MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Release 3, Version 5

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 1 - Overview

Release 3, Version 5.0

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5.0	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 Version 5

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 QnUDEH PLC and a GOT-16 HMI. The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a Q06UDEH CPU and the GOT is a GT-16s 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 Q06UDEHCPU in order to store the Symbolic Information.

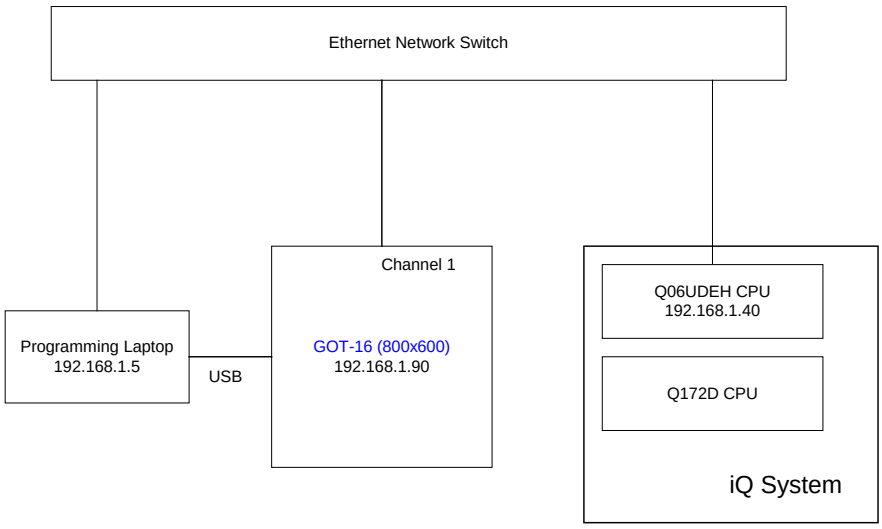


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT using the USB port to download screen information and to the Q06UDEH CPU through the GOT Ethernet Transparent Mode. The Transparent

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

Mode is set up to go through the Ethernet port on the Q06UDEHCPU (with IP Address 192.168.1.40). During the system operation, GOT is configured to work with the iQ PLC through the same Q06UDEHCPU Built-in Ethernet port.

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

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 2 and provided as integral parts of the PackML Template GX Works 2 program in the iQ Works Workspace.

The key PackML function blocks and labels are packaged as a "Library" to allow users to easily import function blocks and global labels associated with PackML.

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 2.

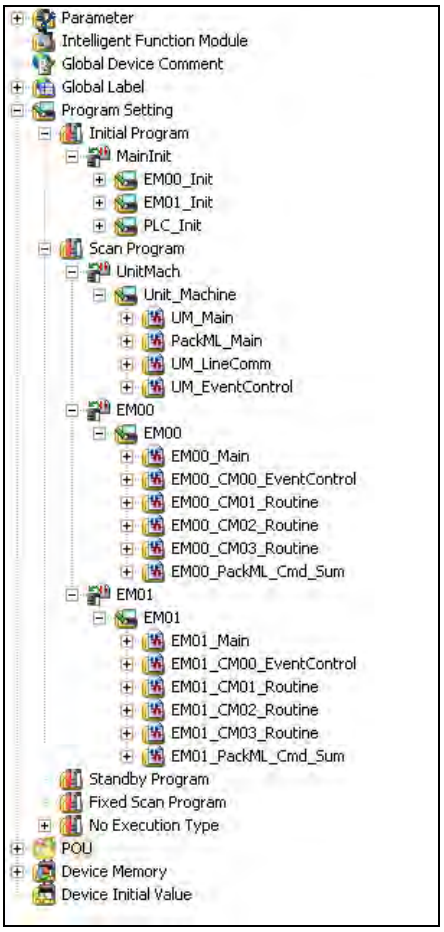


Figure 2 – Mitsubishi PackML Template Program Structure

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 2 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 PackML_R3_V1 Library

The PackML core function blocks and PackTag labels are organized into a GX Works 2 User Library to allow users to easily integrate Mitsubishi PackML functionality to new and existing programs. The template already contains the library.

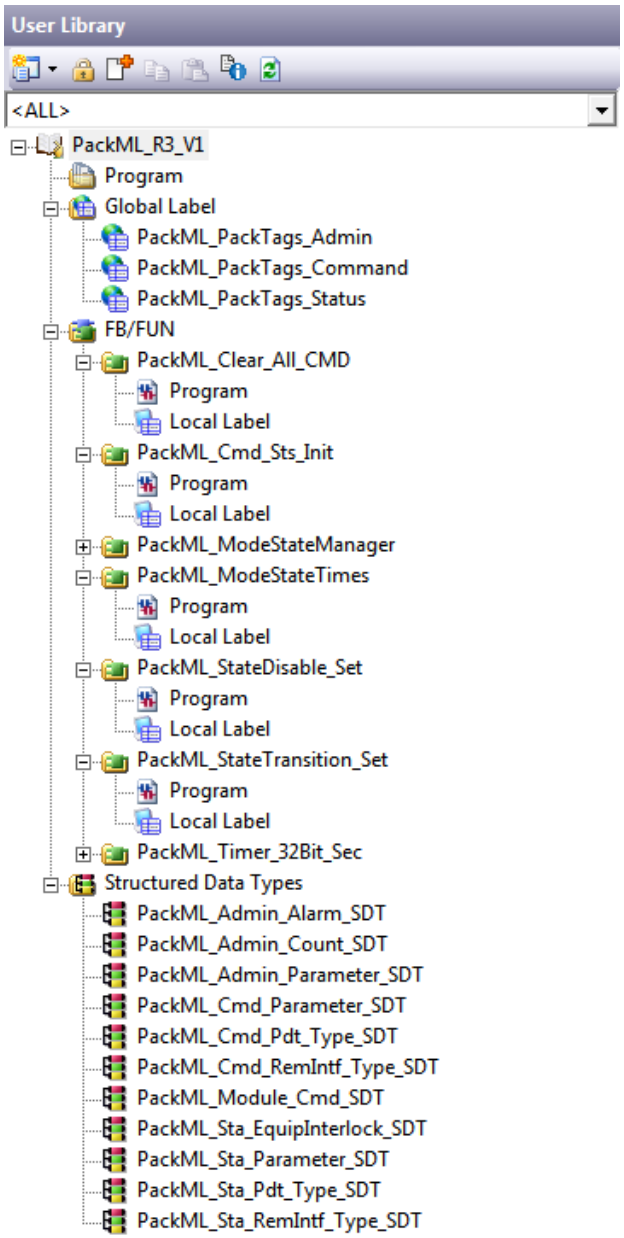


Figure 3 – Mitsubishi PackML Library Structure

6 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 2.
7. Develop the GOT and motion control programs using iQ Works and GT Designer 3 and MT Developer 2 respectively
8. Load programs in PLC, motion control and GOT.

7 Minimum Set PackML Implementation Template

The ISA-88 definition of PackML defines a "minimum" set of PackTags and PackML states that are required for a machine to be considered an appropriate PackML implementation. A Mitsubishi Minimum Set PackML Template has been developed to satisfy the OMAC PackML standard using minimum set of tags and states. The minimum set template is ideal for simple machines that do not require all the PackTags and states that are defined in the full PackML template implementation. This enables the OEMs to use smaller and more cost effective PLCs for their PackML compliant packaging systems.

8 Recommended CPU size for PackML Implementation

The recommended Q series CPU model for implementing the PackML solution will depend on the size of each system. As a general guideline, the recommended CPU model for the full PackML implementation template is Q06UD(E) CPU. For the minimum implementation, smaller Q03UD(E) may be sufficient.

In either of these cases, the "Symbolic File Information" of the program (which includes FB, label and program data) must be downloaded into the "Standard ROM" location in the PLC CPU. This procedure is explained in Part 6 section 8 (Program Structure document).

9 Parts of the PackML Implementation Users Guide

The Mitsubishi PackML Implementation Users Guide consists of 8 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 Library and 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.
Part 8 – Minimum Set Template	Description Mitsubishi Minimum SetPackML Template package and program structure

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 2 - MELSOFT Navigator Configuration

Release 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5.0	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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 Q06UDEHCPU, Q172DCPU Motion Controller and a GOT-16 HMI. The system architecture used to create the Mitsubishi PackML templates is shown in the following block diagram. The PLC is a Q06UDEHCPU and the GOT is a GT-16s with the resolution of 800 x 600.

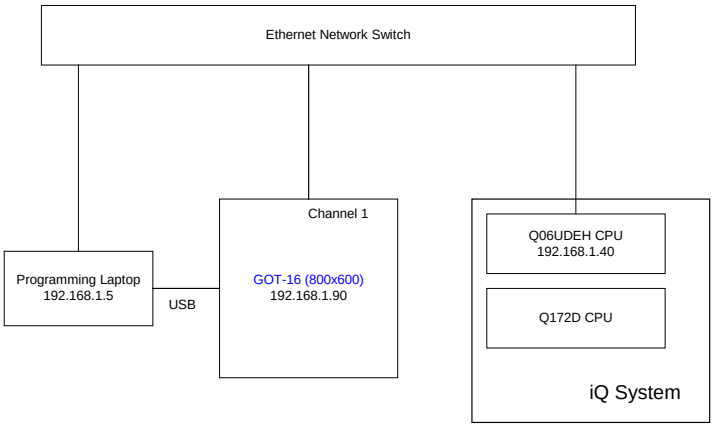


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT using the USB port to download screen information and to the Q06UDEHCPU through the GOT Ethernet Transparent Mode. The Transparent Mode is set up to go through the Ethernet port on the Q06UDEHCPU (with IP Address 192.168.1.40). During the system operation, GOT is configured to work with the iQ PLC through the same Q06UDEHCPU Built-in Ethernet port.

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, as shown in Figure 2 below:

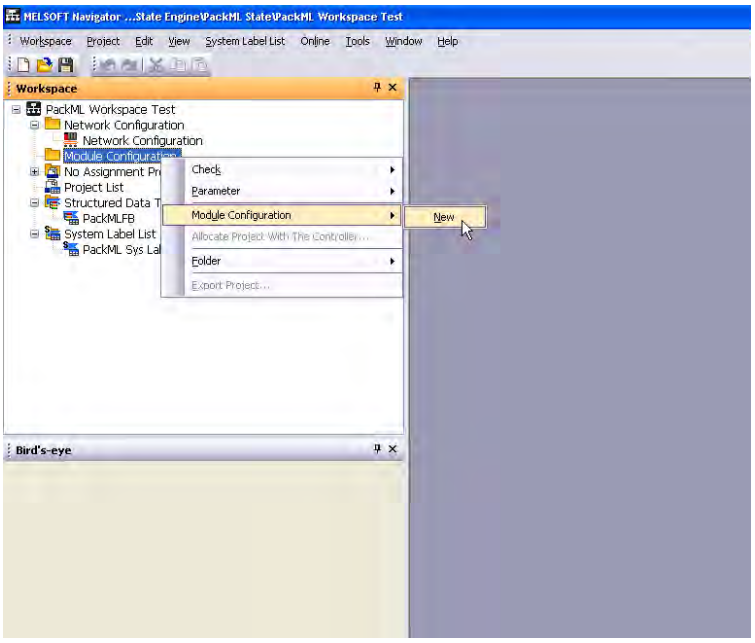


Figure 2 – Creating New Module Configuration

a. When the new Module Configuration workspace is open, one can select all the necessary modules to create an iQ platform that represents the actual hardware platform. In the PackML Template System, an eight-slot main base module (Q38DB) was selected and dragged to the Module Configuration workspace as shown in Figure 3.

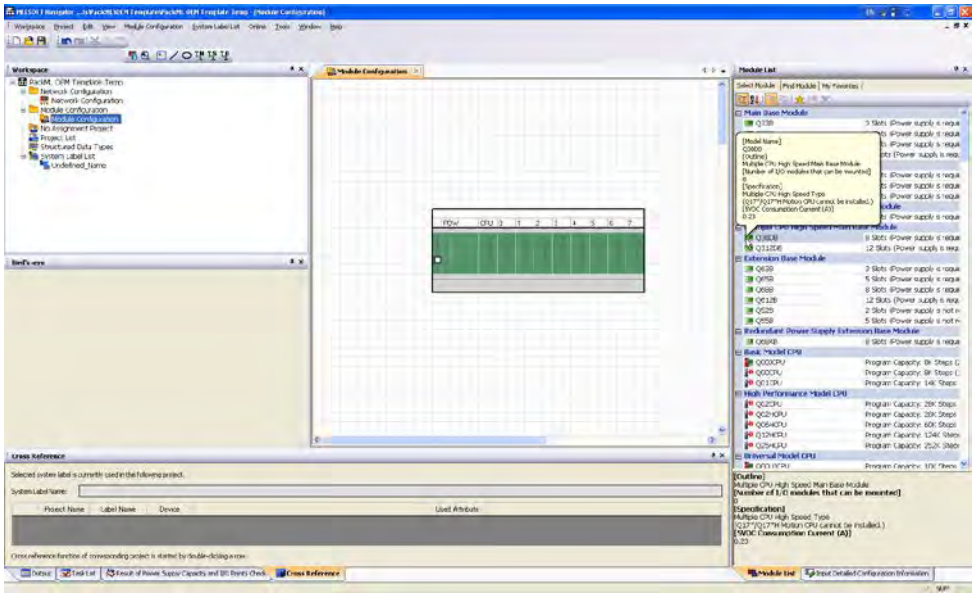


Figure 3 – Module Configuration with Base Module

b. Power supply Q64P, Universal Model CPU Q06UDEHCPU, and Motion Controller Q172D CPU were added to the system. The PLC Module was configured with the proper Station Number and IP address using "Input Detailed Configuration Information" screen as shown in Figure 4.

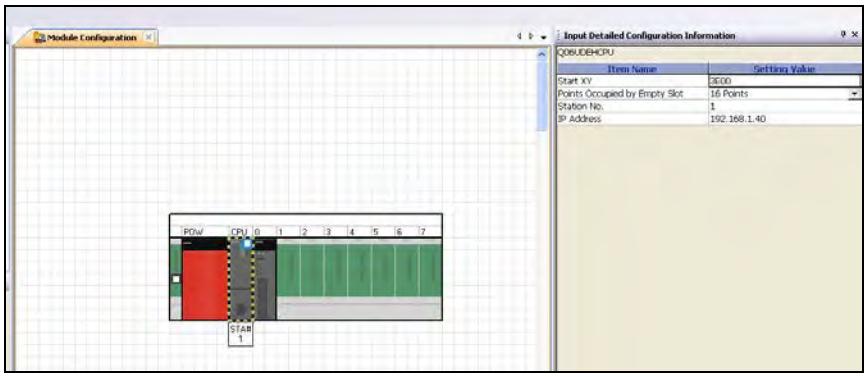


Figure 4 – Q06UDEHCPU Module Configuration

2.2 *Network Configuration*

a. 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 5.

Figure 5 – Initiating PackML Template System Network Configuration

b. Using the Network Cable tools, one can connect the Ethernet port on the Q06UDEHCPU to the network as shown in **Figure 6**.

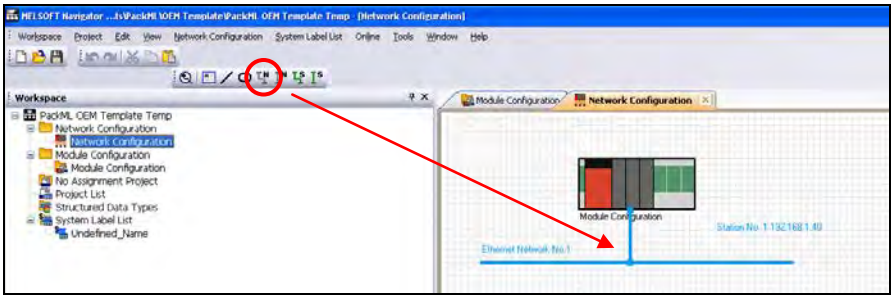


Figure 6 – Connecting Modules to the Network

The "Module Configuration" workspace is now showing the modules are connected to Network #1.

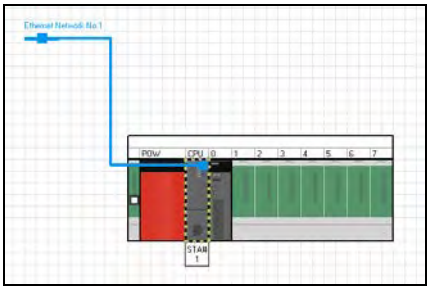


Figure 7 – Module Configuration Reflecting Network Connections

c. From the Module List, one can add a GOT to the system. For the PackML Template System, a GT16 with the resolution of 800 x 600 is added to the system and then configured with the proper channel designations and IP address as shown in Figure 8 and Figure 9 respectively. Please note that the CH1 Station No. has been changed to "3" from the default value "1" reflecting the current configuration. Again, using the Network Cable tool, one can connect the GOT to the Integration System and the network configuration is complete.

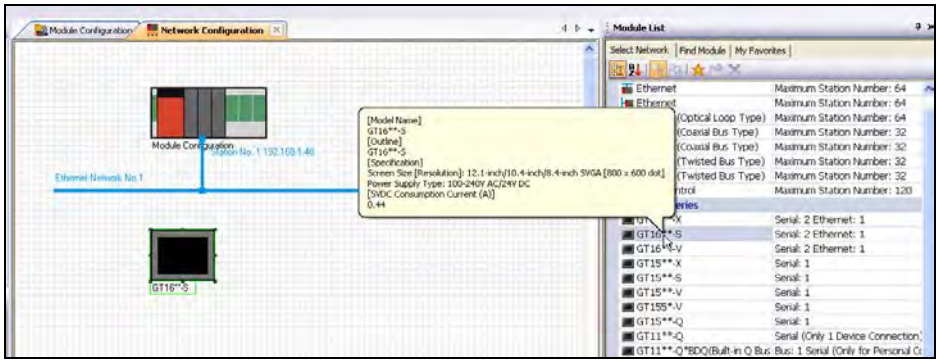


Figure 8 – Adding a GOT to the System

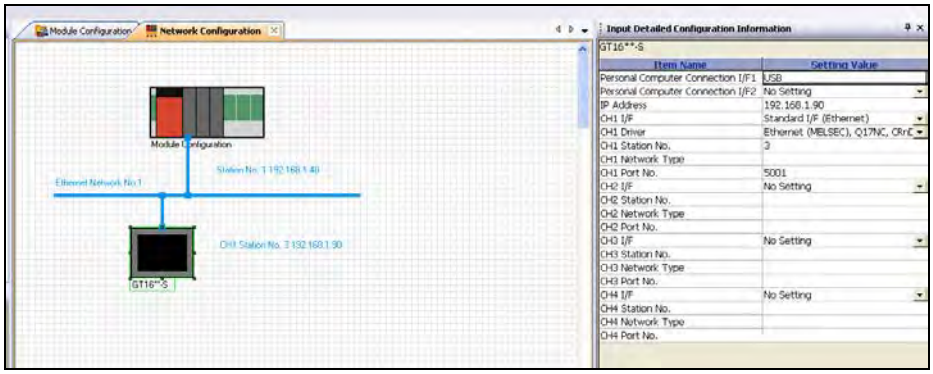


Figure 9 – Configuring the GOT

2.3 Adding Programs to PLC and GOT

Once the PackML Template System is configured with proper modules and network connectivity, one can start to add PLC and GOT programs to the system.

2.3.1. Creating New PLC Program

A new PLC program can be created by double-clicking the CPU module in the Module Configuration workspace and fill in the pop-up window as shown in Figure 10 and click "Create."

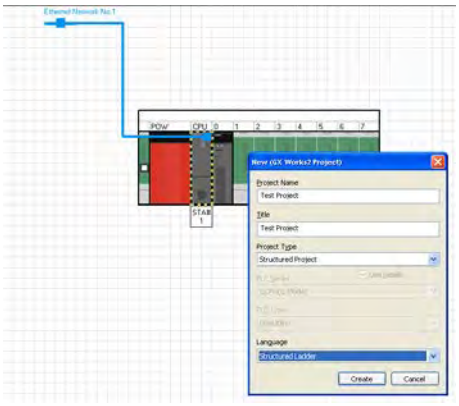


Figure 10 – Creating a New PLC Program

2.3.2. Adding Existing Programs

Another approach is to add existing PLC and GOT projects to the MELSOFT Navigator.

a. From the MELSOFT Navigator project tree, one can right-click the "No Assignment Project" selection and select "Import Project…" as shown below.

