is a R08CPU and the GOT is a GT27 with the resolution of 800 x 600.

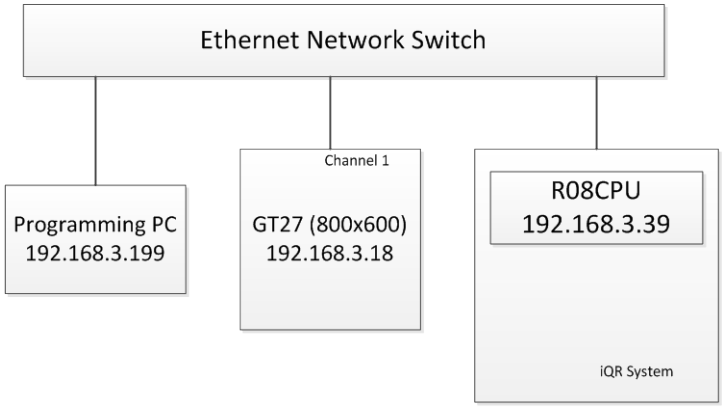


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT and the PLC CPU using the Ethernet connections to download screen information and PLC project.

The version of the MELSOFT Navigator used in creating Release 4 solution is Version 2.07H.

2 MELSOFT Navigator Configuration

Using the MELSOFT Navigator of the iQ Works package, one can create an integrated database that allows system labels to be used harmoniously among the PLC, GOT and Motion control programs.

2.1 Module Configuration

The first step of creating an integrated project is to define the Module Configuration using the MELSOFT Navigator. Figure 2 below shows how the Module Configuration is for the Release 4 solution in Navigator.

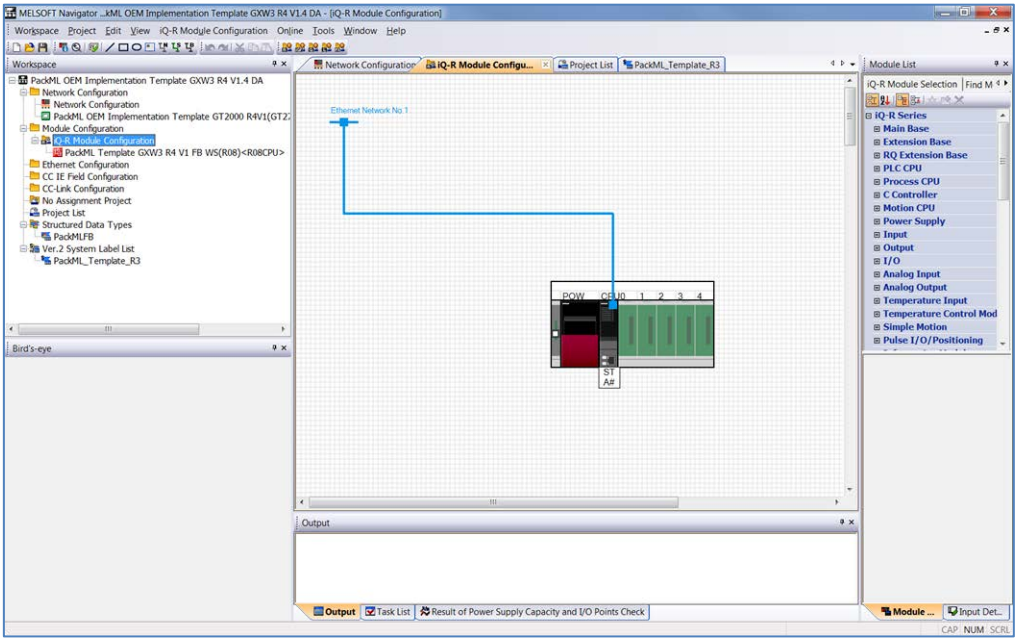


Figure 2 – Module Configuration

2.2 Network Configuration

After the Module Configuration is completed, the Module Configuration is automatically reflected in the Network Configuration workspace. For the PackML Template System, the Ethernet network is added to the Network Configuration as Network No. 1 as shown in Figure 3.

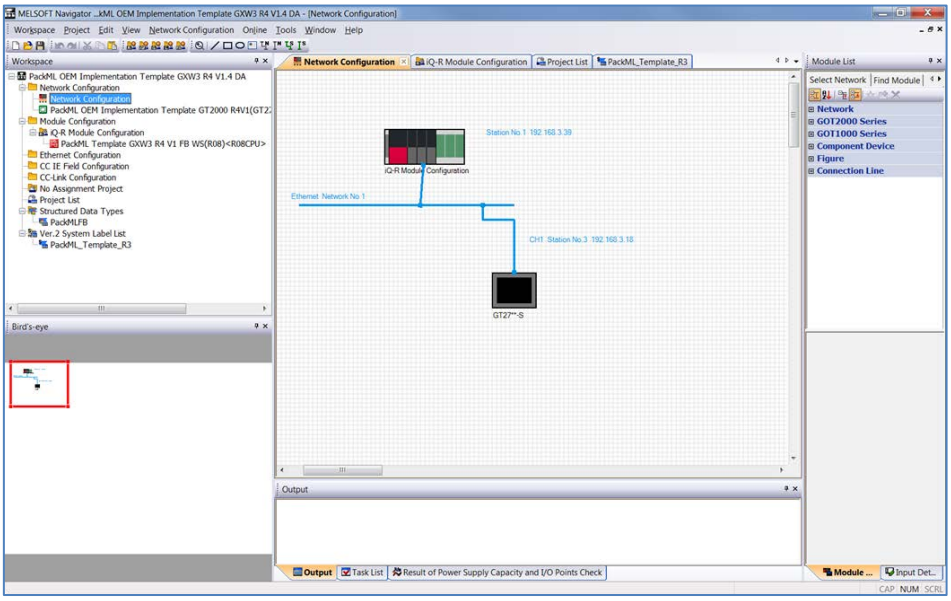


Figure 3 – PackML Template System Network Configuration

For the PackML Template System Release 4, a GT27 with the resolution of 800 x 600 is added to the system and then configured with the proper channel designations and IP address.

2.3 PLC and GOT Programs

In the PackML Template System Release 4 configuration, the PLC and GOT template programs are allocated to the PLC and GOT of the system.

The PLC program "PackML Template GXW3 R4 V1 FB" and the GOT program is "PackML OEM Implementation Template GT2000 R4V1"

Additional details of these programs are described in subsequent parts of this document set.

3 System Label Database

Once the PLC and GOT programs are allocated to the MELSOFT Navigator, the important next steps are to create the system label database so that the labels defined in one of the programs can be shared and used by another program.

In the PackML Template System, all system labels are originated from the PLC program. The figure below is an example showing the global labels defined in the PLC are assigned to the System Label Database and reflected to the System Label Data base being kept in the Navigator. Make sure the "Access from External Device" check boxes are checked to enable other devices, such as a GOT, other than the PLC to access these system labels.

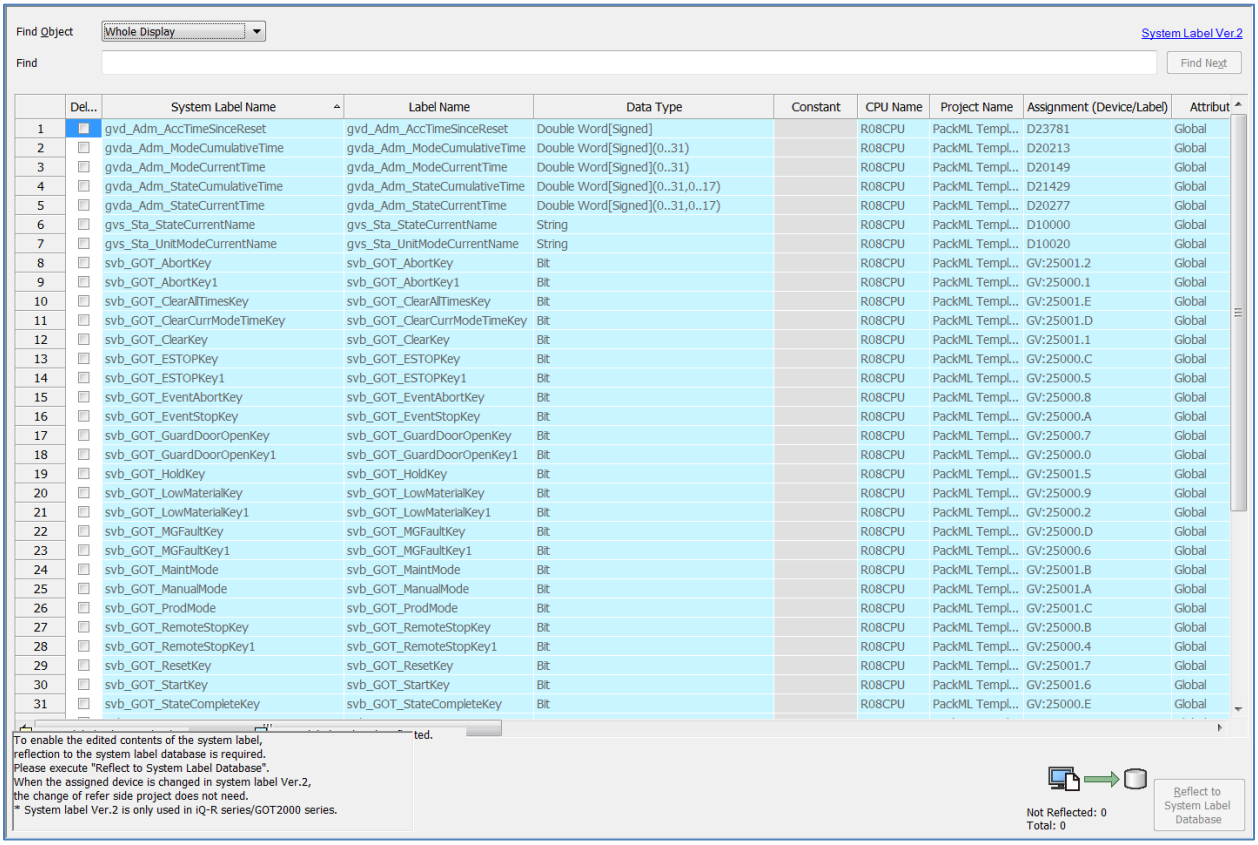
<Filter> | Easy Display | Display Setting | Check

	Label Name	Data Type		Class	Assign (Device/Label)	Comment	System Label Name	Access from External Device
1	svb_GOT_StateCompleteKey	Bit	...	VAR_GLOBAL			svb_GOT_StateCompleteKey	☑
2	svb_GOT_UnSuspendKey	Bit	...	VAR_GLOBAL			svb_GOT_UnSuspendKey	☑
3	svb_GOT_SuspendKey	Bit	...	VAR_GLOBAL			svb_GOT_SuspendKey	☑
4	svb_GOT_ClearKey	Bit	...	VAR_GLOBAL			svb_GOT_ClearKey	☑
5	svb_GOT_AbortKey	Bit	...	VAR_GLOBAL			svb_GOT_AbortKey	☑
6	svb_GOT_UnHoldKey	Bit	...	VAR_GLOBAL			svb_GOT_UnHoldKey	☑
7	svb_GOT_StopKey	Bit	...	VAR_GLOBAL			svb_GOT_StopKey	☑
8	svb_GOT_HoldKey	Bit	...	VAR_GLOBAL			svb_GOT_HoldKey	☑
9	svb_GOT_StartKey	Bit	...	VAR_GLOBAL			svb_GOT_StartKey	☑
10	svb_GOT_ResetKey	Bit	...	VAR_GLOBAL			svb_GOT_ResetKey	☑
11	svb_GOT_User2Mode	Bit	...	VAR_GLOBAL			svb_GOT_User2Mode	☑
12	svb_GOT_User1Mode	Bit	...	VAR_GLOBAL			svb_GOT_User1Mode	☑
13	svb_GOT_ManualMode	Bit	...	VAR_GLOBAL			svb_GOT_ManualMode	☑
14	svb_GOT_MaintMode	Bit	...	VAR_GLOBAL			svb_GOT_MaintMode	☑
15	svb_GOT_ProdMode	Bit	...	VAR_GLOBAL			svb_GOT_ProdMode	☑
16	svw_GOT_Screen_Switch	Word [Signed]	...	VAR_GLOBAL	D10040		svw_GOT_Screen_Switch	☑
17	svb_GOT_ClearCurrModeTimeKey	Bit	...	VAR_GLOBAL			svb_GOT_ClearCurrModeTimeKey	☑
18	svb_GOT_ClearAllTimesKey	Bit	...	VAR_GLOBAL			svb_GOT_ClearAllTimesKey	☑
19	svda_GOT_CurrentStateTimes	Double Word [Signed](0..31,0..17)	...	VAR_GLOBAL			svda_GOT_CurrentStateTimes	☑
20	svda_GOT_CumulativeStateTimes	Double Word [Signed](0..31,0..17)	...	VAR_GLOBAL			svda_GOT_CumulativeStateTimes	☑
21	svda_GOT_ModeCurrentTime	Double Word [Signed](0..31)	...	VAR_GLOBAL			svda_GOT_ModeCurrentTime	☑
22	svda_GOT_ModeCumulativeTime	Double Word [Signed](0..31)	...	VAR_GLOBAL			svda_GOT_ModeCumulativeTime	☑
23	svd_GOT_AccTimeSinceReset	Double Word [Signed]	...	VAR_GLOBAL			svd_GOT_AccTimeSinceReset	☑
24			...					☐

Figure 4 – Examples of System Label Assignments in GX Works 3

The steps required to reserve the global labels and reflect to the System Label Database are straightforward, and documented in Chapter 5 Registering Labels of "GX Works 3 Operating Manual" (Document # SH(NA)-081215ENG-F).

Once the System Labels are registered, they are shown in the Navigator as illustrated in Figure 5 and can be used by external devices.



	Del...	System Label Name	Label Name	Data Type	Constant	CPU Name	Project Name	Assignment (Device/Label)	Attribut
1		gvd_Adm_AccTimeSinceReset	gvd_Adm_AccTimeSinceReset	Double Word[Signed]		R08CPU	PackML Templ...	D23781	Global
2		gvda_Adm_ModeCumulativeTime	gvda_Adm_ModeCumulativeTime	Double Word[Signed](0..31)		R08CPU	PackML Templ...	D20213	Global
3		gvda_Adm_ModeCurrentTime	gvda_Adm_ModeCurrentTime	Double Word[Signed](0..31)		R08CPU	PackML Templ...	D20149	Global
4		gvda_Adm_StateCumulativeTime	gvda_Adm_StateCumulativeTime	Double Word[Signed](0..31,0..17)		R08CPU	PackML Templ...	D21429	Global
5		gvda_Adm_StateCurrentTime	gvda_Adm_StateCurrentTime	Double Word[Signed](0..31,0..17)		R08CPU	PackML Templ...	D20277	Global
6		gvs_Sta_StateCurrentName	gvs_Sta_StateCurrentName	String		R08CPU	PackML Templ...	D10000	Global
7		gvs_Sta_UnitModeCurrentName	gvs_Sta_UnitModeCurrentName	String		R08CPU	PackML Templ...	D10020	Global
8		svb_GOT_AbortKey	svb_GOT_AbortKey	Bit		R08CPU	PackML Templ...	GV:25001.2	Global
9		svb_GOT_AbortKey1	svb_GOT_AbortKey1	Bit		R08CPU	PackML Templ...	GV:25000.1	Global
10		svb_GOT_ClearAllTimesKey	svb_GOT_ClearAllTimesKey	Bit		R08CPU	PackML Templ...	GV:25001.E	Global
11		svb_GOT_ClearCurrModeTimeKey	svb_GOT_ClearCurrModeTimeKey	Bit		R08CPU	PackML Templ...	GV:25001.D	Global
12		svb_GOT_ClearKey	svb_GOT_ClearKey	Bit		R08CPU	PackML Templ...	GV:25001.1	Global
13		svb_GOT_ESTOPKey	svb_GOT_ESTOPKey	Bit		R08CPU	PackML Templ...	GV:25000.C	Global
14		svb_GOT_ESTOPKey1	svb_GOT_ESTOPKey1	Bit		R08CPU	PackML Templ...	GV:25000.5	Global
15		svb_GOT_EventAbortKey	svb_GOT_EventAbortKey	Bit		R08CPU	PackML Templ...	GV:25000.8	Global
16		svb_GOT_EventStopKey	svb_GOT_EventStopKey	Bit		R08CPU	PackML Templ...	GV:25000.A	Global
17		svb_GOT_GuardDoorOpenKey	svb_GOT_GuardDoorOpenKey	Bit		R08CPU	PackML Templ...	GV:25000.7	Global
18		svb_GOT_GuardDoorOpenKey1	svb_GOT_GuardDoorOpenKey1	Bit		R08CPU	PackML Templ...	GV:25000.0	Global
19		svb_GOT_HoldKey	svb_GOT_HoldKey	Bit		R08CPU	PackML Templ...	GV:25001.5	Global
20		svb_GOT_LowMaterialKey	svb_GOT_LowMaterialKey	Bit		R08CPU	PackML Templ...	GV:25000.9	Global
21		svb_GOT_LowMaterialKey1	svb_GOT_LowMaterialKey1	Bit		R08CPU	PackML Templ...	GV:25000.2	Global
22		svb_GOT_MGFaultKey	svb_GOT_MGFaultKey	Bit		R08CPU	PackML Templ...	GV:25000.D	Global
23		svb_GOT_MGFaultKey1	svb_GOT_MGFaultKey1	Bit		R08CPU	PackML Templ...	GV:25000.6	Global
24		svb_GOT_MaintMode	svb_GOT_MaintMode	Bit		R08CPU	PackML Templ...	GV:25001.B	Global
25		svb_GOT_ManualMode	svb_GOT_ManualMode	Bit		R08CPU	PackML Templ...	GV:25001.A	Global
26		svb_GOT_ProdMode	svb_GOT_ProdMode	Bit		R08CPU	PackML Templ...	GV:25001.C	Global
27		svb_GOT_RemoteStopKey	svb_GOT_RemoteStopKey	Bit		R08CPU	PackML Templ...	GV:25000.B	Global
28		svb_GOT_RemoteStopKey1	svb_GOT_RemoteStopKey1	Bit		R08CPU	PackML Templ...	GV:25000.4	Global
29		svb_GOT_ResetKey	svb_GOT_ResetKey	Bit		R08CPU	PackML Templ...	GV:25001.7	Global
30		svb_GOT_StartKey	svb_GOT_StartKey	Bit		R08CPU	PackML Templ...	GV:25001.6	Global
31		svb_GOT_StateCompleteKey	svb_GOT_StateCompleteKey	Bit		R08CPU	PackML Templ...	GV:25000.E	Global

Figure 5 – Example of System Labels

4 Using the System Labels in the GOT Program

To utilize the system labels in the GOT program, one needs to configure the objects in the GT Designer 3. In order for the labels to be recognizable in the GOT program, the Navigator system needs to establish routing information.

4.1 Establish Route Information

In Navigator, select Workspace -> System Label -> Route Information/Routing Parameters Generation

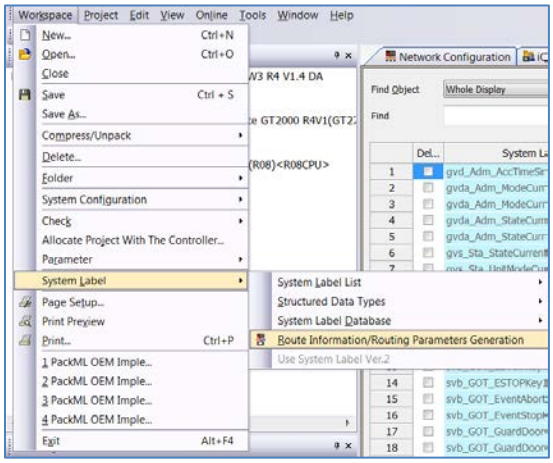


Figure 6 - Generating Routing Information

The routing information is generated based on the GOT and PLC that are in the system and how they are configured to connect to each other as shown in the figure below. This routing information will be used by the GT Designer 3 to connect the GOT objects to the corresponding PLC labels.

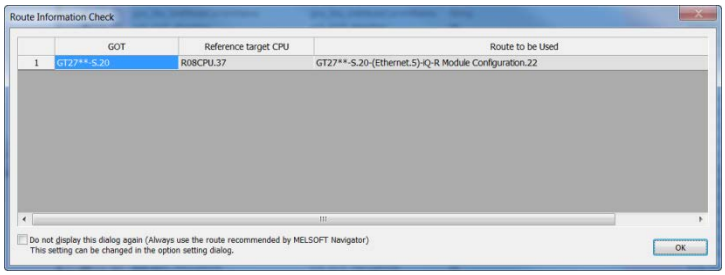


Figure 7 - Example of Generated Routing Information

4.2 Using the System Labels in GOT

Assigning an object to use the System Label is straightforward steps, and one can consult Chapter 6 of "GT Designer3 (GOT2000) Screen Design Manual" (Document No. SH(NA)-081220ENG-M).

As an example shown in Figure 8 below, the Bit Switch "Reset" was assigned to System Label svb_GOT_ResetKey. Once the System Label association is made, the pressing of Bit Switch Reset will cause the label svb_GOT_ResetKey to be "On" in the PLC.

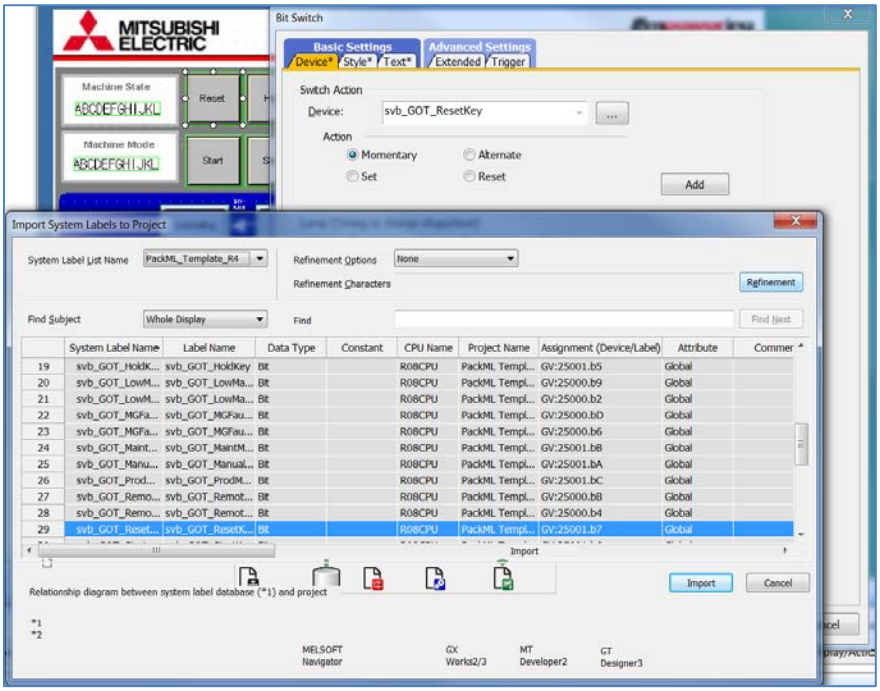


Figure 8 - Example of System Label Assignment in GOT

5 Summary

After completing the configuration steps described in this document, the system foundation is established to support the detailed application program development.

Users should refer to user and system manuals corresponding to the hardware components and software packages that are used in an application for further details.

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 3 – PackTags Design and Implementation

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Release of PackML OEM Implementation Templates Release 4

1 Introduction

The purpose of this document is to describe the design considerations and implementation approaches of implementing PackTags specification in an iQ PLC.

PackTags specification is a part of the overall OMAC PackML standard and defines a set of named data elements used for open architecture, interoperable data exchange in automated machinery. PackTags are useful for machine-to-machine (inter-machine) communications; for example between a Filler and a Capper. PackTags can also be used for data exchange between machines and higher-level information systems like Manufacturing Operations Management and Enterprise Information Systems.

The use of all PackTags is needed to be consistent with the principles for integrated connectivity with systems using this same implementation method. Required tags are those necessary **(1) for the function of the automated machine or (2) the connectivity to supervisory or remote systems.**

This document describes the implementation of the PackTags template files as a part of the Mitsubishi PackML Template system.

2 Key PackTags Design Considerations

The PackTags are implemented as a part of the Mitsubishi PackML Template system. The PackML Template system architecture is described in Part 1 of the Users Guide. Because of the large number of tags required to support the PackTags specification, an extended memory card maybe required to hold the symbolic information depending on the type of PLC that is used.

Generally, PackTags data is passed to higher-level information system using OPC protocol on a standard Ethernet-based communications network. Thus, in addition to the Template System hardware and iQ Works software, a Kepware OPC server is also integrated to work with the iQ-R PLC to form a total solution set. Kepware KEPServerEX V5.19 with enhanced Mitsubishi Ethernet Driver is used in the PackML Template system Release 4 implementation.

Following is a list of critical PackTags design considerations:

- The PackTags are implemented in an iQ-R PLC system as global labels and readily available for use by OEM machine control programs. In other words, the tag values should be accessible and be populated by OEM machine control programs.
- The PackTags should be accessible by external systems compliant to PackML and PackTags standards.
- The PackTags implementation on iQ-R should be directly usable by users of the iQ system. In other words, all PackTag labels should be configured and ready for use by users without additional configuration of the labels. All register assignments should not have to be altered by users of the system.
- Restrictions are placed on the dimensions of the variables to reduce the amount of memory locations that are consumed to support the tags.

3 PackTags Implementation Considerations

All PackTags are implemented as global labels, and the built-in Ethernet port on the iQ-R PLC CPU is used to connect the iQ system to Kepware OPC server.

The label names are shortened from the PackTags specification to be used with the iQ-R platform. PackTags are implemented in three Data Groups: Command, Status, and Admin, and the correlation of standard PackTags names to the shorten iQ-R labels and the Kepware tags is shown in the Appendix A for reference.

Following restrictions are placed on the dimensions of the labels:

- The remote interface number (i.e. the total number of upstream and downstream machines) is set to 10.

- The number of parameters that are given to the unit machine from remote machines is limited to 10. The parameters are typically needed for coordinating the unit machine or production with other machines.
- The number of parameters that are given to the unit machine locally is limited to 10.
- The number of product types that can be produced on a machine is limited to 5.
- The number of process variables needed by a unit machine for processing a specific product is limited to 10.
- The number of raw materials (ingredients) that are used by a unit machine in the processing of a particular product is limited to 10.
- The number of parameter tags associated to the local interface (e.g. parameters that are displayed or used on a unit locally such as an HMI) is limited to 10.
- The number of alarms of a machine is limited to 64.
- The number of alarm history is limited to 256.
- The number of Modes of a machine is limited to 32.
- The number of states in each mode of a machine is limited to 18.
- The number of material used or consumed in a production machine is limited to 10.
- The number of product types that can be processed by a production machine is limited to10.
- The number of product types that can be marked as defective by a production machine is limited to 10.

4 iQ System Configuration

This section documents the configuration of PLC parameters to support the PackTags implementation.

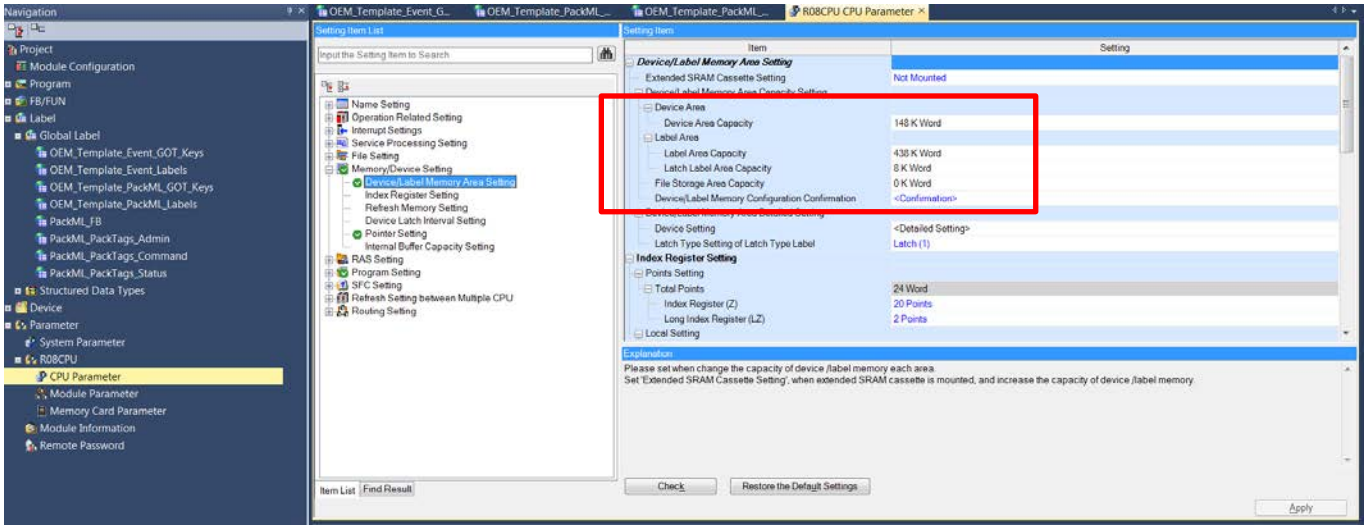


Figure 1 – PLC CPU Parameters for Memory Configuration

4.1 Device Area and Label Area

Because of the large number of tags required to support the PackTags specification, a large PLC Device area and Label Area needs to be assigned. In R4 Template Solution, 348K and 238K Words are allocated to Devices and Label Areas respectively in the R08CPU to accommodate the requirements.

Depending on the type of PLC used in the system, the capacity of memory may vary. However, configuration with 348K Word devices and 238K Word labels is sufficient to address the total number of PackTags and other system labels that are used in the PackML Template system.