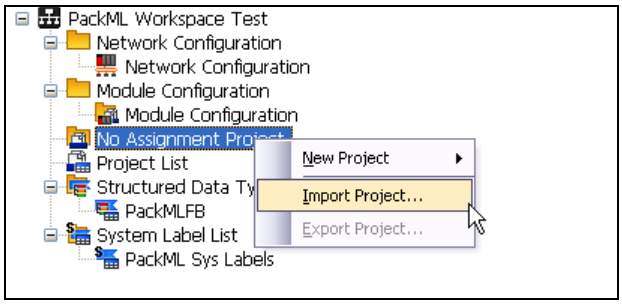


Figure 11 – Importing Projects to iQ Works Navigator

b. One can then select GX Works 2, MT Developer 2, and GT Designer 3 files from the Windows Explorer into the Navigator. The following example shows GX Works 2 file "PackML Implementation" and GT Designer 3 file "PackML Mode State Screen" and MT Developer 2 "Motion Config Examle PackML" were added to the Navigator. Note that the CPU, Motion Controller, and GOT types for the projects are shown next to the file names.

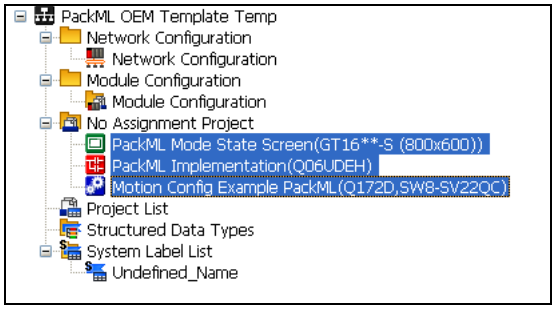


Figure 12 – Example of Imported Files

2.3.3. Allocating Programs

Once the programs are imported, they need to be allocated to the proper modules and network component. From the Navigator project tree, one can right-click the "Module Configuration" and select "Allocate Project

with Controller …" and a pop-up window will appear to allow the user to select the files that match the CPU module configurations from a drop-down list as shown in Figure 13 and Figure 14.

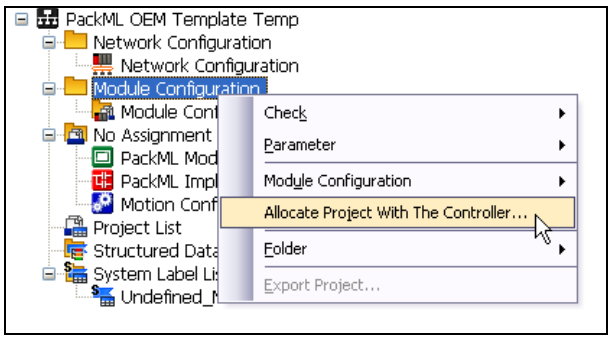


Figure 13 - Allocating Project to CPU Module

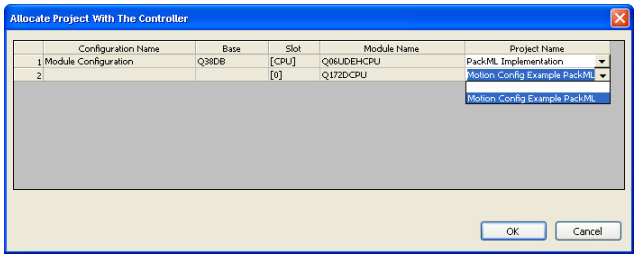


Figure 14 – Allocating CPU Program

a. Similar procedures can be followed to add the GOT program to the configured GOT in the system as shown in Figure 15 and Figure 16.

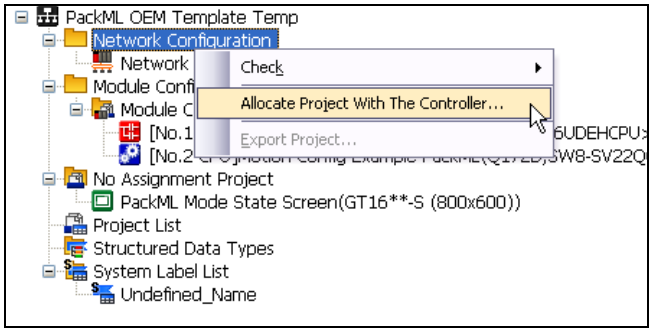


Figure 15 – Allocating Project to GOT on the Network

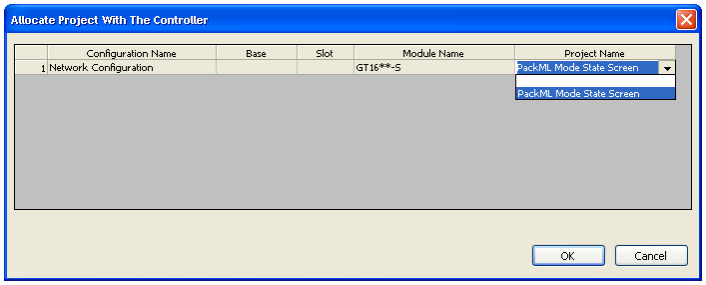


Figure 16 – Allocating GOT Program

3 Registering Labels in the 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, thus the steps in this document describe the procedure relating to creating the system labels from the PLC program. Similar steps can be taken if any GOT labels need to be shared with the PLC. Consult the Motion Controller and GOT manuals on how to define the labels and register them in the System Label Database.

a. Launch the GX Works 2 and in the PLC program, open the global variable window. Select the labels that need to be registered in the System Label Database and then click the "Register Device Name" button and confirm the operation as shown in Figure 17.

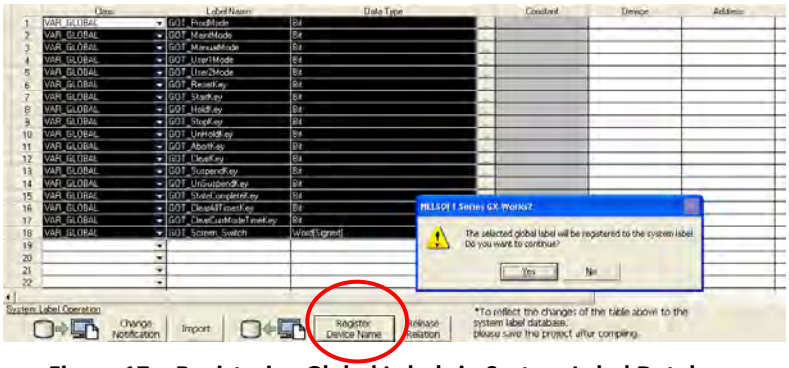


Figure 17 – Registering Global Labels in System Label Database

b. In the Navigator, select Workspace -> Parameter -> Batch Reflect as shown in the following figure:

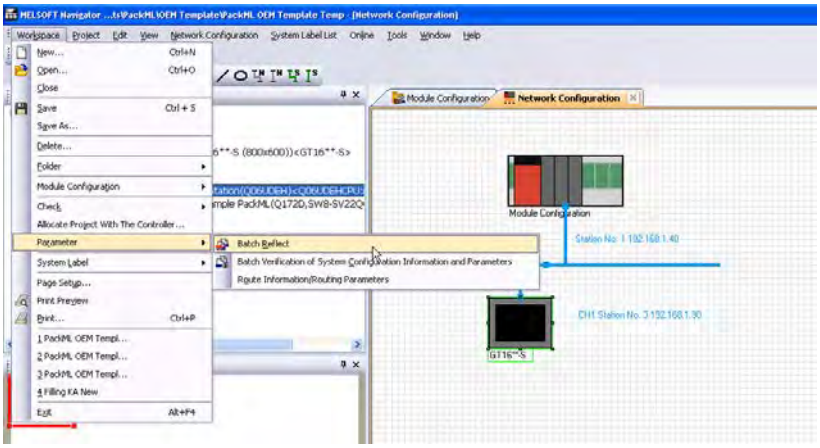


Figure 18 – Batch Reflect the Registered Label

A pop-up window as show below will appear. Confirm the execution of the execution by clicking the "Execute Reflection" button as shown in Figure 19.

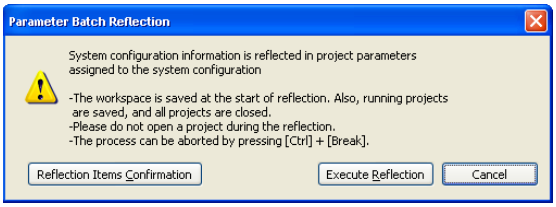


Figure 19 – Selecting Execute Reflection

A second pop-up window will appear and confirm the execution by clicking the "Yes" button in Figure 20.

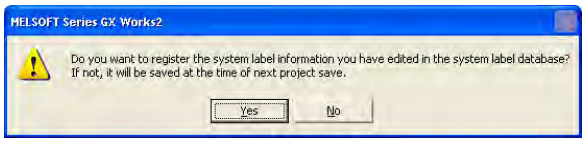


Figure 20 – Confirming Execute Reflection

The labels that will be registered are display in a window similar to Figure 21 below. Make sure the labels are the ones that need to be registered and click the "Reflection" button to initiate the registration. The MELSOFT Navigator will attempt to close the GX Works2 Project. Select "Yes" to close the GX Works 2 project so that the System Labels can be reflected in the data base.

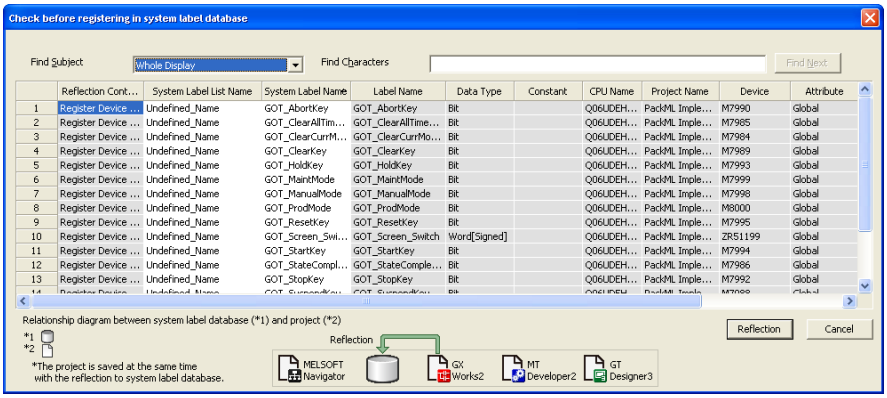


Figure 21 – Reflecting Labels to System Label Database

c. If there is no error in creating the system labels in the database, an indication will be blinking in the Navigator. Right click on the blinking symbol and select the "Change Contents of System Label Database" as shown in Figure 22.

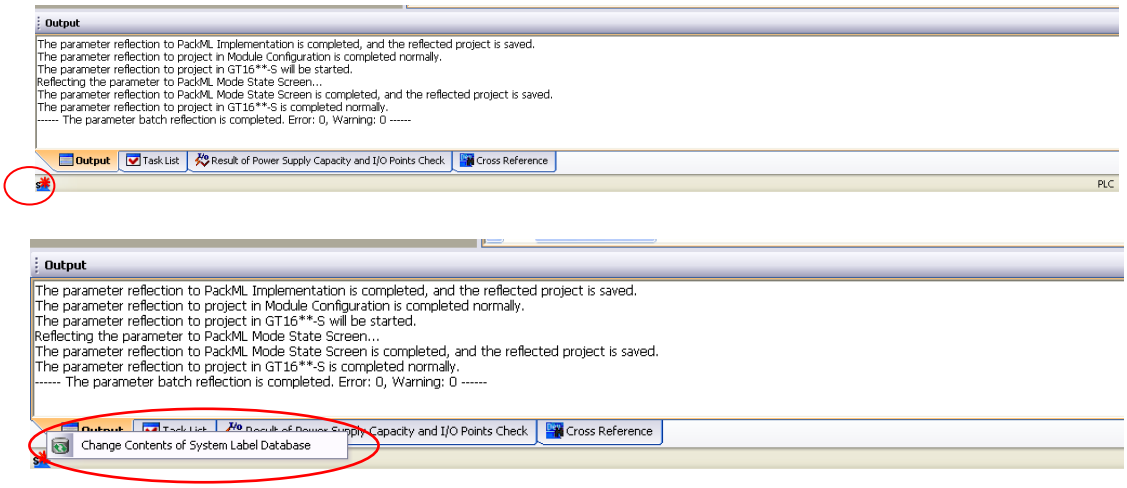


Figure 22 – Updating System Database in the Navigator

The change contents will then be shown similar to the Figure 23 below. Click the "Import" button and import the changes to the Navigator.

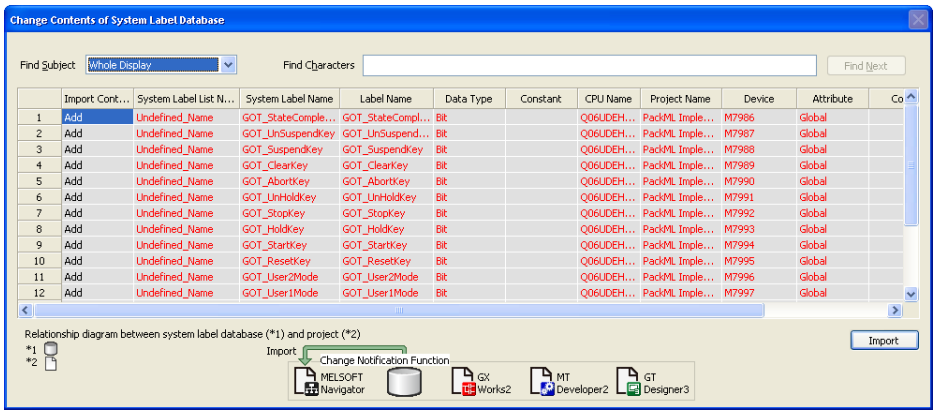


Figure 23 – Database Change Contents

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

From the Navigator, launch the GT Designer 3 with the GOT screens for the application as shown in the figure below:

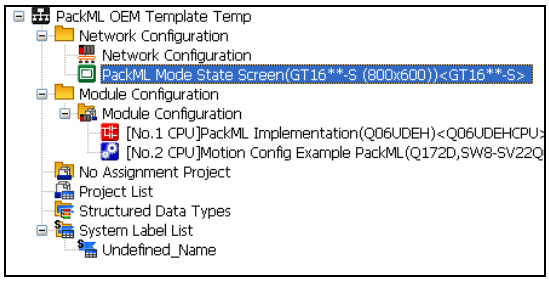


Figure 24 – Launching GT Designer 3

When GT Designer 3 program is launched, the program will perform a system label update/check operation. If there are any system labels that are already in use, the error messages as in Figure 25 will be displayed indicating that there is no route information. In order words, GT Designer 3 does not know where the origins of these system labels are so it can interact with the label properly.

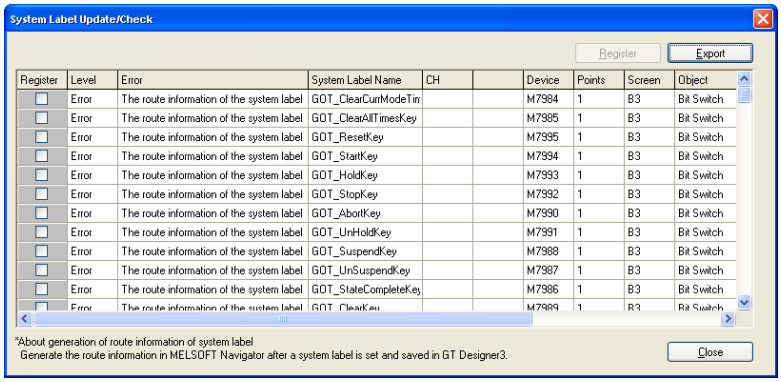


Figure 25 – Error Showing Lack of Route Information

Leaving the GT Designer 3 project open, one can then launch the "Route Information" function from the Navigator using the MELSOFT Navigator project tree by selecting Workspace -> Parameter -> Route Information/ Route Parameters as shown in Figure 26.

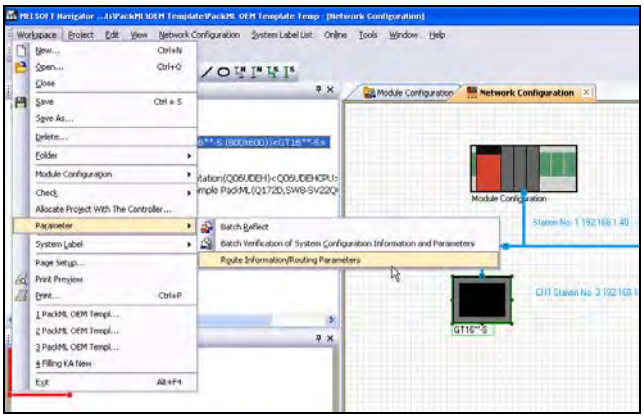


Figure 26 – Establish Route Information

A pop-up window will appear allowing the user to save the GT Designer 3 project and the system will generate the route information as shown in Figure 27. Click "OK" to accept the Route Information.

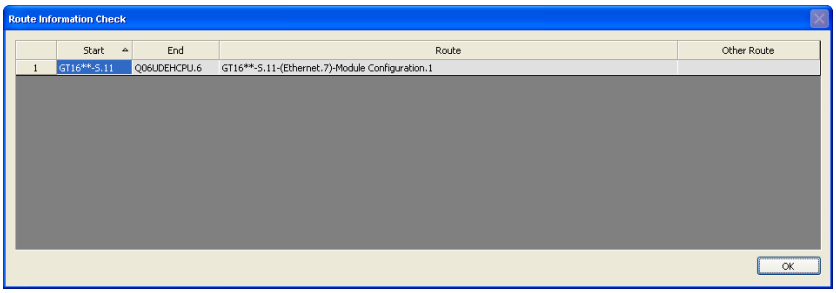


Figure 27 – Route Information Check Display

4.2 Setting Up System Labels for GOT Use

a. In GT Designer 3, select Tools -> System Label Update/Check as shown below to update the system labels in the GOT.

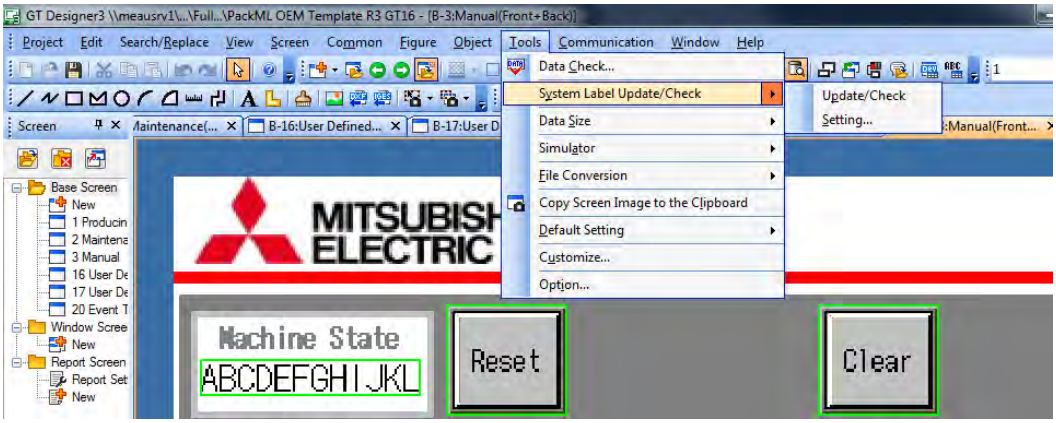


Figure 28 – Using System Label Tool in GT Designer 3

b. From the Navigator, select Workspace -> Check -> Batch Check to verify if there is any error in the system label operation.

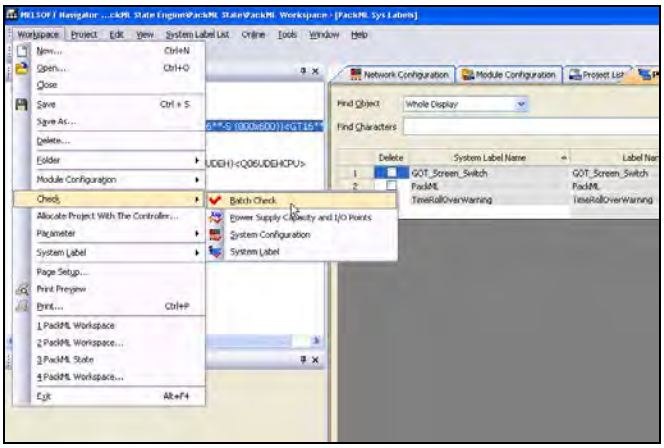


Figure 29 – Batch Check the System Labels

Confirm the Batch Check operation by selecting "Yes" in the pop-up windows that appear as shown in the figures below:

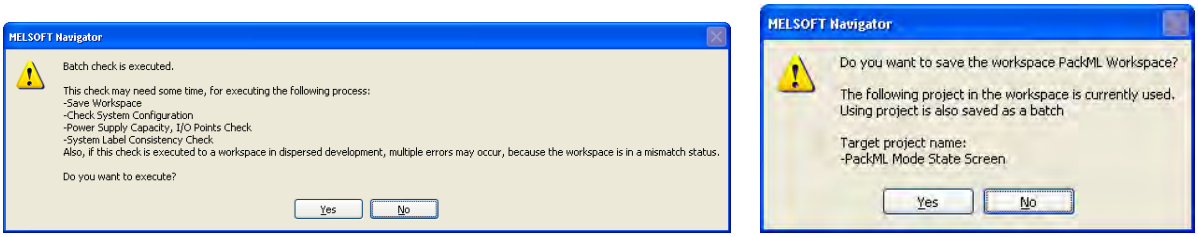


Figure 30 – Pop-Up Windows for Batch Check Confirmation

c. After the Batch Check is completed and there is no error, the following window will appear. Select "Yes" to bring up the next window and then select the "Execute Verify" to continue the Batch Verification process.

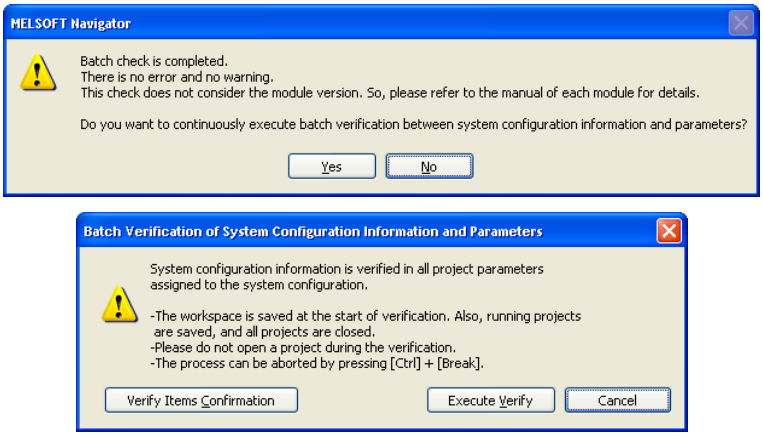


Figure 31 – Pop-Up Windows for Batch Check Verification

d. If there is any error after the Batch Verification process, a window similar to Figure 32 will appear showing the error.

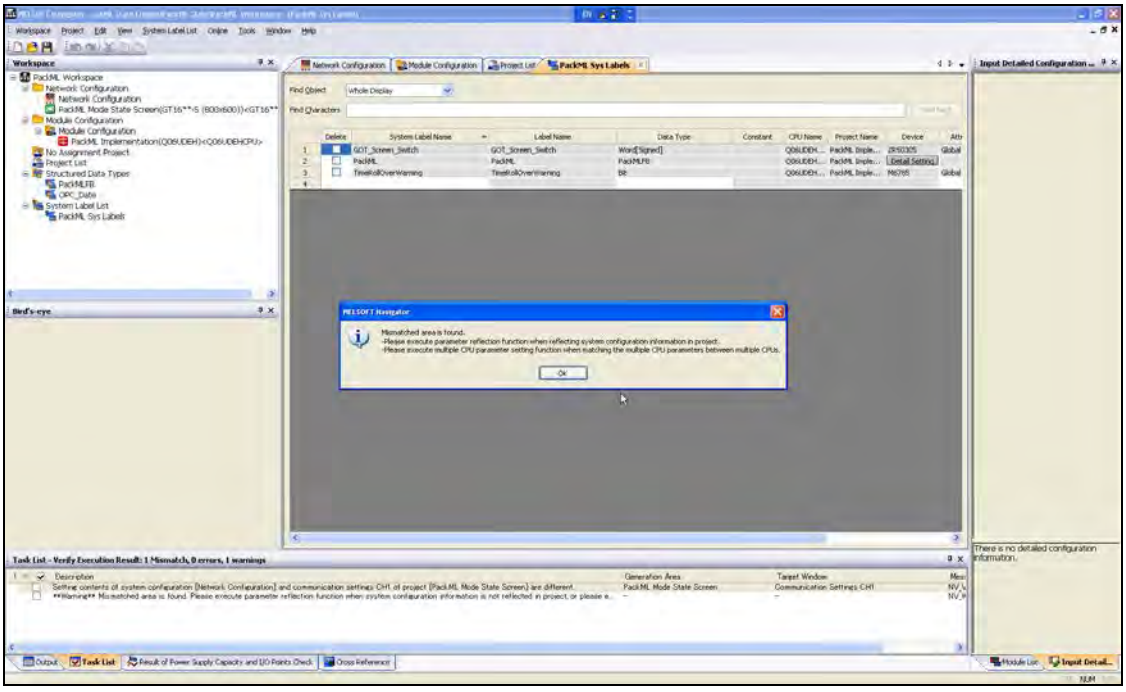


Figure 32 – Batch Verification Result Screen

e. The error in Figure 32 is fairly common indicating that system parameters configured through module and network configurations in the Navigator have not been properly reflected in the PLC and/or GOT programs. Execute Workspace -> Parameter -> Batch Verification of System Configuration Information and Parameters as shown below.

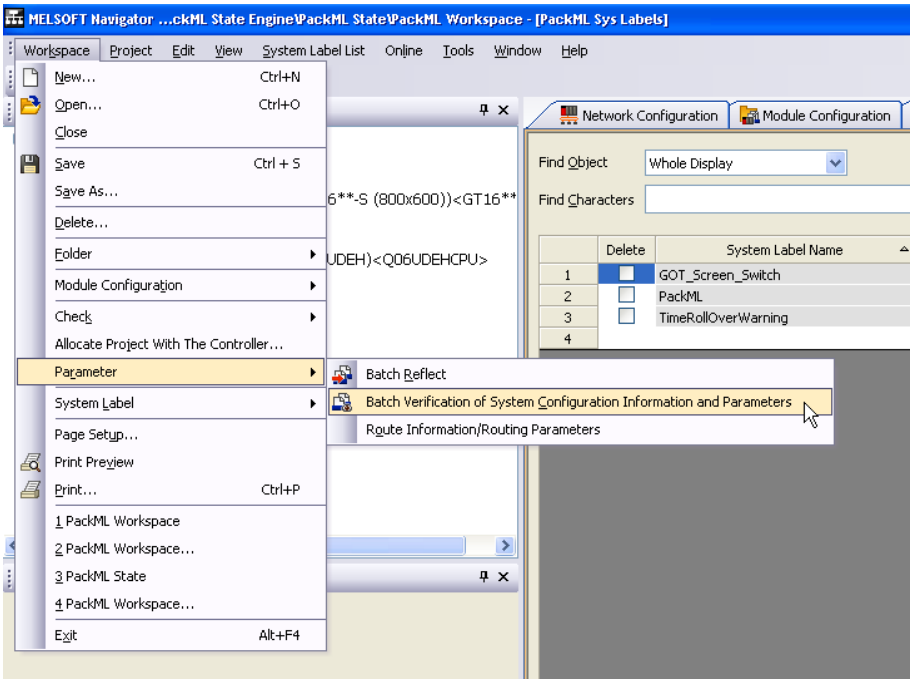
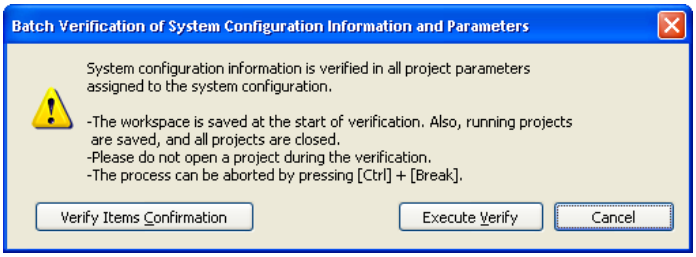


Figure 33 – Bach Verification Configuration and System Labels

Select the "Execute Verify" button to confirm the reflection of system configuration information and parameters.



f. Execute "Batch Reflect" again and then "Check", "Batch Check" to make sure there is no mismatch.

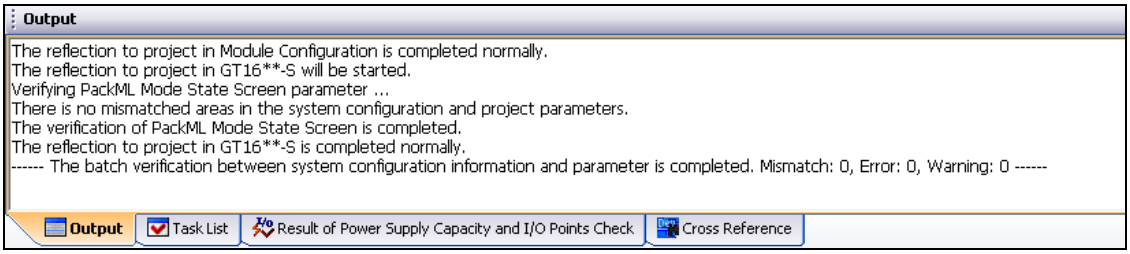


Figure 34 – Output Window Showing Verification Results

4.3 Using the System Labels in GOT

a. From GT Designer 3, select Tools -> Options, select the "Operation" tab, and select the "Always Display" option in "CH No. Selection Dialog Display Setting" to enable the system labels to be shown in DT Designer 3.

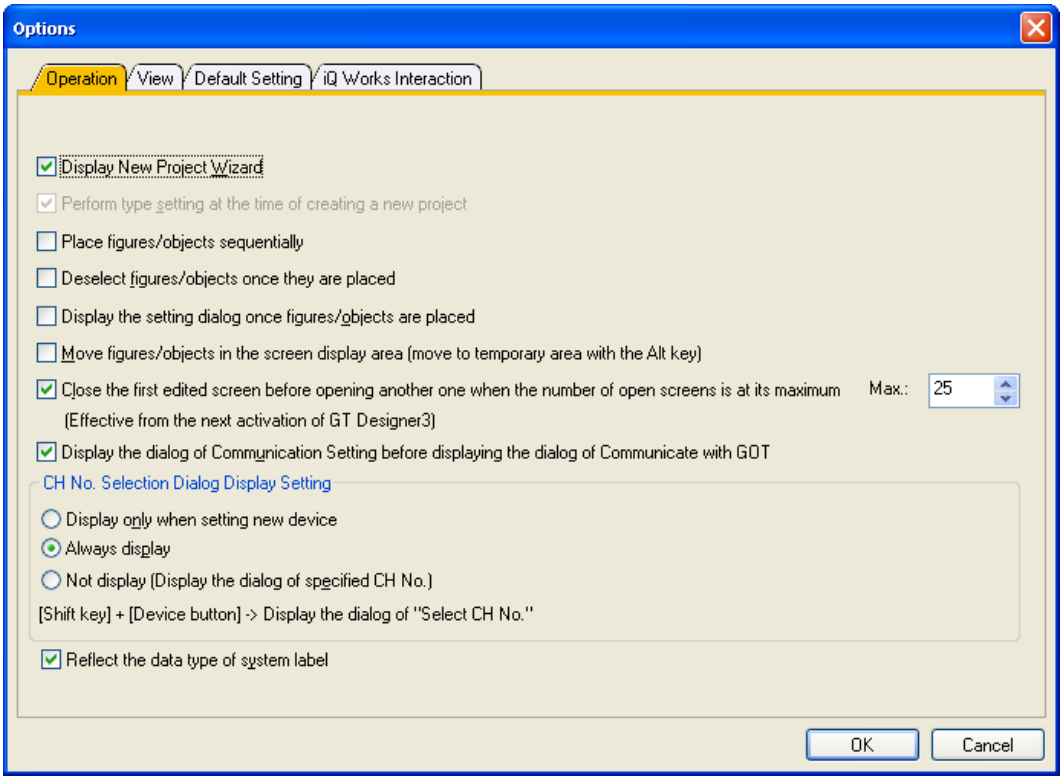


Figure 35 – Enabling System Labels in GOT

b. Double click on one of the objects on a GOT screen where a system label will be use to bring up the corresponding Object Property window as shown in Figure 36 as an example.

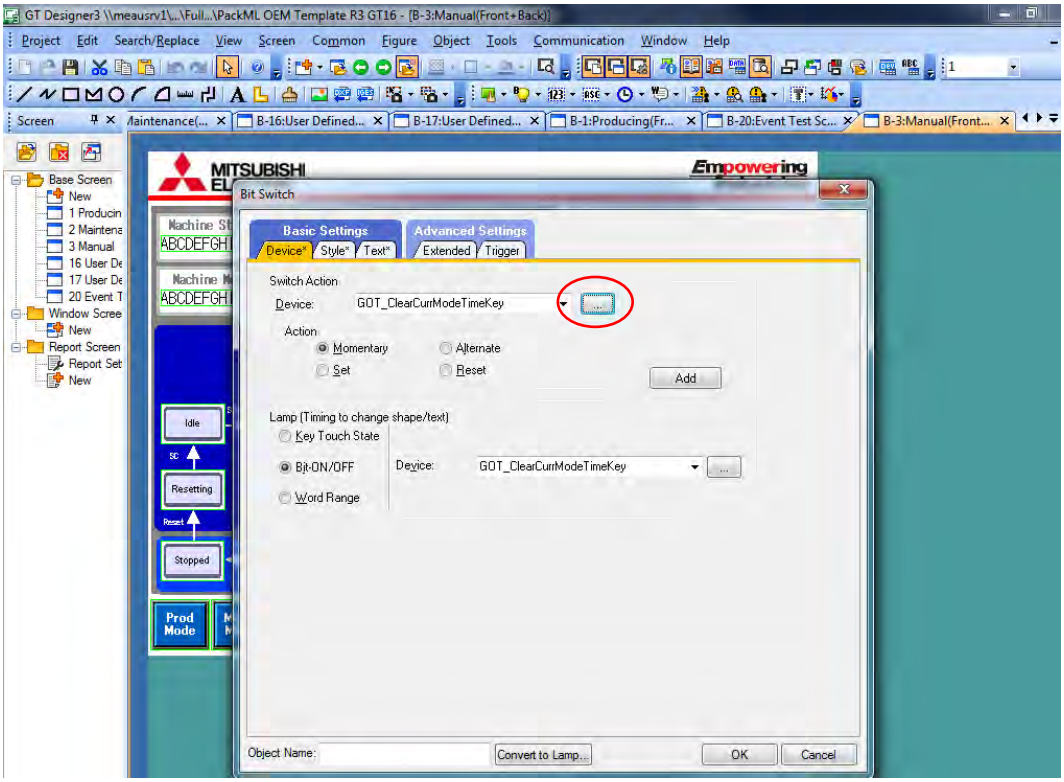


Figure 36 – Configuring a Bit Switch Object to Use the System Label

c. Click the "…" button as shown in Figure 36, the following screen will appear for the user to select the system label that should be used with the selected bit switch device. Click the "Select System Label" button.

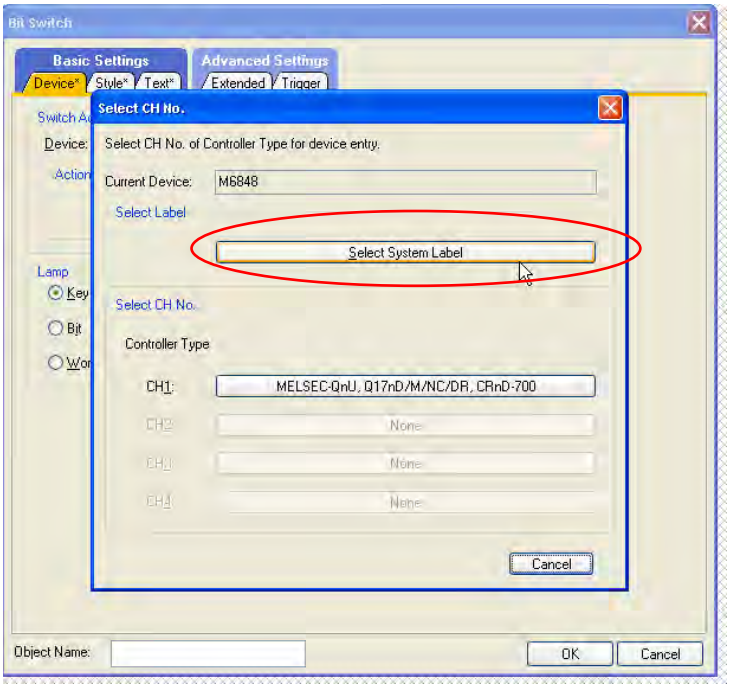


Figure 37 – Selecting System Label in GOT

d. The system labels will be displayed in the table as shown in Figure 38. Select the proper system label to be used with the bit switch and click the "Import" button.

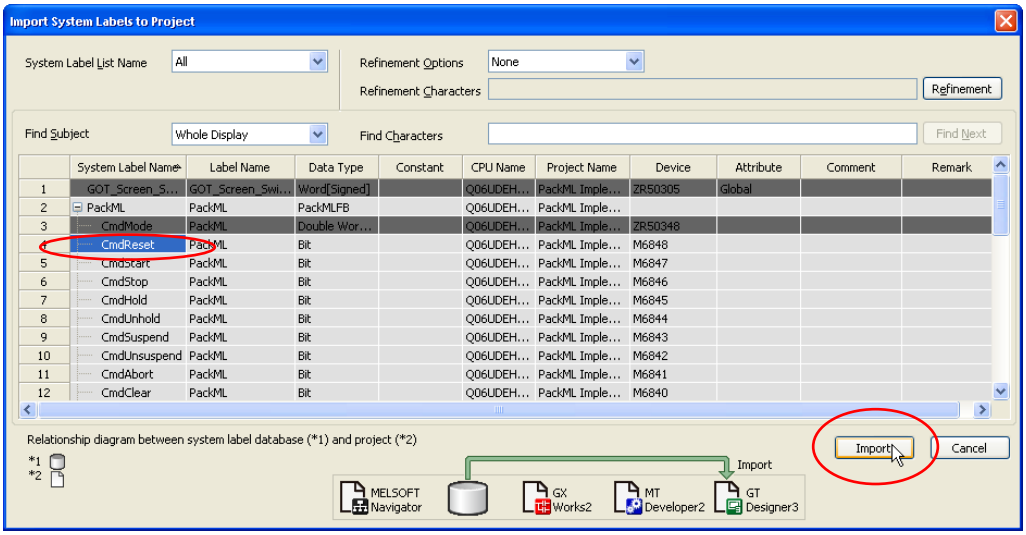


Figure 38 – Selecting System Label from the Database

The label will then show in the object property window as shown below. However, if the routing information has not been properly established, the label will be pre-fixed with two question marks "??" indicating the label is still not yet valid to be used by GOT.

Figure 39 – Verifying System Label Validity in GT Designers 3

e. If that is the case, execute "**Tools -> System Label Update / Check"** in DT Designer 3 and the following error message will be displayed conforming that there is no route information. In other words, GT Designer 3 does not know where the system label is originated.

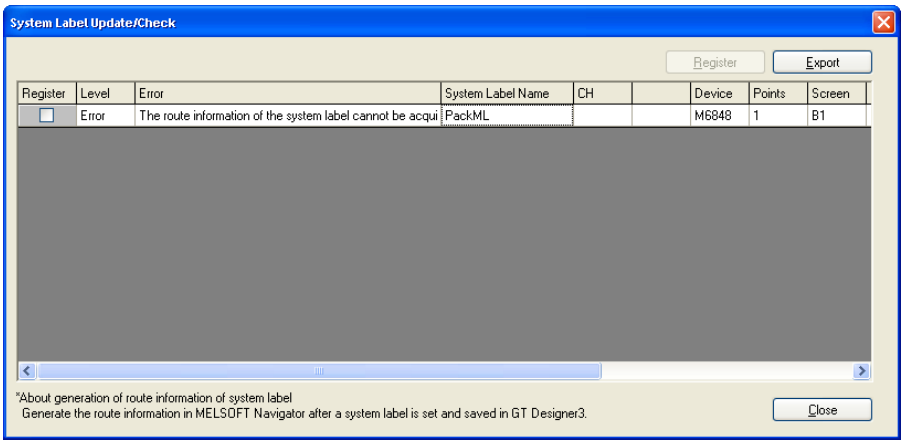


Figure 40 – System Label Update Check Error in GT Designer 3

f. From the Navigator, select "**Workspace -> Parameter -> Route Information/Routing Parameters**" and the proper routing information will then be generated showing where the system label data is originated as shown in Figure 42.