4.2 Device

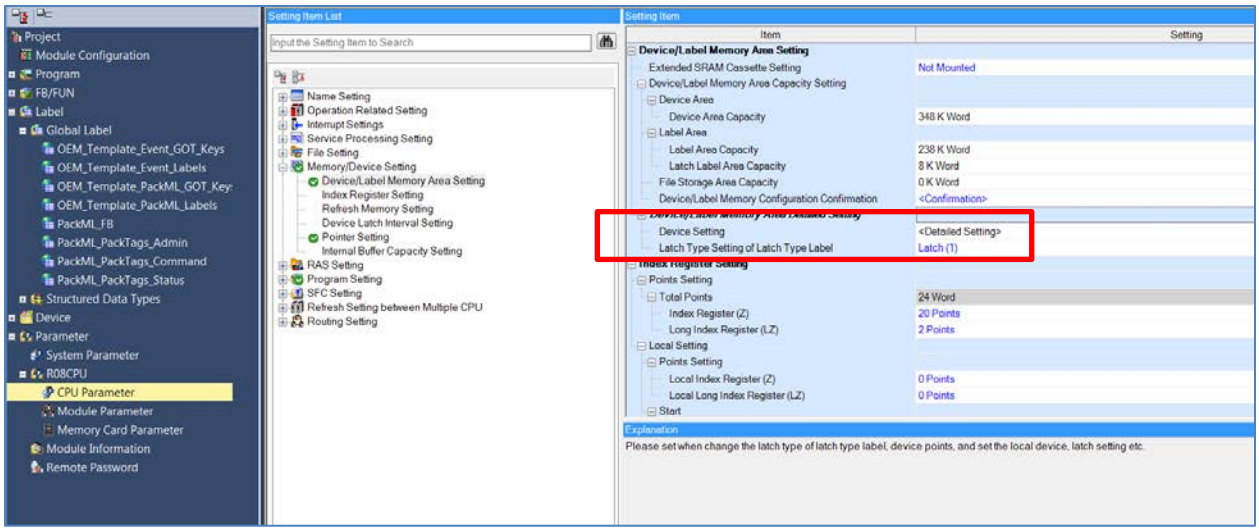


Figure 2 – PLC CPU Device Details

Selecting the Device Setting <Detailed Setting> will reveal the detailed allocation of PLC devices to the PackTags.

Item	Symbol	Device		Local Device			Latch (1)	Latch (2)
		Points	Range	Start	End	Points		
Input	X	12K	0 to 2FFF					
Output	Y	12K	0 to 2FFF					
Internal Relay	M	30K	0 to 30719				No Setting	No Setting
Link Relay	B	8K	0 to 1FFF				No Setting	No Setting
Special Link Relay	SB	2K	0 to 7FF					
Annunciator	F	2K	0 to 2047				No Setting	No Setting
Edge Relay	V	2K	0 to 2047				No Setting	No Setting
Step Relay	S	0						
Timer	T	1K	0 to 1023				No Setting	No Setting
Long Timer	LT	1K	0 to 1023				No Setting	No Setting
Retentive Timer	ST	0					No Setting	No Setting
Long Retentive Timer	LST	0					No Setting	No Setting
Counter	C	512	0 to 511				No Setting	No Setting
Long Counter	LC	512	0 to 511				No Setting	No Setting
Data Register	D	329K	0 to 336895				No Setting	No Setting
Link Register	W	5K	0 to 13FF				No Setting	No Setting
Link Special Register	SW	2K	0 to 7FF					
Latch Relay	L	8K	0 to 8191					No Setting
Total Device		347.5K Word				0.0K Word		
Total Word Device		342.5K Word				0.0K Word		
Total Bit Device		80.0K Bit				0.0K Bit		

In this PackTags implementation, most labels are assigned to D registers M bits. Sufficient D and M devices are allocated to accommodate the requirements. It is necessary to pre-assign the Device Addresses because the Kepware OPC server needs to refer to the memory location by addresses and not label names at this time.

4.3 Built-in Ethernet Port Setting

The built-in Ethernet port of the CPU module is used to communicate with the Kepware OPC server. The Ethernet port should be configured properly as described below and shown in Figure 3.

- Select Module Parameter -> Own Node Settings
- Set the proper IP address, Subnet Mask, and Default Gateway when appropriate.
- Set “Enable/Disable Onlne Change” to “Enable All (SLMP)

- Set “Communicate Data Code” to “Binary”

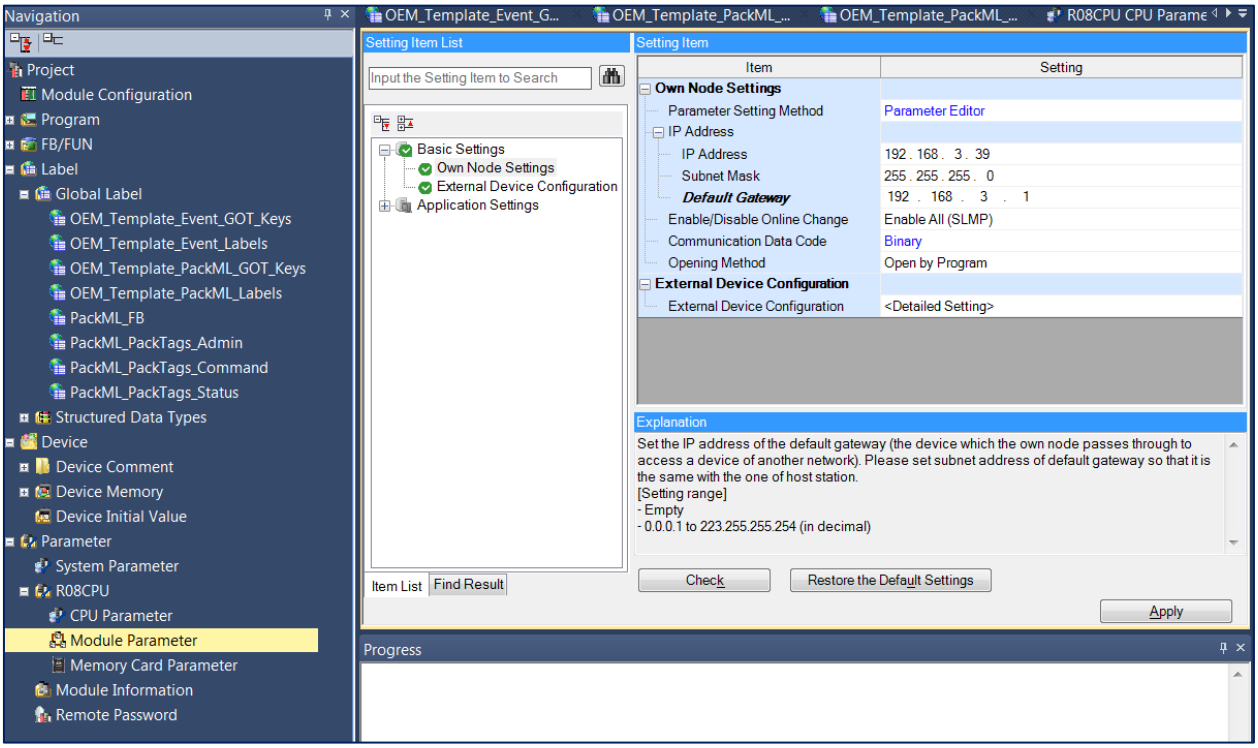


Figure 3 – Configuring the Built-in Ethernet Port

- Click the “External Device Configuration” <Detailed Setting> to configure the channels to communicate using TCP, UDP, SLMP, and port number to the desired value as shown in Figure 4.
  - In the PackML Template System architecture, the port number is set at hexadecimal number 5001(or decimal value 20481) for communication between the GOT and the PLC with UDP protocol.
  - The Port Number 5002 is configured to use TCP for communication with the Kepware OPC server.

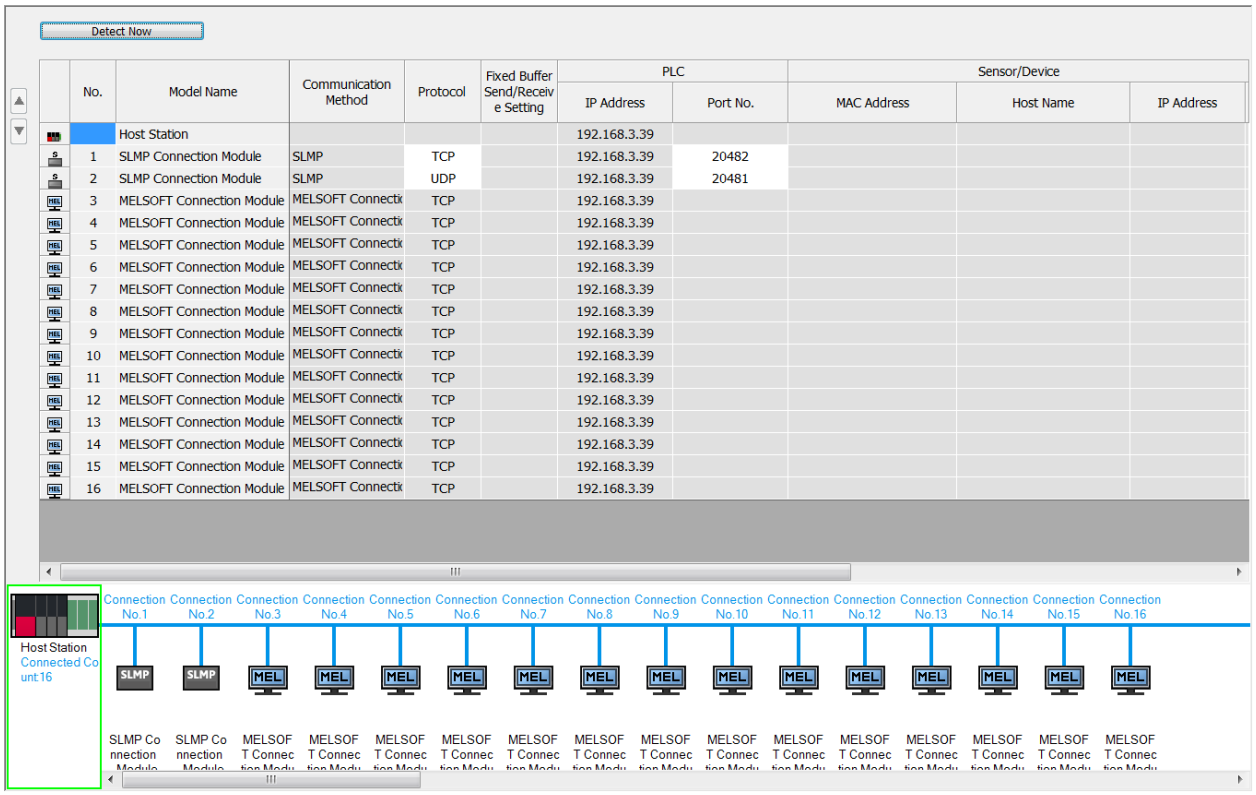


Figure 4 – PLC Communication Channel Configuration for GOT and OPC Communication

5 GX Works3 Label Implementation

As shown in the Appendix, all the labels that are required to support the PackTags standard are implemented in the iQ-R PLC using GX Works3 global labels. The PackTags labels are grouped into three categories: Command labels, Status labels, and Administrative labels. Ten different structured data types are also created to support the label definitions.

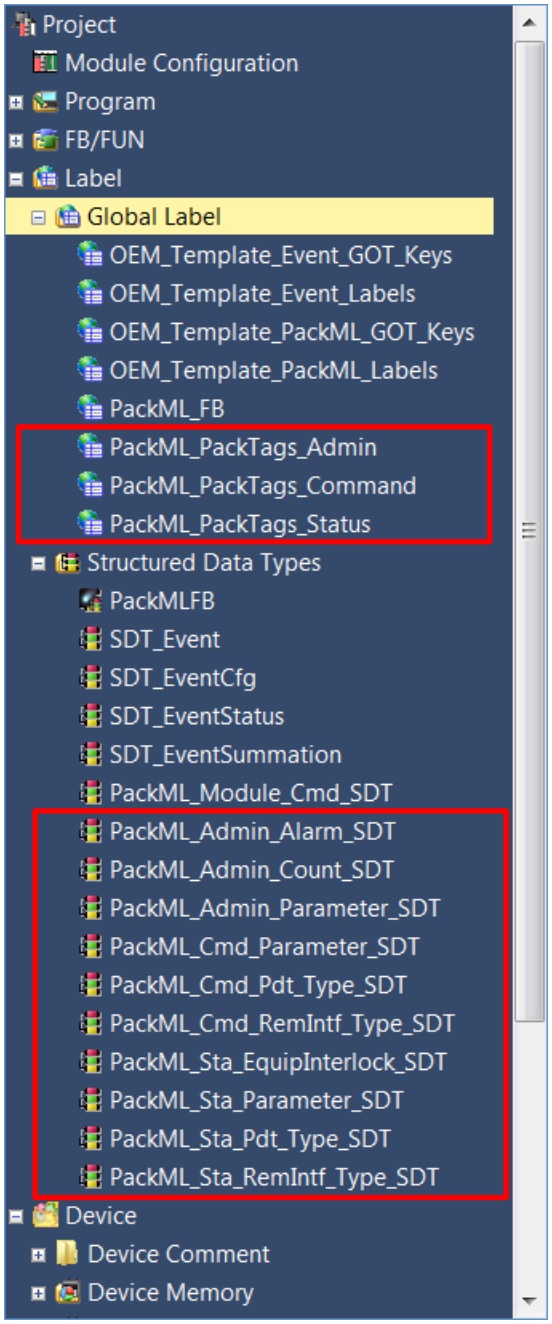


Figure 5 – Global Label Groups and Structure Data Types

5.1 Command Labels – PackTags_Command

The Command labels are created in the Global Label section with the proper data types. The feature of Structure of Structures Data Type is used to align the PackTags labels with the PackML Standard.

	Label Name	Data Type		Class	Assign (Device/Label)
1	gvd_Cmd_UnitMode	Double Word [Signed]	...	VAR_GLOBAL	D150000
2	gvb_Cmd_UnitModeChang	Bit	...	VAR_GLOBAL	M2400
3	gvl_Cmd_MachSpeed	FLOAT [Double Precision]	...	VAR_GLOBAL	D150002
4	gvdu_Cmd_MaterialInterlo	Double Word [Unsigned]/Bit String [32-b	...	VAR_GLOBAL	D150006
5	gvd_Cmd_CntrlCmd	Double Word [Signed]	...	VAR_GLOBAL	D150008
6	gvb_Cmd_CmdChangeRec	Bit	...	VAR_GLOBAL	M2401
7	gvsta_Cmd_RemIntf	PackML_Cmd_RemIntf_Type_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
8	gvsta_Cmd_Parameter	PackML_Cmd_Parameter_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
9	gvsta_Cmd_Pdt	PackML_Cmd_Pdt_Type_SDT(0..4)	...	VAR_GLOBAL	Detailed Setting
10			...		

Figure 6 – PackTags_Command Global Labels

The structured data types used in the Command Labels are:

PackML_Cmd_Remintf_Type_SDT

	Label Name	Data Type
1	d_Number	Double Word [Signed]
2	d_CmdCntlNo	Double Word [Signed]
3	d_Cmd_Value	Double Word [Signed]
4	sa_Parameter	PackML_Cmd_Parameter_SDT(0..9)

PackML_Cmd_Parameter_SDT

1	d_ID	Double Word [Signed]
2	s_Name	String(34)
3	s_Unit	String(34)
4	l_Value	FLOAT [Double Precision]

PackML_Cmd_Pdt_Type_SDT

1	d_Product_ID	Double Word [Signed]
2	sa_Proccess_Var	PackML_Cmd_Parameter_SDT(0..9)
3	sa_Ing	PackML_Cmd_Ing(0..9)

PackML_Cmd_Ing

1	d_Ing_ID	Double Word [Signed]
2	sa_Ing_Para	PackML_Cmd_Parameter_SDT(0..9)

5.2 Status Labels – PackTags_Status

The Status labels are created in the Global Label section with the proper data types.

	Label Name	Data Type		Class	Assign (Device/Label)
1	gvd_Sta_UnitModeCurrent	Double Word [Signed]	...	VAR_GLOBAL	D180000
2	gvb_Sta_UnitModeChangeRequested	Bit	...	VAR_GLOBAL	M2702
3	gvb_Sta_UnitModeChangeInProcess	Bit	...	VAR_GLOBAL	M2703
4	gvd_Sta_StateCurrent	Double Word [Signed]	...	VAR_GLOBAL	D180002
5	gvd_Sta_StateRequested	Double Word [Signed]	...	VAR_GLOBAL	D180004
6	gvb_Sta_StateChangeInProcess	Bit	...	VAR_GLOBAL	M2704
7	gvl_Sta_MachSpeed	FLOAT [Double Precision]	...	VAR_GLOBAL	D180006
8	gvl_Sta_CurMachSpeed	FLOAT [Double Precision]	...	VAR_GLOBAL	D180010
9	gvdu_Sta_MaterialInterlocks	Double Word [Unsigned]/Bit String [32-bit]	...	VAR_GLOBAL	D180014
10	gvst_Sta_EquipmentInterlock	PackML_Sta_EquipInterlock_SDT	...	VAR_GLOBAL	Detailed Setting
11	gvsta_Sta_RemIntf	PackML_Sta_RemIntf_Type_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
12	gvsta_Sta_Parameter	PackML_Sta_Parameter_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
13	gvsta_Sta_Pdt	PackML_Sta_Pdt_Type_SDT(0..4)	...	VAR_GLOBAL	Detailed Setting

Figure 7 – PackTags_Status Global Labels

The structured data types used in the Status Labels are:

PackML_Sta_Remintf_Type_SDT

1	d_Number	Double Word [Signed]
2	d_CmdCntlNo	Double Word [Signed]
3	d_Cmd_Value	Double Word [Signed]
4	sa_Parameter	PackML_Sta_Parameter_SDT(0..9)

PackML_Sta_EquipInterlock_SDT

1	b_Blocked	Bit
2	b_Starved	Bit

PackML_Sta_Parameter_SDT

1	d_ID	Double Word [Signed]
2	s_Name	String(34)
3	s_Unit	String(34)
4	l_Value	FLOAT [Double Precision]
5		

PackML_Sta_Pdt_Type_SDT

1	d_Product_ID	Double Word [Signed]
2	sa_Proccess_Var	PackML_Sta_Parameter_SDT(0..9)
3	sa_Ing	PackML_Sta_Ing(0..9)

PackML_Sta_Ing

1	d_Ing_ID	Double Word [Signed]
2	sa_Ing_Para	PackML_Sta_Parameter_SDT(0..9)

5.3 Administrative Labels

The Administrative labels are created in the Global Label section with the proper data types.

	Label Name	Data Type		Class	Assign (Device/Label)
1	gvst_Adm_Parameter	PackML_Admin_Parameter_SDT(0..19)	...	VAR_GLOBAL	Detailed Setting
2	gvsta_Adm_Alarm	PackML_Admin_Alarm_SDT(0..63)	...	VAR_GLOBAL	Detailed Setting
3	gvd_Adm_AlarmExtent	Double Word [Signed]	...	VAR_GLOBAL	D114168
4	gvda_Adm_ModeCurrentTime	Double Word [Signed](0..31)	...	VAR_GLOBAL	D114170
5	gvda_Adm_ModeCumulativeTime	Double Word [Signed](0..31)	...	VAR_GLOBAL	D114234
6	gvda_Adm_StateCurrentTime	Double Word [Signed](0..31,0..17)	...	VAR_GLOBAL	D114298
7	gvda_Adm_StateCumulativeTime	Double Word [Signed](0..31,0..17)	...	VAR_GLOBAL	D115450
8	gvsta_Adm_ProdConsumedCnt	PackML_Admin_Count_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
9	gvsta_Adm_ProdProcessedCnt	PackML_Admin_Count_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
10	gvsta_Adm_ProdDefectiveCnt	PackML_Admin_Count_SDT(0..9)	...	VAR_GLOBAL	Detailed Setting
11	gvd_Adm_AccTimeSinceReset	Double Word [Signed]	...	VAR_GLOBAL	D117862
12	gvl_Adm_MachDesignSpeed	FLOAT [Double Precision]	...	VAR_GLOBAL	D117864
13	gvd_Adm_StatesDisabled	Double Word [Signed]	...	VAR_GLOBAL	D117868
14	gvst_Adm_AlarmHistory	PackML_Admin_Alarm_SDT(0..255)	...	VAR_GLOBAL	Detailed Setting
15	gvd_Adm_AlarmHistoryExtent	Double Word [Signed]	...	VAR_GLOBAL	D131182
16	gvst_Adm_StopReason	PackML_Admin_Alarm_SDT	...	VAR_GLOBAL	Detailed Setting
17	gvst_Adm_AlarmWarning	PackML_Admin_Alarm_SDT(0..63)	...	VAR_GLOBAL	Detailed Setting
18	gvwa_Adm_PACDateTime_Date	Double Word [Unsigned]/Bit String [32-bit](0..6)	...	VAR_GLOBAL	D134564

Figure 8 – PackTags_Admin Global Labels

The structured data types used in the Admin Labels are:

PackML_Admin_Parameter_SDT

1	b_Trigger	Bit
2	d_ID	Double Word [Signed]
3	d_Value	Double Word [Signed]
4	s_Message	String(34)
5	d_Category	Double Word [Signed]
6	wua_DateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)
7	wua_AckDateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)

PackML_Adm_Alarm_SDT

1	d_ID	Double Word [Signed]
2	s_Name	String(34)
3	s_Unit	String(34)
4	l_Value	FLOAT [Double Precision]

PackML_Adm_Count_SDT

1	ID	Double Word [Signed]
2	Name	String(34)
3	Unit	String(34)
4	Count	Double Word [Signed]
5	AccCount	Double Word [Signed]

6 Kepware Server Configuration

An OPC server is required to connect the PackTags implemented in the iQ PLC to an external world such as a MES system or an HMI.

Kepware OPC server is used for the PackTags implementation because of its functionality and capable Mitsubishi driver to connect with Mitsubishi devices. It also supports long tag names and this capability allows the shortened label names to be mapped to the names as specified in the PackTags specification.

6.1 Adding a Channel of Communication

- Start the Kepware KEPServer Ex software and click to add a channel. In the example, the Channel Name is PackML.

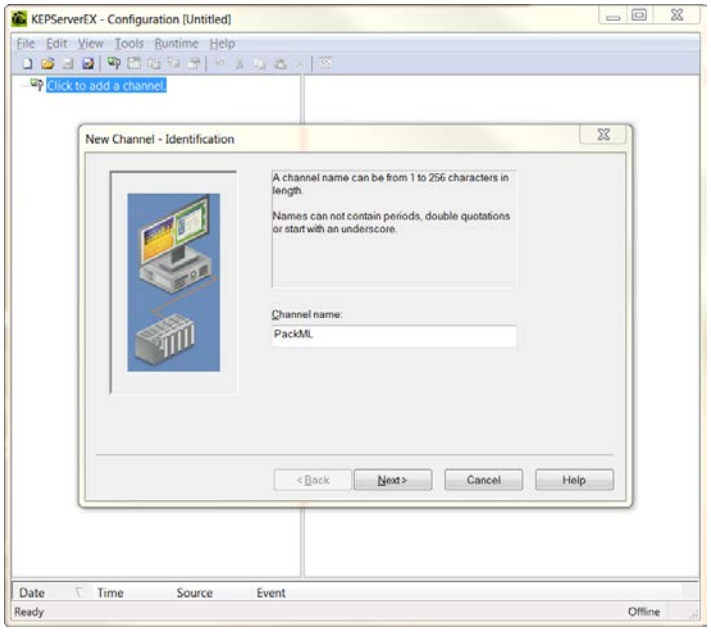


Figure 9 – Adding Channel in Kepware

- Select the Device Driver to be “Mitsubishi Ethernet” from the drop down list

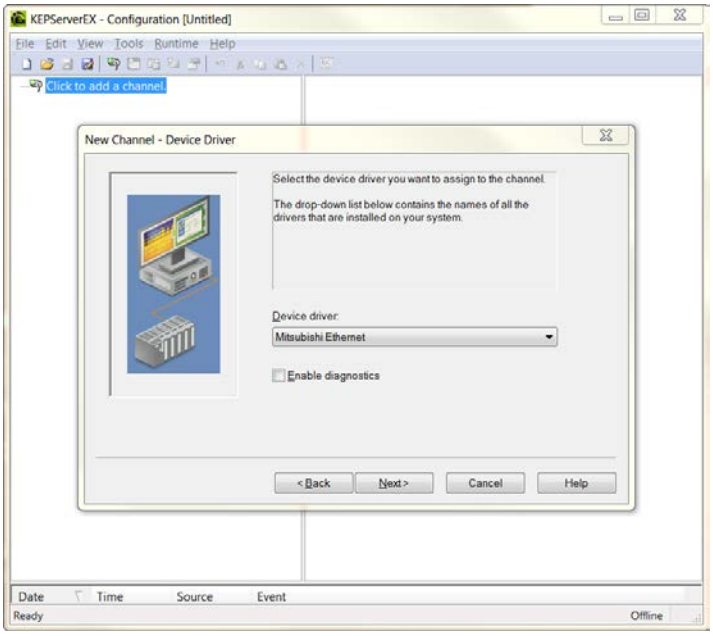


Figure 10 – Adding Device Driver to the Channel

- Define the Network Adapter of the system where the OPC Server is running on. Select the “Default” will use the Ethernet port of the computer the Kepware OPC server is running.

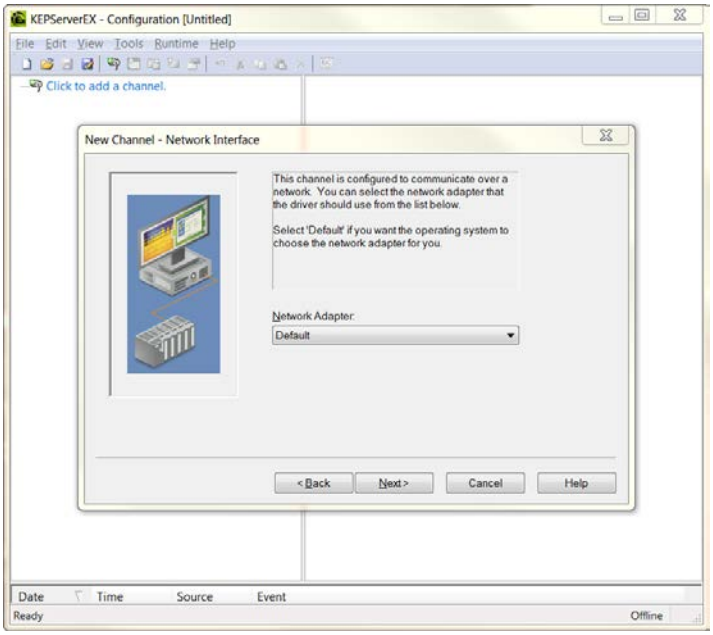


Figure 11 – Selecting the Network Adaptor where the OPC Server is Running

- At the "Write Optimization" screen, a user can determine which method should be used to give the optimized performance of the server for his system. In this example, default values are used.

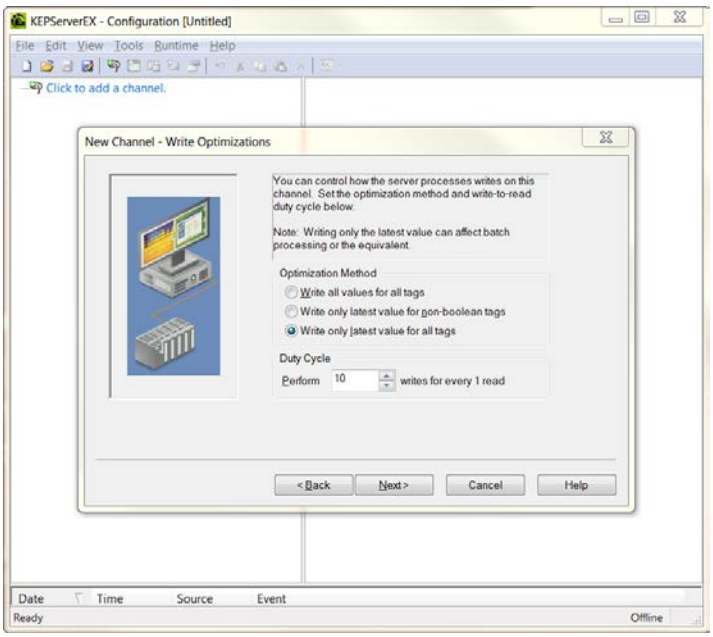
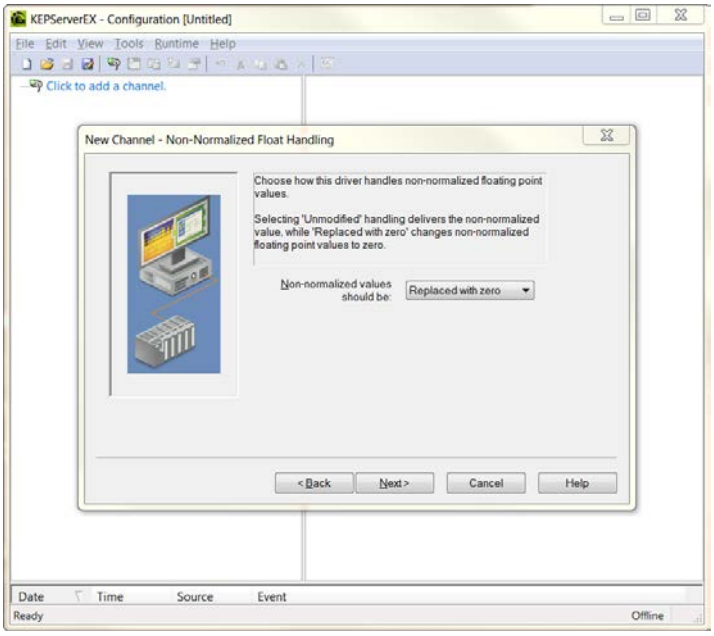


Figure 12 – Selecting Optimization Method

- Click “Next” to use the default “Replaced with Zero” to handle Non-Normalized Float.



- Click the “Finish” to complete adding the Channel.