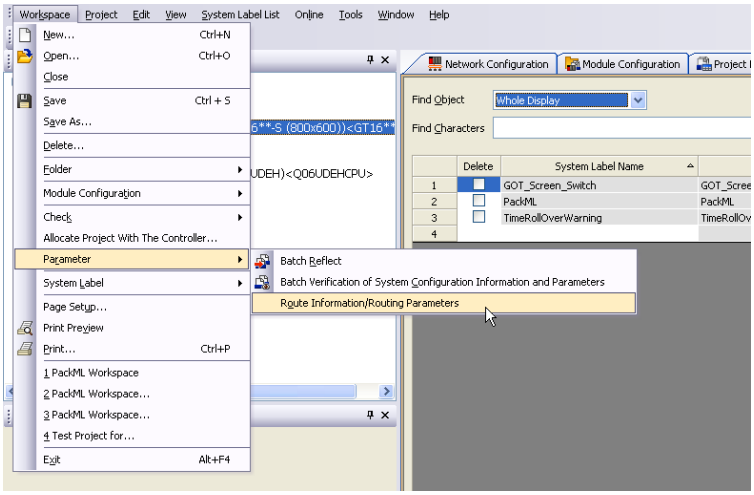


Figure 41 – Generating Routing Information for System Labels to be Used in GT Designer 3

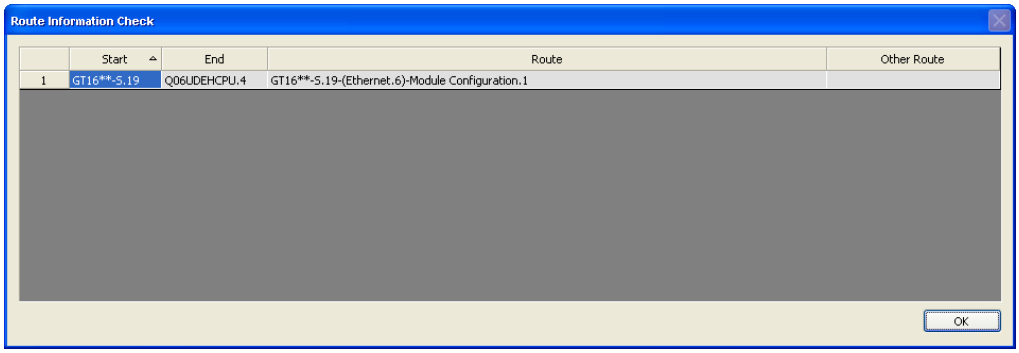


Figure 42 – Routing Information

g. Execute Batch Reflect again on the Navigator. If there is no error, re-launch GT Designer 3, and the System Label Update / Check operation will execute automatically. The system label will now become valid as shown in Figure 43 without the question marks.

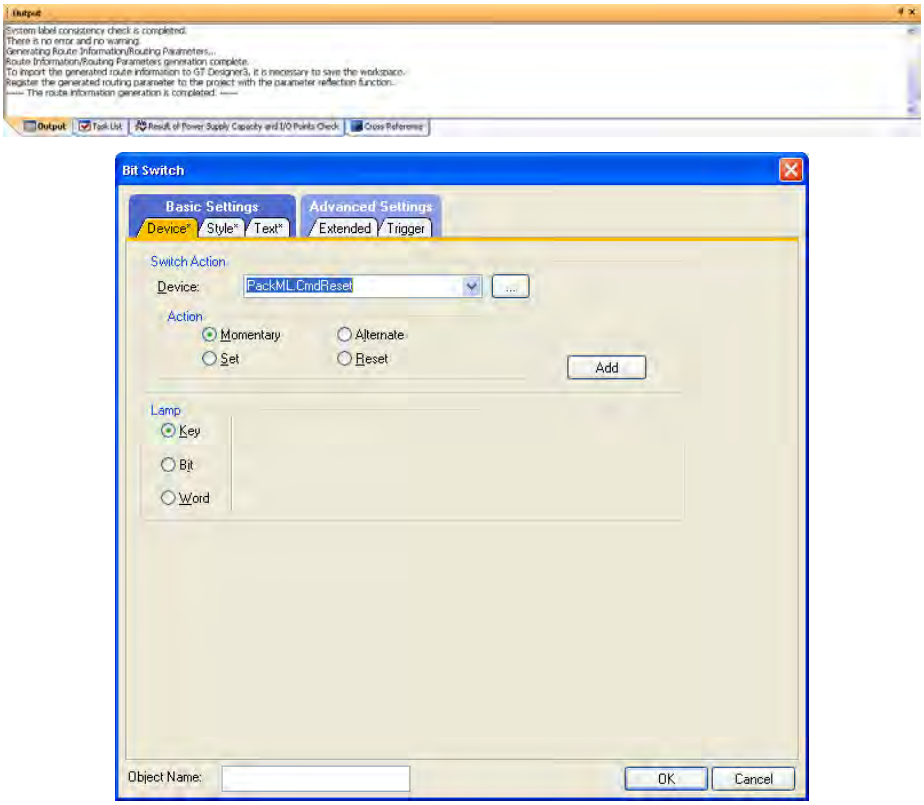


Figure 43 – Valid System Label in GT Designer 3

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 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V1.1	February 19, 2016	Modified PackTags dimensions also newer version of Kepware OPC server
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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 PLC to form a total solution set. Kepware KEPServerEX V5.19 with enhanced Mitsubishi Ethernet Driver is used in the PackML Template system implementation.

Following is a list of critical PackTags design considerations:

- The PackTags are implemented in an iQ 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 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 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 platform. PackTags are implemented in three Data Groups: Command, Status, and Admin, and the correlation of standard PackTag names to the shorten iQ 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 10
- The number of states in each mode of a machine is limited to 20.
- 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 – Selecting PLC Parameters for Configuration

4.1 PLC File

Because of the large number of tags required to support the PackTags specification, a PLC file is used to allocate extended memory locations for the PackTags and system labels that are used to process events (See Part 5 of the Users Guide).

Select the “PCL Parameter” and then the “PLC File” tab to add the extended memory in the system.

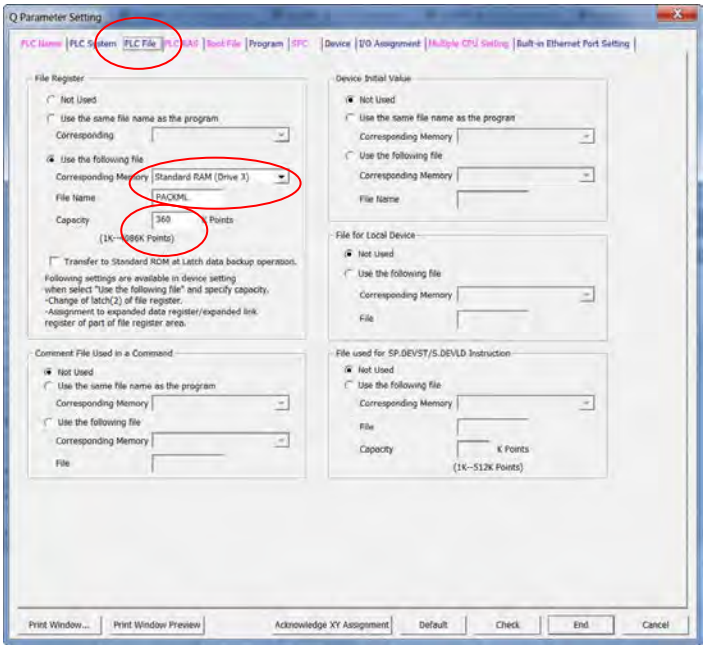


Figure 2 – Configuring PLC File and Capacity

It is critical to note that the PLC file is assigned to Standard RAM of the PLC. One can choose to add an external memory card to the PLC and assign the PLC file to the memory card. However, because of the access time to the memory card is not as efficient comparing to it of the Standard RAM, assigning the PLC file to Standard RAM significantly reduces the PLC program scan time.

Depending on the type of PLC used in the system, the capacity of the Standard RAM may vary. However, configuration with 300K points is sufficient to address the total number of PackTags and system labels that are used in the PackML Template system.

4.2 Device

In this PackTags implementation, most labels are assigned to D registers. A few tags with data type bits are assigned to M bits. In the “Device” tab, assign the extended points to both D and ZR registers. The default is assigning the additional device points to ZR registers only. Make sure that the ZR register points are set to 160K and Extended Data Registers “D” are set to 200K points.

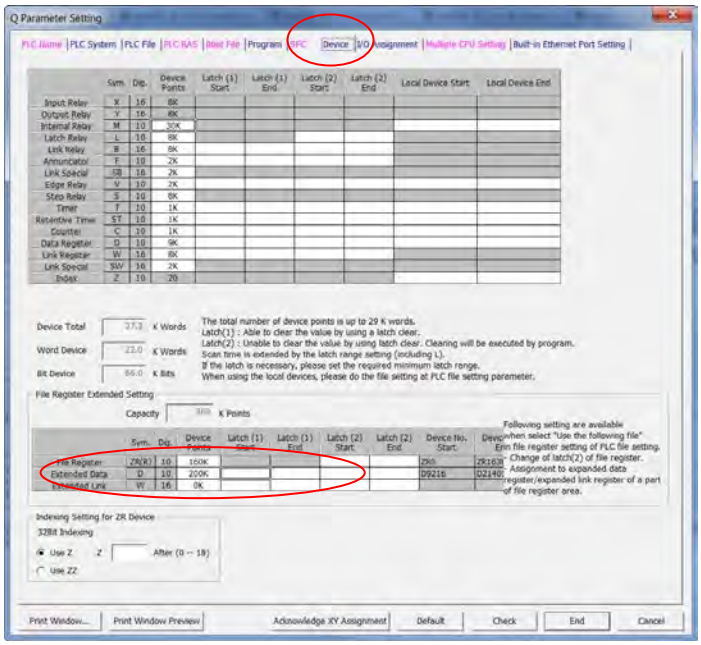


Figure 3 – Configuring File Register Extended Setting

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 4:

- Set the proper IP address and Subnet Mask
- Select “Binary Code”
- Select “Enable onlne change (FTP, MC Protocol)”

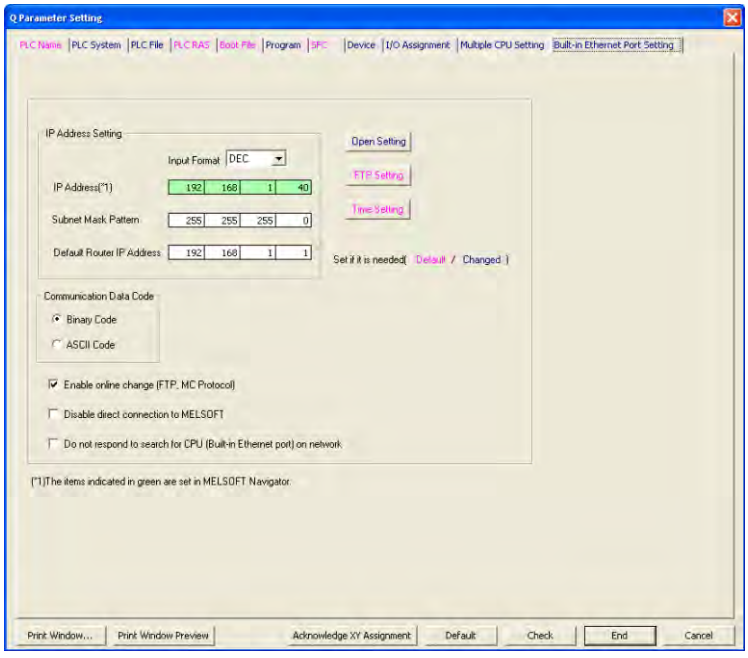


Figure 4 – Configuring the Built-in Ethernet Port

- Click the "Open Setting" and configure the channel to communicate using TCP, UDP, MC Protocol, and port number to the desired value.
    - 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.

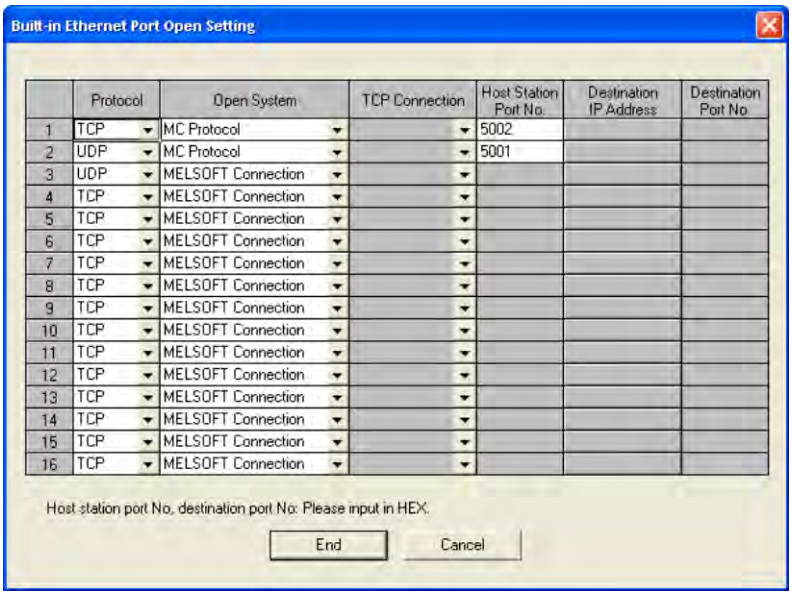


Figure 5 – PLC Communication Channel Configuration for GOT and OPC Communication

5 GX Works2 Label Implementation

As shown in the Appendix, all the labels that are required to support the PackTags standard are implemented in the iQ PLC using GX Works2 global labels. The PackTags labels are grouped into three categories: Command labels, Status labels, and Administrative labels. Five different structured data types are also created to support the label definitions.

The PackTag global labels are part of the "PackML_R3_V1" library which is already part of the template. The library can easily be incorporated into existing OEM programs.

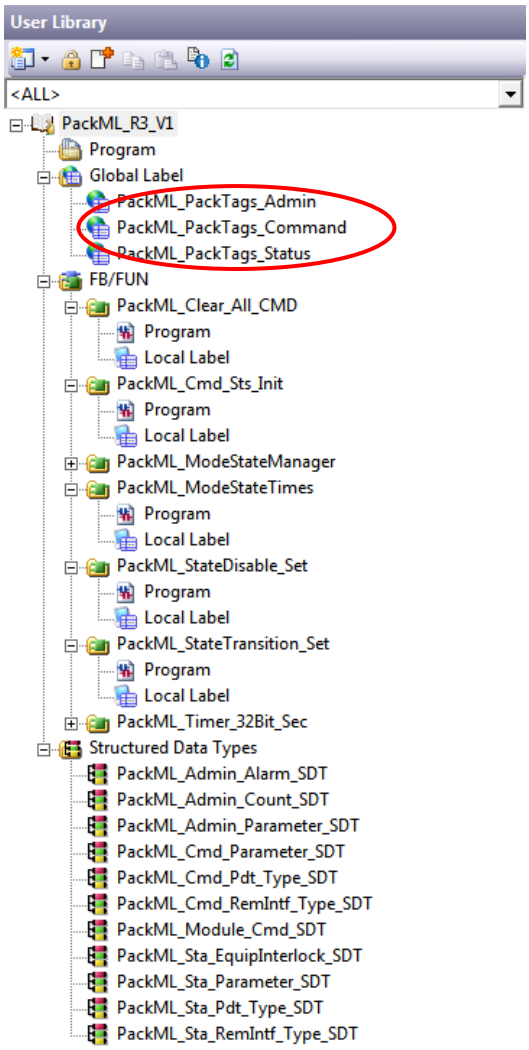


Figure 6 – Global Label Groups and Structure Data Types

5.1 Command Labels – PackTags_Command

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

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	gvd_Cmd_UnitMode	Double Word[Signed]	...		D150000	%MD0.150000
2	VAR_GLOBAL	gvb_Cmd_UnitModeChangeReq	Bit	...		M2400	%MX0.2400
3	VAR_GLOBAL	gvl_Cmd_MachSpeed	FLOAT (Double Precision)	...		D150002	%ML0.150002
4	VAR_GLOBAL	gvdu_Cmd_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]	...		D150006	%MD0.150006
5	VAR_GLOBAL	gvd_Cmd_CntrlCmd	Double Word[Signed]	...		D150008	%MD0.150008
6	VAR_GLOBAL	gvb_Cmd_CmdChangeRequest	Bit	...		M2401	%MX0.2401
7	VAR_GLOBAL	gvda_Cmd_Number_RemInt	Double Word[Signed](0..9)	...		D150010	%MD0.150010
8	VAR_GLOBAL	gvda_Cmd_CntlCmdNum_RemIn	Double Word[Signed](0..9)	...		D150030	%MD0.150030
9	VAR_GLOBAL	gvda_Cmd_CmdValue_RemIntf	Double Word[Signed](0..9)	...		D150050	%MD0.150050
10	VAR_GLOBAL	gvsta_Cmd_RemIntf	PackML_Cmd_RemIntf_Type_SDT(0..9)	...		Detail Setting	Detail Setting
11	VAR_GLOBAL	gvsta_Cmd_Parameter	PackML_Cmd_Parameter_SDT(0..9)	...		Detail Setting	Detail Setting
12	VAR_GLOBAL	gvda_Cmd_PdtID_Pdt	Double Word[Signed](0..4)	...		D154690	%MD0.154690
13	VAR_GLOBAL	gvsta_Cmd_Pdt	PackML_Cmd_Pdt_Type_SDT(0..4)	...		Detail Setting	Detail Setting

Figure 7 – PackTags_Command Global Labels

The structured data types used in the Command Labels are PackML_Cmd_Remintf_Type and PackML_Cmd_Pdt_Type. Their configurations are shown in the follow two screens.

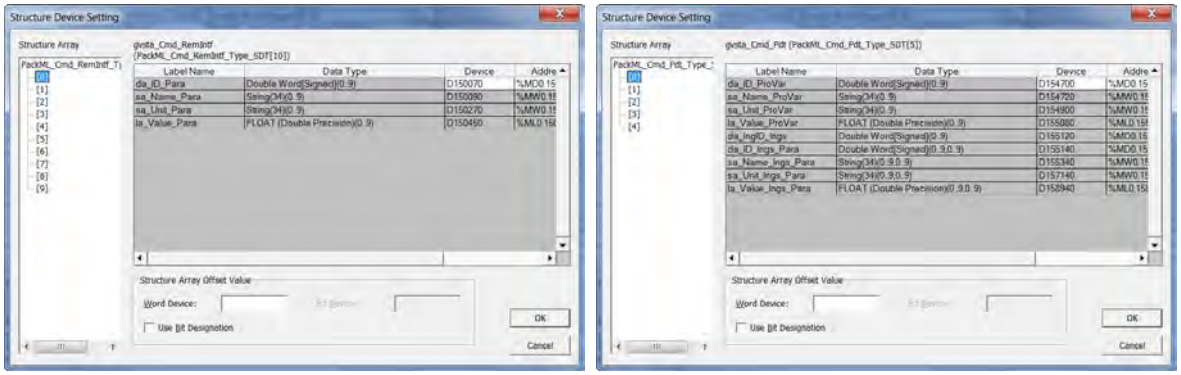


Figure 8 – PackML_Cmd_Remintf_Type and PackML_Cmd_Pdt_Type labels

5.2 Status Labels – PackTags_Status

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

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	gvd_Sta_UnitModeCurrent	Double Word[Signed]	...		D180000	%MD0.180000
2	VAR_GLOBAL	gvb_Sta_UnitModeRequested	Bit	...		M2410	%MX0.2410
3	VAR_GLOBAL	gvb_Sta_UnitModeChangeInProcess	Bit	...		M2411	%MX0.2411
4	VAR_GLOBAL	gvd_Sta_StateCurrent	Double Word[Signed]	...		D180002	%MD0.180002
5	VAR_GLOBAL	gvd_Sta_StateRequested	Double Word[Signed]	...		D180004	%MD0.180004
6	VAR_GLOBAL	gvb_Sta_StateChangeInProcess	Bit	...		M2412	%MX0.2412
7	VAR_GLOBAL	gvl_Sta_MachSpeed	FLOAT (Double Precision)	...		D180006	%ML0.180006
8	VAR_GLOBAL	gvl_Sta_CurMachSpeed	FLOAT (Double Precision)	...		D180010	%ML0.180010
9	VAR_GLOBAL	gvdu_Sta_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]	...		D180014	%MD0.180014
10	VAR_GLOBAL	gvst_Sta_EquipmentInterlock	PackML_Sta_EquipInterlock_SDT	...		Detail Setting	Detail Setting
11	VAR_GLOBAL	gvda_Sta_Number_RemInt	Double Word[Signed](0..9)	...		D180016	%MD0.180016
12	VAR_GLOBAL	gvda_Sta_CntlCmdNum_RemIntf	Double Word[Signed](0..9)	...		D180036	%MD0.180036
13	VAR_GLOBAL	gvda_Sta_CmdValue_RemIntf	Double Word[Signed](0..9)	...		D180056	%MD0.180056
14	VAR_GLOBAL	gvsta_Sta_RemIntf	PackML_Sta_RemIntf_Type_SDT(0..9)	...		Detail Setting	Detail Setting
15	VAR_GLOBAL	gvsta_Sta_Parameter	PackML_Sta_Parameter_SDT(0..9)	...		Detail Setting	Detail Setting
16	VAR_GLOBAL	gvda_Sta_PdtID_Pdt	Double Word[Signed](0..4)	...		D184696	%MD0.184696
17	VAR_GLOBAL	gvsta_Sta_Pdt	PackML_Sta_Pdt_Type_SDT(0..4)	...		Detail Setting	Detail Setting

Figure 9 – PackTags_Status Global Labels

The structured data types used in the Status Labels are Sta_Remintf_Type and Sta_Pdt_Type. Their configurations are shown in the follow two screens.