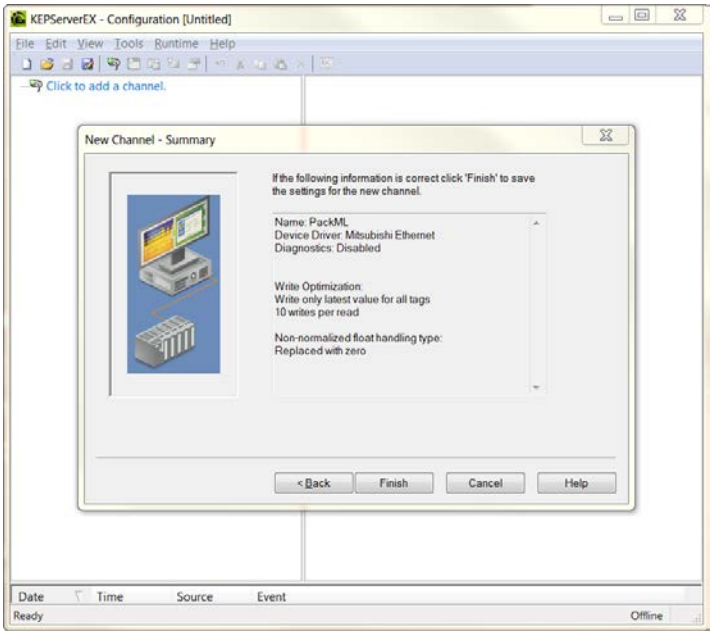


Figure 13 – Completing OPC Channel Configuration

6.2 Adding Devices

After the channel is defined, devices that need to be monitored can be added to the channel. Click to add a device:

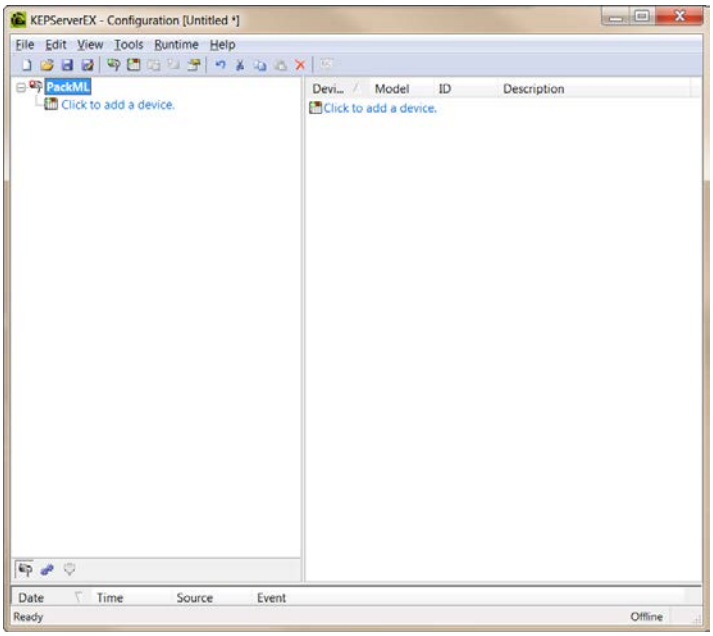


Figure 14 – Adding Device to Channel

- The configuration of the Built-in port is done first with the device name of "QPLC BuiltIn Port"

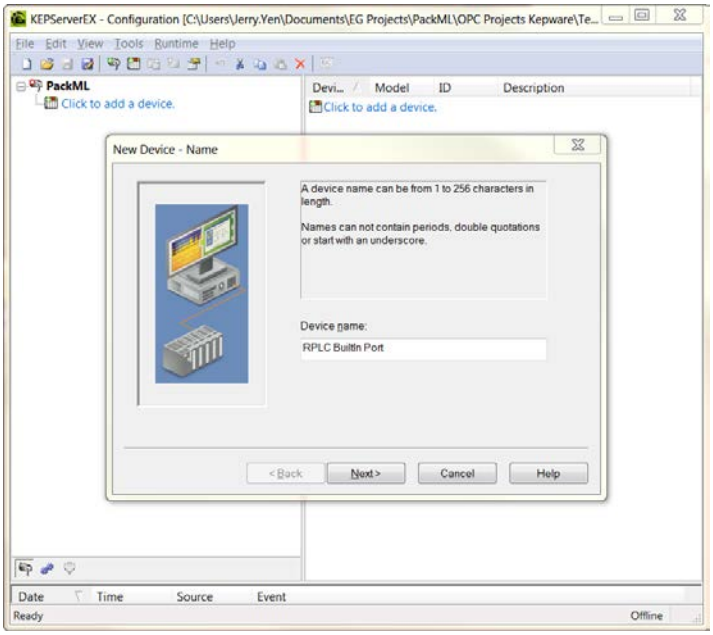


Figure 15 – Naming the Communication Device

- Select Device Model to be “Q Series” from the drop down list even though an iQ-R PLC is being connected. The Kepware OPC server will be updated to support iQ-R series natively.

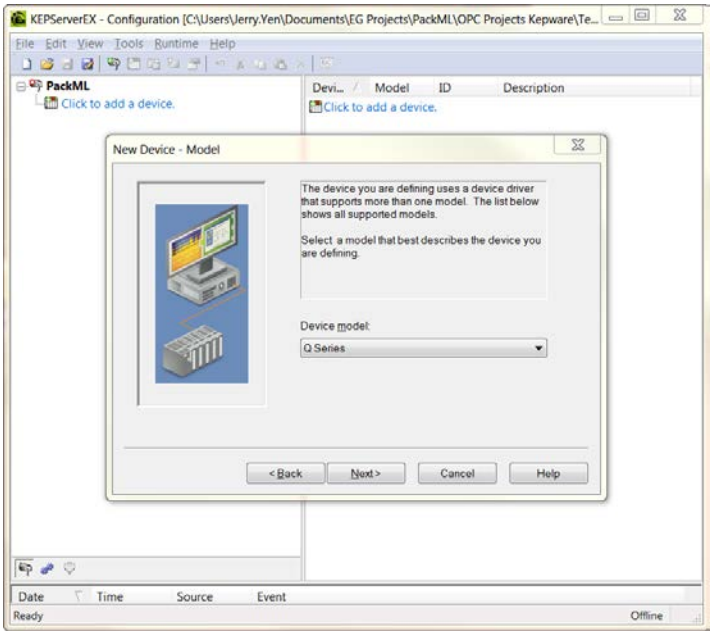


Figure 16 – Selecting the Mitsubishi Device Type

- Define the Device ID to be “192.168.1.40:255.”
    - The normal format of configuring the Device ID for a QPLC in the Kepware server is “IP Address : Network Number : Station Number” However, the Built-In port cannot be addressed using network number and station number. Thus, it is assuming the network number to be zero (thus omitted from the Device ID format) and a general station number of 255.

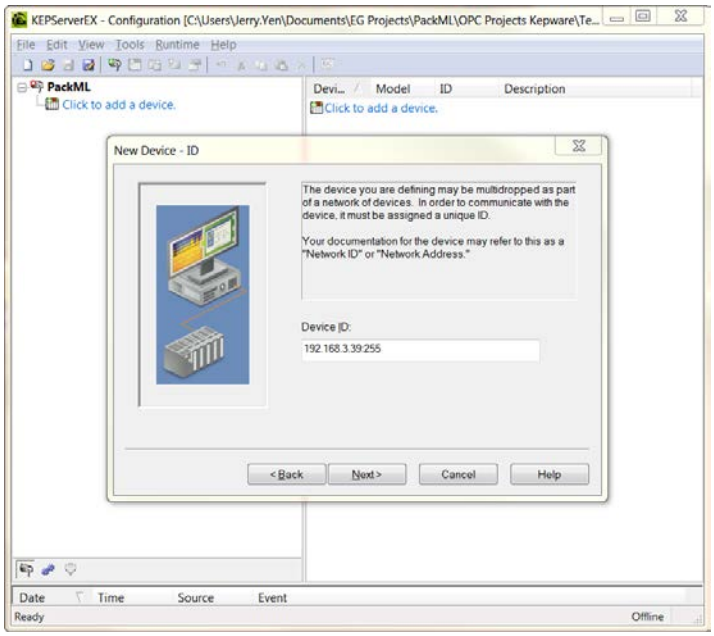
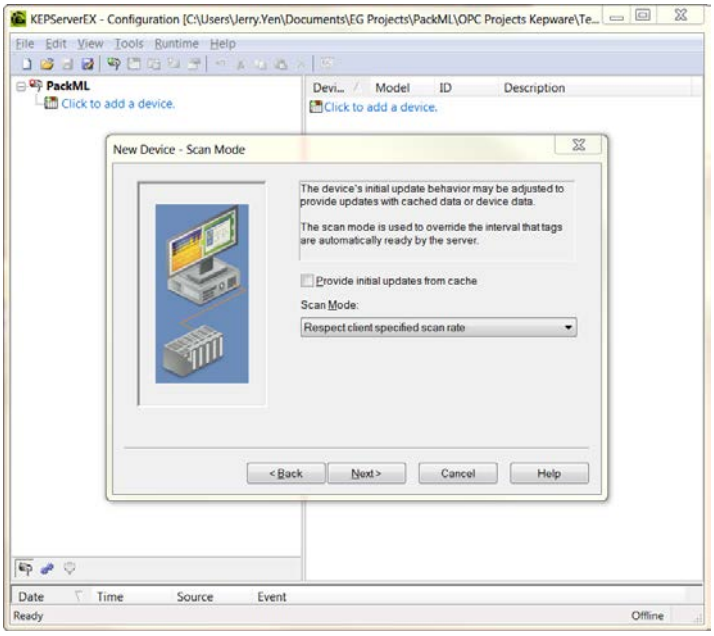


Figure 17 – Entering the Device ID

- Click “Next” to select the default Scan Mode



- The user can configure the timing parameters to optimize the communication performance. In this example, default values are used:

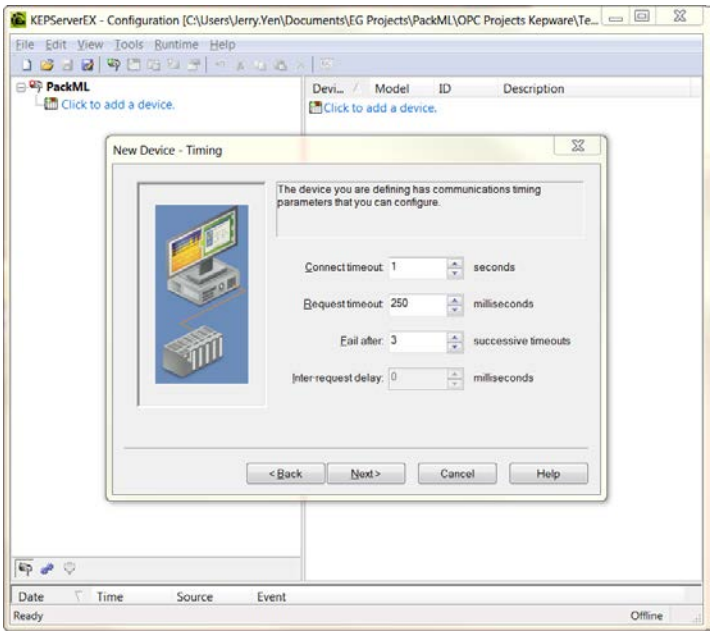


Figure 18 – Selecting the Timing Parameters

- A user can also enable the auto demotion of a device when communication is lost. One should configure this parameter according his application needs. In this example, auto-demotion is not configured.

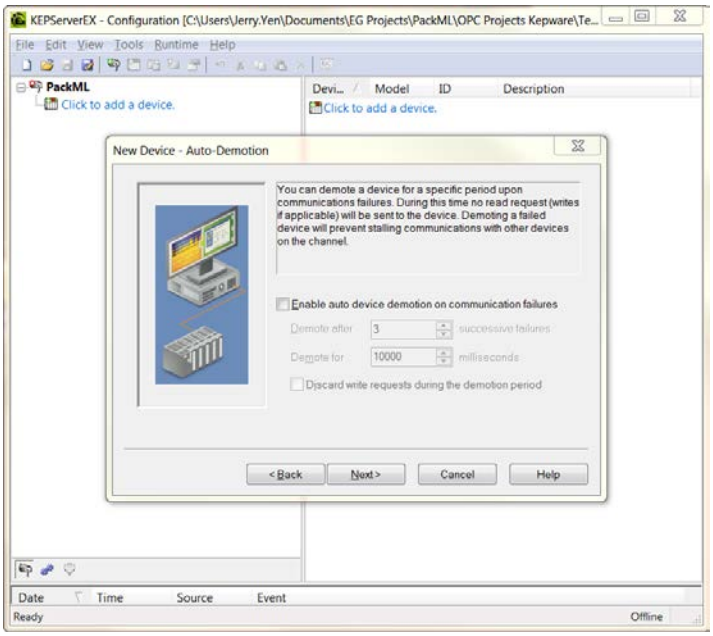


Figure 19 – Configuring Auto-Demotion Function

- The default of the device is set at First Word Low. This configuration is correct for Q PLC. Ensure the check box is selected.

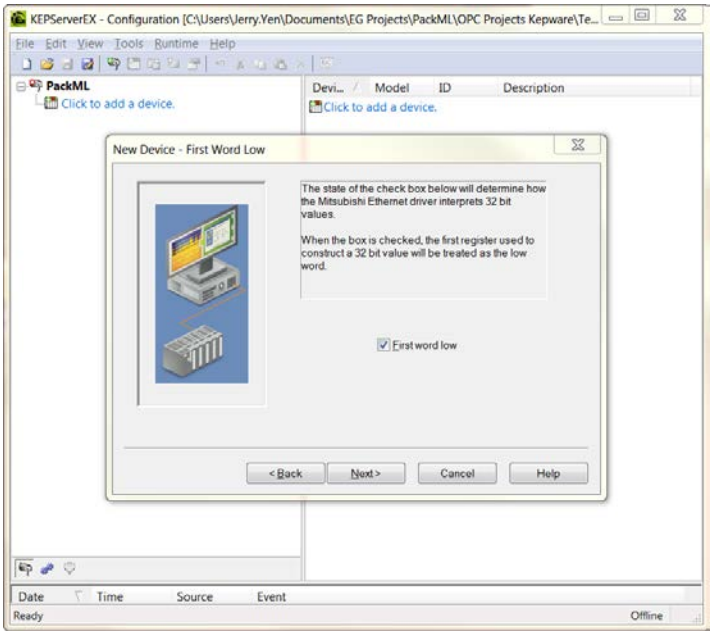


Figure 20 – Selecting Word Order

- Select the IP protocol to TCP/IP and the port number to be 20482 (i.e. 0x5002) as configured earlier for the Built-in Ethernet port in Section 4.3

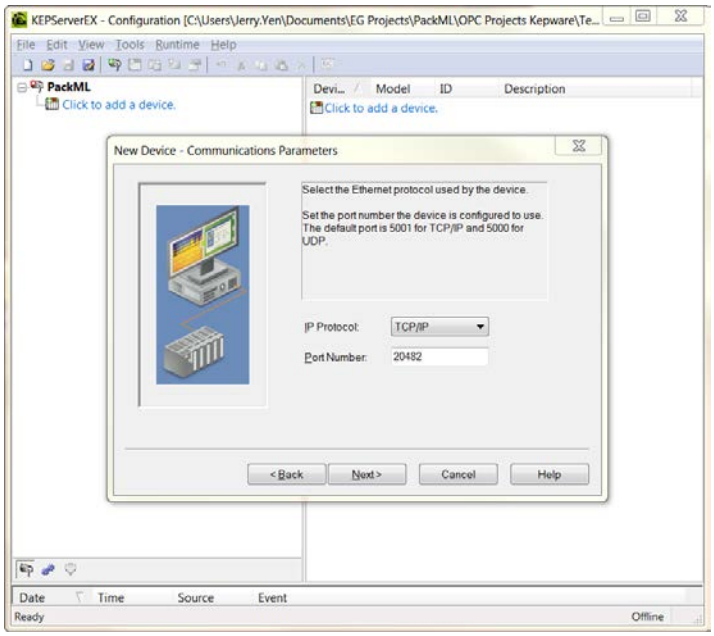


Figure 21 – Configuring IP Protocol and Port Number

- Select the proper time synchronization with the PLC per application requirements. In the example, no synchronization method was used.

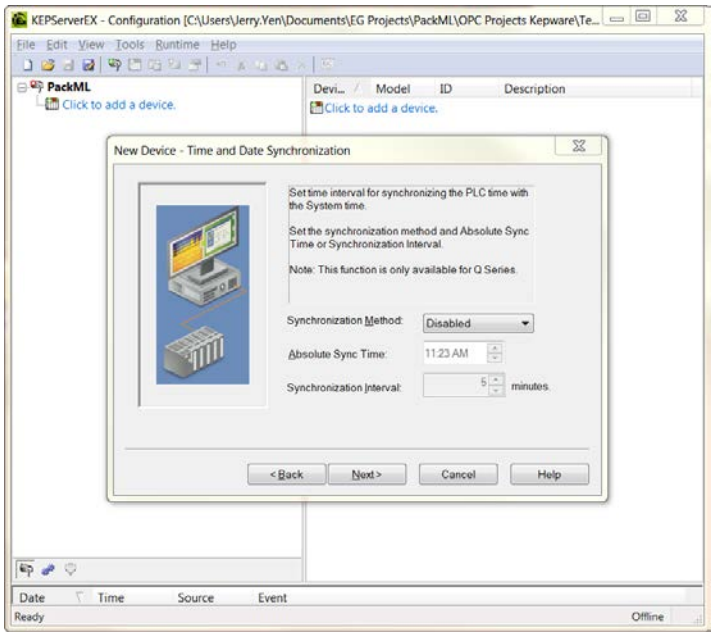


Figure 22 – Selecting Synchronization Method with PLC

- **Select "Finish" to complete adding the device.**

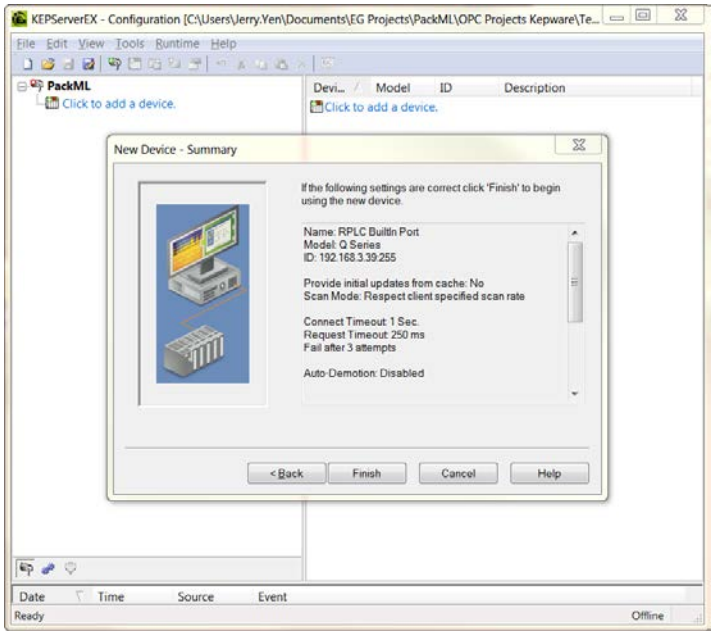


Figure 23 – Completing the Adding Device Process

7 Kepware Tags Implementation

OPC tags can be added manually one at a time. With the large number of tags for the PackTags implementation, it is easier to create the tags in Excel worksheets and import them to the OPC server. For the PackTags implementation, all OPC tags are created in Excel and be imported.

7.1 Creating the Tags

Three tag groups are created for each device for easy monitoring and sorting. The three tag groups are named "Command", "Status", and "Admin."

- Right click on a Device and select "New Tag Group…"

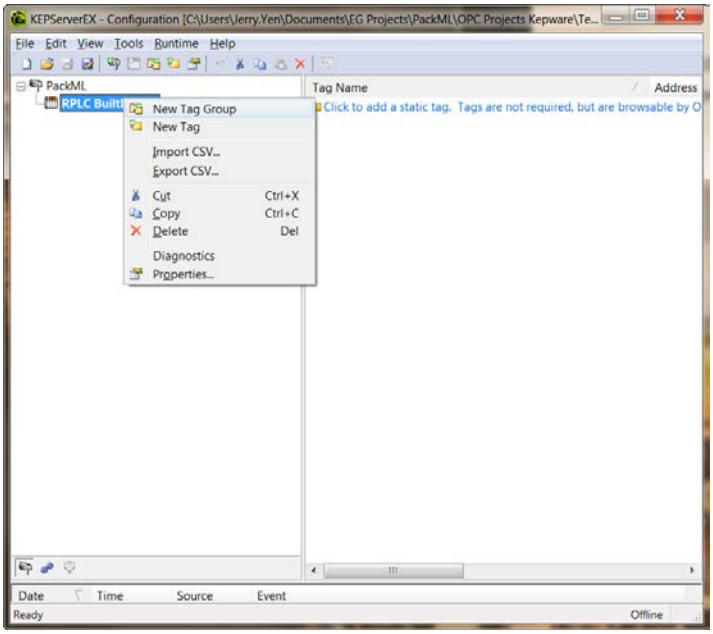


Figure 24 – Creating Tag Groups

- Define the Group Name to be “Command.”

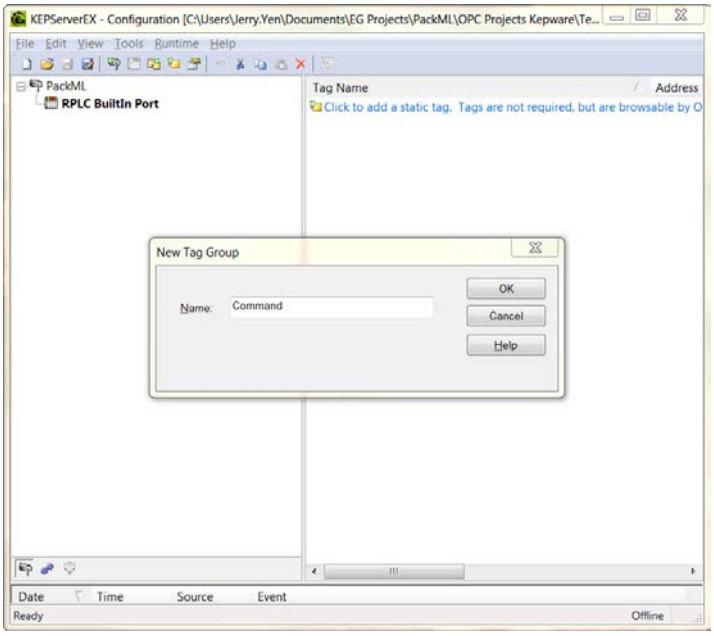


Figure 25 – Adding the Command Group

- Repeat the steps and define Tag Groups.

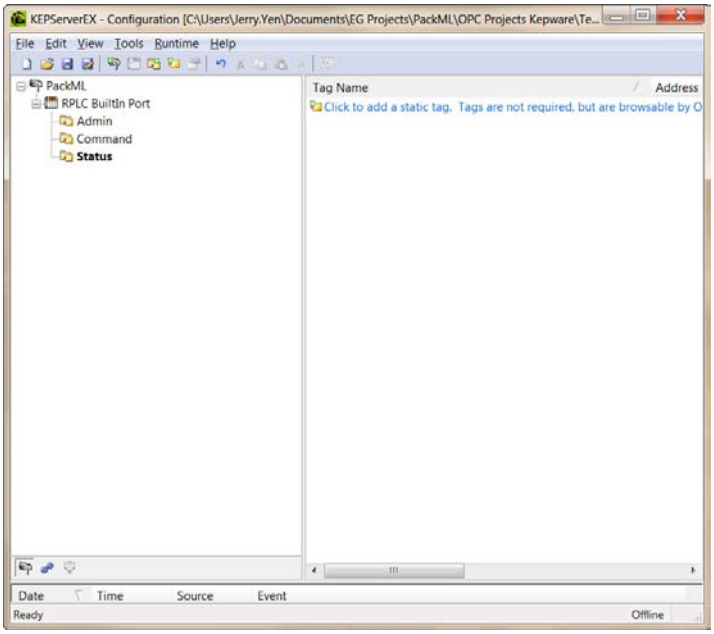


Figure 26 – Adding Other Tag Groups

- Right click on one of the groups and select “Import CSV…” to import the tags for the particular group.

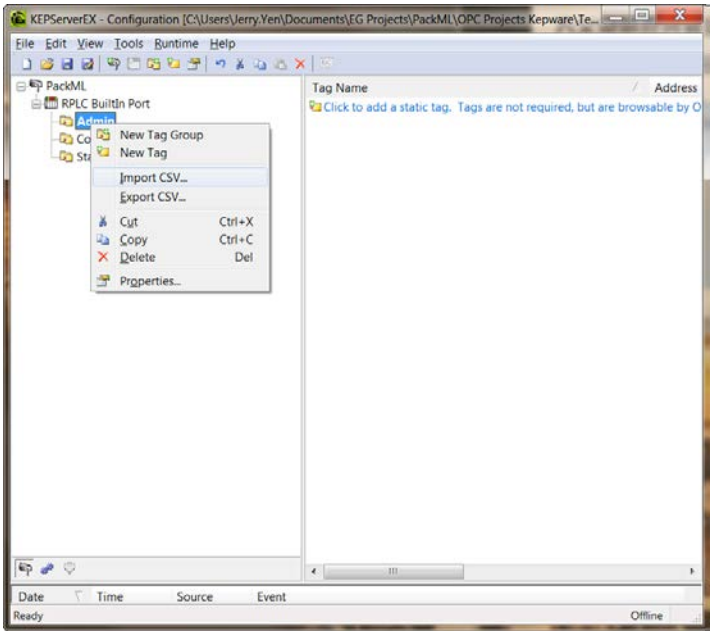


Figure 27 – Importing Tags from CSV Files

- Select the proper tag files and complete the import process for this group. Repeat the same steps to import the rest of the tags.

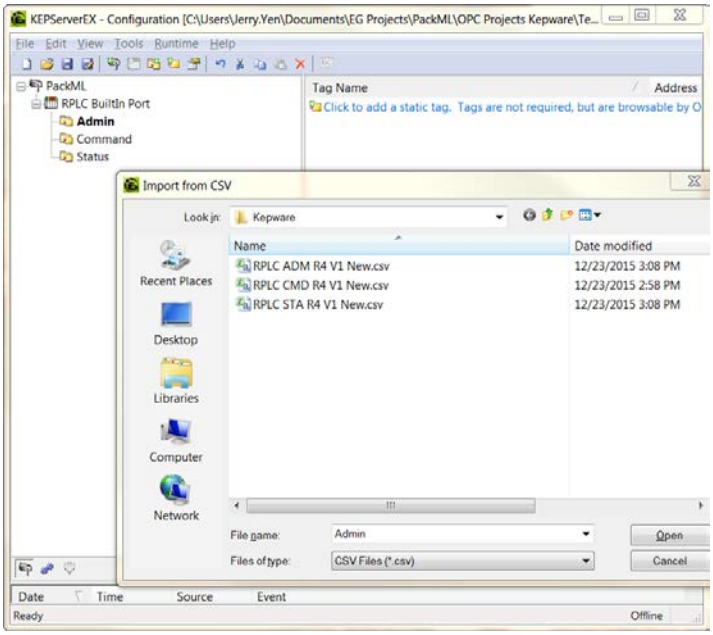


Figure 28 – Selecting the File to Import

8 PackTags Design Template Software Files

The PackTags design template files are included in the Mitsubishi PackML Template System package. Following are the template files:

- GX Works3 File – As a part of the overall iQ Workspace "PackML OEM Template R4"
- Kepware OPC Server File – PackML Template System R4 V1 GXW3 RPLC.opf
- OPC Tag files - RPLC ADM R4 V1 New.csv, RPLC CMD R4 V1 New.csv, RPLC STA R4 V1 New.csv

Appendix A

A.1 Command Tags – Spec to GX Works3 Labels

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvd_Cmd_UnitMode	Double Word [Signed]	UnitName_Command_UnitMode	Long
gvb_Cmd_UnitModeChangeRequest	Bit	UnitName_Command_UnitModeChangeRequest	Boolean
gvl_Cmd_MachSpeed	FLOAT [Double Precision]	UnitName_Command_MachSpeed	Double
gvdu_Cmd_MaterialInterlocks	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Command_MaterialInterlocks	Dword
gvd_Cmd_CntrlCmd	Double Word [Signed]	UnitName_Command_CntrlCmd	Long
gvb_Cmd_CmdChangeRequest	Bit	UnitName_Command_CmdChangeRequest	Boolean
gvsta_Cmd_RemIntf[#]	PackML_Cmd_RemIntf_Type_SDT(0..9)		
gvsta_Cmd_RemIntf[#].d_Number	Double Word[signed]	UnitName_Command_RemoteInterface[#]_Number	Long
gvsta_Cmd_RemIntf[#].d_CmdCrtlNo	Double Word[signed]	UnitName_Command_RemoteInterface[#]_ControlCmdNumber	Long
gvsta_Cmd_RemIntf[#].d_Cmd_Value	Double Word[signed]	UnitName_Command_RemoteInterface[#]_CmdValue	Long
gvsta_Cmd_RemIntf[#].sa_Parameter[#]	PackML_Cmd_Parameter_SDT(0..9)		
gvsta_Cmd_RemIntf[#].sa_Parameter[#].d_ID	Double Word [Signed]	UnitName_Command_RemoteInterface[#]_Parameter[#]_ID	Long
gvsta_Cmd_RemIntf[#].sa_Parameter[#].s_Name	String(34)	UnitName_Command_RemoteInterface[#]_Parameter[#]_Name	String
gvsta_Cmd_RemIntf[#].sa_Parameter[#].s_Unit	String(34)	UnitName_Command_RemoteInterface[#]_Parameter[#]_Unit	String
gvsta_Cmd_RemIntf[#].sa_Parameter[#].l_Value	FLOAT [Double Precision]	UnitName_Command_RemoteInterface[#]_Parameter[#]_Value	Double
gvsta_Cmd_Parameter[#]	PackML_Cmd_Parameter_SDT(0..9)		
gvsta_Cmd_Parameter[#].d_ID	Double Word [Signed]	UnitName_Command_Parameter[#]_ID	Long
gvsta_Cmd_Parameter[#].s_Name	String(34)	UnitName_Command_Parameter[#]_Name	String
gvsta_Cmd_Parameter[#].s_Unit	String(34)	UnitName_Command_Parameter[#]_Unit	String
gvsta_Cmd_Parameter[#].l_Value	FLOAT [Double Precision]	UnitName_Command_Parameter[#]_Value	Double
gvsta_Cmd_Pdt[#]	PackML_Cmd_Pdt_Type_SDT(0..4)		
gvsta_Cmd_Pdt[#].d_Product_ID	Double Word [Signed]	UnitName_Command_Product[#]_ProductID	Long
gvsta_Cmd_Pdt[#].sa_Proccess_Var[#]	PackML_Cmd_Parameter_SDT(0..9)		
gvsta_Cmd_Pdt[#].sa_Proccess_Var[#].d_ID	Double Word [Signed]	UnitName_Command_Product[#]_ProcessVariables[#]_ID	Long
gvsta_Cmd_Pdt[#].sa_Proccess_Var[#].s_Name	String(34)	UnitName_Command_Product[#]_ProcessVariables[#]_Name	String
gvsta_Cmd_Pdt[#].sa_Proccess_Var[#].s_Unit	String(34)	UnitName_Command_Product[#]_ProcessVariables[#]_Unit	String

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvsta_Cmd_Pdt[#].sa_Proccess_Var[#].l_Value	FLOAT [Double Precision]	UnitName_Command_Product[#]_ProcessVariables[#]_Value	Double
gvsta_Cmd_Pdt[#].sa_Ing	PackML_Cmd_Ing(0..9)		
gvsta_Cmd_Pdt[#].sa_Ing[#].d_Ing_ID	Double Word [Signed]	UnitName_Command_Product[#]_Ingredients[#]_IngredientID	Long
gvsta_Cmd_Pdt[#].sa_Ing[#].sa_Ing_Para[#]	PackML_Cmd_Parameter_SDT(0..9)		
gvsta_Cmd_Pdt[#].sa_Ing[#].sa_Ing_Para[#].d_ID	Double Word [Signed]	UnitName_Command_Product[#]_Ingredients[#]_Parameter[#]_ID	Long
gvsta_Cmd_Pdt[#].sa_Ing[#].sa_Ing_Para[#].s_Name	String(34)	UnitName_Command_Product[#]_Ingredients[#]_Parameter[#]_Name	String
gvsta_Cmd_Pdt[#].sa_Ing[#].sa_Ing_Para[#].s_Unit	String(34)	UnitName_Command_Product[#]_Ingredients[#]_Parameter[#]_Unit	String
gvsta_Cmd_Pdt[#].sa_Ing[#].sa_Ing_Para[#].l_Value	FLOAT [Double Precision]	UnitName_Command_Product[#]_Ingredients[#]_Parameter[#]_Value	Double

A.2 Status Tags – Spec to GX Works3 Labels

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvd_Sta_UnitModeCurrent	Double Word [Signed]	UnitName_Status_UnitModeCurrent	Long
gvb_Sta_UnitModeChangeRequested	Bit	UnitName_Status_UnitModeRequested	Boolean
gvb_Sta_UnitModeChangeInProcess	Bit	UnitName_Status_UnitModeChangeInProcess	Boolean
gvd_Sta_StateCurrent	Double Word [Signed]	UnitName_Status_StateCurrent	Long
gvd_Sta_StateRequested	Double Word [Signed]	UnitName_Status_StateRequested	Long
gvb_Sta_StateChangeInProcess	Bit	UnitName_Status_StateChangeInProcess	Boolean
gvl_Sta_MachSpeed	FLOAT [Double Precision]	UnitName_Status_MachSpeed	Double
gvl_Sta_CurMachSpeed	FLOAT [Double Precision]	UnitName_Status_CurMachSpeed	Double
gvdu_Sta_MaterialInterlocks	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Status_MaterialInterlock	Dword
gvst_Sta_EquipmentInterlock	PackML_Sta_EquipInterlock_SDT		Long
gvst_Sta_EquipmentInterlock.B_Blocked	Bit	UnitName_Status_EquipmentInterlocks_Blocked	Boolean
gvst_Sta_EquipmentInterlock.B_Starved	Bit	UnitName_Status_EquipmentInterlocks_Starved	Boolean
gvsta_Sta_RemIntf[#]	PackML_Sta_RemIntf_Type_SDT(0..9)		
gvsta_Sta_RemIntf[#].d_Number	Double Word [Signed]	UnitName_Status_RemoteInterface[#]_Number	Long
gvsta_Sta_RemIntf[#].d_CmdCtrlNo	Double Word [Signed]	UnitName_Status_RemoteInterface[#]_ControlCmdNumber	Long
gvsta_Sta_RemIntf[#].d_Cmd_Value	Double Word [Signed]	UnitName_Status_RemoteInterface[#]_CmdValue	Long
gvsta_Sta_RemIntf[#].sa_Parameter[#]	PackML_Sta_Parameter_SDT(0..9)		
gvsta_Sta_RemIntf[#].sa_Parameter[#].d_ID	Double Word [Signed]	UnitName_Status_RemoteInterface[#]_Parameter[#]_ID	Long
gvsta_Sta_RemIntf[#].sa_Parameter[#].s_Name	String(34)	UnitName_Status_RemoteInterface[#]_Parameter[#]_Name	String
gvsta_Sta_RemIntf[#].sa_Parameter[#].s_Unit	String(34)	UnitName_Status_RemoteInterface[#]_Parameter[#]_Unit	String
gvsta_Sta_RemIntf[#].sa_Parameter[#].l_Value	FLOAT [Double Precision]	UnitName_Status_RemoteInterface[#]_Parameter[#]_Value	Double
gvsta_Sta_Parameter[#]	PackML_Sta_Parameter_SDT(0..9)		
gvsta_Sta_Parameter[#].d_ID	Double Word [Signed]	UnitName_Status_Parameter[#]_ID	Long
gvsta_Sta_Parameter[#].s_Name	String(34)	UnitName_Status_Parameter[#]_Name	String
gvsta_Sta_Parameter[#].s_Unit	String(34)	UnitName_Status_Parameter[#]_Unit	String
gvsta_Sta_Parameter[#].l_Value	FLOAT [Double Precision]	UnitName_Status_Parameter[#]_Value	Double
gvsta_Sta_Pdt[#]	PackML_Sta_Pdt_Type_SDT(0..4)		
gvsta_Sta_Pdt[#].d_Product_ID	Double Word [Signed]	UnitName_Status_Product[#]_ProductID	Long

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvsta_Sta_Pdt[#].sa_Proccess_Var[#]	PackML_Sta_Parameter_SDT(0..9)		
gvsta_Sta_Pdt[#].sa_Proccess_Var[#].d_ID	Double Word [Signed]	UnitName_Status_Product[#]_ProcessVariables[#]_ID	Long
gvsta_Sta_Pdt[#].sa_Proccess_Var[#].s_Name	String(34)	UnitName_Status_Product[#]_ProcessVariables[#]_Name	String
gvsta_Sta_Pdt[#].sa_Proccess_Var[#].s_Unit	String(34)	UnitName_Status_Product[#]_ProcessVariables[#]_Unit	String
gvsta_Sta_Pdt[#].sa_Proccess_Var[#].l_Value	FLOAT [Double Precision]	UnitName_Status_Product[#]_ProcessVariables[#]_Value	Double
gvsta_Sta_Pdt[#].sa_Ing	PackML_Sta_Ing(0..9)		
gvsta_Sta_Pdt[#].sa_Ing[#].d_Ing_ID	Double Word [Signed]	UnitName_Status_Product[#]_Ingredients[#]_IngredientID	Long
gvsta_Sta_Pdt[#].sa_Ing[#].sa_Ing_Para[#]	PackML_Sta_Parameter_SDT(0..9)		
gvsta_Sta_Pdt[#].sa_Ing[#].sa_Ing_Para[#].d_ID	Double Word [Signed]	UnitName_Status_Product[#]_Ingredients[#]_Parameter[#]_ID	Long
gvsta_Sta_Pdt[#].sa_Ing[#].sa_Ing_Para[#].s_Name	String(34)	UnitName_Status_Product[#]_Ingredients[#]_Parameter[#]_Name	String
gvsta_Sta_Pdt[#].sa_Ing[#].sa_Ing_Para[#].s_Unit	String(34)	UnitName_Status_Product[#]_Ingredients[#]_Parameter[#]_Unit	String
gvsta_Sta_Pdt[#].sa_Ing[#].sa_Ing_Para[#].l_Value	FLOAT [Double Precision]	UnitName_Status_Product[#]_Ingredients[#]_Parameter[#]_Value	Double

A.3 Admin Tags – Spec to GX Works3 Labels

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvst_Adm_Parameter[#]	PackML_Admin_Parameter_SDT(0..19)		
gvst_Adm_Parameter[#].d_ID	Double Word [Signed]	UnitName_Admin_Parameter[#]_ID	Long
gvst_Adm_Parameter[#].s_Name	String(34)	UnitName_Admin_Parameter[#]_Name	String
gvst_Adm_Parameter[#].s_Unit	String(34)	UnitName_Admin_Parameter[#]_Unit	String
gvst_Adm_Parameter[#].l_Value	FLOAT [Double Precision]	UnitName_Admin_Parameter[#]_Value	Double
gvsta_Adm_Alarm[#].	PackML_Admin_Alarm_SDT(0..63)		
gvsta_Adm_Alarm[#].b_Trigger	Bit	UnitName_Admin_Alarm[#]_Trigger	Boolean
gvsta_Adm_Alarm[#].d_ID	Double Word [Signed]	UnitName_Admin_Alarm[#]_ID	Long
gvsta_Adm_Alarm[#].d_Value	Double Word [Signed]	UnitName_Admin_Alarm[#]_Value	Long
gvsta_Adm_Alarm[#].s_Message	String(34)	UnitName_Admin_Alarm[#]_Message	String
gvsta_Adm_Alarm[#].d_Category	Double Word [Signed]	UnitName_Admin_Alarm[#]_Category	
gvsta_Adm_Alarm[#].wua_DateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_Alarm[#].wua_DateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[0]	Long
gvsta_Adm_Alarm[#].wua_DateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[1]	Long
gvsta_Adm_Alarm[#].wua_DateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[2]	Long
gvsta_Adm_Alarm[#].wua_DateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[3]	Long
gvsta_Adm_Alarm[#].wua_DateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[4]	Long
gvsta_Adm_Alarm[#].wua_DateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[5]	Long
gvsta_Adm_Alarm[#].wua_DateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_DateTime[6]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_Alarm[#].wua_AckDateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[0]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[1]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[2]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[3]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[4]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[5]	Long
gvsta_Adm_Alarm[#].wua_AckDateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_Alarm[#]_AckDateTime[6]	Long
gvd_Adm_AlarmExtent	Double Word [Signed]	UnitName_Admin_AlarmExtent	Long

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvst_Adm_AlarmHistory[#]	PackML_Admin_Alarm_SDT(0..255)		
gvsta_Adm_AlarmHistory[#].b_Trigger	Bit	UnitName_Admin_AlarmHistory[#]_Trigger	Boolean
gvsta_Adm_AlarmHistory[#].d_ID	Double Word [Signed]	UnitName_Admin_AlarmHistory[#]_ID	Long
gvsta_Adm_AlarmHistory[#].d_Value	Double Word [Signed]	UnitName_Admin_AlarmHistory[#]_Value	Long
gvsta_Adm_AlarmHistory[#].s_Message	String(34)	UnitName_Admin_AlarmHistory[#]_Message	String
gvsta_Adm_AlarmHistory[#].d_Category	Double Word [Signed]	UnitName_Admin_AlarmHistory[#]_Category	
gvsta_Adm_AlarmHistory[#].wua_DateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_AlarmHistory[#].wua_DateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[0]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[1]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[2]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[3]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[4]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[5]	Long
gvsta_Adm_AlarmHistory[#].wua_DateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_DateTime[6]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[0]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[1]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[2]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[3]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[4]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[5]	Long
gvsta_Adm_AlarmHistory[#].wua_AckDateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmHistory[#]_AckDateTime[6]	Long
gvd_Adm_AlarmHistoryExtent	Double Word [Signed]	UnitName_Admin_AlarmHistoryExtent	Long
gvst_Adm_StopReason	PackML_Admin_Alarm_SDT		
gvsta_Adm_StopReason.b_Trigger	Bit	UnitName_Admin_StopReason_Trigger	Boolean
gvsta_Adm_StopReason.d_ID	Double Word [Signed]	UnitName_Admin_StopReason_ID	Long
gvsta_Adm_StopReason.d_Value	Double Word [Signed]	UnitName_Admin_StopReason_Value	Long
gvsta_Adm_StopReason.s_Message	String(34)	UnitName_Admin_StopReason_Message	String
gvsta_Adm_StopReason.d_Category	Double Word [Signed]	UnitName_Admin_StopReason_Category	

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvsta_Adm_StopReason.wua_DateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_StopReason.wua_DateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[0]	Long
gvsta_Adm_StopReason.wua_DateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[1]	Long
gvsta_Adm_StopReason.wua_DateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[2]	Long
gvsta_Adm_StopReason.wua_DateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[3]	Long
gvsta_Adm_StopReason.wua_DateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[4]	Long
gvsta_Adm_StopReason.wua_DateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[5]	Long
gvsta_Adm_StopReason.wua_DateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_DateTime[6]	Long
gvsta_Adm_StopReason.wua_AckDateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_StopReason.wua_AckDateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[0]	Long
gvsta_Adm_StopReason.wua_AckDateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[1]	Long
gvsta_Adm_StopReason.wua_AckDateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[2]	Long
gvsta_Adm_StopReason.wua_AckDateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[3]	Long
gvsta_Adm_StopReason.wua_AckDateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[4]	Long
gvsta_Adm_StopReason.wua_AckDateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[5]	Long
gvsta_Adm_StopReason.wua_AckDateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_StopReason_AckDateTime[6]	Long
gvd_Adm_StopReasonExtent	Double Word [Signed]	UnitName_Admin_StopReasonExtent	Long
gvst_Adm_AlarmWarning[#]	PackML_Admin_Alarm_SDT(0..63)		
gvsta_Adm_AlarmWarning[#].b_Trigger	Bit	UnitName_Admin_AlarmWarning[#]_Trigger	Boolean
gvsta_Adm_AlarmWarning[#].d_ID	Double Word [Signed]	UnitName_Admin_AlarmWarning[#]_ID	Long
gvsta_Adm_AlarmWarning[#].d_Value	Double Word [Signed]	UnitName_Admin_AlarmWarning[#]_Value	Long
gvsta_Adm_AlarmWarning[#].s_Message	String(34)	UnitName_Admin_AlarmWarning[#]_Message	String
gvsta_Adm_AlarmWarning[#].d_Category	Double Word [Signed]	UnitName_Admin_AlarmWarning[#]_Category	
gvsta_Adm_AlarmWarning[#].wua_DateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_AlarmWarning[#].wua_DateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[0]	Long
gvsta_Adm_AlarmWarning[#].wua_DateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[1]	Long
gvsta_Adm_AlarmWarning[#].wua_DateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[2]	Long
gvsta_Adm_AlarmWarning[#].wua_DateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[3]	Long
gvsta_Adm_AlarmWarning[#].wua_DateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[4]	Long

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvsta_Adm_AlarmWarning[#].wua_DateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[5]	Long
gvsta_Adm_AlarmWarning[#].wua_DateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_DateTime[6]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[0]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[0]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[1]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[1]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[2]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[2]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[3]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[3]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[4]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[4]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[5]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[5]	Long
gvsta_Adm_AlarmWarning[#].wua_AckDateTime[6]	Double Word [Unsigned]/Bit String [32-bit]	UnitName_Admin_AlarmWarning[#]_AckDateTime[6]	Long
gvd_Adm_AlarmWarningExtent	Double Word [Signed]	UnitName_Admin_AlarmWarningExtent	Long
gvda_Adm_ModeCurrentTime	Double Word [Signed](0..31)	UnitName_Admin_ModeCurrentTime[#]	Long
gvda_Adm_ModeCumulativeTime	Double Word [Signed](0..31)	UnitName_Admin_ModeCumulativeTime[#]	Long
gvda_Adm_StateCurrentTime	Double Word [Signed](0..31,0..17)	UnitName_Admin_StateCurrentTime[#]	Long
gvda_Adm_StateCumulativeTime	Double Word [Signed](0..31,0..17)	UnitName_Admin_StateCumulativeTime[#]	Long
gvsta_Adm_ProdConsumedCnt[#]	PackML_Admin_Count_SDT(0..9)		
gvsta_Adm_ProdConsumedCnt[#].d_ID	Double Word [Signed]	UnitName_Admin_ProdConsumedCount[#]_ID	Long
gvsta_Adm_ProdConsumedCnt[#].s_Name	String(34)	UnitName_Admin_ProdConsumedCount[#]_Name	String
gvsta_Adm_ProdConsumedCnt[#].s_Unit	String(34)	UnitName_Admin_ProdConsumedCount[#]_Unit	String
gvsta_Adm_ProdConsumedCnt[#].d_Count	Double Word [Signed]	UnitName_Admin_ProdConsumedCount[#]_Count	Long
gvsta_Adm_ProdConsumedCnt[#].d_AccCount	Double Word [Signed]	UnitName_Admin_ProdConsumedCount[#]_AccCount	Long
gvsta_Adm_ProdProcessedCnt[#]	PackML_Admin_Count_SDT(0..9)		
gvsta_Adm_ProdProcessedCnt[#].d_ID	Double Word [Signed]	UnitName_Admin_ProdProcessedCount[#]_ID	Long
gvsta_Adm_ProdProcessedCnt[#].s_Name	String(34)	UnitName_Admin_ProdProcessedCount[#]_Name	String
gvsta_Adm_ProdProcessedCnt[#].s_Unit	String(34)	UnitName_Admin_ProdProcessedCount[#]_Unit	String
gvsta_Adm_ProdProcessedCnt[#].d_Count	Double Word [Signed]	UnitName_Admin_ProdProcessedCount[#]_Count	Long
gvsta_Adm_ProdProcessedCnt[#].d_AccCount	Double Word [Signed]	UnitName_Admin_ProdProcessedCount[#]_AccCount	Long
gvsta_Adm_ProdDefectiveCnt[#]	PackML_Admin_Count_SDT(0..9)		
gvsta_Adm_ProdDefectiveCnt[#].d_ID	Double Word [Signed]	UnitName_Admin_ProdDefectiveCount[#]_ID	Long

GX Works3 Labels		Kepware OPC Server Tags	
Label	Data Type	Label	Data Type
gvsta_Adm_ProdDefectiveCnt[#].s_Name	String(34)	UnitName_Admin_ProdDefectiveCount[#]_Name	String
gvsta_Adm_ProdDefectiveCnt[#].s_Unit	String(34)	UnitName_Admin_ProdDefectiveCount[#]_Unit	String
gvsta_Adm_ProdDefectiveCnt[#].d_Count	Double Word [Signed]	UnitName_Admin_ProdDefectiveCount[#]_Count	Long
gvsta_Adm_ProdDefectiveCnt[#].d_AccCount	Double Word [Signed]	UnitName_Admin_ProdDefectiveCount[#]_AccCount	Long
gvd_Adm_AccTimeSinceReset	Double Word [Signed]	UnitName_Admin_AccTimeSinceReset	Long
gvl_Adm_MachDesignSpeed	FLOAT [Double Precision]	UnitName_Admin_MachDesignSpeed	Double
gvd_Adm_StatesDisabled	Double Word [Signed]	UnitName_Admin_StatesDisabled	Long
gvwa_Adm_PACDateTime_Date	Double Word [Unsigned]/Bit String [32-bit](0..6)		
gvwa_Adm_PACDateTime_Date[0]	Double Word [Signed]	UnitName_Admin_PLCDateTime[0]	Long
gvwa_Adm_PACDateTime_Date[1]	Double Word [Signed]	UnitName_Admin_PLCDateTime[1]	Long
gvwa_Adm_PACDateTime_Date[2]	Double Word [Signed]	UnitName_Admin_PLCDateTime[2]	Long
gvwa_Adm_PACDateTime_Date[3]	Double Word [Signed]	UnitName_Admin_PLCDateTime[3]	Long
gvwa_Adm_PACDateTime_Date[4]	Double Word [Signed]	UnitName_Admin_PLCDateTime[4]	Long
gvwa_Adm_PACDateTime_Date[5]	Double Word [Signed]	UnitName_Admin_PLCDateTime[5]	Long
gvwa_Adm_PACDateTime_Date[6]	Double Word [Signed]	UnitName_Admin_PLCDateTime[6]	Long

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 4 – PackML Core Function Blocks

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 29, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

The purpose of this document is to describe the design considerations and implementation approaches of implementing PackML specification in an iQ PLC.

PackML specification is a part of the overall OMAC PackML standard and consists of PackTags and PacKML State Engine definitions. PackTags defines a set of named data elements used for open architecture, interoperable data exchange in automated machinery. PackTags are useful for machine-to-machine (inter-machine) communications; for example between a Filler and a Capper. PackTags can also be used for data exchange between machines and higher-level information systems like Manufacturing Operations Management and Enterprise Information Systems. PackML State Engine defines common procedural programming structures, consistent mode and state definitions that drive a common look and feel between equipment.

The Mitsubishi design of PackTags is documented in Part 3 of this Users Guide. This document describes the implementation of Mitsubishi PackML core function blocks that handle the PackML Machine State Transitions, Mode Manager, and State and Mode Timers, and also the execution of machine breakdown structures.

The function blocks are implemented using Mitsubishi GX Works3 Functional Block Diagram Programming language and label programming methods.

2 Overview of PackML State and Mode Core Function Blocks

There are two Mitsubishi PackML State and Mode core function blocks:

- PackML_ModeStateManager
- PackML_ModeStateTimes

The two key functions of the PackML_ModeStateManger are: (1) transitioning the machine from current state to the proper next state based on external commands and state completion status, and (2) handling the transitions of machine modes. The PackML_ModeStateTimes (1) accumulates the current and accumulated time of the machine in each mode and state, and (2) provides the timer values and stores them in appropriate PackTags.

These function blocks, together with their associated global and local labels, are packaged in the overall Mitsubishi PackML OEM Implementation template project.

3 Function Block: PackML_ModeStateManager

3.1 Description

The PackML_ModeStateManager handles the state and mode transitions of a unit machine according to the State and Mode Models defined in the OMAC PackML specification.

To use this Function Block properly in an OEM program, one should ensure the following requirements are satisfied:

1. When an OEM programs a machine to use the PackML Function Blocks, it should initialize the machine to start up with the machine mode set to 3, the **Manual Mode** condition, and the state to be at the "Stopped" stage during the first scan of the PLC.
2. When an OEM designs the machine, he should determine how many modes the machine will have and how many and what states each mode should have. The selection of modes and states should follow the OMAC PackML Standard when appropriate. Each mode, when defined, should have at least three states: Stopped, Execute, Aborted.
3. Since not all states are configured for all modes, the OEM is responsible for setting up which states are not configured for each mode. He is also responsible for setting up at which states the machine is allowed to change mode. Refer to Part 6 Section 5.1.1 3.3 of the Users Guide for more details.

4. The OEM is also responsible for configuring the names of all modes and states. Refer to Part 6 Section 5.1.1 of the Users Guide for more details.
5. When this FB is used in an OEM program, it is preferable that the FB is always enabled.
6. The label "PackML" is a structured data type and its members should be properly defined before the FB is called.

3.2 Function Block Operations

The main functions of the FB consist of (1) managing state transitions, (2) managing mode transitions, and (3) updating current mode and state information.

When the FB is first call, it determines the current state of the machine and whether there is a valid command to transition the machine to a new state. If a valid command is set, the machine will be transitioned to the valid new state and the corresponding output bit will be set to reflect the new current state.

The FB will then examine any mode change command is set. It will verify the machine is in the proper mode and state that a change of mode is allowed. If the mode change command is not valid, the ModeChangeNotAllowed output will be set high for 3 seconds and then reset. The mode and state of the unit machine will remain in the current mode and state respectively.

The FB finally updates the current mode and state information and exit to the FB calling programs.

The PackML_ModeStateManager Function Block is shown in the figure below:

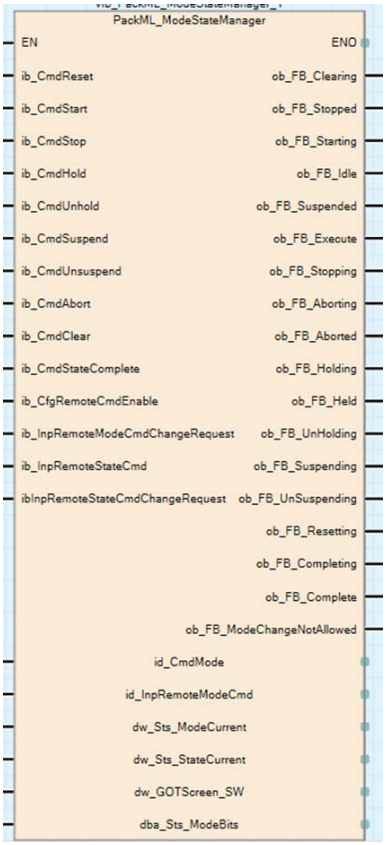


Figure 1 PackML_ModeStateManager Function Block with Inputs and Outputs

3.3 Function Block Local Variables

The local variables that are used by the FB are described in this section. There are three types of local variables:

- Input Variables – The values of these variables need to be properly set / defined by connecting proper logic or variable inputs to the FB before execution.
- Output Variables – The values of these variables will be properly set or defined after the execution of the FB is completed. If a user of the FB can chose not to use the results of the output variables.
- Variables – Those labels used internally in the FB that will not be exposed externally.

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	ob_FB_Clearing	Bit	When the bit is set, the Machine is in the "Clearing" State
VAR_OUTPUT	ob_FB_Stopped	Bit	When the bit is set, the Machine is in the "Stopped" State
VAR_OUTPUT	ob_FB_Starting	Bit	When the bit is set, the Machine is in the "Starting" State
VAR_OUTPUT	ob_FB_Idle	Bit	When the bit is set, the Machine is in the "Idle" State
VAR_OUTPUT	ob_FB_Suspended	Bit	When the bit is set, the Machine is in the "Suspended" State
VAR_OUTPUT	ob_FB_Execute	Bit	When the bit is set, the Machine is in the "Execute" State
VAR_OUTPUT	ob_FB_Stopping	Bit	When the bit is set, the Machine is in the "Stopping" State
VAR_OUTPUT	ob_FB_Aborting	Bit	When the bit is set, the Machine is in the "Aborting" State
VAR_OUTPUT	ob_FB_Aborted	Bit	When the bit is set, the Machine is in the "Aborted" State
VAR_OUTPUT	ob_FB_Holding	Bit	When the bit is set, the Machine is in the "Holding" State
VAR_OUTPUT	ob_FB_Held	Bit	When the bit is set, the Machine is in the "Held" State
VAR_OUTPUT	ob_FB_UnHolding	Bit	When the bit is set, the Machine is in the "Unholding" State
VAR_OUTPUT	ob_FB_Suspending	Bit	When the bit is set, the Machine is in the "Suspending" State
VAR_OUTPUT	ob_FB_UnSuspending	Bit	When the bit is set, the Machine is in the "UnSuspending" State
VAR_OUTPUT	ob_FB_Resetting	Bit	When the bit is set, the Machine is in the "Resetting" State
VAR_OUTPUT	ob_FB_Completing	Bit	When the bit is set, the Machine is in the "Completing" State
VAR_OUTPUT	ob_FB_Complete	Bit	When the bit is set, the Machine is in the "Complete" State
VAR_OUTPUT	ob_FB_ModeChangeNotAllowed	Bit	When the bit is set, the requested new mode is not valid and the "Mode Change" command is not allowed. The machine will remain in the current mode and current state. This bit will remain on for 3 seconds and then reset itself.
VAR_INPUT	ib_CmdReset	Bit	Setting this bit, the user program indicating the "Reset" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Reset" transition, the machine will remain in the current state and the "Reset" command will be ignored.
VAR_INPUT	ib_CmdStart	Bit	Setting this bit, the user program indicating the "Start" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Start" transition, the machine will remain in the current state and the "Start" command will be ignored.
VAR_INPUT	ib_CmdStop	Bit	Setting this bit, the user program indicating the "Stop" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Stop" transition, the machine will remain in the current state and the "Stop" command will be ignored.
VAR_INPUT	ib_CmdHold	Bit	Setting this bit, the user program indicating the "Hold" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Hold" transition, the machine will remain in the current state and the "Hold" command will be ignored.

Variable Type	Variable Label	Data Type	Description
VAR_INPUT	ib_CmdUnhold	Bit	Setting this bit, the user program indicating the "UnHold" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "UnHold" transition, the machine will remain in the current state and the "UnHold" command will be ignored.
VAR_INPUT	ib_CmdSuspend	Bit	Setting this bit, the user program indicating the "Suspend" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Suspend" transition, the machine will remain in the current state and the "Suspend" command will be ignored.
VAR_INPUT	ib_CmdUnsuspend	Bit	Setting this bit, the user program indicating the "UnSuspend" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "UnSuspend" transition, the machine will remain in the current state and the "UnSuspend" command will be ignored.
VAR_INPUT	ib_CmdAbort	Bit	Setting this bit, the user program indicating the "Abort" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Abort" transition, the machine will remain in the current state and the "Abort" command will be ignored.
VAR_INPUT	ib_CmdClear	Bit	Setting this bit, the user program indicating the "Clear" command has been received and the machine should transition to the proper next state. If the current machine state does not support the "Clear" transition, the machine will remain in the current state and the "Clear" command will be ignored.
VAR_INPUT	ib_CmdStateComplete	Bit	Setting this bit, the user program indicating the "State Complete" condition has been received and the machine should transition to the proper next state. If the current machine state does not support the "State Complete" transition, the machine will remain in the current state and the "State Complete" command will be ignored.
VAR_INPUT	ib_CfgRemoteCmdEnable	Bit	When this bit is set, the machine is allowing state and mode transition commands to be issued remotely in addition to the Command Bits to the FB.
VAR_INPUT	ib_InpRemoteModeCmdChangeRequest	Bit	When this bit is set and the machine is allowing mode transition commands to be issued remotely, the mode change command will then be evaluated and accepted if it is valid.
VAR_INPUT	id_InpRemoteStateCmd	Double Word	This input contains the Remote State Command value and is the value of the new state the machine should transition to. If the input value does not change, no state change will occur and the machine will remain in the current state. The valid State Command values are defined as follows and others are ignored: 1: Reset 2: Start 3: Stop 4: Hold 5: UnHold 6: Suspend 7: UnSuspend 8: Abort 9: Clear
VAR_INPUT	ib_InpRemoteStateCmdChangeRequest	Bit	When this bit is set and the machine is allowing state transition commands to be issued remotely, the state change command will then be evaluated and accepted if it is valid.

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	id_CmdMode	Double Word	The value of the new mode the machine will transition to. If the CmdMode input value does not change, no mode change will occur and the machine will remain in the current mode. The valid values of CmdMode are 0 – 31. The FB allows up to 31 valid modes and "0" being "NoMode". However, the user programs should have proper logic in place to handle all valid machine modes.
VAR_IN_OUT	id_InpRemoteModeCmd	Double Word	This input contains the Remote Mode Command value and is the value of the new mode the machine should transition to. If the input value does not change, no mode change will occur and the machine will remain in the current mode. The valid values are 0 – 31. The FB allows up to 31 valid modes and "0" being "NoMode". However, the user programs should have proper logic in place to handle all valid machine modes.
VAR_IN_OUT	dw_Sts_ModeCurrent	Double Word	Current mode value
VAR_IN_OUT	dw_Sts_StateCurrent	Double Word	Current state value
VAR_IN_OUT	dw_GOTScreen_SW	Word	Current GOT screen selection
VAR_IN_OUT	dba_Sts_ModeBits(0..31)	Bit Array	Current mode Bit value

4 Function Block: PackML_ModeStateTimes

4.1 Description

The PackML_ModeStateTimes accumulates the timer values for all configured states and modes of a unit-machine. It also accumulate the overall machine time since the last reset. The time unit of each timer is "second," and each timer will roll over at 900,000,000 seconds (i.e. 10416.67 days, or 28.5 years).