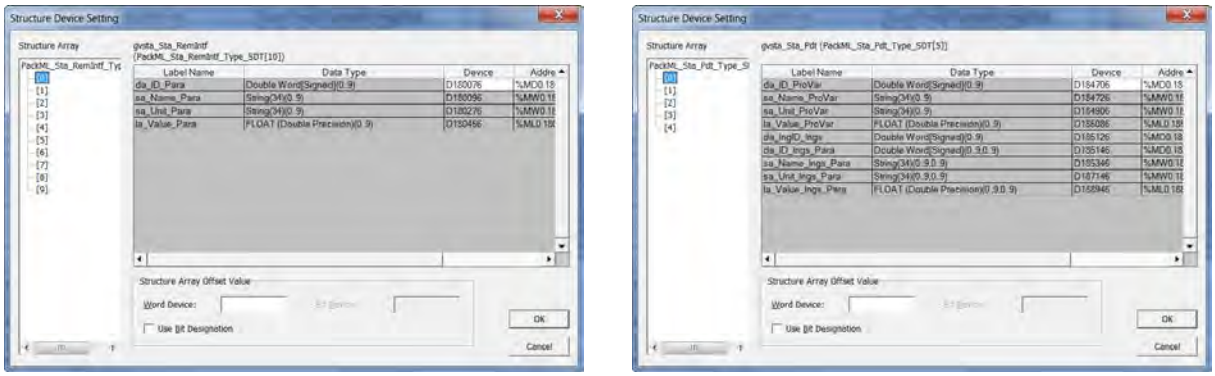


Figure 10 – Sta_Remintf_Type and Sta_Pdt_Type labels

5.3 Administrative Labels

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

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	gvst_Adm_Parameter	PackML_Admin_Parameter_SDT(0..19)	...		Detail Setting	Detail Setting
2	VAR_GLOBAL	gvsta_Adm_Alarm	PackML_Admin_Alarm_SDT(0..63)	...		Detail Setting	Detail Setting
3	VAR_GLOBAL	gvd_Adm_AlarmExtent	Double Word[Signed]	...		D114168	%MD0.114168
4	VAR_GLOBAL	gvst_Adm_AlarmHistory	PackML_Admin_Alarm_SDT(0..255)	...		Detail Setting	Detail Setting
5	VAR_GLOBAL	gvd_Adm_AlarmHistoryExtent	Double Word[Signed]	...		D127482	%MD0.127482
6	VAR_GLOBAL	gvst_Adm_AlarmStopReason	PackML_Admin_Alarm_SDT	...		Detail Setting	Detail Setting
7	VAR_GLOBAL	gvd_Adm_AlarmStopReasonExte	Double Word[Signed]	...		D127536	%MD0.127536
8	VAR_GLOBAL	gvst_Adm_AlarmWarning	PackML_Admin_Alarm_SDT(0..63)	...		Detail Setting	Detail Setting
9	VAR_GLOBAL	gvd_Adm_AlarmWarningExtent	Double Word[Signed]	...		D130866	%MD0.130866
10	VAR_GLOBAL	gvda_Adm_ModeCurrentTime	Double Word[Signed](0..31)	...		D130868	%MD0.130868
11	VAR_GLOBAL	gvda_Adm_ModeCumulativeTim	Double Word[Signed](0..31)	...		D130932	%MD0.130932
12	VAR_GLOBAL	gvda_Adm_StateCurrentTime	Double Word[Signed](0..31,0..17)	...		D130996	%MD0.130996
13	VAR_GLOBAL	gvda_Adm_StateCumulativeTime	Double Word[Signed](0..31,0..17)	...		D132148	%MD0.132148
14	VAR_GLOBAL	gvsta_Adm_ProdConsumedCnt	PackML_Admin_Count_SDT(0..9)	...		Detail Setting	Detail Setting
15	VAR_GLOBAL	gvsta_Adm_ProdProcessedCnt	PackML_Admin_Count_SDT(0..9)	...		Detail Setting	Detail Setting
16	VAR_GLOBAL	gvsta_Adm_ProdDefectiveCnt	PackML_Admin_Count_SDT(0..9)	...		Detail Setting	Detail Setting
17	VAR_GLOBAL	gvd_Adm_AccTimeSinceReset	Double Word[Signed]	...		D134560	%MD0.134560
18	VAR_GLOBAL	gvl_Adm_MachDesignSpeed	FLOAT (Double Precision)	...		D134562	%ML0.134562
19	VAR_GLOBAL	gvd_Adm_StatesDisabled	Double Word[Signed]	...		D134566	%MD0.134566
20	VAR_GLOBAL	gvwa_Adm_PLCDateTime_Date	Word[Unsigned]/Bit String[16-bit](0..6)	...		D134568	%MW0.134568

Figure 11 – PackTags_Admin Global Labels

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.

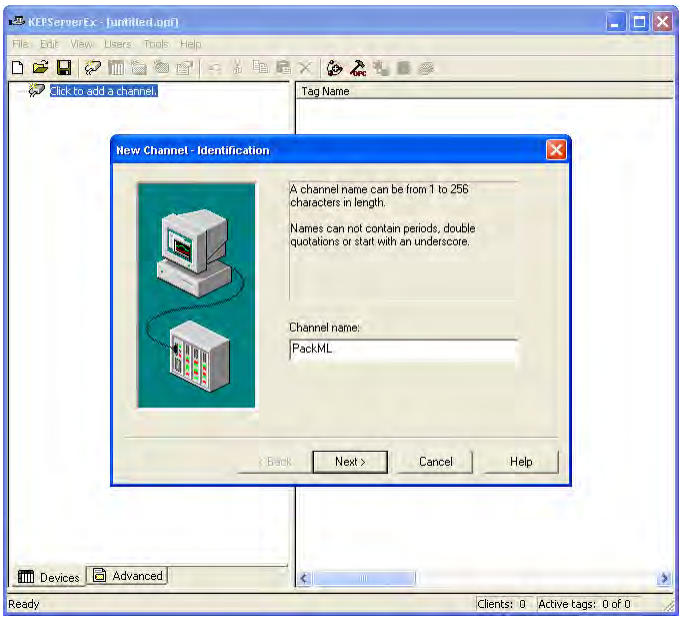


Figure 12 – Adding Channel in Kepware

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

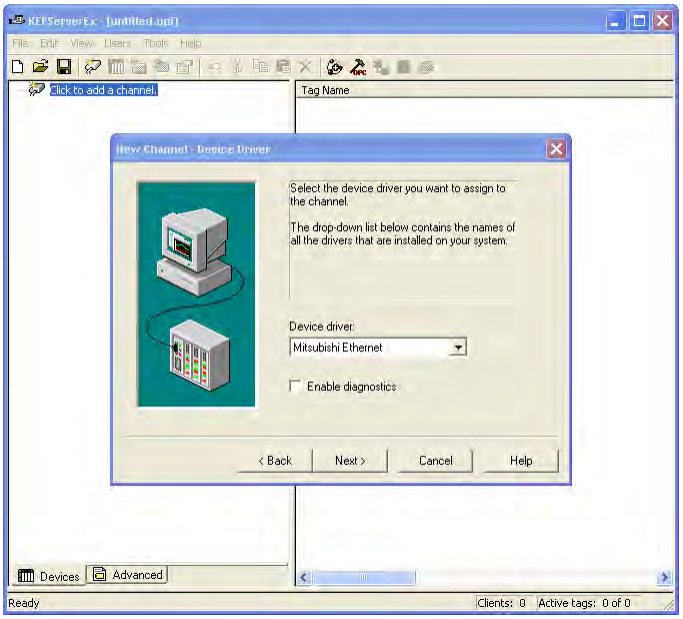


Figure 13 – Adding Device Driver to the Channel

- Define the Network Adapter of the system where the OPC Server is running on. In this example, the interface card is at IP address 192.168.1.5.

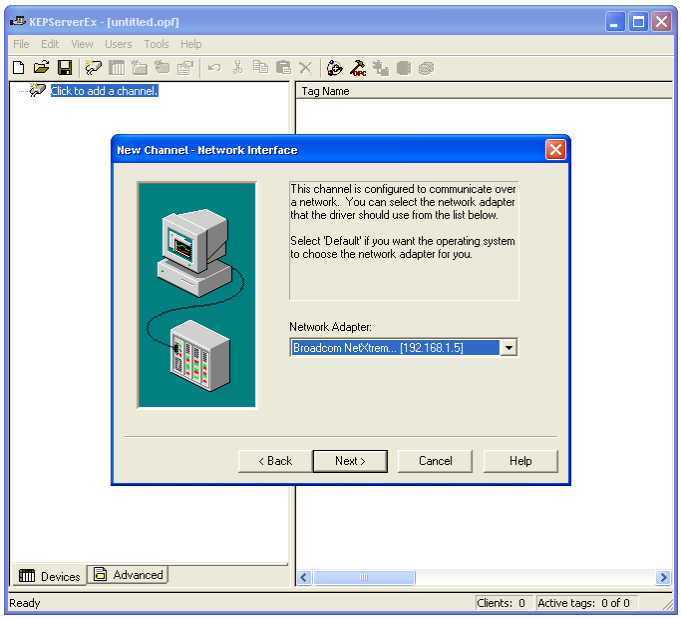


Figure 14 – Selecting the Network Adaptor where the OPC Server is Running

- 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.

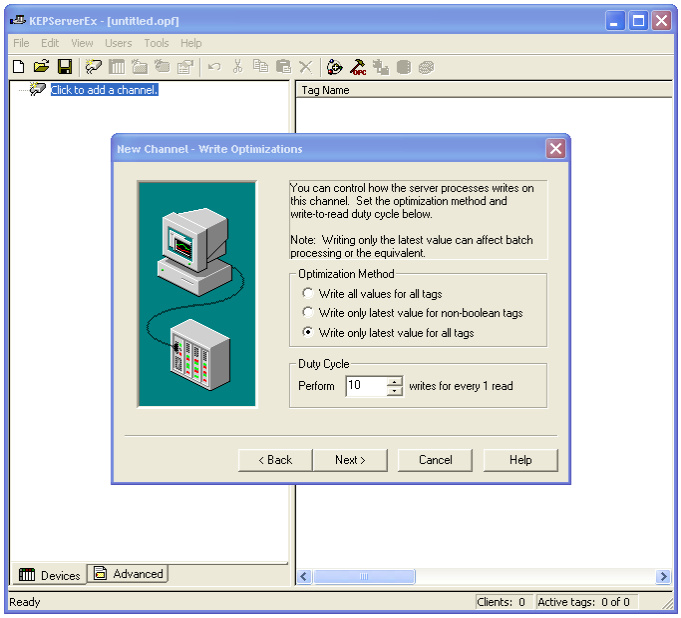


Figure 15 – Selecting Optimization Method

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

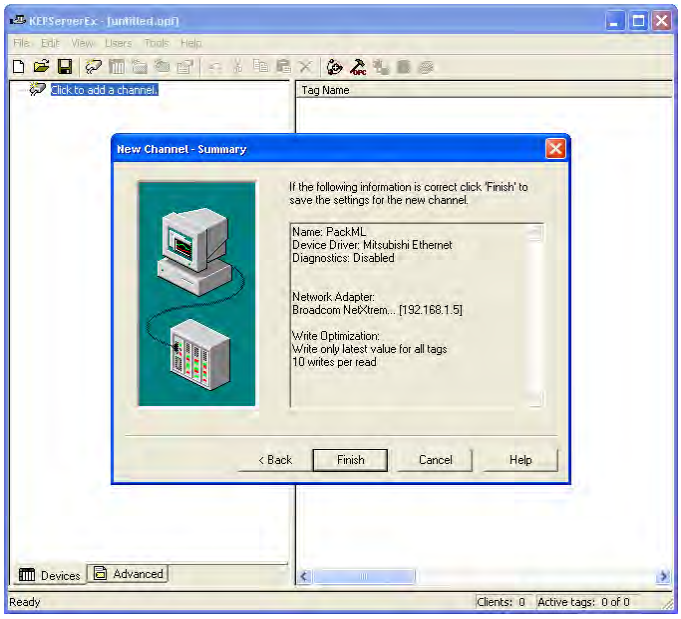


Figure 16 – Completing OPC Channel Configuration

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:

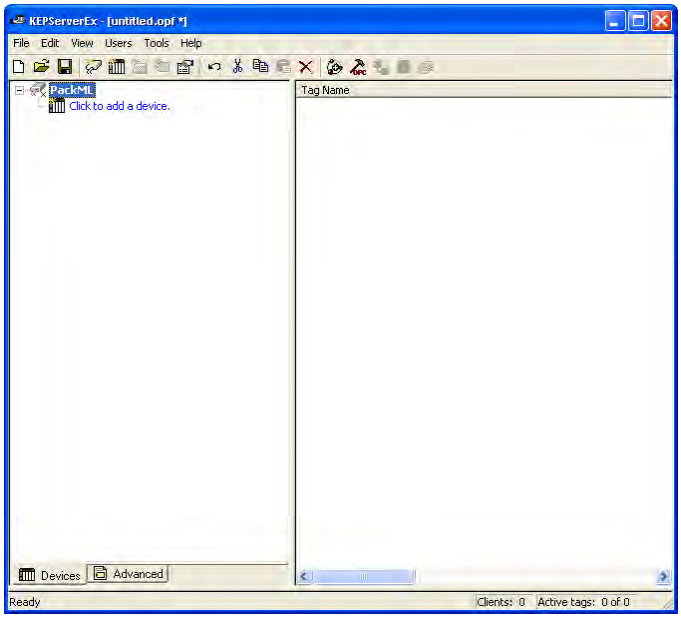


Figure 17 – Adding Device to Channel

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

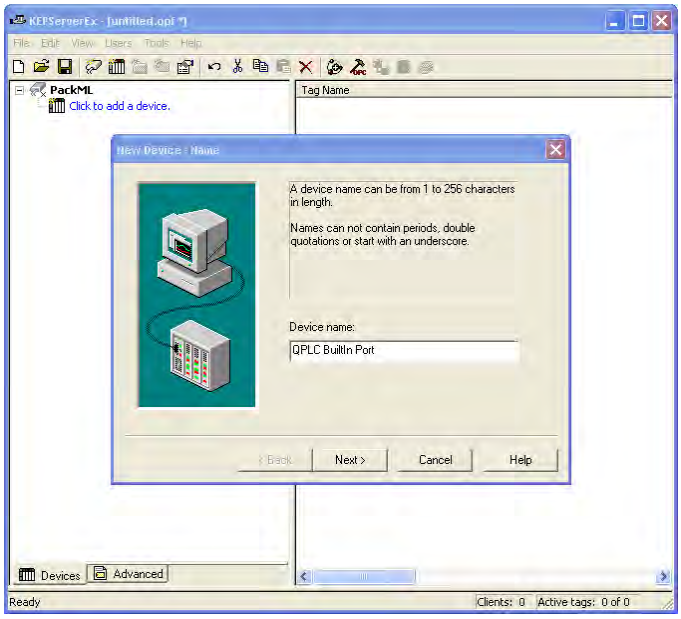


Figure 18 – Naming the Communication Device

- Select Device Model to be “Q Series” from the drop down list:

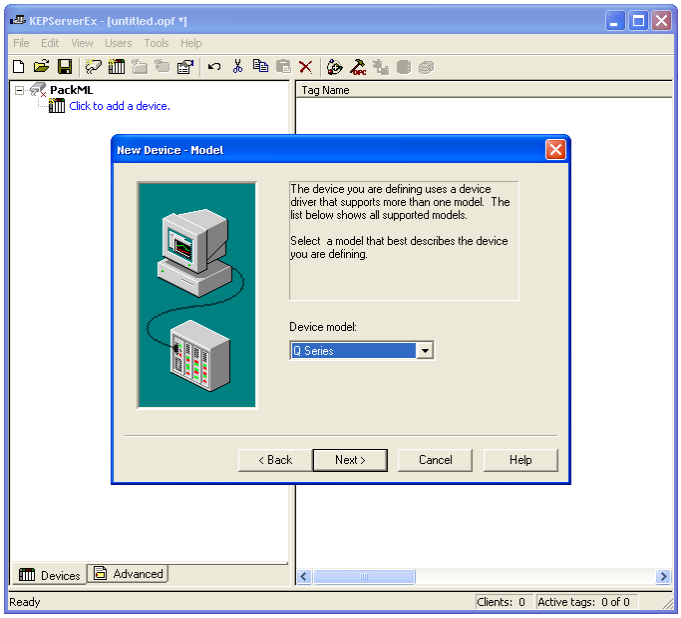


Figure 19 – Selecting the Mitsubishi Device Type

- 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 can not 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.

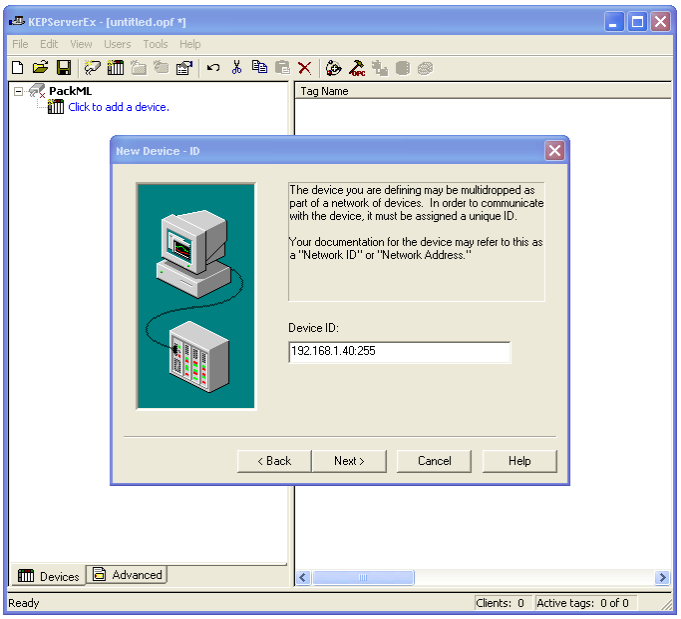


Figure 20 – Entering the Device ID

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

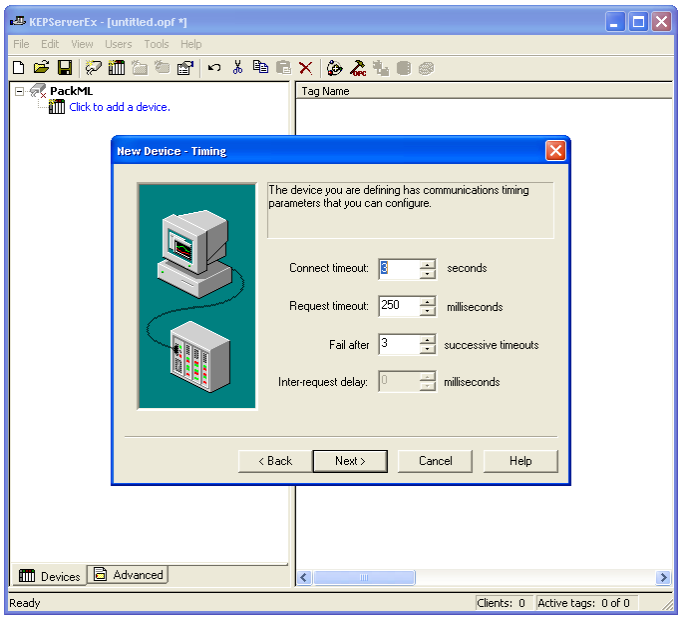


Figure 21 – Selecting the Timing Parameters

- 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.