When any of the timers is rolled over, a TimeRollOverWarning bit will be set. However, the OEM programs will have to check the timers of current mode and current state to determine which timer has overflown.

The PackML_ModeStateTimes FB utilizes a custom function block "Timer_32Bit_Sec" FB to accumulate current mode and state time. The function of this FB is also described here.

The PackML_ModeStateTimes function block should be used right after the PackML_ModeStateManager function block in order to accumulate the time values of the current mode and state properly.

4.2 Timer_32Bit_Sec Function Block

To use the timer, define the beginning value of the timer by inputting the value to the "Start_Timer_Value_FB." When the Timer_Enable_FB is set high, the timer will accumulate in seconds and the current timer value can be read from the Current_Timer_Value_FB label. The maximum value of the timer is 900,000,000 seconds. It will overflow to zero when it passes the maximum value.

Figure 2 Timer_32Bit_Sec Function Block

If the Timer_Enable_FB bit is off, the timer will hold the current value and shown in Current_Timer_Value_FB. The timer will be reset to the Start_Timer_Value when it is enabled and the Timer_Reset_FB bit is high. It will continue in the Reset State until the Timer_Reset_FB bit is off.

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	Current_Timer_Value_FB	Double Word	Contains the current timer value.
VAR_INPUT	Start_Timer_Value_FB	Double Word	Contains the initial value of the timer before it starts accumulating.
VAR_INPUT	Timer_Enable_FB	Bit	Enables the timer to start accumulating.
VAR_INPUT	Timer_Reset_FB	Bit	Resets the timer value to the Start Timer Value. The reset is effective only when the timer is enabled.

4.3 Function Block Operations

The main functions of the FB consist of (1) managing the mode timer and updating the values of current of accumulated time of the mode, (2) managing the state timer and updating the values of current and accumulated time of the state, and (3) managing the timer of the overall machine and updating the values of the machine timer since last reset. The function block is shown in the figure below:

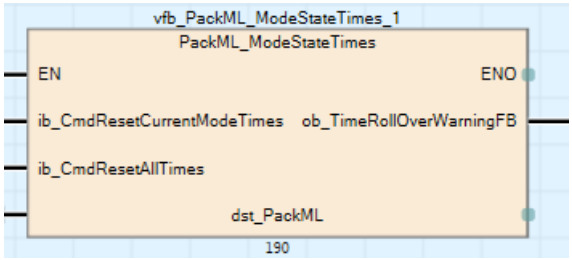


Figure 3 PackML_ModeStateTimes Function Block

4.4 Function Block Local Variables

The local variables that are used by the FB are described in this section.

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	ob_TimerRollOverWarningFB	Bit	If any of the timers over flows, this bit will be set until the timer is reset.
VAR_INPUT	ib_CmdResetrCurrentModeTimes	Bit	When this bit is set, the current mode timer values will be cleared to zero and timers of all states of the mode will also be cleared to zero.
VAR_INPUT	ib_CmdResetAllTimes	Bit	When this bit is set, all timer values, including the overall machine timer (i.e. TimeSinceLastReset) will be cleared to zero.
VAR_IN_OUT	dst_PackML	SDT	PackML FB global variables that need to be updated

5 Example Use of the PackML Function Blocks

The following example programs demonstrate how these function blocks can be used in an OEM program. The OEM PackML Implementation Template project will be described in Part 6 of this Users Guide and will have more complete program routines describing the use of PackML Core Function Blocks.

5.1 Initialization Example

The following rungs only need to be executed on the first scan of the PLC after power-up. They initialize the machine modes and states for proper operation.

The program file can be register to run under “Initial Program” area of Program Setting in the Project Tree as shown below:

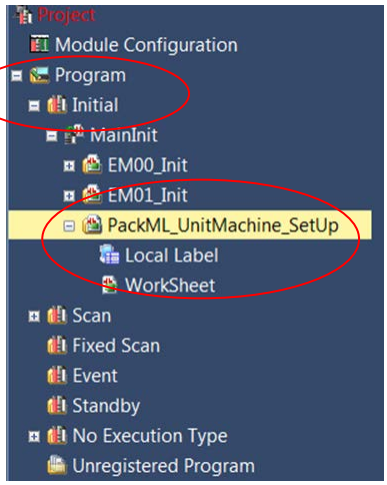


Figure 4 –Example of Setting the Program for Initial Scan Only

1. In this example, there are five valid modes: Mode 1 – Producing, Mode 2 - Maintenance, Mode 3 – Manual, Mode 16 – User Defined 1, and Mode 17 – User Define 2. Rung 2 set up these mode names in PackML_ModeNames. If there are additional modes for the machine, the user needs to set up additional mode names and logic to populate the proper labels.

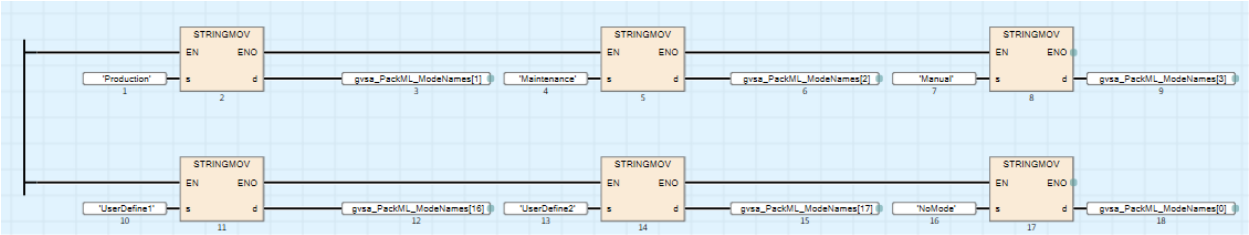


Figure 5 – Example of Setting PackML Modes for a Unit Machine

2. The following rung sets up all the state names in PackML_StateNames.

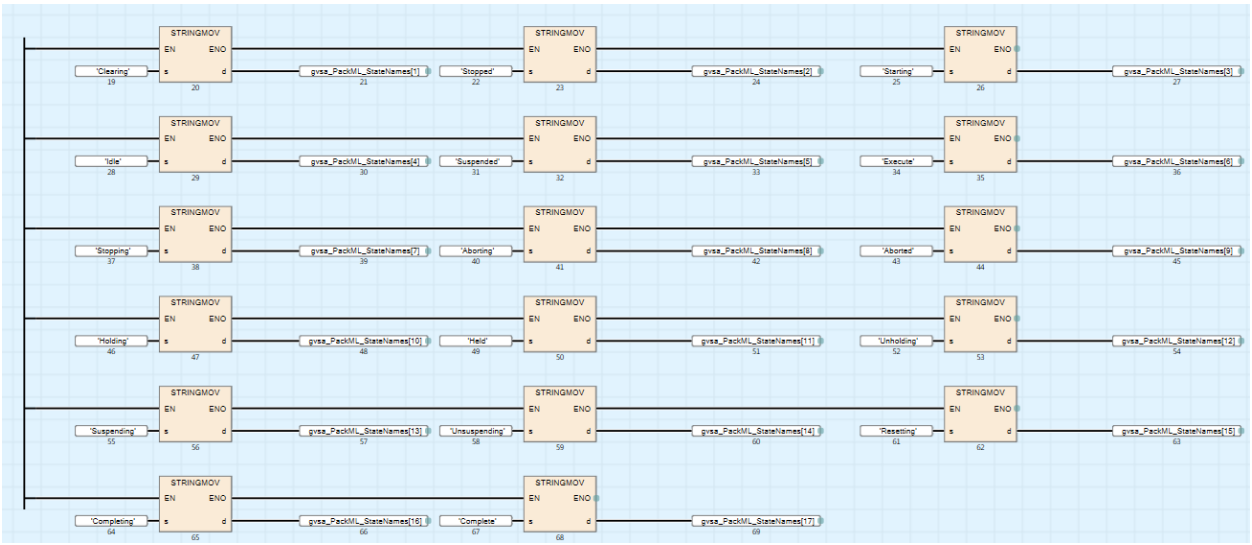


Figure 6 - Example of Setting PackML States for a Unit Machine

3. The following rung configures the states in each mode that mode transitions are allowed.

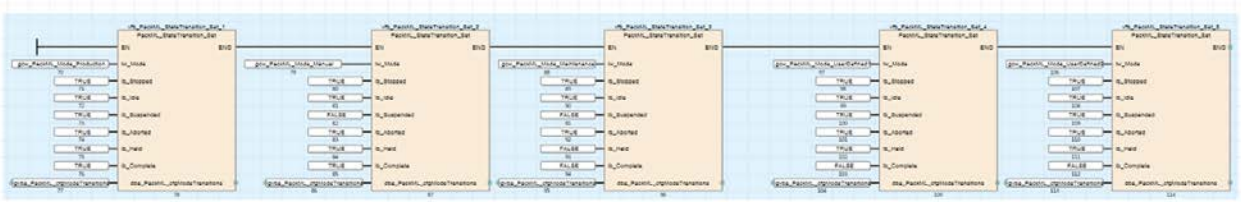


Figure 7 - Example of Setting States that Mode Transition are Allowed

4. The Following rung configures the states that are disabled in each mode.

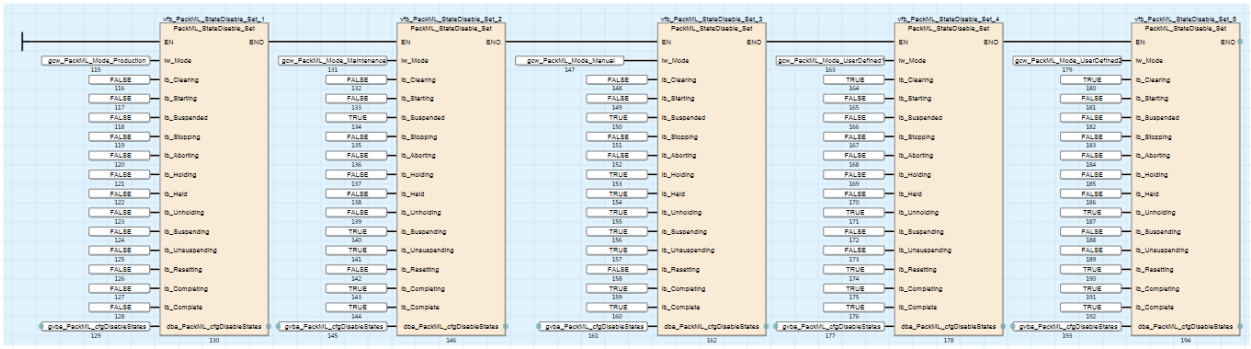


Figure 8 - Example of Setting States that are Disabled in Each Mode

5. The following rungs initializes the machine to Mode 0 (NoMode) and State 2 (Stopped) before execution, and updates the current mode name and state name in the proper labels.

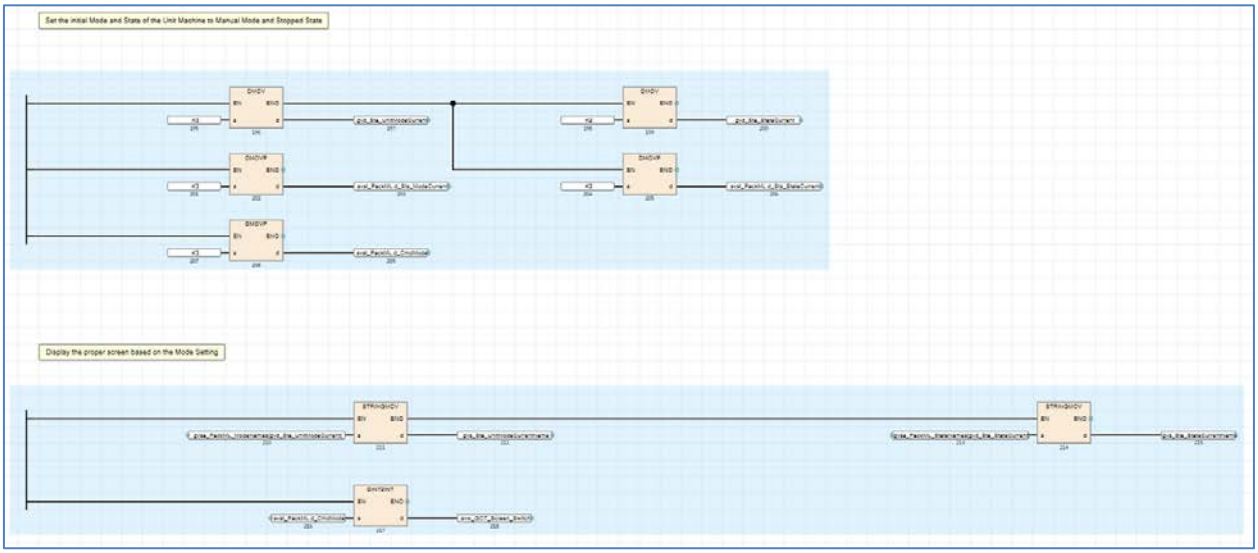


Figure 9 - Example of Setting Initial State and Mode of the Unit Machine

5.2 Example of Calling Function Blocks

1. The following rung calls the PackML_ModeStateManager function block to start the PackML operation. One should realize that the variables connected to the inputs and outputs of the FB members of the label PackML with the structured data type. The OEM programs should properly set up these values before the function block is called.

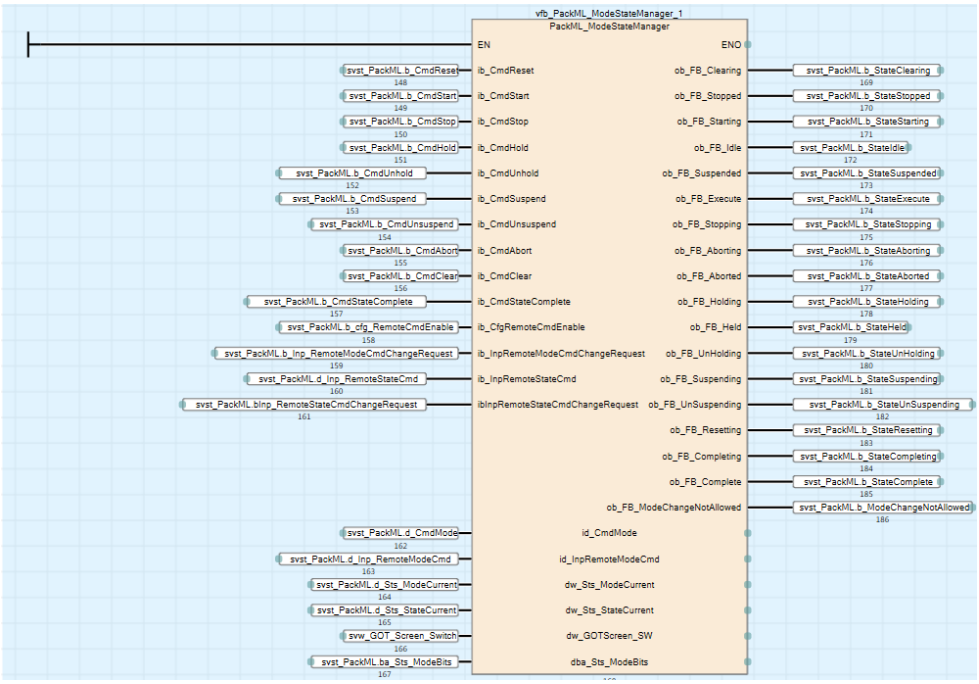


Figure 10 – Calling the PackML_ModeStateManager Function Block

2. The PackML_ModeStateTimes function block is then called to start accumulate timer values.

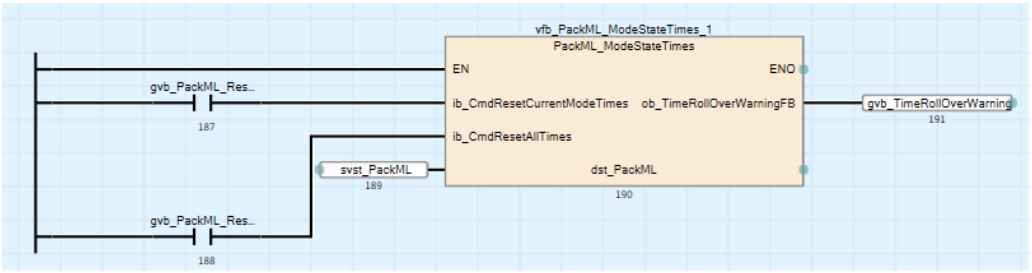


Figure 11 – Calling the PackML_ModeStateTimes Function Block

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 5 –Event Handling FBs

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

The purpose of this document is to describe the design considerations and implementation approaches of implementing PackML Alarm and Event handling function blocks in the Mitsubishi PackML OEM Implementation Template.

Even though alarm and event handling methods are not specified in or required by the PackML standard, it is beneficial to end users since standard alarm and event handling implementations in machine program promote consistent operations across multiple machines in a factory.

The Mitsubishi implementation of PackML Alarm and Event Handling Function Blocks follows the philosophy and methods shown in the OMAC Users Group PackML Implementation Guideline.

The design of these function blocks and how they can be used in a machine program implementation are described in this document. Descriptions are included on how to expand or modify the design of these function blocks to accommodate different fault handling philosophy of a particular user. However, the modifications that can be made are minor in nature. Totally different fault handling philosophy will require re-write of these function blocks.

The function blocks are implemented using Mitsubishi GX Works2 Structure Text Programming language and label programming methods.

2 Overview of the Event Handling Philosophy

The high-level overview of the implementation of Mitsubishi Alarm and Warning Event handling methods is described in this section. The alarm and warning events are together referred to as events in this document.

- The function blocks can be used to handle alarms and warning events separately or as one type of events. If a user determines to treat alarms and warning events separately, the same function blocks can be used to handle both types of events. The user will have to keep track of the alarms and warnings separately. For example, one Event_Manager FB is used to process alarm events of an equipment module and a second Event_Manager FB used to process warning events of the same module.
- The design of the function blocks can handle events up to 10 different categories. A user can determine how many categories are necessary for his applications.
- A user can also determine how the machine should react to any of these categories of events. The implemented actions in the template software (as described below) can be easily modified to behave differently.
    - In the Mitsubishi PackML Implementation Template package, Category 0 and Category 1 events will cause a PackML Abort command to be issued and the PackML state machine will transition into “Aborting” state.
    - In the Mitsubishi PackML Implementation Template package, Category 2, Category 3, and Category 4 events will cause a PackML Stop command to be issued and the PackML state machine will transition into “Stopping” state.
    - In the Mitsubishi PackML Implementation Template package, Category 5 through Category 9 events will not cause any PackML command to be issued. The PackML state machine will remain at its current state. The user is responsible to determine what action(s) the machine should take.
- When an event becomes active then inactive and active again before an “Event Reset” command is issued to the Event_Manager, the second activation of the event is considered the same event as before and not a new event. The Trigger bit of the event (refer to Section 4.3.1below for the Structured Data Type of an event) will reflect the status of the event.
- The Event_Manager and Event_Summation FBs capture the first out event of an equipment module and the unit machine respectively. These FBs also capture the first out event of each event category separately.
- When PackML State Machine is in the Resetting State or Clearing State, an Alarm Reset Command will be issued to Even_Manager function blocks to clear all latched event flags.

3 Overview of the Alarm and Event Handling Function Blocks

3.1 CM_Event Function Block

The CM_Event FB is mainly used in Control Module routines to capture an alarm or warning event. It copies an event configuration in the overall Equipment Module event list when the input event occurs. The event is latched on even when the event condition becomes false. The event will stay latched until a reset command is issued to the Event_Manager FB for the particular Equipment Module. Each event will require an instance of this FB.

3.2 Event_Manager Function Block

One and only one Event_Manager Function Block is required to manage the alarms for each Equipment Module. When required, a second Event_Manager is used to manage warning events of the same Equipment Module.

The main function of the Event_Manager Function block is to collect all events from all Control Modules of a particular Equipment Module and create one list of the events for this Equipment Module.

The FB summarizes the data and identifies the "First-Out" event of the Equipment Module as well as the "First Out" events of each Event Category. The FB also unlatches Control Module events when a Reset Command is received.

3.3 Event_Summation Function Block

The Event_Summation FB is used at the Unit Machine level and it aggregates the alarms or warning events from all Equipment Modules within the Unit Machine.

It identifies the "First Out" event for the Unit Machine and also the "First Out" event for each fault categories at the Unit Machine Level.

The Event List of one of the Equipment Module is fed to the FB and the FB is called to produce an event list of the Unit Machine. The Event List of the second Equipment Module is then fed to the FB and the FB is called again to aggregate the two Equipment Module Event lists to the Unit Machine Event List. This process continues until all the event lists of Equipment Modules of the Unit Machine are consolidated.

3.3.1. Event_SummationBegin Function Block

The Event_SummantionBegin Function Block should be called before the first Event_Summation FB is executed. The purpose of this FB is to initialize the Event List of the Unit Machine before it is populated with the events from all equipment modules.

3.3.2. Event_SummationEnd Function Block

The Event_SummantionEnd Function Block should be called after the last Event_Summation FB is executed. The purpose of this FB is to clear the un-used array elements of the Unit Machine Event List.

3.4 Event_Sort Function Block

The Event_Sort Function Block performs the following functions:

- Processing the Unit Machine Event List and producing a list of all active events;
- Processing the Unit Machine Event List and producing a list of all active events of a selected category of events;
- Processing the Unit Machine Event List and producing a list of all events (both active and non-active) of a selected event category;
- Sorting the Unit Machine Event list by event time, regardless of event category, from the earliest event to the latest event;
- Sorting the Unit Machine Event list by event time of a selected category, from the earliest event to the latest event;

4 Function Block: CM_Event

4.1 Description

CM_Event FBs are used in Control Module routines to capture alarm or warning events. One instance of the CM_Event function block is required to capture a single alarm or event. This FB is generally used in a Control Module routine to capture events generated in that particular Control Module operation.

The structure of a CM_Event Function Block is shown in the figure below:

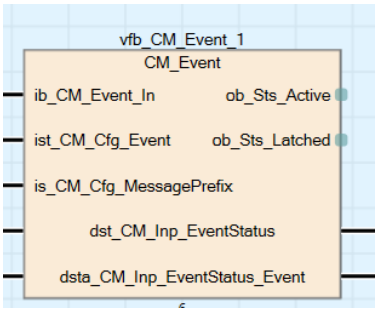


Figure 1 – CM_Event Function Block with Inputs and Outputs

It copies an event configuration in the overall Equipment Module event list when the input event occurs. The event is latched on even when the event condition becomes false. The event will stay latched until a reset command is issued to the Event_Manager FB for the particular Equipment Module. Each event will require an instance of this FB.

4.2 Function Block Local Variables

The local variables that are used by the FB are described in this section. There are four types of local variables:

- Input Variables – The values of these variables need to be properly set / defined by connecting proper logic or variable inputs to the FB before execution.
- Output Variables – The values of these variables will be properly set or defined after the execution of the FB is completed. If a user of the FB can chose not to use the results of the output variables.
- In_Out Variables – The variables that are set by the connecting logic, but the values are then processed by the FB logic and the resulting values are written to the variables connected to the outputs.
- Variables – Those labels used internally in the FB that will not be exposed externally.

The following table describes the Input, Output, and In_Out variables used by the function block. The details of Structured Data Types are defined in Section 4.3 below.

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	ob_Sts_Active	Bit	This bit is on when the event is currently active.
VAR_OUTPUT	ob_Sts_Latched	Bit	This bit is on when this event has been active at least once since the last reset of events.
VAR_INPUT	ib_CM_Event_In	Bit	When this bit is on, it indicates that an alarm or event active and may need to be captured.
VAR_INPUT	ist_CM_Cfg_Event	SDT_EventCfg	This structured variable contains the information of the alarm or event
VAR_INPUT	is_CM_Cfg_MessagePrefix	String(16)	This variable contains any prefix that can be appended to the alarm message

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dst_CM_Inp_EventStatus	SDT_EventStatus	This structured variable contains the event status of an equipment module where this particular event needs to be stored and used to update the equipment module event status
VAR_IN_OUT	dsta_CM_Inp_EventStatus_Event	SDT_Event(0..29)	The actual list of events of an equipment module. This array is limited to the maximum of 30 events per equipment module.

4.3 Event Related Structured Data Type

4.3.1. SDT_Event Structured Data Type

The SDT_Event is used to describe each alarm or warning event.

Label	Data Type	Description
d_ID	Double Word[Signed]	An unique value assigned to each event as defined by an end user
d_Value	Double Word[Signed]	Value can be used to specify additional details associated with an event. For example, an alarm ID of 1 may indicate an E-Stop PB is pressed, but the alarm value may be used to indicate which E-Stop PB is pressed. The use of this field is determined by users of the FB.
s_Message	String(50)	Text message of the event, up to the maximum of 50 characters.
wa_TimeEventArray	Word[Signed](0..6)	The actual time, read from the PLC, when the event occurs. The QPLC format is as follows: [0]: Year (1980 to 2079) [1]: Month (1-12) [2]: Day (1-31) [3]: Hour in 24 hour clock format (0 to 23) [4]: Minutes (0-59) [5]: Seconds (0-59) [6]: Day of week (0-6)
wa_TimeAckArray	Word[Signed](0..6)	The actual time, read from the PLC, when the event condition clears. [0]: Year (1980 to 2079) [1]: Month (1-12) [2]: Day (1-31) [3]: Hour in 24 hour clock format (0 to 23) [4]: Minutes (0-59) [5]: Seconds (0-59) [6]: Day of week (0-6)
d_Category	Double Word[Signed]	The category of the event. The current design allows Categories 0 through 9.
b_Trigger	Bit	1: the event is active 0: the event is not active

4.3.2. SDT_EventCfg Structured Data Type

This SDT is mainly used to define a particular alarm or warning event.

Label	Data Type	Description
d_ID	Double Word[Signed]	An unique value assigned to each event as defined by an end user
d_Value	Double Word[Signed]	Value can be used to specify additional details associated with an event. For example, an alarm ID of 1 may indicate an E-Stop PB is pressed, but the alarm value may be used to indicate which E-Stop PB is pressed. The use of this field is determined by users of the FB.

Label	Data Type	Description
s_Message	String(34)	Text message of the event, up to the maximum of 34 characters.
d_Category	Double Word[Signed]	The category of the event. The current design allows Categories 0 through 9.

4.3.3. SDT_EventStatus

Label	Data Type	Description
d_Sts_NumEvents	Double Word[Signed]	The number of events in the Event List
d_Sts_NumAllEvents	Double Word[Signed]	The total number of events that are active
b_Sts_Category_0_Latched	Bit	A Category 0 event has occurred and flag is latched until event is reset
b_Sts_Category_1_Latched	Bit	A Category 1 event has occurred and flag is latched until event is reset
b_Sts_Category_2_Latched	Bit	A Category 2 event has occurred and flag is latched until event is reset
b_Sts_Category_3_Latched	Bit	A Category 3 event has occurred and flag is latched until event is reset
b_Sts_Category_4_Latched	Bit	A Category 4 event has occurred and flag is latched until event is reset
b_Sts_Category_5_Latched	Bit	A Category 5 event has occurred and flag is latched until event is reset
b_Sts_Category_6_Latched	Bit	A Category 6 event has occurred and flag is latched until event is reset
b_Sts_Category_7_Latched	Bit	A Category 7 event has occurred and flag is latched until event is reset
b_Sts_Category_8_Latched	Bit	A Category 8 event has occurred and flag is latched until event is reset
b_Sts_Category_9_Latched	Bit	A Category 9 event has occurred and flag is latched until event is reset
b_Sts_Category_0_NotLatched	Bit	A Category 0 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_1_NotLatched	Bit	A Category 1 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_2_NotLatched	Bit	A Category 2 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_3_NotLatched	Bit	A Category 3 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_4_NotLatched	Bit	A Category 4 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_5_NotLatched	Bit	A Category 5 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_6_NotLatched	Bit	A Category 6 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_7_NotLatched	Bit	A Category 7 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_8_NotLatched	Bit	A Category 8 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.

Label	Data Type	Description
b_Sts_Category_9_NotLatched	Bit	A Category 9 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_EventActive	Bit	This flag is set when there is an active event
d_Wrk_ResetID	Double Word[Signed]	This is an internal variable for FBs use. Do not modify the values.
d_Wrk_EventArraySize	Double Word[Signed]	This is an internal variable for FBs use. Do not modify the values.
d_Wrk_StringEventSize	Double Word[Signed]	This is an internal variable for FBs use. Do not modify the values.
d_Wrk_UpdateEventListID	Double Word[Signed]	This is an internal variable for FBs use. Do not modify the values.

4.4 Function Block Operations

The following example is used to describe how to use the CM_Event function block:

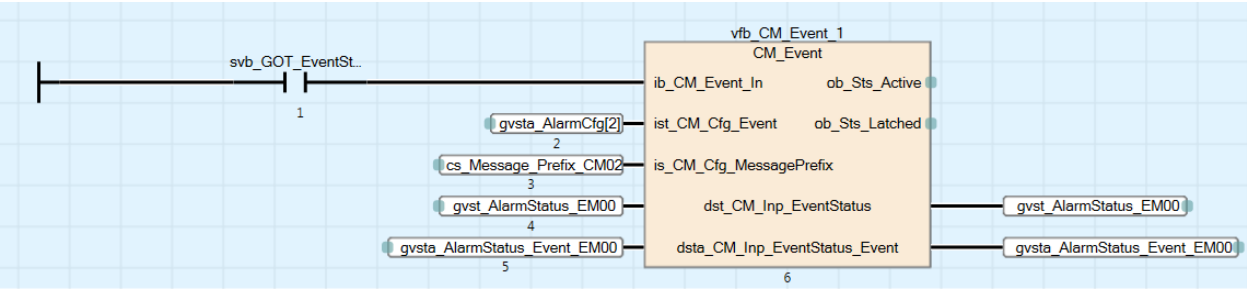


Figure 2 – Example of CM_Event Function Block

- When a "STOP" key for "Equipment Module 00" on an operator interface is pressed, the status of AlarmStatus_EM00 is updated with the information of the alarm that is configured in AlarmCfg[2] and Message_Prefix_CM02. This alarm configuration needs to be initialized by the end user before calling the FB and the happening of the actual event.
    - AlarmCfg[2].id:=65;
    - AlarmCfg[2].Value:=0;
    - AlarmCfg[2].Message:="Stop PB Pressed";
    - AlarmCfg[2].Category:=2;
    - Message_Prefix_CM02:= "HMI "
- The actual alarm information (e.g. ID, Message, Time of Active) is recorded in the AlarmStatus_Event_EM00 array. The Sts_Active and Sts_Latched bits are set high.
- When the "STOP" key of "Equipment Module 00" becomes inactive, the Acknowledge Time is recorded and the Sts_Active bit is reset. However, the Sts_Latched bit is still high until a reset command is issued to the Event_Manager FB (Refer to Section 5 of this document).

4.4.1. Examples of Alarm Data

4.4.1.1. AlarmStatus_EM00

The before values are the values of AlarmStatus_EM00 before any event occurs. After the GOT_StopKey is pressed, the values of AlarmStatus_EM00 were updated by the CM_Event FB and are shown as below:

Label	Value (before)	Value (After)
Sts_NumEvents	0	1
Sts_NumAllEvents	0	1
Sts_Category_0_Latched	0	0
Sts_Category_1_Latched	0	0
Sts_Category_2_Latched	0	1
Sts_Category_3_Latched	0	0
Sts_Category_4_Latched	0	0
Sts_Category_5_Latched	0	0
Sts_Category_6_Latched	0	0
Sts_Category_7_Latched	0	0
Sts_Category_8_Latched	0	0
Sts_Category_9_Latched	0	0
Sts_Category_0_NotLatched	0	0
Sts_Category_1_NotLatched	0	0
Sts_Category_2_NotLatched	0	1
Sts_Category_3_NotLatched	0	0
Sts_Category_4_NotLatched	0	0
Sts_Category_5_NotLatched	0	0
Sts_Category_6_NotLatched	0	0
Sts_Category_7_NotLatched	0	0
Sts_Category_8_NotLatched	0	0
Sts_Category_9_NotLatched	0	0
Sts_EventActive	0	1

4.4.1.2. AlarmStatus_Event_EM00 (when Event is active)

The actual event list of Equipment Module 00 is an array of 30 elements. Before any event, the array contains elements with zero values after initialization.

After the GOT_StopKey is set high, the alarm is recorded in the event list, and the configuration AlarmCfg[2] was copied into the first array location together with the time value when the event occurs.

Label	Value (before)	Value (After)
[0]		
ID	0	65
Value	0	0
Message		HMI STOP PB Pressed
TimeEventArray		
[0]	0	2010
[1]	0	6
[2]	0	28
[3]	0	9
[4]	0	15
[5]	0	7
[6]	0	1
TimeAckArray		
[0]	0	0
[1]	0	0
[2]	0	0
[3]	0	0
[4]	0	0
[5]	0	0
[6]	0	0
Category	0	2
Trigger	0	1

Label	Value (before)	Value (After)
[1]		
[2]		
…….		
[29]		

4.4.1.3. AlarmStatus_Event_EM00 (when Event becomes inactive)

After the GOT_StopKey becomes low, the alarm is acknowledged and the time value when the event is acknowledged is recorded. The Trigger bit is reset to indicate that the event is no longer active.

Label	Value (Before)	Value (After)
[0]		
ID	65	65
Value	0	0
Message	EM00 HMI STOP PB Pressed	HMI STOP PB Pressed
TimeEventArray		
[0]	2010	2010
[1]	6	6
[2]	28	28
[3]	9	9
[4]	15	15
[5]	7	7
[6]	1	1
TimeAckArray		
[0]	0	2010
[1]	0	6
[2]	0	28
[3]	0	9
[4]	0	26
[5]	0	6
[6]	0	1
Category	2	2
Trigger	1	0
[1]		
[2]		
…….		
[29]		

5 Function Block: Event_Manager

5.1 Description

The main purpose of the Event_Manager Function Block is to consolidate all events created by all Control Modules of a particular Equipment Module and create one list of the events for this Equipment Module.

One and only one Event_Manager Function Block is required to manage the alarms for each Equipment Module. When required, a second Event_Manager instance is used to manage warning events of the same Equipment Module if a user chooses to handle alarms and warning events separately.

The FB summarizes the events and identifies the “First-Out” event of the Equipment Module as well as the “First Out” events of each Event Category. The FB also unlatches Control Module events of this Equipment Module when a Reset Command is received.

The structure of the function block is shown in Figure 3 below.

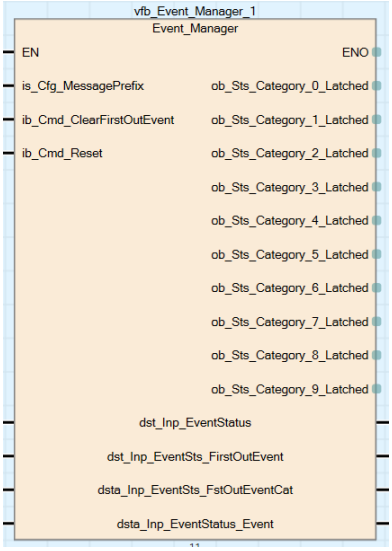


Figure 3 – Event_Manager Function Block

5.2 Function Block Local Variables

The Input, Output, and In_Out variables that are used by the FB are described in this section. The details of Structured Data Types are defined in Section 4.3 .

Variable Type	Variable Label	Data Type	Description
VAR_INPUT	is_Cfg_MessagePrefix	String(16)	This variable contains any prefix that can be appended to the alarm messages of this equipment module alarm list.
VAR_INPUT	ib_Cmd_ClearFirstOutEvent	Bit	When this bit is set, all First-Out events of this equipment module will be cleared.
VAR_INPUT	ib_Cmd_Reset	Bit	When this bit is set, all latched events that are not currently active will be cleared.
VAR_OUTPUT	ob_Sts_Category_0_Latched	Bit	A Category 0 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_1_Latched	Bit	A Category 1 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_2_Latched	Bit	A Category 2 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_3_Latched	Bit	A Category 3 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_4_Latched	Bit	A Category 4 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_5_Latched	Bit	A Category 5 event has occurred and flag is latched until event is reset
VAR_OUTPUT	Sts_Category_6_Latched	Bit	A Category 6 event has occurred and flag is latched until event is reset
VAR_OUTPUT	Sts_Category_7_Latched	Bit	A Category 7 event has occurred and flag is latched until event is reset

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	ob_Sts_Category_8_Latched	Bit	A Category 8 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_9_Latched	Bit	A Category 9 event has occurred and flag is latched until event is reset
VAR_IN_OUT	dst_Inp_EventStatus	SDT_EventStatus	This structured variable contains the event status of an equipment module
VAR_IN_OUT	dst_Inp_EventStatus_FirstOutEvent	SDT_Event	This structured variable contains the information of the first out event of the equipment module
VAR_IN_OUT	dsta_Inp_EventStatus_FirstOutEventCat	SDT_Event(0..9)	This array of structured variables contains the information of first out events of all event categories
VAR_IN_OUT	dsta_Inp_EventStatus_Event	SDT_Event(0..29)	The actual list of events of an equipment module. This array is limited to the maximum of 30 events per equipment module.

5.3 Function Block Operations

The Event_Manager FB should be called in a program scan after all control modules have been executed, and the events associated with all control modules have been recorded using CM_Event FBs. In the example below, AlarmStatus_EM00 and AlarmStatus_Event_EM00 arrays of SDTs have been updated by all CM_Event FBs before the Event_Manager FB is called.

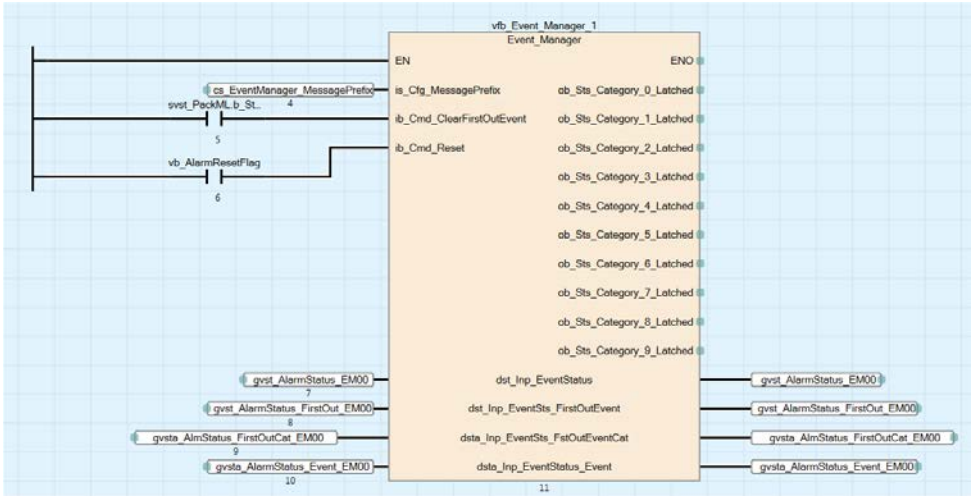


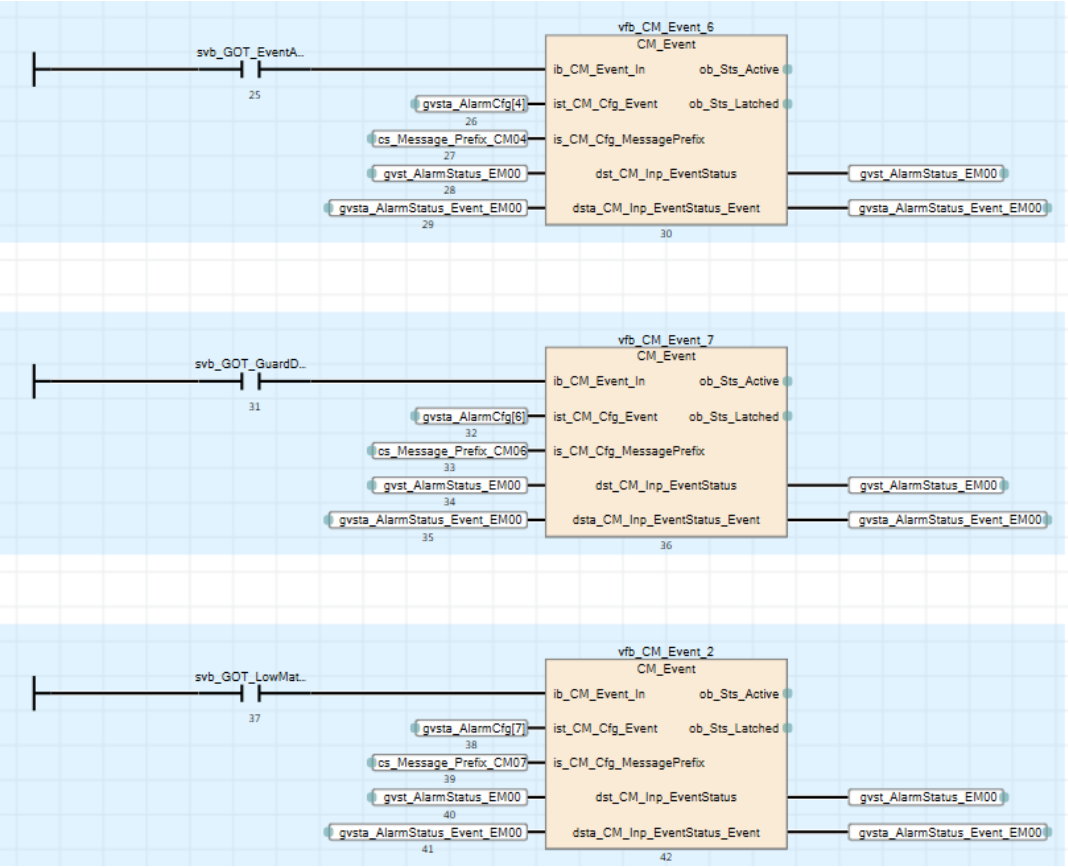
Figure 4 – Example of Event_Manager Function Block

The Event_Manager function block then appends the EventManager_MessagePrefix to each event message, sorts out the list of events in AlarmStatus_Event_EM00 and identifies the First Out event for the Equipment module as well as the First Out events for each event category. The results are updated in AlarmStatus_EM00, AlarmStatus_FirstPut_EM00, AlarmStatus_FirstOutCat_EM00, and AlarmStatus_Event_EM00.

5.4 Example of Using CM_Event and Event_Manager FBs

The following logic will capture the events of “Abort”, “GuardDoorOpen”, and “LowMaterial” keys are pressed on a GOT by the instances of the CM_Event function block. The AlarmStatus_EM00 and AlarmStatus_Event_EM00 will be updated with these events.

The Event_Manager will then consolidate the events and record the overall first out event of EM00 and first out events for each category. The results will then be used by Event_Summation FB to produce an event list for the overall unit machine.



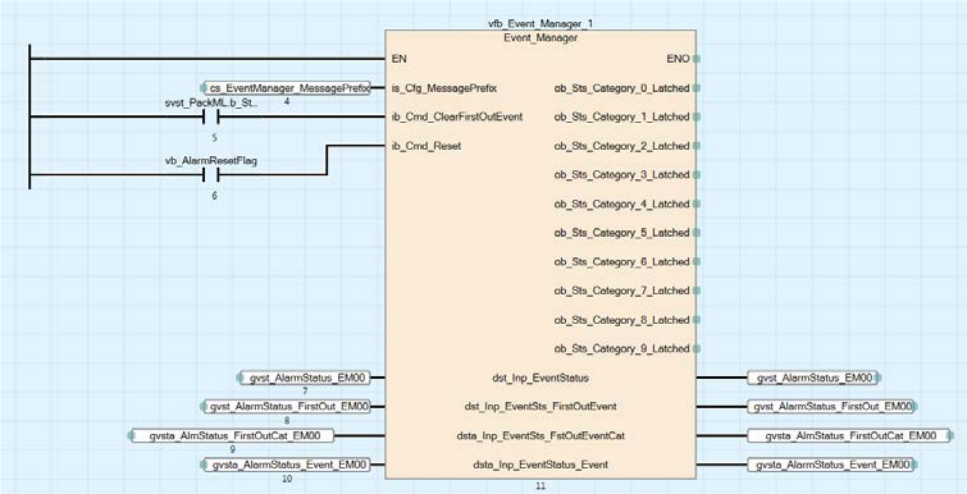


Figure 5 – Examples of using CM_Event and Event_Manager Function Blocks

6 Function Blocks: Event_Summation, Event_SummationBegin, Event_SummationEnd

6.1 Description

The Event_Summation Function Block summarizes the event data from equipment modules (created by using Event_Manager FBs) and creates a summarized list of events for the unit machine. It summarizes the events by locating the unit machine first out event and unit machine first out events for all categories that are active.

Additionally, associated function blocks **Event_SummationBegin** and **Event_SummationEnd** initializes and cleans-up the list of events after the summation of events from all equipment modules.

The structure of the Event_Summation function block, Event_SummationBegin, and Event_SummationEnd are shown in Figure 6, Figure 7, and Figure 8 respectively.

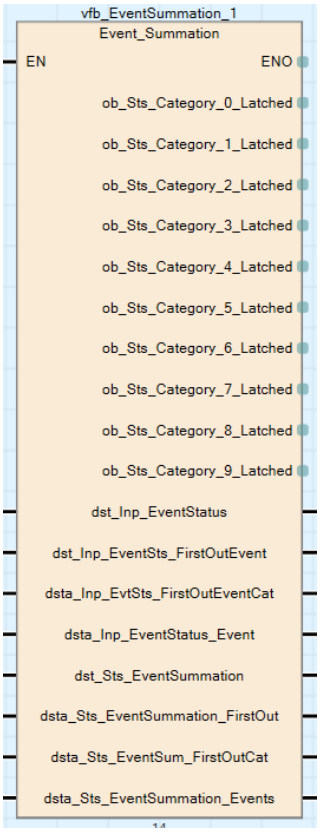


Figure 6 – Event_Summation Function Block

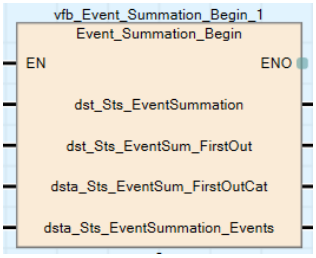


Figure 7 – Event_SummationBegin Function Block

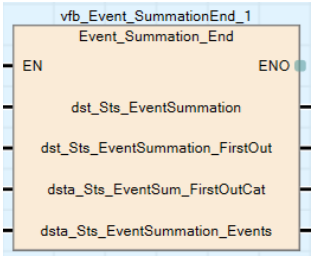


Figure 8 – Event_SummationEnd Function Block

6.2 Function Block Local Variables

The Input, Output, and In_Out variables that are used by the FBs are described in this section. The details of Structured Data Types are defined in Section 4.3 and Section 6.3 of this document.

6.2.1. Even_Summation FB Variables

Variable Type	Variable Label	Data Type	Description
VAR_OUTPUT	ob_Sts_Category_0_Latched	Bit	A Category 0 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_1_Latched	Bit	A Category 1 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_2_Latched	Bit	A Category 2 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_3_Latched	Bit	A Category 3 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_4_Latched	Bit	A Category 4 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_5_Latched	Bit	A Category 5 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_6_Latched	Bit	A Category 6 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_7_Latched	Bit	A Category 7 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_8_Latched	Bit	A Category 8 event has occurred and flag is latched until event is reset
VAR_OUTPUT	ob_Sts_Category_9_Latched	Bit	A Category 9 event has occurred and flag is latched until event is reset

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dst_Inp_EventStatus	SDT_EventStatus	This structured variable contains the event status of an equipment module that will be summarized
VAR_IN_OUT	dst_Inp_EventStatus_FirstOutEvent	SDT_Event	This structured variable contains the information of the first out event of the equipment module that will be summarized
VAR_IN_OUT	dsta_Inp_EventStatus_FirstOutEventCat	SDT_Event(0..9)	This array of structured variables contains the information of first out events of all event categories that will be summarized
VAR_IN_OUT	dsta_Inp_EventStatus_Event	SDT_Event(0..29)	The actual list of events of an equipment module that will be summarized. This array is limited to the maximum of 30 events per equipment module.
VAR_IN_OUT	dst_Sts_EventSummation	SDT_EventSummation	This structured variable contains the event status of the overall unit machine
VAR_IN_OUT	dst_Sts_EventSummation_FirstOut	SDT_Event	This structured variable contains the information of the first out event of the unit machine
VAR_IN_OUT	dsta_Sts_EventSummation_FirstOutCat	SDT_Event(0..9)	This array of structured variables contains the information of first out events of all event categories of the unit machine
VAR_IN_OUT	dsta_Sts_EventSummation_Events	SDT_Event(0..99)	The actual list of events of the unit machine. This array is limited to the maximum of 100 events per unit machine.

6.2.2. Even_SummationBegin FB Variables

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dst_Sts_EventSummation	SDT_EventSummation	This structured variable contains the event status of the overall unit machine
VAR_IN_OUT	dst_Sts_EventSummation_FirstOut	SDT_Event	This structured variable contains the information of the first out event of the unit machine
VAR_IN_OUT	dsta_Sts_EventSummation_FirstOutCat	SDT_Event(0..9)	This array of structured variables contains the information of first out events of all event categories of the unit machine
VAR_IN_OUT	dsta_Sts_EventSummation_Events	SDT_Event(0..99)	The actual list of events of the unit machine. This array is limited to the maximum of 100 events per unit machine.

6.2.3. Even_SummationEnd FB Variables

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dst_Sts_EventSummation	SDT_EventSummation	This structured variable contains the event status of the overall unit machine
VAR_IN_OUT	dst_Sts_EventSummation_FirstOut	SDT_Event	This structured variable contains the information of the first out event of the unit machine
VAR_IN_OUT	dsta_Sts_EventSummation_FirstOutCat	SDT_Event(0..9)	This array of structured variables contains the information of first out events of all event categories of the unit machine

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dsta_Sts_EventSummation_Events	SDT_Event(0..99)	The actual list of events of the unit machine. This array is limited to the maximum of 100 events per unit machine.

6.3 Event Summation Related Structured Data Type

6.3.1. SDT_EventSummation Structured Data Type

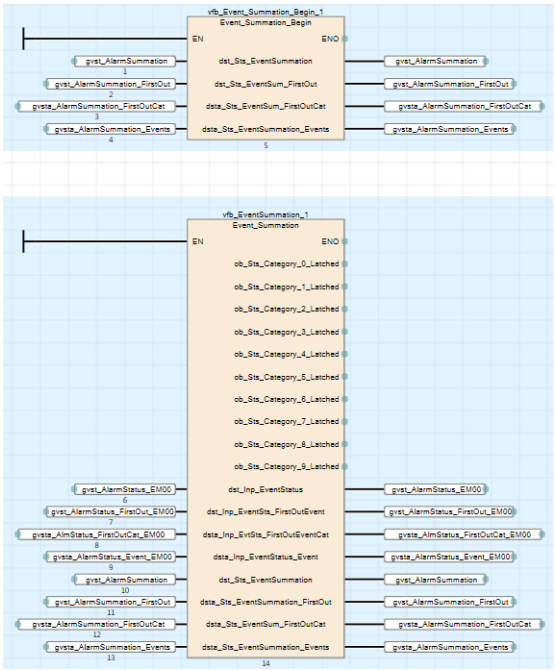
Label	Data Type	Description
d_Sts_NumEvents	Double Word[Signed]	Total number of events
b_Sts_Category_0_Latched	Bit	A Category 0 event has occurred and flag is latched until event is reset
b_Sts_Category_1_Latched	Bit	A Category 1 event has occurred and flag is latched until event is reset
b_Sts_Category_2_Latched	Bit	A Category 2 event has occurred and flag is latched until event is reset
b_Sts_Category_3_Latched	Bit	A Category 3 event has occurred and flag is latched until event is reset
b_Sts_Category_4_Latched	Bit	A Category 4 event has occurred and flag is latched until event is reset
b_Sts_Category_5_Latched	Bit	A Category 5 event has occurred and flag is latched until event is reset
b_Sts_Category_6_Latched	Bit	A Category 6 event has occurred and flag is latched until event is reset
b_Sts_Category_7_Latched	Bit	A Category 7 event has occurred and flag is latched until event is reset
b_Sts_Category_8_Latched	Bit	A Category 8 event has occurred and flag is latched until event is reset
b_Sts_Category_9_Latched	Bit	A Category 9 event has occurred and flag is latched until event is reset
b_Sts_Category_0_NotLatched	Bit	A Category 0 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_1_NotLatched	Bit	A Category 1 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_2_NotLatched	Bit	A Category 2 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_3_NotLatched	Bit	A Category 3 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_4_NotLatched	Bit	A Category 4 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_5_NotLatched	Bit	A Category 5 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_6_NotLatched	Bit	A Category 6 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_7_NotLatched	Bit	A Category 7 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_8_NotLatched	Bit	A Category 8 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.
b_Sts_Category_9_NotLatched	Bit	A Category 9 event has occurred and flag is NOT latched. The flag is reset when the event becomes inactive.

Label	Data Type	Description
b_Sts_CurrentActiveEvent	Bit	This flag is set when there is an active event
d_Wrk_EventArraySize	Double Word[Signed]	This is an internal variable for FBs use. Do not modify the values.

6.4 Function Block Operations

This example in Figure 9 below illustrates how the event list of a unit machine is created using the Event_Summation FBs and Event_SummationBegin and Event_SummationEnd FBs.

- The Event_SummationBegin FB is called first to initialize all structured labels and arrays of structured labels. After the execution of this FB, all the labels are initialized to zero (values) or blank (strings).
- The Event_Summation FB is called next in Rung 2 to process the alarm information for Equipment Module 00. The event lists of Equipment 00 are processed and documented in the unit machine event lists AlarmSummation, AlarmSummation_FirstOut, Alarm_Summation_FirstOutCat, and AlarmSummation_Events.
- The Event_SummationFB is called again in Rung 3 to process the alarm information for Equipment Module 01. The event lists of Equipment 01 are processed and compared to the information from EM00 that has already been recorded in the Unit Machine lists. The overall First Out event will be determined and updated when necessary, and so are the first out events for all categories. The combined information is then recorded in the unit machine event lists AlarmSummation, AlarmSummation_FirstOut, Alarm_Summation_FirstOutCat, and AlarmSummation_Events.
- The step is repeated for all other equipment modules when necessary. In this example, there are only two equipment modules so that the Event_SummationEnd FB is called to clear out the arrays that are not used if the total number of events is less than the maximum event size of 100.



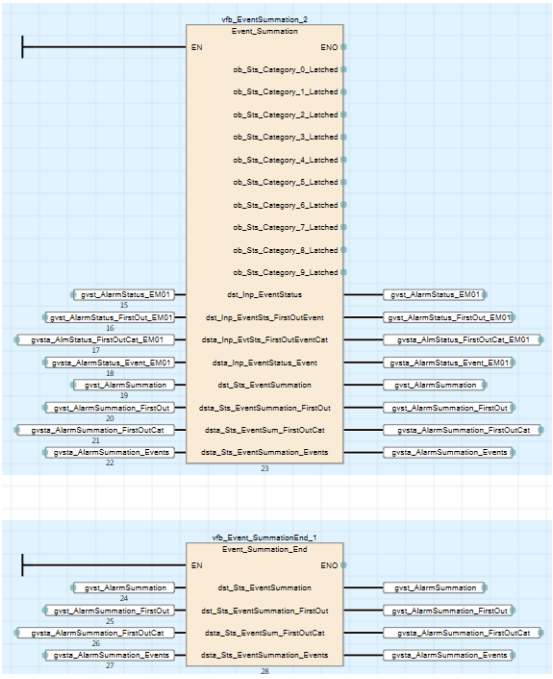


Figure 9 – Example – Event Summation of a Unit Machine

7 Function Block: Event_Sort

7.1 Description

The Event_Sort function block is used to filter and sort the event list of a unit machine for reporting or display purposes. It is capable of:

- Processing the Unit Machine Event List and producing a list of all active events;
- Processing the Unit Machine Event List and producing a list of all active events of a selected category of events;
- Processing the Unit Machine Event List and producing a list of all events (both active and non-active) of a selected event category;
- Sorting the Unit Machine Event list by event time, regardless of event category, from the earliest event to the latest event;
- Sorting the Unit Machine Event list by event time of a selected category, from the earliest event to the latest event.

The structure of the Event_Sort function block is shown in Figure 10 below:

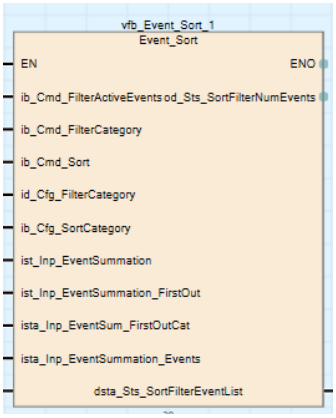


Figure 10 – Event_Sort Function Block

7.2 Function Block Local Variables

The Input, Output, and In_Out variables that are used by the FBs are described in this section. The details of Structured Data Types are defined in Section 4.3 and Section 6.3 of this document.

Variable Type	Variable Label	Data Type	Description
VAR_INPUT	ib_Cmd_FilterActiveEvents	Bit	Setting this bit will filter events that are currently active and document them in the Sorted List of the Unit Machine
VAR_INPUT	ib_Cmd_FilterCategory	Bit	Setting this bit will filter events in the Category as specified by Cfg_FilterCategory and document them in the Sorted List of the Unit Machine
VAR_INPUT	ib_Cmd_Sort	Bit	Setting this bit will sort the events in the Sorted List in chronological order, from the earliest event to the last event.
VAR_INPUT	id_Cfg_FilterCategory	Double Word[Signed]	This parameter specifies the Category of events that are filtered and displayed.
VAR_INPUT	ib_Cfg_SortCategory	Bit	Setting this bit together with the Cmd_Sort will not only sort the events in chronological order but also group them by category.
VAR_INPUT	ist_Inp_EventSummation	SDT_EventSummation	The structured variable contains the information of the summarized list of events
VAR_INPUT	ist_Inp_EventSummation_FirstOut	SDT_Event	The structured variable contains the first out event information of the summarized list of the Unit Machine
VAR_INPUT	ista_Inp_EventSummation_FirstOutCat	SDT_Event(0..9)	The array of structured variables contains the information of first out event per category of the summarized list of the Unit Machine.
VAR_INPUT	ista_Inp_EventSummation_Events	SDT_Event(0..99)	The array of events in the summarized list
VAR_IN_OUT	Sts_SortFilterEventList	SDT_Event(0..399)	The list of events after sorting
VAR_OUTPUT	od_Sts_SortFilterNumEvents	Double Word[Signed]	The number of events in the sorted list

7.3 Function Block Operations

An example of using the Event_Sort function block is shown in Figure 11 below.

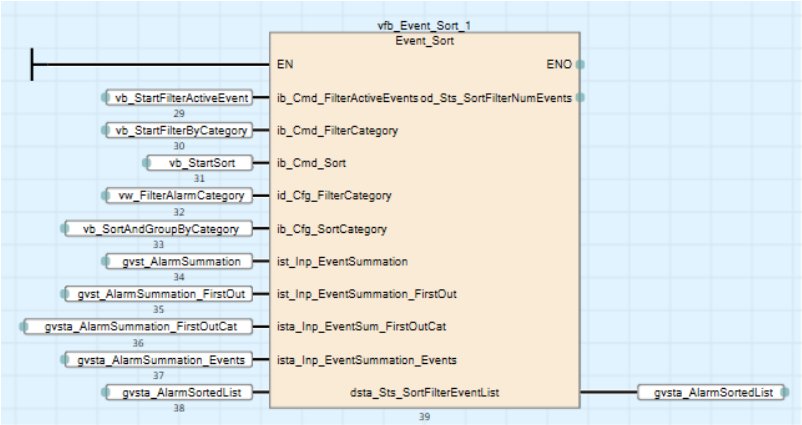


Figure 11 – Example of Event_Sort FB Operations

It is important to know the sequence of the command flags will have effects on how the filtering and sorting work. The table below summarizes the functions that are executed by setting various flags.

The operations of the commands are summarized in the following table:

StartFilterActiveEvent	StartFilterByCategory	StartSort	SortAndGroupByCategory	FilterAlarmCategory	Operations
0	0	0	X	X	The Sort List will be updated with the AlarmSummation_Events List
1	0	0	X	X	The events in the AlarmSummation_Events will be filtered and only active events will be recorded in the AlarmSortedList
0	1	0	X	n	The events in the AlarmSummation_Events will be filtered and events in Category n will be recorded in the AlarmSortedList, regardless whether the events are currently active or not.
1	1	0	X	n	The events in the AlarmSummation_Events will be filtered and only ACTIVE events in Category n will be recorded in the AlarmSortedList.
0	X	1	0	X	The events in the **AlarmSummation_Events** will be sorted in chronological order and recorded in the AlarmSortedList with the earliest event first.
0	X	1	1	X	The events in the **AlarmSummation_Events** will be sorted in chronological order grouped by category. The events will be recorded in the AlarmSortedList with the lowest category and earliest event first. **StartFilterByCategory** flag needs to be set before the StartSort command.
1	X	1	X	X	When StartFilterActiveEvent flag is on then the StartSortFlag is on, the events in the **AlarmSortedList** will be sorted in chronological order and recorded in the AlarmSortedList with the earliest event first. If the StartSortFlag is on first then StartFilterActiveEvent is on, the StartFilterActiveEvent flag will have no effect and the sorting will be done on the AlarmSummation_Events array instead of AlarmSortedList.
1	X	0	1	X	The events in the **AlarmSortedList** will be sorted in chronological order grouped by category. The events will be recorded in the AlarmSortedList with the lowest category and earliest event first. If the StartSort Flag is on first then StartFilterActiveEvent is on, the StartFilterActiveEvent flag will have no effect and the sorting will be done on the AlarmSummation_Events array instead of AlarmSortedList.