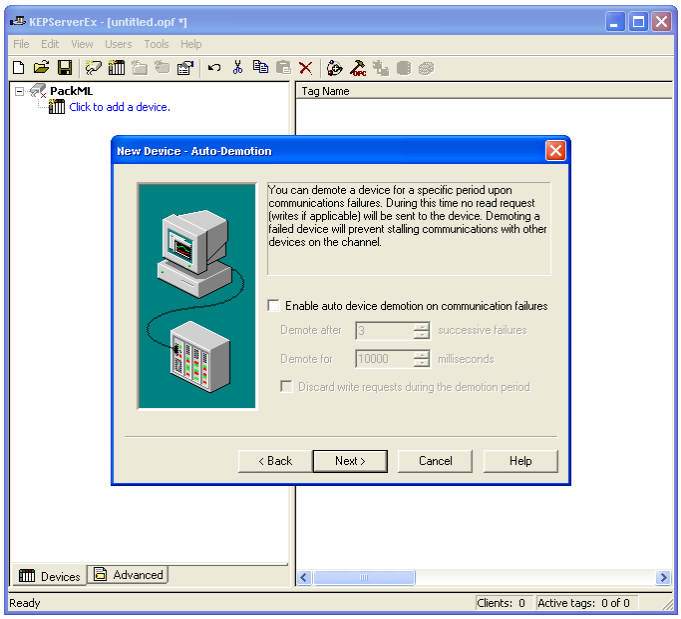


Figure 22 – Configuring Auto-Demotion Function

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

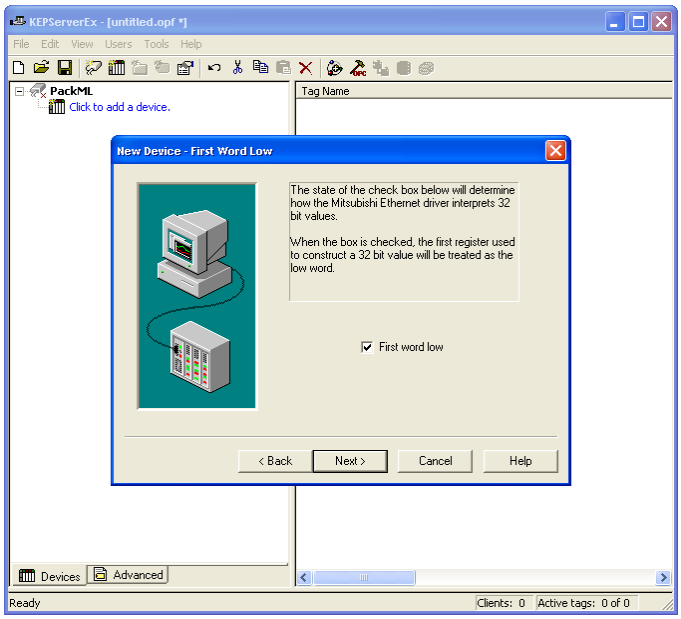


Figure 23 – Selecting Word Order

- 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

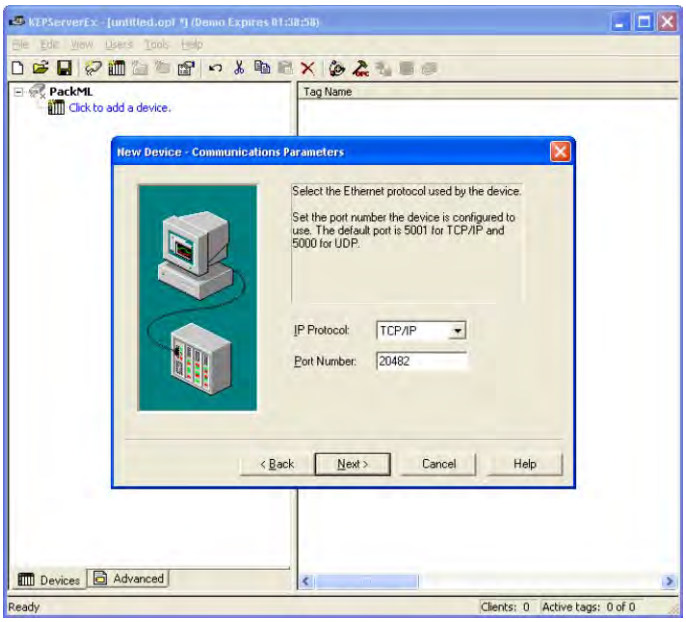


Figure 24 – Configuring IP Protocol and Port Number

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

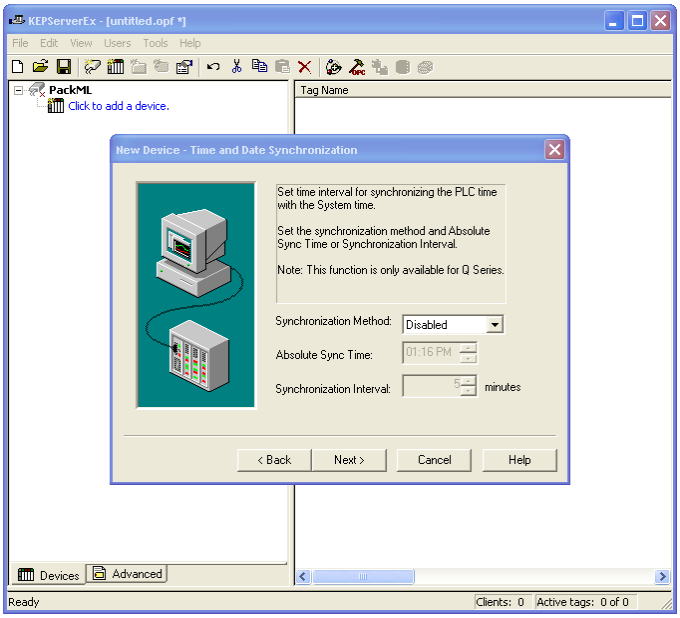


Figure 25 – Selecting Synchronization Method with PLC

Select "Finish" to complete adding the device.

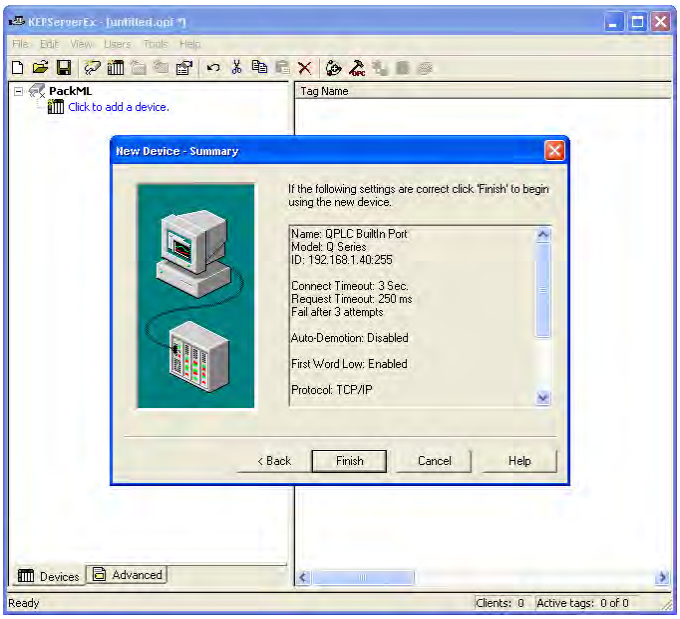


Figure 26 – Completing the Adding Device Process

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…"

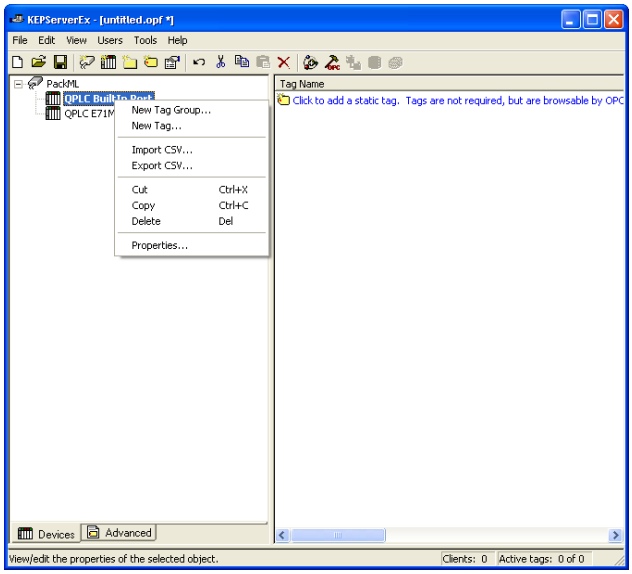


Figure 27 – Creating Tag Groups

- Define the Group Name to be "Command."

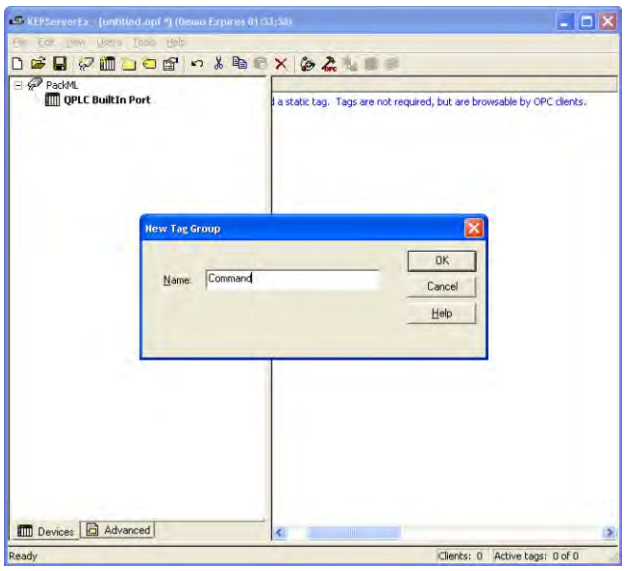


Figure 28 – Adding the Command Group

- Repeat the steps and define Tag Groups.

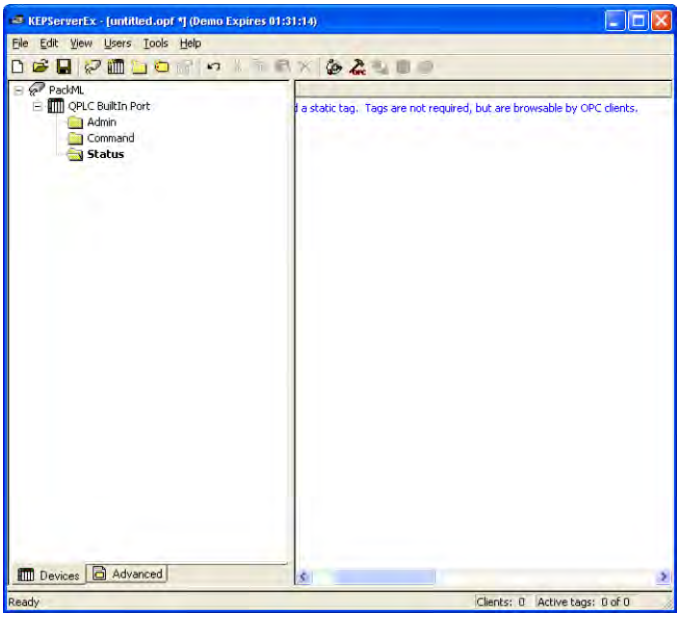


Figure 29 – Adding Other Tag Groups

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

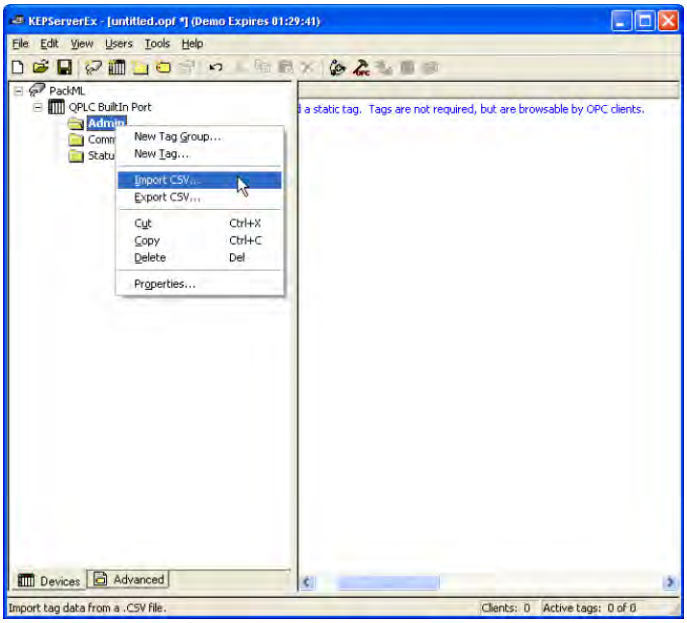


Figure 30 – Importing Tags from CSV Files

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

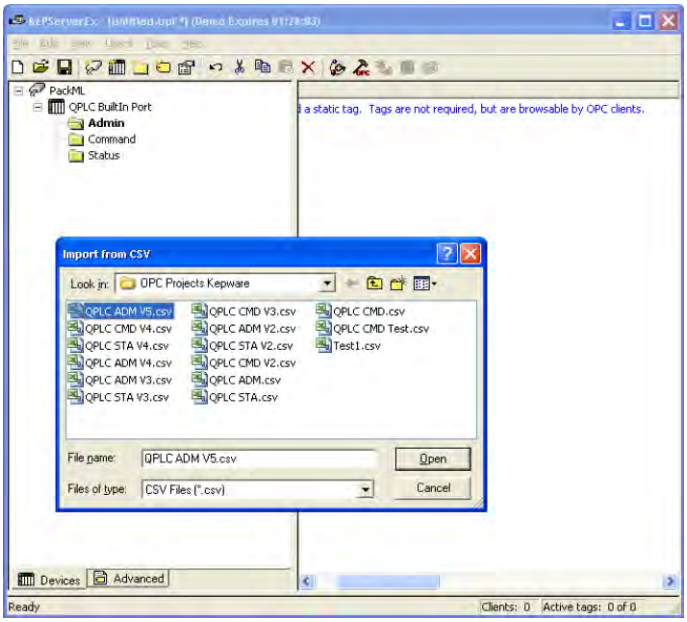


Figure 31 – Selecting the File to Import

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 Works2 File – As a part of the overall iQ Workspace "PackML OEM Template R3 V5"
- Kepware OPC Server File – PackTags Release 3 V5.opf
- OPC Tag files - QPLC ADM R3 V5.csv, QPLC CMD R3 V5.csv, QPLC STA R3 V5.csv

Appendix A

A.1 Command Tags – Spec to GX Works2 Labels

PackTag Specification		GX Works 2 Labels	
Tags	**Data Type**	**Label**	**Data Type**
UnitName.Command.UnitMode	Int (32bit)	gvd_Cmd_UnitMode	Double Word[Signed]
UnitName.Command.UnitModeChangeRequest	Bool	gvb_Cmd_UnitModeChangeRequest	Bit
UnitName.Command.MachSpeed	Real	gvl_Cmd_MachSpeed	FLOAT (Double Precision)
UnitName.Command.MaterialInterlock	Bool Structure	gvdu_Cmd_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]
UnitName.Command.CntrlCmd	Int (32bit)	gvd_Cmd_CntrlCmd	Double Word[Signed]
UnitName.Command.CmdChangeRequest	Bool	gvb_Cmd_CmdChangeRequest	Bit
UnitName.Command.RemoteInterface[#].Number	Int (32bit)	gvda_Cmd_Number_RemInt[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].ControlCmdNumber	Int (32bit)	gvda_Cmd_CntlCmdNum_RemIntf[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].CmdValue	Int (32bit)	gvda_Cmd_CmdValue_RemIntf[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].Parameter[#]	Descriptor Structure	gvsta_Cmd_RemIntf[#]	PackML_Cmd_RemIntf_Type_SDT(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].ID	Int (32bit)	gvsta_Cmd_RemIntf[#].da_ID_Para[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Name	String	gvsta_Cmd_RemIntf[#].sa_Name_Para[#]	String(34)(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Unit	String	gvsta_Cmd_RemIntf[#].sa_Unit_Para[#]	String(34)(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Value	Real	gvsta_Cmd_RemIntf[#].la_Value_Para[#]	FLOAT (Double Precision)(0..9)
UnitName.Command.Parameter[#]	Descriptor Structure	gvsta_Cmd_Parameter[#]	PackML_Cmd_Parameter_SDT(0..9)
UnitName.Command.Parameter[#].ID	Int (32bit)	gvsta_Cmd_Parameter[#].d_ID	Double Word[Signed]
UnitName.Command.Parameter[#].Name	String	gvsta_Cmd_Parameter[#].s_Name	String(32)
UnitName.Command.Parameter[#].Unit	String	gvsta_Cmd_Parameter[#].s_Unit	String(32)
UnitName.Command.Parameter[#].Value	User Defined	gvsta_Cmd_Parameter[#].l_Value	FLOAT (Double Precision)
UnitName.Command.Product[#]	Product Structure	gvsta_Cmd_Pdt[#]	PackML_Cmd_Pdt_Type_SDT(0..4)
UnitName.Command.Product[#].ProductID	Int (32bit)	gvsta_Cmd_PdtID_Pdt[#]	Double Word[Signed](0..4)

UnitName.Command.Product[#].ProcessVariables[#].ID	Int (32bit)	gvsta_Cmd_Pdt[#].da_ID_ProVar[#]	Double Word[Signed](0..9)
UnitName.Command.Product[#].ProcessVariables[#].Name	String	gvsta_Cmd_Pdt[#].sa_Name_ProVar[#]	String(34)(0..9)
UnitName.Command.Product[#].ProcessVariables[#].Unit	String	gvsta_Cmd_Pdt[#].sa_Unit_ProVar[#]	String(34)(0..9)
UnitName.Command.Product[#].ProcessVariables[#].Value	Real	gvsta_Cmd_Pdt[#].la_Value_ProVar[#]	FLOAT (Double Precision)(0..9)
UnitName.Command.Product[#].Ingredients[#].IngredientID	Int (32bit)	gvsta_Cmd_Pdt[#].da_IngID_Ings[#]	Double Word[Signed](0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].ID	Int (32bit)	gvsta_Cmd_Pdt[#].da_ID_Ings_Para[#,#]	Double Word[Signed](0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Name	String	gvsta_Cmd_Pdt[#].sa_Name_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Unit	String	gvsta_Cmd_Pdt[#].sa_Unit_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Value	Real	gvsta_Cmd_Pdt[#].la_Value_Ings_Para[#,#]	FLOAT (Double Precision)(0..9,0..9)

A.2 Command Tags –GX Works2 Labels to Kepware Tags

PackTag Specification		GX Works 2 Labels	
Tags	**Data Type**	**Label**	**Data Type**
UnitName.Command.UnitMode	Int (32bit)	gvd_Cmd_UnitMode	Double Word[Signed]
UnitName.Command.UnitModeChangeRequest	Bool	gvb_Cmd_UnitModeChangeRequest	Bit
UnitName.Command.MachSpeed	Real	gvl_Cmd_MachSpeed	FLOAT (Double Precision)
UnitName.Command.MaterialInterlock	Bool Structure	gvdu_Cmd_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]
UnitName.Command.CntrlCmd	Int (32bit)	gvd_Cmd_CntrlCmd	Double Word[Signed]
UnitName.Command.CmdChangeRequest	Bool	gvb_Cmd_CmdChangeRequest	Bit
UnitName.Command.RemoteInterface[#].Number	Int (32bit)	gvda_Cmd_Number_RemInt[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].ControlCmdNumber	Int (32bit)	gvda_Cmd_CntlCmdNum_RemIntf[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].CmdValue	Int (32bit)	gvda_Cmd_CmdValue_RemIntf[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].Parameter[#]	Descriptor Structure	gvsta_Cmd_RemIntf[#]	PackML_Cmd_RemIntf_Type_SDT(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].ID	Int (32bit)	gvsta_Cmd_RemIntf[#].da_ID_Para[#]	Double Word[Signed](0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Name	String	gvsta_Cmd_RemIntf[#].sa_Name_Para[#]	String(34)(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Unit	String	gvsta_Cmd_RemIntf[#].sa_Unit_Para[#]	String(34)(0..9)
UnitName.Command.RemoteInterface[#].Parameter[#].Value	Real	gvsta_Cmd_RemIntf[#].la_Value_Para[#]	FLOAT (Double Precision)(0..9)
UnitName.Command.Parameter[#]	Descriptor Structure	gvsta_Cmd_Parameter[#]	PackML_Cmd_Parameter_SDT(0..9)
UnitName.Command.Parameter[#].ID	Int (32bit)	gvsta_Cmd_Parameter[#].d_ID	Double Word[Signed]
UnitName.Command.Parameter[#].Name	String	gvsta_Cmd_Parameter[#].s_Name	String(32)
UnitName.Command.Parameter[#].Unit	String	gvsta_Cmd_Parameter[#].s_Unit	String(32)
UnitName.Command.Parameter[#].Value	User Defined	gvsta_Cmd_Parameter[#].l_Value	FLOAT (Double Precision)
UnitName.Command.Product[#]	Product Structure	gvsta_Cmd_Pdt[#]	PackML_Cmd_Pdt_Type_SDT(0..4)
UnitName.Command.Product[#].ProductID	Int (32bit)	gvsta_Cmd_PdtID_Pdt[#]	Double Word[Signed](0..4)
UnitName.Command.Product[#].ProcessVariables[#].ID	Int (32bit)	gvsta_Cmd_Pdt[#].da_ID_ProVar[#]	Double Word[Signed](0..9)
UnitName.Command.Product[#].ProcessVariables[#].Name	String	gvsta_Cmd_Pdt[#].sa_Name_ProVar[#]	String(34)(0..9)

UnitName.Command.Product[#].ProcessVariables[#].Unit	String	gvsta_Cmd_Pdt[#].sa_Unit_ProVar[#]	String(34)(0..9)
UnitName.Command.Product[#].ProcessVariables[#].Value	Real	Cmd_Pdt[#].la_Value_ProVar[#]	FLOAT (Double Precision)(0..9)
UnitName.Command.Product[#].Ingredients[#].IngredientID	Int (32bit)	Cmd_Pdt[#].da_IngID_Ings[#]	Double Word[Signed](0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].ID	Int (32bit)	Cmd_Pdt[#].da_ID_Ings_Para[#,#]	Double Word[Signed](0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Name	String	Cmd_Pdt[#].sa_Name_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Unit	String	Cmd_Pdt[#].sa_Unit_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Command.Product[#].Ingredients[#].Parameter[#].Value	Real	Cmd_Pdt[#].la_Value_Ings_Para[#,#]	FLOAT (Double Precision)(0..9,0..9)

A.3 Status Tags – Spec to GX Works2 Labels

PackTag Specification		GX Works 2 Labels	
Tags	**Data Type**	**Label**	**Data Type**
UnitName.Status.UnitModeCurrent	Int (32bit)	gvd_Sta_UnitModeCurrent	Double Word[Signed]
UnitName.Status.UnitModeRequested	Bool	gvb_Sta_UnitModeChangeRequested	Bit
UnitName.Status.UnitModeChangeInProcess	Bool	gvb_Sta_UnitModeChangeInProcess	Bit
UnitName.Status.StateCurrent	Int (32bit)	gvd_Sta_StateCurrent	Double Word[Signed]
UnitName.Status.StateRequested	Int (32bit)	gvd_Sta_StateRequested	Double Word[Signed]
UnitName.Status.StateChangeInProcess	Bool	gvb_Sta_StateChangeInProcess	Bit
UnitName.Status.MachSpeed	Real	gvl_Sta_MachSpeed	FLOAT (Double Precision)
UnitName.Status.CurMachSpeed	Real	gvl_Sta_CurMachSpeed	FLOAT (Double Precision)
UnitName.Status.MaterialInterlock	Bool Array [32]	gvdu_Sta_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]
UnitName.Status.EquipmentInterlock	Bool Structure [2]	gvst_Sta_EquipmentInterlock	PackML_Sta_EquipInterlock_SDT
UnitName.Status.EquipmentInterlock.Blocked	Bool	gvst_Sta_EquipmentInterlock.b_Blocked	Bit
UnitName.Status.EquipmentInterlock.Starved	Bool	gvst_Sta_EquipmentInterlock.b_Starved	Bit
UnitName.Status.RemoteInterface[#].Number	Int (32bit)	gvda_Sta_Number_RemInt	Double Word[Signed](0..9)
UnitName.Status.RemoteInterface[#].ControlCmdNumber	Int (32bit)	gvda_Sta_CntlCmdNum_RemIntf	Double Word[Signed](0..9)
UnitName.Status.RemoteInterface[#].CmdValue	Int (32bit)	gvda_Sta_CmdValue_RemIntf	Double Word[Signed](0..9)
UnitName.Status.RemoteInterface[#].Parameter[#]	Descriptor Structure	gvsta_Sta_RemIntf[#]	PackML_Sta_RemIntf_Type_SDT(0..9)
UnitName.Status.RemoteInterface[#].Parameter[#].ID	Int (32bit)	gvsta_Sta_RemIntf[#].da_ID_Para[#]	Double Word[Signed](0..9)
UnitName.Status.RemoteInterface[#].Parameter[#].Name	String	gvsta_Sta_RemIntf[#].sa_Name_Para[#]	String(34)(0..9)
UnitName.Status.RemoteInterface[#].Parameter[#].Unit	String	gvsta_Sta_RemIntf[#].sa_Unit[#]	String(34)(0..9)
UnitName.Status.RemoteInterface[#].Parameter[#].Value	Real	gvsta_Sta_RemIntf[#].la_Value[#]	FLOAT (Double Precision)(0..9)
UnitName.Status.Parameter[#]	Descriptor Structure	gvsta_Sta_Parameter[#]	PackML_Sta_Parameter_SDT (0..9)
UnitName.Status.Parameter[#].ID	Int (32bit)	gvsta_Sta_Parameter[#].d_ID	Double Word[Signed]
UnitName.Status.Parameter[#].Name	String	gvsta_Sta_Parameter[#].s_Name	String(32)

UnitName.Status.Parameter[#].Unit	String	gvsta_Sta_Parameter[#].s_Unit	String(32)
UnitName.Status.Parameter[#].Value	User Defined	gvsta_Sta_Parameter[#].l_Value	FLOAT (Double Precision)
UnitName.Status.Product[#]	Product Structure	gvsta_Sta_Pdt[#]	PackML_Sta_Pdt_Type_SDT(0..4)
UnitName.Status.Product[#].ProductID	Int (32bit)	gvsta_Sta_PdtID_Pdt[#]	Double Word[Signed](0..4)
UnitName.Status.Product[#].ProcessVariables[#].ID	Int (32bit)	gvsta_Sta_Pdt[#].da_ID_ProVar	Double Word[Signed](0..9)
UnitName.Status.Product[#].ProcessVariables[#].Name	String	gvsta_Sta_Pdt[#].sa_Name_ProVar	String(34)(0..9)
UnitName.Status.Product[#].ProcessVariables[#].Unit	String	gvsta_Sta_Pdt[#].sa_Unit_ProVar	String(34)(0..9)
UnitName.Status.Product[#].ProcessVariables[#].Value	Real	gvsta_Sta_Pdt[#].la_Value_ProVar	FLOAT (Double Precision)(0..9)
UnitName.Status.Product[#].Ingredients[#].IngredientID	Int (32bit)	gvsta_Sta_Pdt[#].da_IngID_Ings	Double Word[Signed](0..9)
UnitName.Status.Product[#].Ingredients[#].Parameter[#].ID	Int (32bit)	gvsta_Sta_Pdt[#].da_ID_Ings_Para[#,#]	Double Word[Signed](0..9,0..9)
UnitName.Status.Product[#].Ingredients[#].Parameter[#].Name	String	gvsta_Sta_Pdt[#].sa_Name_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Status.Product[#].Ingredients[#].Parameter[#].Unit	String	gvsta_Sta_Pdt[#].sa_Unit_Ings_Para[#,#]	String(34)(0..9,0..9)
UnitName.Status.Product[#].Ingredients[#].Parameter[#].Value	Real	gvsta_Sta_Pdt[#].la_Value_Ings_Para[#,#]	FLOAT (Double Precision)(0..9,0..9)

A.4 Status Tags –GX Works2 Labels to Kepware Tags

Command Tags –GX Works2 Labels to Kepware Tags

GX Works 2 Labels		Kepware Tags	
Label	Data Type	Tags	Data Type
Sta_UnitModeCurrent	Double Word[Signed]		
Sta_UnitModeChangeRequested	Bit		
Sta_UnitModeChangeInProcess	Bit		
Sta_StateCurrent	Double Word[Signed]		
Sta_StateRequested	Double Word[Signed]		
Sta_StateChangeInProcess	Bit		
Sta_MachSpeed	FLOAT (Double Precision)		
Sta_CurMachSpeed	FLOAT (Double Precision)		
Sta_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]		
Sta_EquipmentInterlock	SDT_Sta_EquipInterlock		
Sta_EquipmentInterlock.Blocked	Bit		
Sta_EquipmentInterlock.Starved	Bit		
Sta_Number_RemInt	Double Word[Signed](0..9)		
Sta_CntlCmdNum_RemIntf	Double Word[Signed](0..9)		
Sta_CmdValue_RemIntf	Double Word[Signed](0..9)		
Sta_RemIntf[#]	Sta_RemIntf_Type(0..9)		
Sta_RemIntf[#].ID_Para[#]	Double Word[Signed](0..9)		
Sta_RemIntf[#].Name_Para[#]	String(34)(0..9)		
Sta_RemIntf[#].Unit[#]	String(34)(0..9)		
Sta_RemIntf[#].Value[#]	FLOAT (Double Precision)(0..9)		
Sta_Parameter[#]	SDT_Sta_Parameter(0..9)		
Sta_Parameter[#].ID	Double Word[Signed]		
Sta_Parameter[#].Name	String(32)		
Sta_Parameter[#].Unit	String(32)		
Sta_Parameter[#].Value	FLOAT (Double Precision)		
Sta_Pdt[#]	Sta_Pdt_Type(0..4)		

Sta_PdtID_Pdt[#]	Double Word[Signed](0..4)		
Sta_Pdt[#].ID_ProVar	Double Word[Signed](0..9)		
Sta_Pdt[#].Name_ProVar	String(34)(0..9)		
Sta_Pdt[#].Unit_ProVar	String(34)(0..9)		
Sta_Pdt[#].Value_ProVar	FLOAT (Double Precision)(0..9)		
Sta_Pdt[#].IngID_Ings	Double Word[Signed](0..9)		
ID_Ings_Para[#,#]	Double Word[Signed](0..9,0..9)		
Sta_Pdt[#].Name_Ings_Para[#,#]	String(34)(0..9,0..9)		
Sta_Pdt[#].Unit_Ings_Para[#,#]	String(34)(0..9,0..9)		
Sta_Pdt[#].Value_Ings_Para[#,#]	FLOAT (Double Precision)(0..9,0..9)		
Sta_Pdt.Value_Ings_Para	FLOAT (Double Precision)(0..9,0..9)		

A.5 Admin Tags – Spec to GX Works2 Labels

PackTag Specification		GX Works 2 Labels	
Tags	**Data Type**	**Label**	**Data Type**
UnitName.Admin.Parameter[#]	Descriptor Structure	gvst_Adm_Parameter	PackML_Admin_Parameter_SDT
UnitName.Admin.Parameter[#].ID	Int (32bit)	gvst_Adm_Parameter[#].d_ID	Double Word[Signed]
UnitName.Admin.Parameter[#].Name	String	gvst_Adm_Parameter[#].s_Name	String(32)
UnitName.Admin.Parameter[#].Unit	String	gvst_Adm_Parameter[#].s_Unit	String(32)
UnitName.Admin.Parameter[#].Value	Real	gvst_Adm_Parameter[#].l_Value	FLOAT (Double Precision)
UnitName.Admin.Alarm[#]	Alarm Structure	gvsta_Adm_Alarm[#]	PackML_Admin_Alarm_SDT(0..255)
UnitName.Admin.Alarm[#].Trigger	Bool	gvsta_Adm_Alarm[#].b_Trigger	Bit
UnitName.Admin.Alarm[#].ID	Int (32bit)	gvsta_Adm_Alarm[#].d_ID	Double Word[Signed]
UnitName.Admin.Alarm[#].Value	Int (32bit)	gvsta_Adm_Alarm[#].d_Value	Double Word[Signed]
UnitName.Admin.Alarm[#].Message	String	gvsta_Adm_Alarm[#].s_Message	String(32)
UnitName.Admin.Alarm[#].Category (Event Grouping)	Int (32bit)	gvsta_Adm_Alarm[#].d_Category	Double Word[Signed]
UnitName.Admin.Alarm[#].DateTime[0]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[1]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[2]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[3]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[4]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[5]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].DateTime[6]	Int (32bit)	gvsta_Adm_Alarm[#].wu_DateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[0]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[1]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[2]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[3]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[4]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[5]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Alarm[#].AckDateTime[6]	Int (32bit)	gvsta_Adm_Alarm[#].wu_AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]

UnitName.Admin.AlarmExtent	Int(32bit)	gvd_Adm_AlarmExtent	Double Word[Signed]
UnitName.Admin.AlarmHistory[#]	Alarm Structure	gvst_Adm_AlarmHistory[#]	SDT_Admin_Alarm(0..255)
UnitName.Admin.AlarmHistory[#].Trigger	Bool	gvst_Adm_AlarmHistory[#].b_Trigger	Bit
UnitName.Admin.AlarmHistory[#].ID	Int (32bit)	gvst_Adm_AlarmHistory[#].d_ID	Double Word[Signed]
UnitName.Admin.AlarmHistory[#].Value	Int (32bit)	gvst_Adm_AlarmHistory[#].d_Value	Double Word[Signed]
UnitName.Admin.AlarmHistory[#].Message	String	gvst_Adm_AlarmHistory[#].s_Message	String(32)
UnitName.Admin.AlarmHistory[#].Category (Event Grouping)	Int (32bit)	gvst_Adm_AlarmHistory[#].d_Category	Double Word[Signed]
UnitName.Admin.AlarmHistory[#].DateTime[0]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[1]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[2]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[3]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[4]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[5]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].DateTime[6]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_DateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[0]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[1]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[2]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[3]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[4]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[5]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistory[#].AckDateTime[6]	Int (32bit)	gvst_Adm_AlarmHistory[#].wu_AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.AlarmHistoryExtent	Int (32bit)	gvd_Adm_AlarmStopReasonExtent	Double Word[Signed]

UnitName.Admin.StopReason	Alarm Structure	gvst_Adm_AlarmStopReason	PackML_Admin_Alarm_SDT
UnitName.Admin.StopReason.Trigger	Bool	gvst_Adm_AlarmStopReason.b_Trigger	Bit
UnitName.Admin.StopReason.ID	Int (32bit)	gvst_Adm_AlarmStopReason.d_ID	Double Word[Signed]
UnitName.Admin.StopReason.Value	Int (32bit)	gvst_Adm_AlarmStopReason.d_Value	Double Word[Signed]
UnitName.Admin.StopReason.Message	String	gvst_Adm_AlarmStopReason.s_Message	String(32)
UnitName.Admin.StopReason.Category (Event Grouping)	Int (32bit)	gvst_Adm_AlarmStopReason.d_Category	Double Word[Signed]
UnitName.Admin.StopReason[#].DateTime[0]	Int (32bit)	gvst_Adm_AlarmStopReason.wu_DateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[1]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[2]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[3]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[4]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[5]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].DateTime[6]	Int (32bit)	gvst_Adm_AlarmStopReason. wu_DateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[0]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[1]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[2]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[3]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[4]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[5]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReason[#].AckDateTime[6]	Int (32bit)	gvst_Adm_AlarmStopReason[#].wu_AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.StopReasonExtent	Int (32bit)	gvd_Adm_AlarmStopReasonExtent	Double Word[Signed]
UnitName.Admin.Warning[#]	Alarm Structure	gvst_Adm_AlarmWarning[#]	PackML_Admin_Alarm_SDT
UnitName.Admin.Warning[#].Trigger	Bool	gvst_Adm_AlarmWarning[#].b_Trigger	Bit
UnitName.Admin.Warning[#].ID	Int (32bit)	gvst_Adm_AlarmWarning[#].d_ID	Double Word[Signed]
UnitName.Admin.Warning[#].Value	Int (32bit)	gvst_Adm_AlarmWarning[#].d_Value	Double Word[Signed]

UnitName.Admin.Warning[#].Message	String	gvst_Adm_AlarmWarning[#].s_Message	String(32)
UnitName.Admin.Warning[#].Category (Event Grouping)	Int (32bit)	gvst_Adm_AlarmWarning[#].d_Category	Double Word[Signed]
UnitName.Admin.Warning[#].DateTime[0]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[1]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[2]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[3]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[4]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[5]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].DateTime[6]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_DateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[0]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[1]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[2]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[3]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[4]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[5]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.Warning[#].AckDateTime[6]	Int (32bit)	gvst_Adm_AlarmWarning[#].wu_AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.WarningExtent	Int (32bit)	gvd_Adm_AlarmWarningExtent	Double Word[Signed]
UnitName.Admin.ModeCurrentTime[#]	Int (32bit)	gvda_Adm_ModeCurrentTime[#]	Double Word[Signed](0..31)
UnitName.Admin.ModeCumulativeTime[#]	Int (32bit)	gvda_Adm_ModeCumulativeTime[#]	Double Word[Signed](0..31)
UnitName.Admin.StateCurrentTime[#,#] (Mode,State)	Int (32bit)	gvda_Adm_StateCurrentTime[#,#]	Double Word[Signed](0..31,0..17)
UnitName.Admin.StateCumulativeTime[#,#] (Mode,State)	Int (32bit)	gvda_Adm_StateCumulativeTime[#,#]	Double Word[Signed](0..31,0..17)
UnitName.Admin.ProdConsumedCount[#]	Count Structure	gvsta_Adm_ProdConsumedCnt[#]	PackML_Admin_Count_SDT(0..9)
UnitName.Admin.ProdConsumedCount[#].ID	Int(32bit)	gvsta_Adm_ProdConsumedCnt[#].d_ID	Double Word[Signed]
UnitName.Admin.ProdConsumedCount[#].Name	String	gvsta_Adm_ProdConsumedCnt[#].s_Name	String(32)
UnitName.Admin.ProdConsumedCount[#].Unit	String	gvsta_Adm_ProdConsumedCnt[#].s_Unit	String(32)
UnitName.Admin.ProdConsumedCount[#].Count	Int(32bit)	gvsta_Adm_ProdConsumedCnt[#].d_Count	Double Word[Signed]

UnitName.Admin.ProdConsumedCount[#].AccCount	Int(32bit)	gvsta_Adm_ProdConsumedCnt[#].d_AccCount	Double Word[Signed]
UnitName.Admin.ProdProcessedCount[#]	Count Structure	gvsta_Adm_ProdProcessedCnt[#]	PackML_Admin_Count_SDT(0..9)
UnitName.Admin.ProdProcessedCount[#].ID	Int(32bit)	gvsta_Adm_ProdProcessedCnt[#].d_ID	Double Word[Signed]
UnitName.Admin.ProdProcessedCount[#].Name	String	gvsta_Adm_ProdProcessedCnt[#].s_Name	String(32)
UnitName.Admin.ProdProcessedCount[#].Unit	String	gvsta_Adm_ProdProcessedCnt[#].s_Unit	String(32)
UnitName.Admin.ProdProcessedCount[#].Count	Int(32bit)	gvsta_Adm_ProdProcessedCnt[#].d_Count	Double Word[Signed]
UnitName.Admin.ProdProcessedCount[#].AccCount	Int(32bit)	gvsta_Adm_ProdProcessedCnt[#].d_AccCount	Double Word[Signed]
UnitName.Admin.ProdDefectiveCount[#]	Count Structure	gvsta_Adm_ProdDefectiveCnt[#]	PackML_Admin_Count_SDT(0..9)
UnitName.Admin.ProdDefectiveCount[#].ID	Int(32bit)	gvsta_Adm_ProdDefectiveCnt[#].d_ID	Double Word[Signed]
UnitName.Admin.ProdDefectiveCount[#].Name	String	gvsta_Adm_ProdDefectiveCnt[#].s_Name	String(32)
UnitName.Admin.ProdDefectiveCount[#].Unit	String	gvsta_Adm_ProdDefectiveCnt[#].s_Unit	String(32)
UnitName.Admin.ProdDefectiveCount[#].Count	Int(32bit)	gvsta_Adm_ProdDefectiveCnt[#].d_Count	Double Word[Signed]
UnitName.Admin.ProdDefectiveCount[#].AccCount	Int(32bit)	gvsta_Adm_ProdDefectiveCnt[#].d_AccCount	Double Word[Signed]
UnitName.Admin.AccTimeSinceReset	Int(32bit)	gvsta_Adm_AccTimeSinceReset	Double Word[Signed]
UnitName.Admin.MachDesignSpeed	Real	gvsta_Adm_MachDesignSpeed	FLOAT (Double Precision)
UnitName.Admin.StatesDisabled	Int(32bit)	gvsta_Adm_StatesDisabled	Double Word[Signed]
UnitName.Admin.PLCDateTime	Date-Time Array	gvwa_Adm_PLCDateTime_Date	Word[Unsigned]/Bit String[16-bit](0..6)
UnitName.Admin.PLCDateTime[0]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[0]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[1]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[1]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[2]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[2]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[3]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[3]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[4]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[4]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[5]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[5]	Word[Unsigned]/Bit String[16-bit]
UnitName.Admin.PLCDateTime[6]	Int (32bit)	gvwa_Adm_PLCDateTime_Date[6]	Word[Unsigned]/Bit String[16-bit]