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 6 – OEM PackML Template Project and Implementation

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

This document describes the program structure of the PackML Implementation Template project, the functions and implementation details of each program, and how an OEM can tailor the template routines that are included in the template project to conform to the actual mechanical systems.

Release 4 of the Mitsubishi PackML OEM Implementation Templates also includes the function blocks that handle events of a unit machine.

This project structure is based on PackML Implementation Guidelines released by the OMAC Users Group and follows the ISA-88 Make2Pack modularization concept.

2 PackML Template System Hardware Architecture

The PackML templates are designed to run on a system with the minimum of a R08CPU and a GT27 HMI. The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a R08CPU and the GOT is a GT27 with the resolution of 800 x 600.

Because of the large number of tags required to support the PackTags specification, an extended memory card may be required to be installed in the R08CPU.

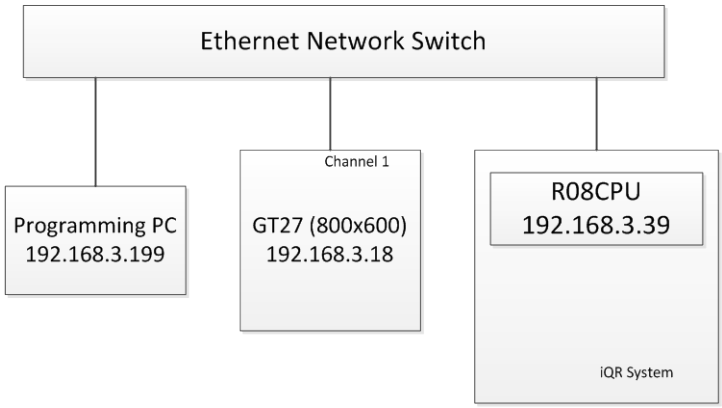


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT and the PLC CPU using the Ethernet connections to download screen information and PLC project.

The configurations of these components are described in more details in other parts of the Mitsubishi PackML Implementation Users Guide.

3 Mitsubishi PackML Template Project Structure Overview

The Mitsubishi PackML Template Project provides a pre-defined project structure that can be used by an OEM to implement the control programs for a machine. The project is structured with the hierarchy of Project -> Program Files - > Program Block or Program Organization Unit (POU).

Figure 2 below shows the overall organization of the PackML Template Project:

- The Project “PackML Template GXW3 R4 V2 FB” contains many Program Files.
    - Program File “MainInit” contains the POUs necessary to initialize the unit machine and all the equipment modules (designated as EM00 to Emxx) of the unit machine. The number of Equipment Module required for the Unit Machine depends on the actual mechanical design of the machine and the logical division of the machine.

- The Program File UnitMach contains all the necessary Program Blocks at the Unit Machine level. The PackML state and mode transitions occur at the Unit Machine level so that the PackML core function blocks are used only in the Unit_Machine level.
- Each Equipment Module of the Unit Machine has its own Program File and associated program blocks. The number of Program Files required depends on the number of the Equipment Module. Each Equipment Module can have as many Control Modules as necessary to perform the control functions of the Equipment Module.
    - The Mitsubishi PackML Template Project assigns each Equipment Module a main program block. The main program block will call the Event_Control function block (FB), and an Equipment Module PackML Command and Status Summation FB together with as many control module FBs as necessary to control the actual equipment.

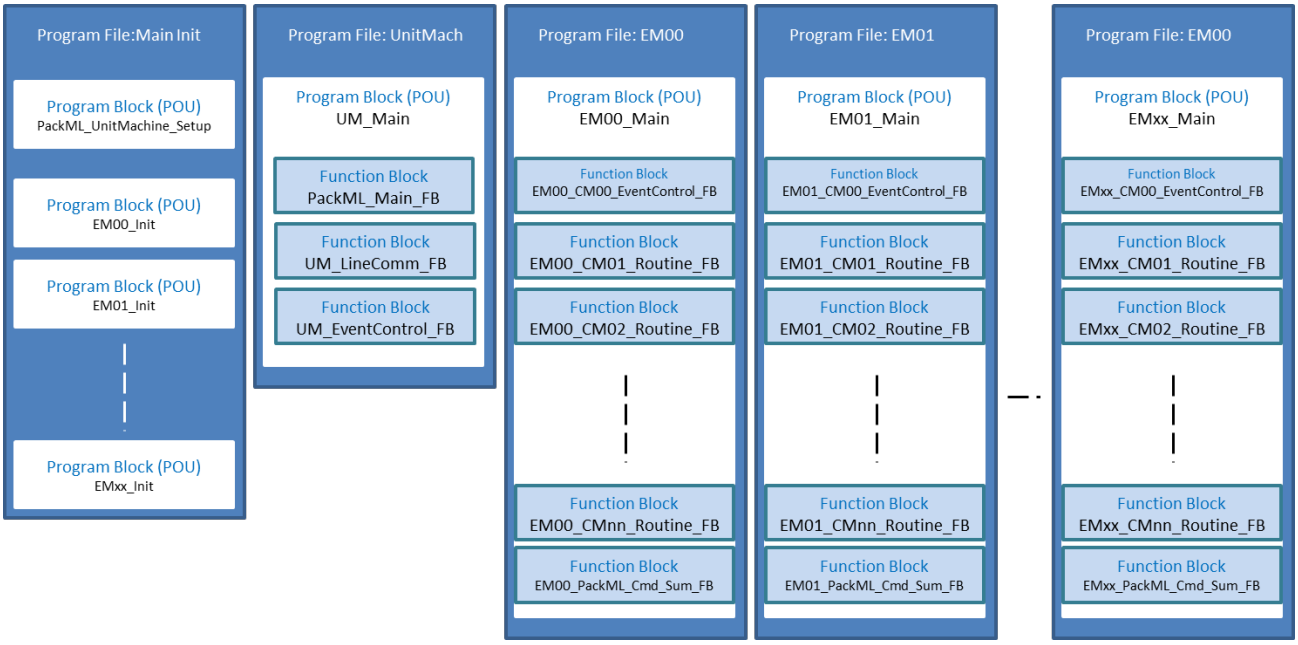


Figure 2 – The Overall Structure of the PackML Template Project

4 Mitsubishi PackML Template Project

The actual Mitsubishi PackML Template Project contains the structure to support a Unit Machine with two Equipment Modules (EM) and five Control Module (CM) Function Blocks in each EM (including one CM FB for event control and one for PackML command and status summation) within each Equipment Module.

The structure can be easily expanded to match the actual Unit Machine structure. For example, if the actual machine has three Equipment Modules instead of two, the OEM can duplicate the complete Program File EM00 and modify the all names (such as Program File Name, Task Name, Control Module Names, etc.) and labels referenced in the new equipment module from EM00 to EM02. An example is given in Section **Error! Reference source not found.** of this document.

Figure 3 shows the actual project structure in GX Works 3. After all Program Organization Units (POU) are created, they are registered in the proper program setting areas.

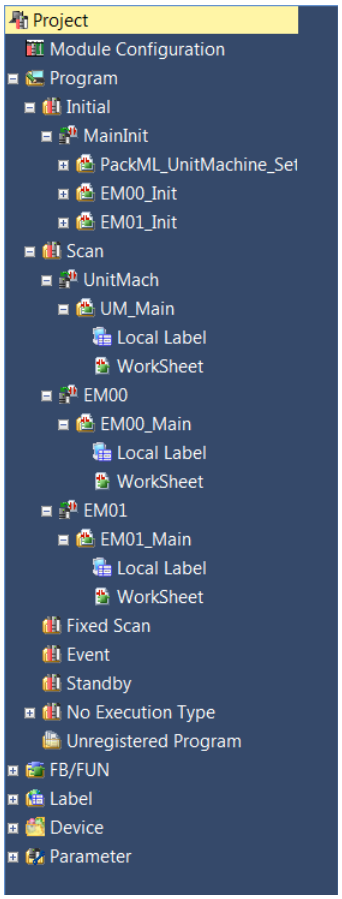


Figure 3 – Project Structure of PackML Implementation Template

4.1 Initial Program Type

Programs registered as Initial Program Type will only be executed during the first scan of the PLC after it is first powered up or reset.

The Initial Program type contains the Program File **MainInit** with the POU EM00_Init which has the actual program routine initializing the Equipment Module EM00 and local variables associated with the routine; POU EM01_Init which has the actual program routine initializing the Equipment Module EM01 and local variables associated with the routine; and POU PackML_UnitMachine_Setup which has the actual program routine initializing the Unit Machine with proper PackML modes and states and the local variables associated with the routines.

4.2 Scan Program Type

Most the program files are registered as Scan Program type that will be executed on every scan of the PLC. The PackML template contains Program files for a machine with two Equipment Modules.

- The UniMach Program File contains one POU: UM_Main. Function blocks PackML_Main_FB, UM_LineComm_FB, and UM_EventControl_FB are called within the UM_Main.
- The EM00 Program File contains one POU: EM00_Main. Function blocks EM00_CM00_EventControl_FB, EM00_CM01_Routine_FB, EM00_CM02_Routine_FB, EM00_CM03_Routine_FB, and EM00_PackML_Cmd_Sum_FB are executed within EM00_Main.

- The EM01 Program File contains one POU: EM01_Main. Function blocks EM01_CM00_EventControl_FB, EM01_CM01_Routine_FB, EM01_CM02_Routine_FB, EM01_CM03_Routine_FB, and EM01_PackML_Cmd_Sum_FB are executed within EM01_Main.

4.3 Other Program Types

The PackML Template project does not use the Standby, Fixed Scan, and No Execution Program Types.

5 PackML Global Labels

This section contains the detailed descriptions of the global labels used in the PackML Implementation Template project. There are five groups of Global Labels used in the PackML Template Project. The groups **PackTags_Adm, PackTags_Command,** and **PackTags_Status** are labels related to PackTags that are described in Part 3 of the Users Guide and will not be described in this document.

5.1 PackML_FB Group

This group contains global labels that are critical to the operation of PackML Core function blocks. The Structured Data Types PackMLFB is defined to support the core PackML FB operations.

Variable Label	Data Type	Description
gvba_PackML_cfgModeTransitions	Bit(0..31,0..17)	These are the configuration variables that OEM programs need to define at which states where mode transitions are allowed for each mode. For example, PackML_cfgModeTransitions[1, 2] = 1 means at Mode 1 (“Producing” mode) State 2 (“Stopped”) state, the machine is allow to switch mode. However, PackML_cfgModeTransitions[2, 2] should also be set to 1 to allow the mode change from Mode 1 to Mode 2. Otherwise, the mode change from 1 to 2 is not allowed.
gvba_PackML_cfgDisableStates	Bit(0..31,0..17)	These are the configuration variables that OEM programs need to define at which states are not enabled for each mode. For example, PackML_cfgModeTransitions[2, 5] = 1 means at Mode 2 (“Maintenance” mode) State 5 (“Suspended”) state is not configured as a part of the state model for Mode 2.
gvsa_PackML_ModeNames	String(32)(0..31)	These are the configuration variables that OEM programs need to configure all the names of the modes in the machine.
gvsa_PackML_StateNames	String(32)(0..17)	These are the configuration variables that OEM programs need to configure for all the names of the states in the machine.
gvb_TimeRollOverWarning	Bit	This bit is on when any of the Mode or State timers roll over its limit.
gvs_Sta_StateCurrentName	String(32)	This is a status variable where the name of the current state is stored.
gvs_Sta_UnitModeCurrentName	String(32)	This is a status variable where the name of the current mode is stored.
gvb_PackML_ResetAllTimes	Bit	This bit is used to reset all mode and state timers of the unit machine
gvb_PackML_ResetCurrentModeTimes	Bit	This bit is used to reset all mode timers of the unit machine
svst_PackML	PackMLFB	These are the labels of PackML commands and status used to interact with the ModeStateManager function block. The PackMLFB Structured Data Type is detailed below.

5.1.1. PackMLFB Structured Data Type

This structured data type contains key elements to support the operation of the PackML_ModeStateManager Function Block.

Variable Label	Data Type	Description
PackML	**PackMLFB**	The overall PackML label that contains various values and parameters for the machine states and modes

Variable Label	Data Type	Description
b_CmdMode	Double Word	When OEM machine programs require the machine to be in a certain mode, this label should be set to the desired value of the mode. For example, when PackML.CmdMode is set to "1" the machine is intended to be in the "Producing" mode. Per PackML specification, the Mode Numbers 1, 2 and 3 are defined as "Producing", "Maintenance" and "Manual" respectively. User Define Modes can be from Model 16 and above. Mode Numbers 4 through 15 are reserved for future use.
b_CmdReset	Bit	When OEM machine programs receive a "Reset" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Reset" command is no longer valid.
b_CmdStart	Bit	When OEM machine programs receive a "Start" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Start" command is no longer valid.
b_CmdStop	Bit	When OEM machine programs receive a "Stop" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Stop" command is no longer valid.
b_CmdHold	Bit	When OEM machine programs receive a "Hold" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Hold" command is no longer valid.
b_CmdUnhold	Bit	When OEM machine programs receive a "UnHold" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "UnHold" command is no longer valid.
b_CmdSuspend	Bit	When OEM machine programs receive a "Suspend" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Suspend" command is no longer valid.
b_CmdUnsuspend	Bit	When OEM machine programs receive a "UnSuspend" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "UnSuspend" command is no longer valid.
b_CmdAbort	Bit	When OEM machine programs receive a "Abort" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Abort" command is no longer valid.
b_CmdClear	Bit	When OEM machine programs receive a "Clear" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "Reset" command is no longer valid.
b_CmdStateComplete	Bit	When OEM machine programs receive a "State Complete" command, this bit should be set by the programs. It should be cleared by the OEM machine programs when the "State Complete" command is no longer valid.
b_cfg_RemoteCmdEnable	Bit	When OEM machine programs allow mode and state change commands to be issued remotely, this bit should be set
d_Inp_RemoteModeCmd	Double Word	This label contains the Remote Mode Command value and is the value of the new mode the machine should transition to. The valid values are 0 – 31.
b_Inp_RemoteModeCmdChangeRequest	Bit	When OEM machine programs request a remote mode change command, this bit should be set

Variable Label	Data Type	Description
d_Inp_RemoteStateCmd	Double Word	This label contains the Remote State Command value and is the value of the new state the machine should transition to. The valid State Command values are defined as follows and others are ignored: 1: Reset 2: Start 3: Stop 4: Hold 5: UnHold 6: Suspend 7: UnSuspend 8: Abort 9: Clear
b_Inp_RemoteStateCmdChangeRequest	Bit	When OEM machine programs request a remote state change command, this bit should be set.
b_StateClearing	Bit	This is a status bit. When it is set, the machine is in "Clearing" mode.
b_StateStopped	Bit	This is a status bit. When it is set, the machine is in "Stopped" mode.
b_StateStarting	Bit	This is a status bit. When it is set, the machine is in "Starting" mode.
b_StateIdle	Bit	This is a status bit. When it is set, the machine is in "Idle" mode.
b_StateSuspended	Bit	This is a status bit. When it is set, the machine is in "Suspended" mode.
b_StateExecute	Bit	This is a status bit. When it is set, the machine is in "Execute" mode.
b_StateStopping	Bit	This is a status bit. When it is set, the machine is in "Stopping" mode.
b_StateAborting	Bit	This is a status bit. When it is set, the machine is in "Aborting" mode.
b_StateAborted	Bit	This is a status bit. When it is set, the machine is in "Aborted" mode.
b_StateHolding	Bit	This is a status bit. When it is set, the machine is in "Holding" mode.
b_StateHeld	Bit	This is a status bit. When it is set, the machine is in "Held" mode.
b_StateUnHolding	Bit	This is a status bit. When it is set, the machine is in "UnHolding" mode.
b_StateSuspending	Bit	This is a status bit. When it is set, the machine is in "Suspending" mode.
b_StateUnSuspending	Bit	This is a status bit. When it is set, the machine is in "UnSuspending" mode.
b_StateResetting	Bit	This is a status bit. When it is set, the machine is in "Resetting" mode.
b_StateCompleting	Bit	This is a status bit. When it is set, the machine is in "Completing" mode.
b_StateComplete	Bit	This is a status bit. When it is set, the machine is in "Complete" mode.
b_ModeChangeNotAllowed	Bit	This is a status bit. When it is set, the mode change of the machine is not allowed.
d_Sts_StateCurrent	Double Word	This label shows the current state of the machine.
b_Sts_Modebits[0..31]	Bit	This array label shows the current bit of the machine mode. It can be used to as test conditions for machine programs. For example, when bit #2 of the Sts_ModeBits is set, the State Machine is in the Maintenance mode.
d_Sts_ModeCurrent	Double Word	This label shows the current mode of the machine. The values are as defined in the CmdMode label of this table.

5.2 OEM_Template_PackML_Labels

This group contains global labels that are used by Unit Machine and Equipment Modules to operate PackML states and commands. The Structure Data Type PackM_Module_Cmd is defined to support the operations.

Label	Data Type	Description
gvst_EM00_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Equipment Module EM00 (which is the aggregate of all the Control Modules within the Equipment Module)
gvst_EM01_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Equipment Module EM01 (which is the aggregate of all the Control Modules within the Equipment Module)
gvst_UN_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of the Unit Machine (which is the aggregate of all Equipment Modules)
gvst_EM00_CM00_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM00_CM00
gvst_EM00_CM01_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM00_CM01
gvst_EM00_CM02_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM00_CM02
gvst_EM00_CM03_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM00_CM03
gvst_EM01_CM00_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM01_CM00
gvst_EM01_CM01_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM01_CM01
gvst_EM01_CM02_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM01_CM02
gvst_EM01_CM03_PackML_Sts	PackML_Module_Cmd	The label indicating the PackML commands and status of Control Module EM01_CM03
gvb_RemoteCmd_ResetAllTimes	Bit	The command that comes from external to the machine to reset all the timers within the PackML State Machine of this Unit Machine.

5.2.1. PackML_Module_Cmd Structured Data Type

This structured data type is used by each Equipment or Control Module to issue PackML commands as well as reflects its PackML state status.

Label	Data Type	Description
b_Cmd_Reset	Bit	When the bit is set TRUE, the Control Module is issuing a “Reset” command to the State Machine.
b_Sts_Resetting_SC	Bit	When “Resetting” state operations are completed, this bit is should be set to send a “State Complete” command to the State Machine.
b_Cmd_Start	Bit	When the bit is set TRUE, the Control Module is issuing a “Start” command to the State Machine.
b_Sts_Starting_SC	Bit	When “Starting” state operations are completed, this bit is should be set to send a “State Complete” command to the State Machine.
b_Cmd_Stop	Bit	When the bit is set TRUE, the Control Module is issuing a “Stop” command to the State Machine.
b_Sts_Stopping_SC	Bit	When “Stopping” state operations are completed, this bit is should be set to send a “State Complete” command to the State Machine.
b_Cmd_Hold	Bit	When the bit is set TRUE, the Control Module is issuing a “Hold” command to the State Machine.
b_Sts_Holding_SC	Bit	When “Holding” state operations are completed, this bit is should be set to send a “State Complete” command to the State Machine.
b_Cmd_UnHold	Bit	When the bit is set TRUE, the Control Module is issuing an “Unhold” command to the State Machine.

Label	Data Type	Description
b_Sts_UnHolding_SC	Bit	When "UnHolding" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Cmd_Suspend	Bit	When the bit is set TRUE, the Control Module is issuing a "Suspend" command to the State Machine.
b_Sts_Suspending_SC	Bit	When "Suspending" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Cmd_UnSuspend	Bit	When the bit is set TRUE, the Control Module is issuing a "UnSuspend" command to the State Machine.
b_Sts_UnSuspending_SC	Bit	When "UnSuspending" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Cmd_Abort	Bit	When the bit is set TRUE, the Control Module is issuing an "Abort" command to the State Machine.
b_Sts_Aborting_SC	Bit	When "Aborting" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Cmd_Clear	Bit	When the bit is set TRUE, the Control Module is issuing a "Clear" command to the State Machine.
b_Sts_Clearing_SC	Bit	When "Clearing" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Sts_Execute_SC	Bit	When "Execute" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_Sts_Completing_SC	Bit	When "Completing" state operations are completed, this bit is should be set to send a "State Complete" command to the State Machine.
b_ONS	Bit	The internal flag bit that is used for an one-shot Function Block
b_ModuleActive	Bit	When this bit is set, it indicates that the control module is active

5.3 OEM_Template_PackML_GOT_Keys

This group of global labels is defined in the template to support the User Interface Screens that are parts of the PackML template. The user interface screens are implemented in the Mitsubishi GT-16 GOT hardware. These screens can be easily modified and used by the actual OEM machine control project. The descriptions of GOT screens and GT Designer projects are documented in Part 7 of the Users Guide.

The GOT interface programs of the PackML Template Project are implemented as Control Module CM01 of Equipment Module 00 as examples. The user can implement any operator interface routines in other control modules when appropriate.

Label	Data Type	Description
svb_GOT_ProdMode	Bit	Reflecting the status of "Produce Mode" key on the GOT
svb_GOT_MaintMode	Bit	Reflecting the status of "Maintenance Mode" key on the GOT
svb_GOT_ManualMode	Bit	Reflecting the status of "Manual Mode" key on the GOT
svb_GOT_User1Mode	Bit	Reflecting the status of "User Mode 1" key on the GOT
svb_GOT_User2Mode	Bit	Reflecting the status of "User Mode 2" key on the GOT
svb_GOT_ResetKey	Bit	Reflecting the status of "Reset Command" key on the GOT
svb_GOT_StartKey	Bit	Reflecting the status of "Start Command" key on the GOT
svb_GOT_HoldKey	Bit	Reflecting the status of "Hold Command" key on the GOT
svb_GOT_StopKey	Bit	Reflecting the status of "Stop Command" key on the GOT
svb_GOT_UnHoldKey	Bit	Reflecting the status of "UnHold Command" key on the GOT
svb_GOT_AbortKey	Bit	Reflecting the status of "Abort Command" key on the GOT

svb_GOT_ClearKey	Bit	Reflecting the status of "Clear Command" key on the GOT
svb_GOT_SuspendKey	Bit	Reflecting the status of "Suspend Command" key on the GOT
svb_GOT_UnSuspendKey	Bit	Reflecting the status of "UnSuspend Command" key on the GOT
svb_GOT_StateCompleteKey	Bit	Reflecting the status of "State Complete Command" key on the GOT
svb_GOT_ClearAllTimesKey	Bit	Reflecting the status of "Clear All Timers" key on the GOT
svb_GOT_ClearCurrModeTimeKey	Bit	Reflecting the status of "Clear Current Mode Timers" key on the GOT
svw_GOT_Screen_Switch	Word[Signed]	Label that is used to instruct the GOT which screen to display.
svda_GOT_CurrentStateTimes	Double Words[signed](0..31, 0..17)	Contain the current time (in seconds) in the current state of a particular mode
svda_GOT_CumulativeStateTimes	Double Words[signed](0..31, 0..17)	Contain the cumulated time (in seconds) of the state of a particular mode
svda_GOT_ModeCurrentTime	Double Words[signed](0..31)	Contain the current time (in seconds) in the current mode
svda_GOT_ModeCumulativeTime	Double Words[signed](0..31)	Contain the cumulated time (in seconds) of the mode
svd_GOT_AccTimeSinceReset	Double Words[signed]	Contain the cumulated time (in seconds) of the machine since the last reset timer command was issued.

5.4 OEM_Template_Event_Labels

This group of global labels is defined in the template to support the event handling functions. The descriptions of the Structured Data Types are documented in Part 5 of the Users Guide where Event Handling Function Blocks are described.

Label	Data Type	Description
gvsta_AlarmCfg	SDT_EventCfg(0..19)	This array is used to define the information of alarms in the system such as ID, Value, Alarm message, and Category.
gvst_ZeroEvent	SDT_Event	This structure is used to initialize any list of events.
gvst_AlarmStatus_EM00	SDT_EventStatus	This structure contains the event status for Equipment Module 00
gvsta_AlarmStatus_Event_EM00	SDT_Event(0..29)	This event list contains the events of Equipment Module 00. The array size is pre-defined to 30.
gvst_AlarmStatus_FirstOut_EM00	SDT_Event	This structure is used to hold the First Out Event of Equipment Module 00
gvsta_AlarmStatus_FirstOutCat_EM00	SDT_Event(0..9)	This array of structures is used to hold the First Out Event of each event category of Equipment Module 00
gvst_AlarmStatus_EM01	SDT_EventStatus	This structure contains the event status for Equipment Module 01
gvsta_AlarmStatus_Event_EM01	SDT_Event(0..29)	This event list contains the events of Equipment Module 01. The array size is pre-defined to 30.
gvst_AlarmStatus_FirstOut_EM01	SDT_Event	This structure is used to hold the First Out Event of Equipment Module 01
gvsta_AlarmStatus_FirstOutCat_EM01	SDT_Event(0..9)	This array of structures is used to hold the First Out Event of each event category of Equipment Module 01
gvst_AlarmSummation	SDT_EventSummation	This structure contains the Event Summation status of the Unit Machine
gvst_AlarmSummation_FirstOut	SDT_Event	This structure is used to hold the First Out Event of the Unit Machine
gvsta_AlarmSummation_FirstOutCat	SDT_Event(0..9)	This array of structures is used to hold the First Out Event of each event category of the Unit Machine
gvsta_AlarmSummation_Events	SDT_Event(0..99)	This event list contains the events of Unit Machine. The array size is pre-defined to 100.

Label	Data Type	Description
gvsta_AlarmSortedList	SDT_Event(0..399)	This event list is used to sort the events of the Unit Machine.

5.5 OEM_Template_Event_GOT_Keys

This group of global labels is defined in the template to support the Event Handling Test Screen that is a part of the PackML template. The Event Handling Test Screen is implemented in the Mitsubishi GT-16 GOT hardware. The screen can be easily modified and used by the actual OEM machine control project. The descriptions of GOT screens and GT Designer projects are documented in Part 7 of the Users Guide.

The Event Handling Test Screen programs of the PackML Template Project are implemented as Control Module CM02 of Equipment Module 00 and Control Module CM02 of Equipment Module 01 as examples. The user can implement any operator interface routines in other control modules when appropriate.

Label	Data Type	Description
svb_GOT_GuardDoorOpenKey1	Bit	Bit to set or clear “Guard Door Open” error for EM01
svb_GOT_AbortKey1	Bit	Bit to set or clear “Abort” error for EM01
svb_GOT_LowMaterialKey1	Bit	Bit to set or clear “Low Materials” error for EM01
svb_GOT_StopKey1	Bit	Bit to set or clear “Stop” error for EM01
svb_GOT_RemoteStopKey1	Bit	Bit to set or clear “Remote Stop” error for EM01
svb_GOT_ESTOPKey1	Bit	Bit to set or clear “E-Stop” error for EM01
svb_GOT_MGFaultKey1	Bit	Bit to set or clear “Motion Group Fault” error for EM01
svb_GOT_GuardDoorOpenKey	Bit	Bit to set or clear “Guard Door Open” error for EM00
svb_GOT_EventAbortKey	Bit	Bit to set or clear “Abort” error for EM00
svb_GOT_LowMaterialKey	Bit	Bit to set or clear “Low Materials” error for EM00
svb_GOT_EventStopKey	Bit	Bit to set or clear “Stop” error for EM00
svb_GOT_RemoteStopKey	Bit	Bit to set or clear “Remote Stop” error for EM00
svb_GOT_ESTOPKey	Bit	Bit to set or clear “E-Stop” error for EM00
svb_GOT_MGFaultKey	Bit	Bit to set or clear “Motion Group Fault” error for EM00

6 PackML Template Project Program Organization Units

6.1 Initialization POU

6.1.1. Equipment Module PackML Initialization POUs

EM00_Init and EM01_Init are two POUs that perform the initialization of PackML related labels only during the first scan of the PLC. It is important to note that an OEM using this PackML Template Project will need to add the necessary operation-related initialization code for each equipment module.

The EM00_Init and EM01_Init functions are identical except that labels corresponding to each equipment module are initialized in their respective POU. Figure 4 shows the Structured Ladder code. The label with the PackML_Module_Cmd Structured Data Type of each Control Module in the Equipment Module is input to the Function Block PackML_Cmd_Sts_Init and all PackML commands and status are initialized to the default values.

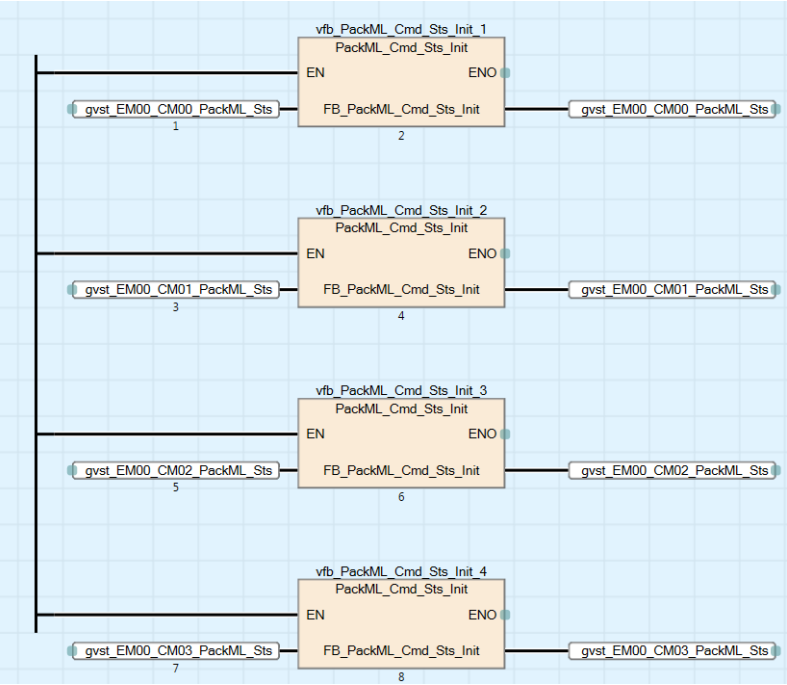


Figure 4 – Equipment Module PackML Initialization

Figure 5 below shows the actual code of the PackML_Cmd_Sts_Init function block. The initialization routine clears all PackML commands and set the State Complete status for states that require StateComplete flags to transition.

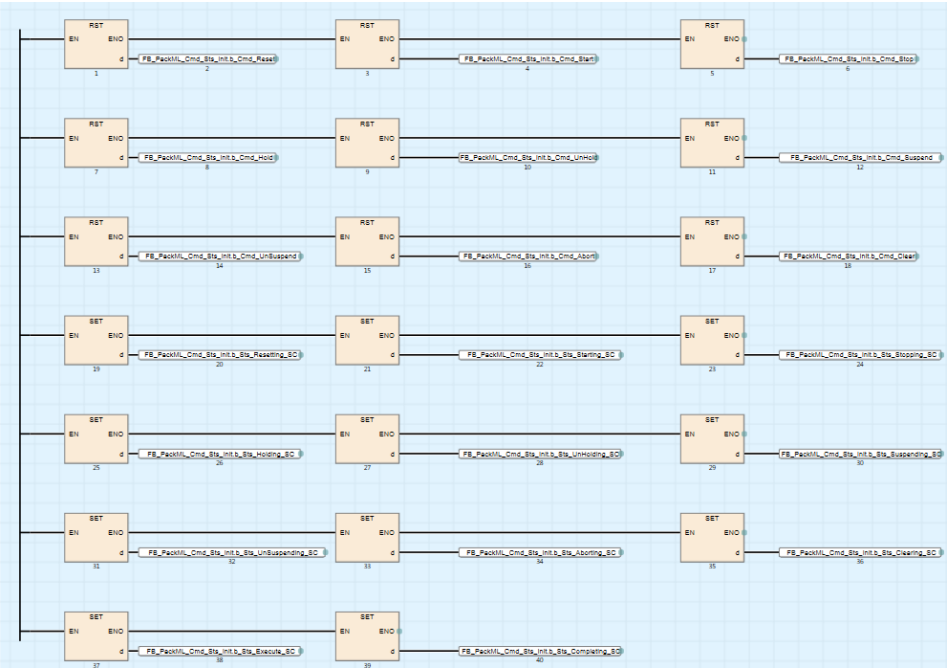


Figure 5 Function Block: PackML_Cmd_Sts_Init

6.1.2. Unit Machine PackML Initialization POUs

PackML_UnitMachine_SetUp is used to configure PackML modes and states for the Unit Machine and set up initial alarm configurations and zero event parameters.

The structured ladder programs of this POU included in the PackML Template Project are used to configure the modes and states for this template system only. This POU needs to be modified by the OEM to properly configure his Unit Machine to represent the actual modes and states of the machine.

The functions of this POU include:

- Configuring names of all modes available in the Unit Machine,
    - i.e. populating global variable PackML_ModeNames[0..31]
- Configuring names of all states available in the Unit Machine,
    - i.e. populating global variable PackML_StateNames[0..31]
- Defining all states within each mode that mode transitions are allowed,
    - i.e. defining global variable PackML_cfgModeTransitions[0..31, 0..17]
- Defining all states that are not required in each mode,
    - i.e. defining global variable PackML_cfgDisableStates[0..31, 0..17]
- Defining the initial mode and state of the Unit Machine,
    - In the Template System, the Unit Machine is set to “Manual Mode” and “Stopped State”
- Selecting the GOT screen to be displayed based on the mode of the Unit Machine,
- Configuring event information for all events in the Unit Machine,
- Configuring event handling parameters for proper operation.

6.2 Unit Machine Level POU and FBs

It is recommended that a user should refer to the actual FBD code in the Mitsubishi PackML Implementation Template project to get a better understanding of the functions of the POU and the FBs.

6.2.1. UM_Main

The purpose of this POU is to control the flow of the other routines at the Unit Machine level. It contains all the calls to all other unit machine level POUs.

Since the Event Summation and Sorting functions do not have to be executed on every scan to save overall machine scan time, a timer is programmed to call the UM_EventControl subroutine every second. A user can modify this time period to make the system operate appropriately per application requirements.

6.2.2. PackML_Main_FB

This FB operates the PackML state machine and its functions include:

- Aggregating PackML commands and status of all equipment modules,
    - If there are more than two equipment modules in the Unit Machine, this part of the template routine needs to be modified to include commands and status from the additional equipment modules
- Setting proper PackML command or status based on the aggregated results,
- Calling the PackML_ModeStateManager function block to set the PackML state machine in the correct mode and state and then clearing the command and status
- Calling the PackML_ModeStateTimes function block to accumulate the time in the current mode and state.

6.2.3. UM_LineComm_FB

The purpose of this POU is to handle PackML commands external to the Unit Machine. The commands may come from a Unit Machine downstream or Upstream. The current PackML Template project does not include the code to handle remote command communication.

It is assumed that an external system uses the PackTags Cmd_UnitMode and Cmd_CntrlCmd to configure the desired mode and command the Unit Machine needs to handle respectively. Then the remote machine can enable the PackTags Cmd_UnitModeChangeRequest and Cmd_CmdChangeRequest to initiate the remote operations.

The code in this POU is simply to take the remote commands and enable the proper labels within the template project and the PackML_Main POU will function accordingly.

6.2.4. UM_EventControl_FB

The purpose of this POU is to aggregate events and alarms from all equipment modules and handle them at the Unit Machine level.

In the PackML Template, the events from Equipment Modules 00 and 01 are summed into an overall list AlarmSummation at the Unit Machine level. The Event_Sort FB can then be used to perform filtering and sorting functions on the list. The details of filtering and sorting functions are documented in Part 5 of the Users Guide.

6.3 Equipment Module Level POU and FBs

Programs of each Equipment Module are grouped in the Program File EMxx. In the Template Project, each equipment module contains the main POU, an event control FB, three control module FBs, and a PackML Command Summation FB. For an actual implementation, the OEM can add or subtract the control module FBs as appropriate.

6.3.1. EMxx_Main

The purpose of this POU is to control the flow of the other routines at the Equipment Module level. It contains all the calls to all other equipment module level FBs.

6.3.2. EMxx_CMnn_Routine_FB

This POU contains the logic for Control Module nn of this particular equipment module xx. The OEM should incorporate the appropriate control logic in this POU to perform the actual control functions. For an actual implementation, the OEM can add or subtract control module POU's as appropriate. The names of these POU's can also be modified to better reflect the actual control module functions, for example, instead of EM00_CM01_Routine, the POU can be named as Filling_Station_HMI_Interface.

In the Mitsubishi PackML Implementation Template project, EM00_CM01_Routine_FB contains the GOT interface routine for PackML state and mode transitions. It takes the key pressed on the GOT and set or reset the proper flags to drive the PackML state machine operations. EM00_CM02_Routine_FB and EM01_CM02_Routine_FB contain the GOT interface routine for simulating events in the Unit Machine through GOT key presses.

6.3.3. EMxx_EventControl_FB

The purpose of this FB is to use the Event_Manager FB to aggregate events from all control modules into the equipment module level. It also contains the logic to issue PackML Stop or Abort command depending on which category of events have occurred.

This EventControl is referred to as Control Module 00 of the equipment module.

6.3.4. EMxx_PackML_Cmd_Sum

The purpose of this POU is to aggregate all PackML related commands and status from all control modules of this particular equipment module.

The consolidated command or status will then be used in the PackML_Main POU to set the command and status at the Unit Machine level.

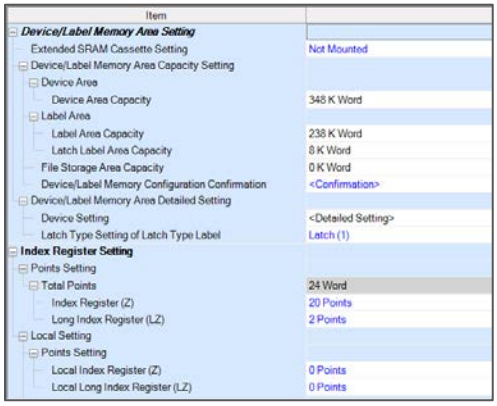
7 PLC CPU Parameters and Settings

This section contains some of the key PLC CPU parameter settings for the PackML Implementation Template project.

7.1 Memory/Device Setting

Because of the large number of PackTags and labels that are required to support the PackML and Event Handling functions, it is important to allocate sufficient Device Area Capacity and label Area Capacity properly.

The capacity of the Device Area is set at 348K words but can be adjusted according to the application needs. This is necessary to allocate PagTags to direct devices addresses so that they can be read or written by the OPC server. The Kepware OPC server does not support direct Label Access capability at this time.

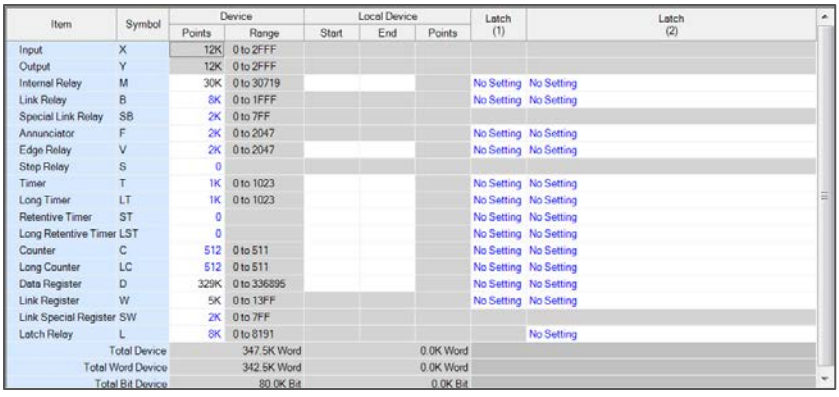


Item	
Device/Label Memory Area Setting	
Extended SRAM Cassette Setting	Not Mounted
Device/Label Memory Area Capacity Setting	
Device Area	
Device Area Capacity	348 K Word
Label Area	
Label Area Capacity	238 K Word
Latch Label Area Capacity	8 K Word
File Storage Area Capacity	0 K Word
Device/Label Memory Configuration Confirmation	<Confirmation>
Device/Label Memory Area Detailed Setting	
Device Setting	<Detailed Setting>
Latch Type Setting of Latch Type Label	Latch (1)
Index Register Setting	
Points Setting	
Total Points	24 Word
Index Register (Z)	20 Points
Long Index Register (LZ)	2 Points
Local Setting	
Points Setting	
Local Index Register (Z)	0 Points
Local Long Index Register (LZ)	0 Points

Figure 6 – Device / Label Area Setting

7.2 Device Settings

The key settings on this screen are the allocated size for D registers (329K points) to allocation pre-defined PackTags in D Registers.



Item	Symbol	Device		Local Device			Latch (1)	Latch (2)
		Points	Range	Start	End	Points		
Input	X	12K	0 to 2FFF					
Output	Y	12K	0 to 2FFF					
Internal Relay	M	30K	0 to 30719				No Setting	No Setting
Link Relay	B	8K	0 to 1FFF				No Setting	No Setting
Special Link Relay	SB	2K	0 to 7FF					
Annunciator	F	2K	0 to 2047				No Setting	No Setting
Edge Relay	V	2K	0 to 2047				No Setting	No Setting
Step Relay	S	0						
Timer	T	1K	0 to 1023				No Setting	No Setting
Long Timer	LT	1K	0 to 1023				No Setting	No Setting
Retentive Timer	ST	0					No Setting	No Setting
Long Retentive Timer	LST	0					No Setting	No Setting
Counter	C	512	0 to 511				No Setting	No Setting
Long Counter	LC	512	0 to 511				No Setting	No Setting
Data Register	D	329K	0 to 336895				No Setting	No Setting
Link Register	W	5K	0 to 13FF				No Setting	No Setting
Link Special Register	SW	2K	0 to 7FF					
Latch Relay	L	8K	0 to 8191					No Setting
Total Device			347.5K Word			0.0K Word		
Total Word Device			342.5K Word			0.0K Word		
Total Bit Device			80.0K Bit			0.0K Bit		

Figure 7 – PLC Parameters, Device Settings

7.3 Built In Ethernet Port Setting

The Built-In Ethernet port is configured so that it can be used with the Kepware OPC server to send PackTags data to external systems. It is also used to communicate with the GOT in the system.

Port 20482 using TCP protocol is configured to communicate with the Kepware OPC server. Port 20481 using UDP protocol is configured to communicate with the GOT.

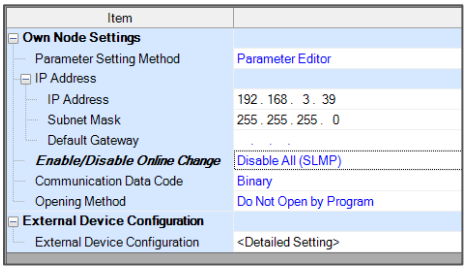


Figure 8 – PLC Parameters, Built-In Ethernet Port Settings

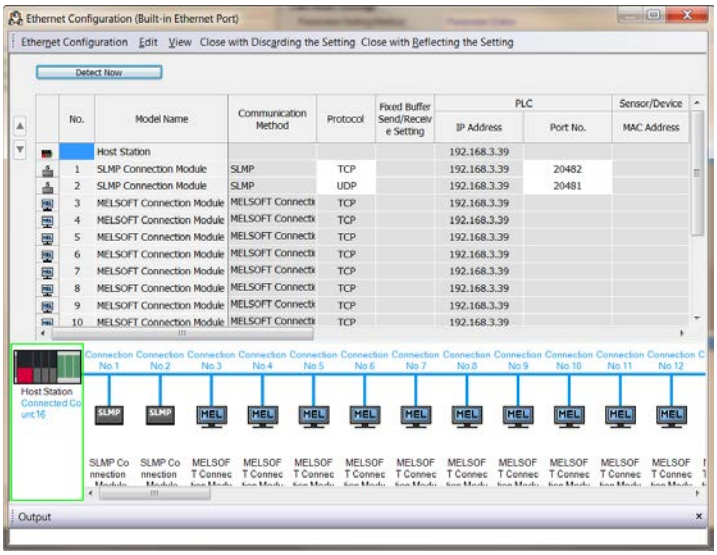


Figure 9 – PLC Parameters, Built-In Ethernet Port External Settings

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 7 – GOT Screens

Release 4, Version 1.0

Content

Revision History

Version	Revision Date	Description
R4 V1.0	January 31, 2016	Initial release of PackML OEM Implementation Templates Release 4

1 Introduction

This document describes the example screens that are used with the Mitsubishi PackML Implementation Template project. Many of the screens and screen elements can be used by OEMs on actual operator screens for the machine. The GT Designer 3 project for the example screens is a part of the Mitsubishi PackML Implementation Template package.

The use of iQ Works system labels is described in Part 2 of the Mitsubishi PackML Implementation Template Users Guide.

2 PackML Template System Architecture

The PackML templates are designed to run on a system with the minimum of a R08CPU and a GT27 HMI. The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a R08CPU and the GOT is a GT27 with the resolution of 800 x 600.

Because of the large number of tags required to support the PackTags specification, an extended memory card may be required to be installed in the R08CPU.

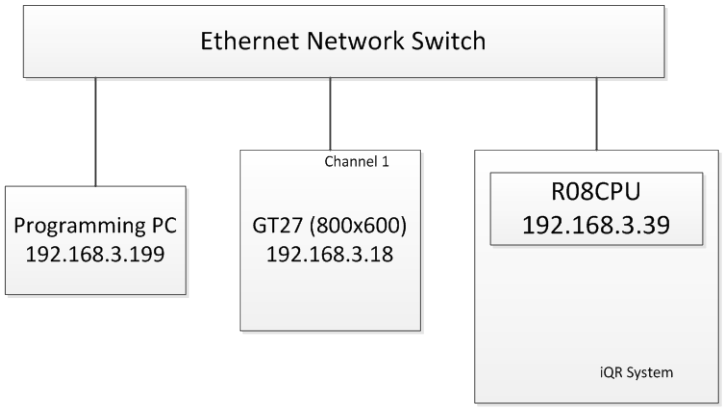


Figure 1 – Mitsubishi PackML GXW3 Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT and the PLC CPU using the Ethernet connections to download screen information and PLC project.

The configurations of these components are described in more details in other parts of the Mitsubishi PackML Implementation Users Guide.

3 GOT Communication Channel Configuration

The GOT in the Template system uses the Ethernet port to communicate with the programming laptop and an Ethernet channel to communicate with the PLC.

When using iQ Works to define the system architecture, the communication channel between GOT and the PLC should have already been set up.

In Figure 2 below, all parameters shown with the green background are defined in iQ Works and transferred over to the GT Designer 3.

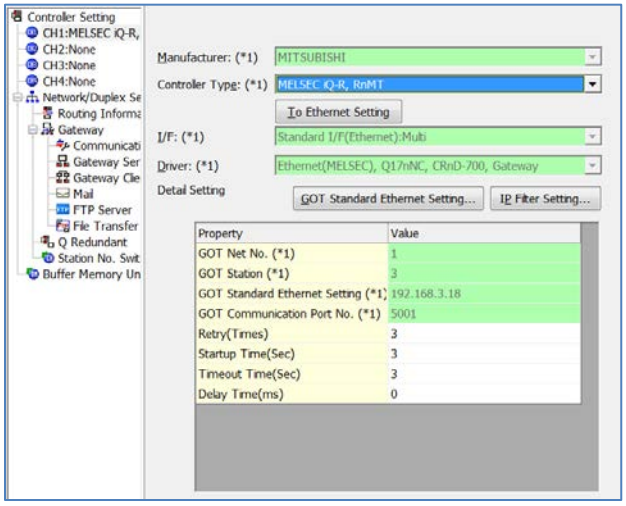


Figure 2 – Communication Channel 1 Configuration

Set the channel 1 as the Host to the PLC.

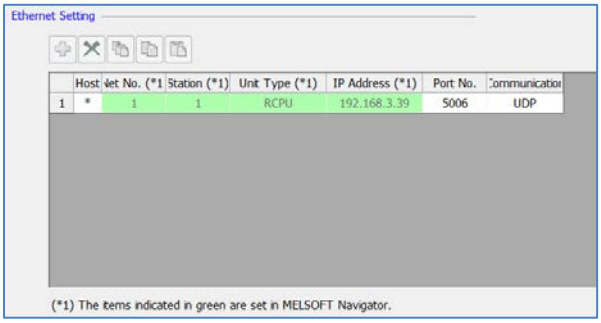


Figure 3 – Setting Channel 1 as Host

And then select the Communication Setting to verify all parameters.

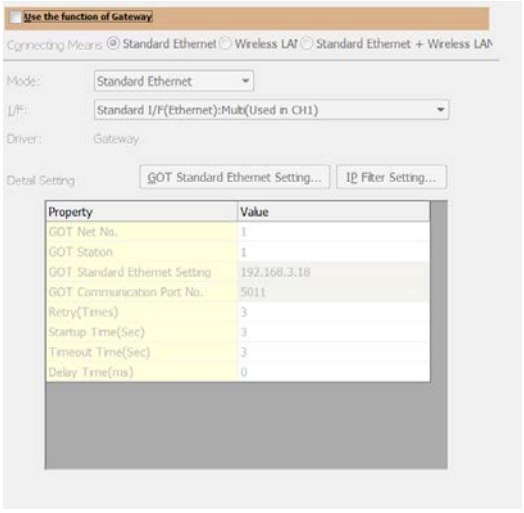


Figure 4 – Verifying Channel 1 Communication Settings

One should ensure the configurations are correct. If for whatever reasons the parameters do not match with the actual system configuration, one can select Tools -> Options -> iQ Works Interaction tab as shown in Figure 5 and check the box to

enable editing of parameters set in MELSOFT Navigator. However, the best practice is to make the necessary changes in the Navigator and "reflect" the parameters using the methods described in Part 2 of the Users Guide.

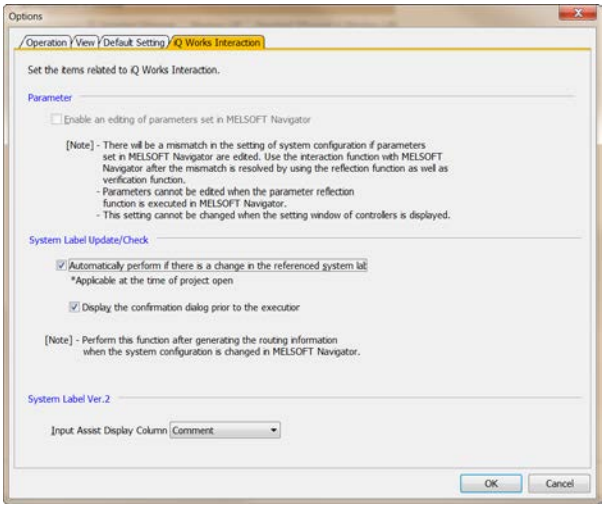


Figure 5 – Option to Modify Parameters Set in iQ Works

4 Sample Screens

Six sample screens, consisting of five PackML Mode Screens and one Event Test Screen, are included in the Template project.

Each screen of PackML Mode Screens displays the state machine of the mode and the accumulated and current time values for the mode and states. Keys are provided for the user to change modes, reset timers, and issue PackML commands. The state machine will display the transition of the states and highlight the state the state machine is in. The Event Test Screen has total of 14 keys that generate simulated events in the Template system. The details of this screen are described in this section.

Elements on these screens can be copied and used on other screens created by OEMs.

4.1 PackML Mode Screens

The PackML Mode Screens are used to demonstrate the PackML state and mode transition functions and display timer values PackML states and modes.

The functions of these screens are documented below:

- This screen of a particular mode displays the state diagram of the mode and the active state is highlighted and shown in Current Machine State display box.
- The Current Machine Mode is shown in the "Current Machine Mode" display box
- The Mode keys at the bottom of the screen allows the Unit Machine to change Mode and the screen of the new mode will be displayed. If the Unit Machine is at a state that mode change is not allowed, the "Mode Change Not Allowed" lamp will be lit.
- All timer values valid for the particular mode are displayed. Reset Current Mode Times and Rest All Times keys will reset the proper timers accordingly.
- The PackML Command Keys simulate commands to the State Machine and will cause state transition

4.1.1. Producing Mode Screen

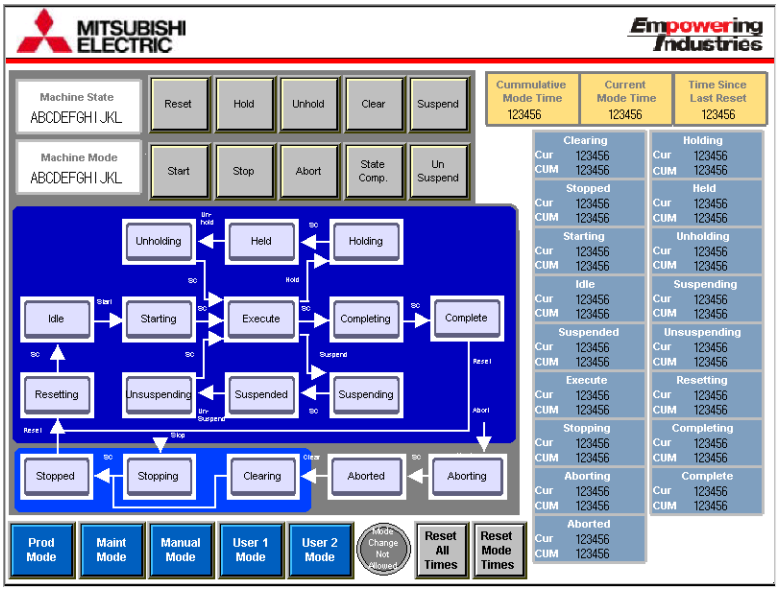


Figure 6 – Producing Mode Screen

4.1.2. Maintenance Mode Screen

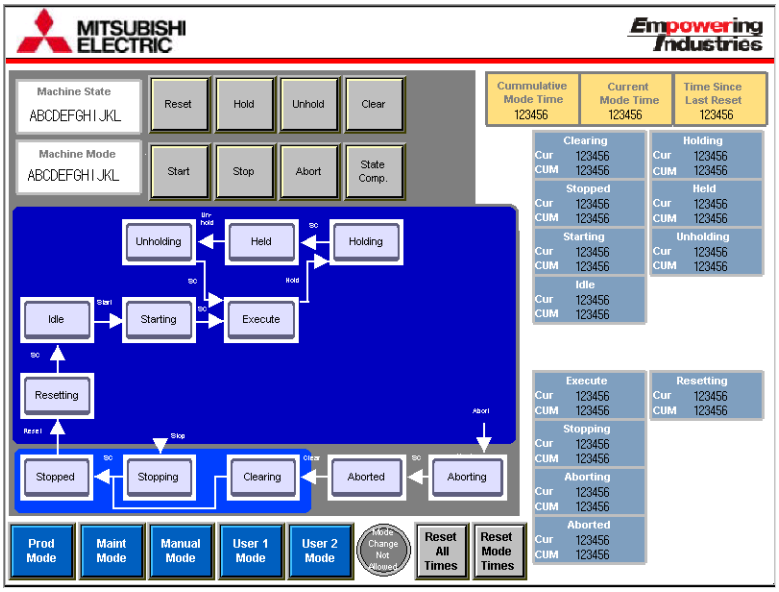


Figure 7 – Maintenance Mode Screen

4.1.3. Manual Mode Screen

The Manual Mode Screen has an additional key “Go To Event Test Screen” which allows the Event Simulation screen to be displayed and events being generated.

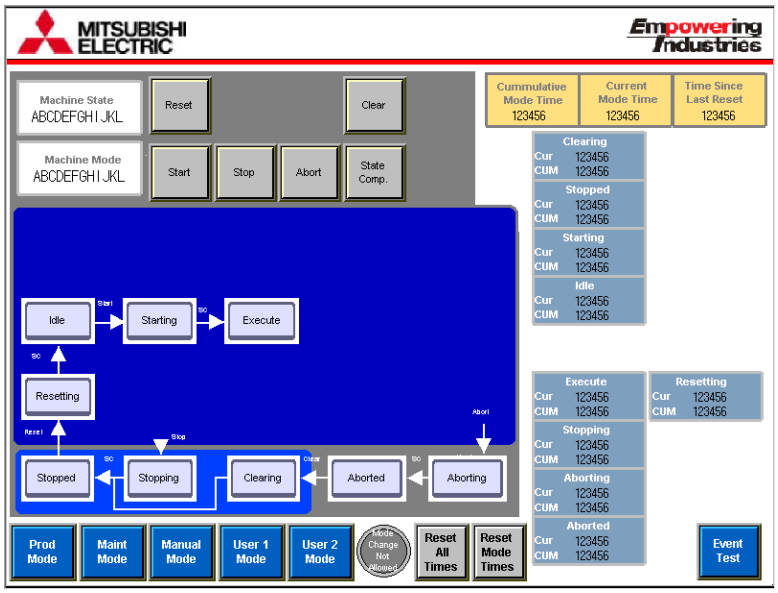


Figure 8 – Manual Mode Screen

4.1.4. User Defined Mode 1 / User Defined Mode 2 Screen

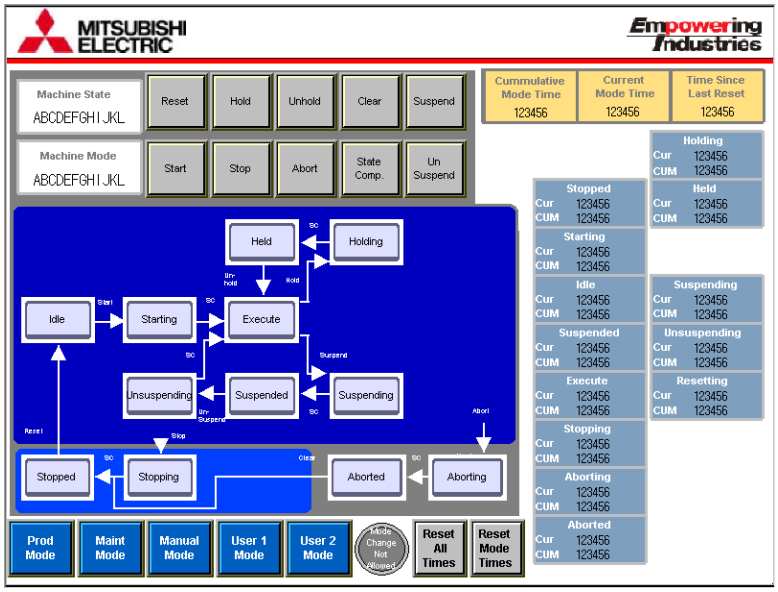


Figure 9 – User Defined Mode Screen

4.2 Event Test Screen

The purpose of Event Test Screen is to allow a user to simulate an event being generated and cleared in the Unit Machine. When a key is pressed, an event is created and the event will remain active and the key will be lit. When the key is pressed again, the event will be cleared and the light will be turned off.

The keys in the group EM00 will generate and clear events associated in Equipment Module 00. The logic to handle these key presses is in EM00_CM02_Routines. Similarly, the keys in the group EM01 will generate and clear events associated in Equipment Module 01. The logic to handle these key presses is in EM01_CM02_Routines.

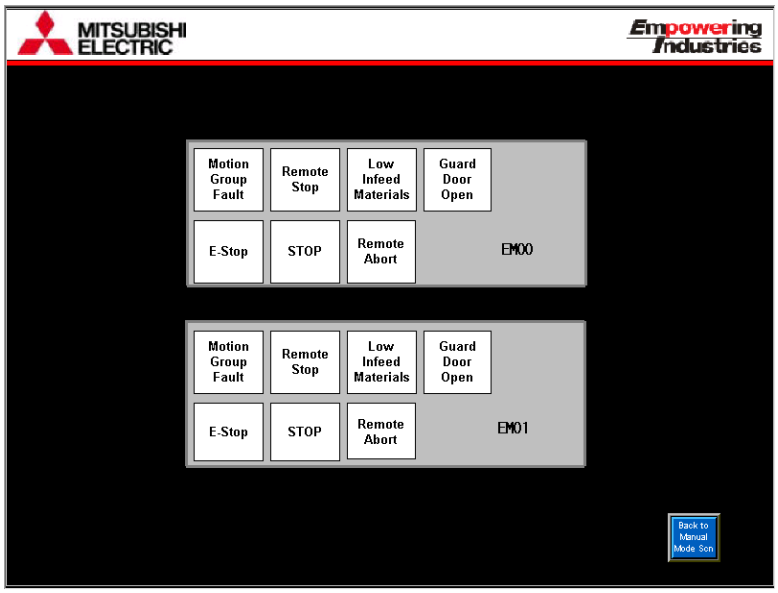


Figure 10 – Event Test Screen