A.6 Admin Tags –GX Works2 Labels to Kepware Tags

GX Works 2 Labels		Kepware Tags	
Label	Data Type	Tags	Data Type
Adm_Parameter	SDT_Admin_Parameter(0..19)		
Adm_Parameter[#].ID	Double Word[Signed]		
Adm_Parameter[#].Name	String(32)		
Adm_Parameter[#].Unit	String(32)		
Adm_Parameter[#].Value	FLOAT (Double Precision)		
Adm_Alarm[#]	SDT_Admin_Alarm(0..255)		
Adm_Alarm[#].Trigger	Bit		
Adm_Alarm[#].ID	Double Word[Signed]		
Adm_Alarm[#].Value	Double Word[Signed]		
Adm_Alarm[#].Message	String(32)		
Adm_Alarm[#].Category	Double Word[Signed]		
Adm_Alarm[#].DateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].DateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]		

Adm_Alarm[#].AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_Alarm[#].AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmExtent	Double Word[Signed]		
Adm_AlarmHistory[#]	SDT_Admin_Alarm(0..255)		
Adm_AlarmHistory[#].Trigger	Bit		
Adm_AlarmHistory[#].ID	Double Word[Signed]		
Adm_AlarmHistory[#].Value	Double Word[Signed]		
Adm_AlarmHistory[#].Message	String(32)		
Adm_AlarmHistory[#].Category	Double Word[Signed]		
Adm_AlarmHistory[#].DateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].DateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmHistory[#].AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReasonExtent	Double Word[Signed]		
Adm_AlarmStopReason	SDT_Admin_Alarm		
Adm_AlarmStopReason.Trigger	Bit		
Adm_AlarmStopReason.ID	Double Word[Signed]		
Adm_AlarmStopReason.Value	Double Word[Signed]		
Adm_AlarmStopReason.Message	String(32)		
Adm_AlarmStopReason.Category	Double Word[Signed]		

Adm_AlarmStopReason.DateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason.DateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReason[#].AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmStopReasonExtent	Double Word[Signed]		
Adm_AlarmWarning[#]	SDT_Admin_Alarm(0..255)		
Adm_AlarmWarning[#].Trigger	Bit		
Adm_AlarmWarning[#].ID	Double Word[Signed]		
Adm_AlarmWarning[#].Value	Double Word[Signed]		
Adm_AlarmWarning[#].Message	String(32)		
Adm_AlarmWarning[#].Category	Double Word[Signed]		
Adm_AlarmWarning[#].DateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].DateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].DateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].DateTime[3]	Word[Unsigned]/Bit String[16-bit]		

Adm_AlarmWarning[#].DateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].DateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].DateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarning[#].AckDateTime[6]	Word[Unsigned]/Bit String[16-bit]		
Adm_AlarmWarningExtent	Double Word[Signed]		
Adm_ModeCurrentTime[#]	Double Word[Signed](0..31)		
Adm_ModeCumulativeTime[#]	Double Word[Signed](0..31)		
Adm_StateCurrentTime[#,#]	Double Word[Signed](0..31,0..17)		
Adm_StateCumulativeTime[#,#]	Double Word[Signed](0..31,0..17)		
Adm_ProdConsumedCnt[#]	SDT_Admin_Count(0..9)		
Adm_ProdConsumedCnt[#].ID	Double Word[Signed]		
Adm_ProdConsumedCnt[#].Name	String(32)		
Adm_ProdConsumedCnt[#].Unit	String(32)		
Adm_ProdConsumedCnt[#].Count	Double Word[Signed]		
Adm_ProdConsumedCnt[#].AccCount	Double Word[Signed]		
Adm_ProdProcessedCnt[#]	SDT_Admin_Count(0..9)		
Adm_ProdProcessedCnt[#].ID	Double Word[Signed]		
Adm_ProdProcessedCnt[#].Name	String(32)		
Adm_ProdProcessedCnt[#].Unit	String(32)		
Adm_ProdProcessedCnt[#].Count	Double Word[Signed]		
Adm_ProdProcessedCnt[#].AccCount	Double Word[Signed]		
Adm_ProdDefectiveCnt[#]	SDT_Admin_Count(0..9)		
Adm_ProdDefectiveCnt[#].ID	Double Word[Signed]		
Adm_ProdDefectiveCnt[#].Name	String(32)		

Adm_ProdDefectiveCnt[#].Unit	String(32)		
Adm_ProdDefectiveCnt[#].Count	Double Word[Signed]		
Adm_ProdDefectiveCnt[#].AccCount	Double Word[Signed]		
Adm_AccTimeSinceReset	Double Word[Signed]		
Adm_MachDesignSpeed	FLOAT (Double Precision)		
Adm_StatesDisabled	Double Word[Signed]		
Adm_PLCDateTime_Date	Word[Unsigned]/Bit String[16-bit](0..6)		
Adm_PLCDateTime_Date[0]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[1]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[2]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[3]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[4]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[5]	Word[Unsigned]/Bit String[16-bit]		
Adm_PLCDateTime_Date[6]	Word[Unsigned]/Bit String[16-bit]		

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 4 – PackML Library and Core Function Blocks

Release 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R2 V1.1	August 12, 2010	Minor corrections of figure and heading reference errors.
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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 PackML Library and the implementation of four Mitsubishi PackML core function blocks that handle the PackML Machine State Transitions, Mode Manager, State and Mode Timers, State Disable Set and State Transition Set

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

2 Overview of PackML Library

PackML Library is a GX Works 2 “User Library” that contains all the four core PackML function blocks and the PackTags. This allows users to simply install the “PackML_R3_V1.sul” library file to a new or an existing project to be able to access the PackTags labels and function blocks.

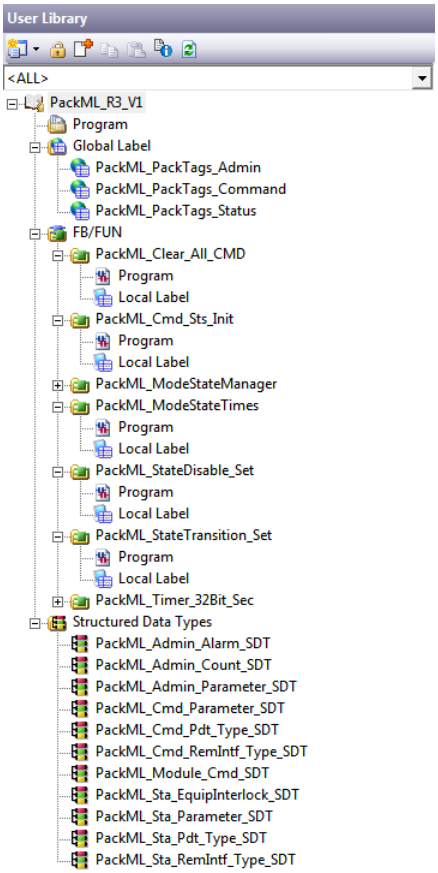


Figure 1: PackML Library

2.1 PackML Library Items

The PackML Library contains the following global label groups:

- PackML_PackTags_Admin
- PackML_PackTags_Command
- PackML_PackTags_Status

The following function blocks are part of the library:

- PackML_ModeStateManager
- PackML_ModeStateTimes
- PackML_StateDisable_Set
- PackML_StateTransition_Set

Structured data types associated with the PackML_PackTags labels are declared here

- PackML_Admin_Alarm_SDT
- PackML_Admin_Count_SDT
- PackML_Admin_Parameter_SDT
- PackML_Cmd_Paramter_SDT
- PackML_Cmd_Pdt_Type_SDT
- PackML_Cmd_RemIntf_SDT
- PackML_Module_Cmd_SDT
- PackML_Sta_EquipInterlock_SDT
- PackML_Sta_Parameter_SDT
- PackML_Sta_Pdt_Type
- PackML_Sta_RemIntf_Type_SDT

2.2 Installing the PackML User Library

To install a user library, switch to the User Library view in the project Navigation Window.

While in the User Library view, either select the ”Install” option in the Project ⇒ Library menu or click on the down arrow of the New Library button and select ”Install”.

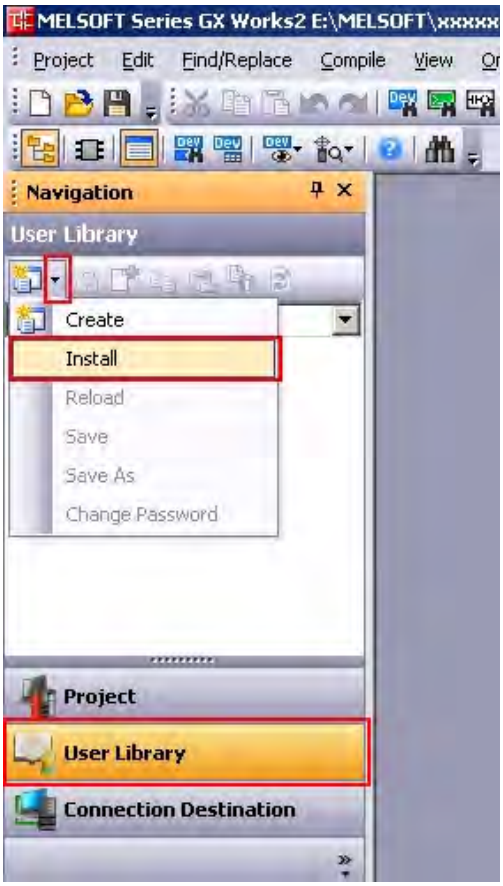


Figure 2: PackML Library Install

In the Install Library dialog window press the ”Browse...” button and select the *.sul user library file in its current location. After the selection of the library file, press the OK button to install the library.

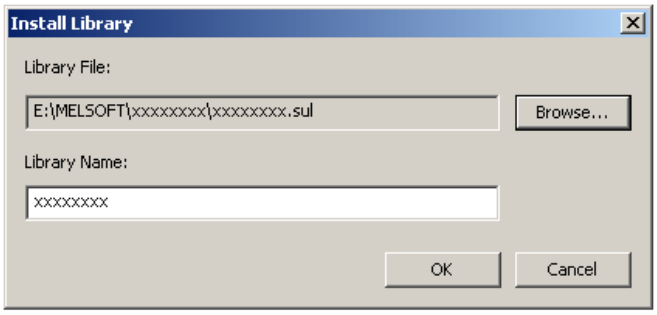


Figure 3: Library Selection

Once the library is successfully installed, it is possible to use the provided function blocks in the project.

2.3 PackML User Library Help

The PackML Library also contains a help file that can be associated to display the help from the GX Works2 programming software. When using any of the function blocks, the user can select one of the function library blocks and press “F1” to bring up help for that specific function block. The help file “PackML R3 V1.CHM” is included in the library. To associate this file to the library function blocks, follow the procedure below.

Select the library and right-click on it. In the displayed menu, select "Open" to open the library for editing.

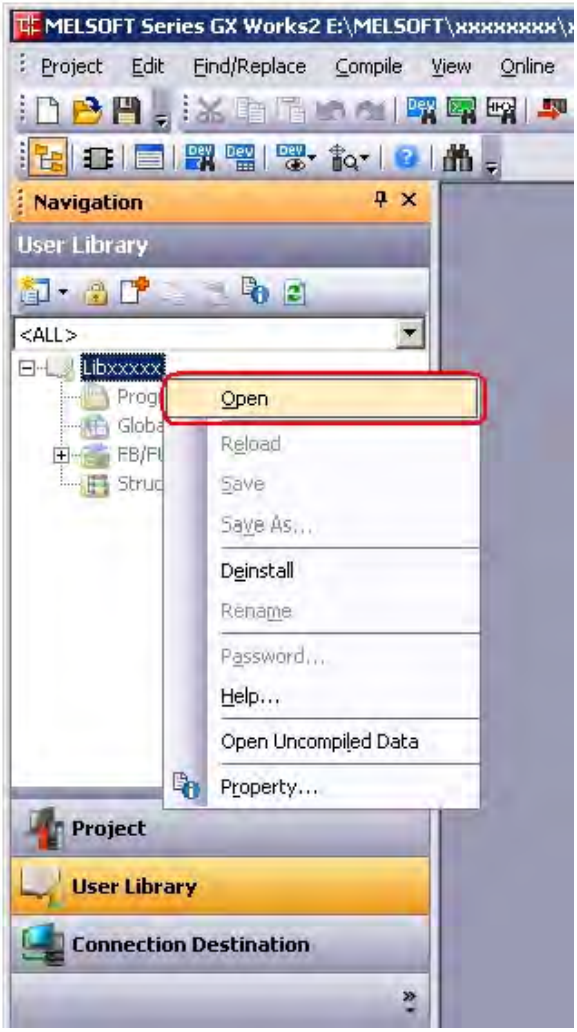


Figure 4: Library Selection

Once the library is open, select it again and righ-click on it. In the displayed menu, select "Property..." to display the current library properties (Title, associated help file, library comment).

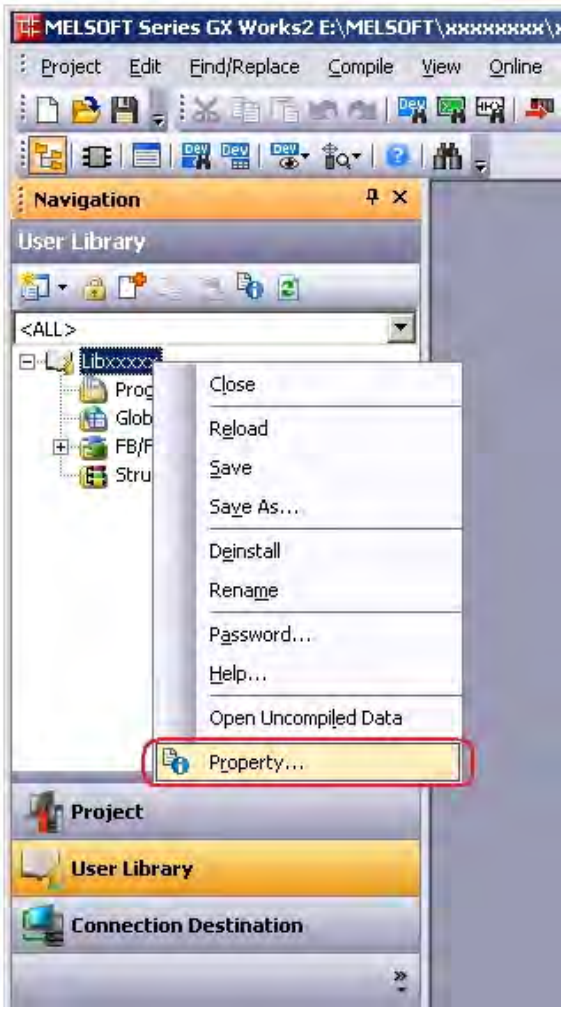


Figure 5: Library Property

In the Property screen, click the "Browse" button and select the HTML Help file that will be associated to the library, then click the "OK" button to register the new settings.

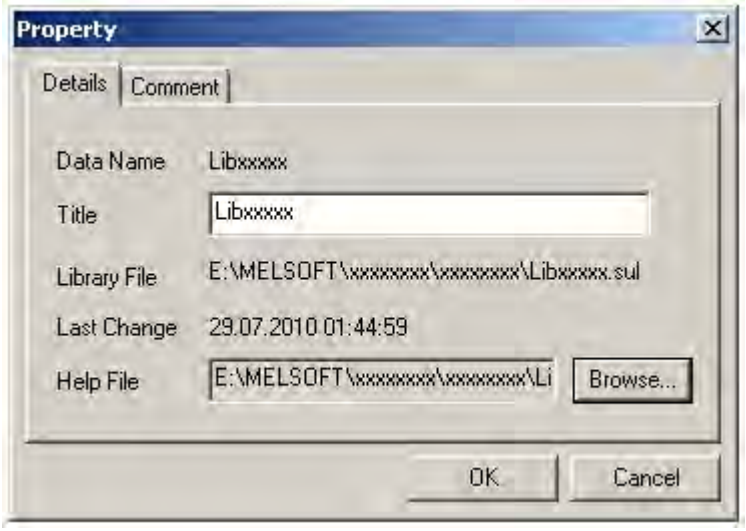


Figure 6: Help file selection

To display the help file for a library, whether it is open for editing or closed, in the User Library view, select the library and right-click on it. In the displayed menu, select "Help..." to open the associated help file and view the contained information.

To use the help file, drop the FB in the program, select the FB and then press F1 to bring up the help menu.

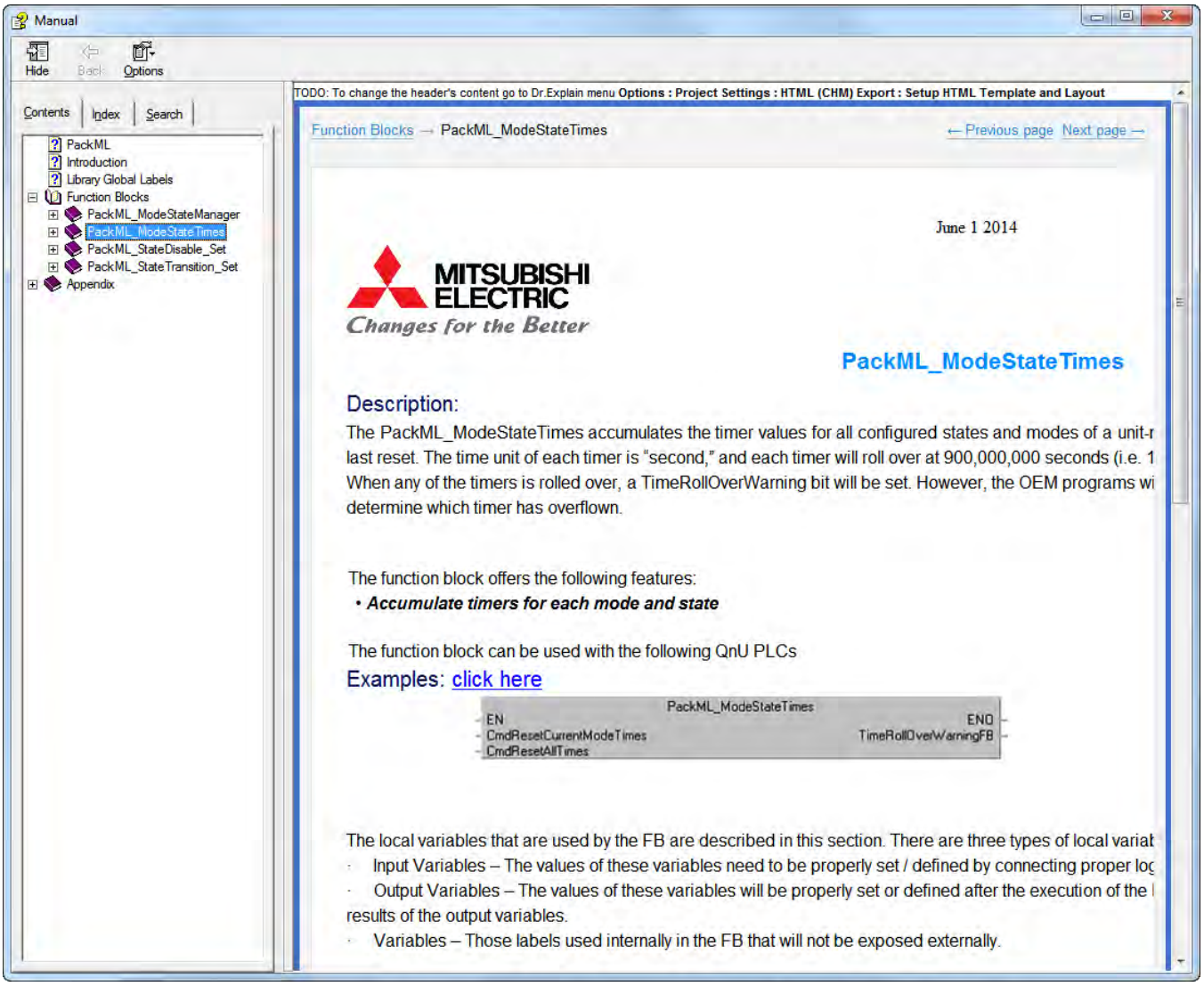


Figure 7: Help Menu

3 Overview of PackML State and Mode Core Function Blocks

There are two key Mitsubishi PackML State and Mode core function blocks included with the library:

- PackML_ModeStateManager
- PackML_ModeStateTimes

There are also two additional function blocks for setting up the states and transitions.

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.

4 Function Block: PackML_ModeStateManager

4.1 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 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.

4.2 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:

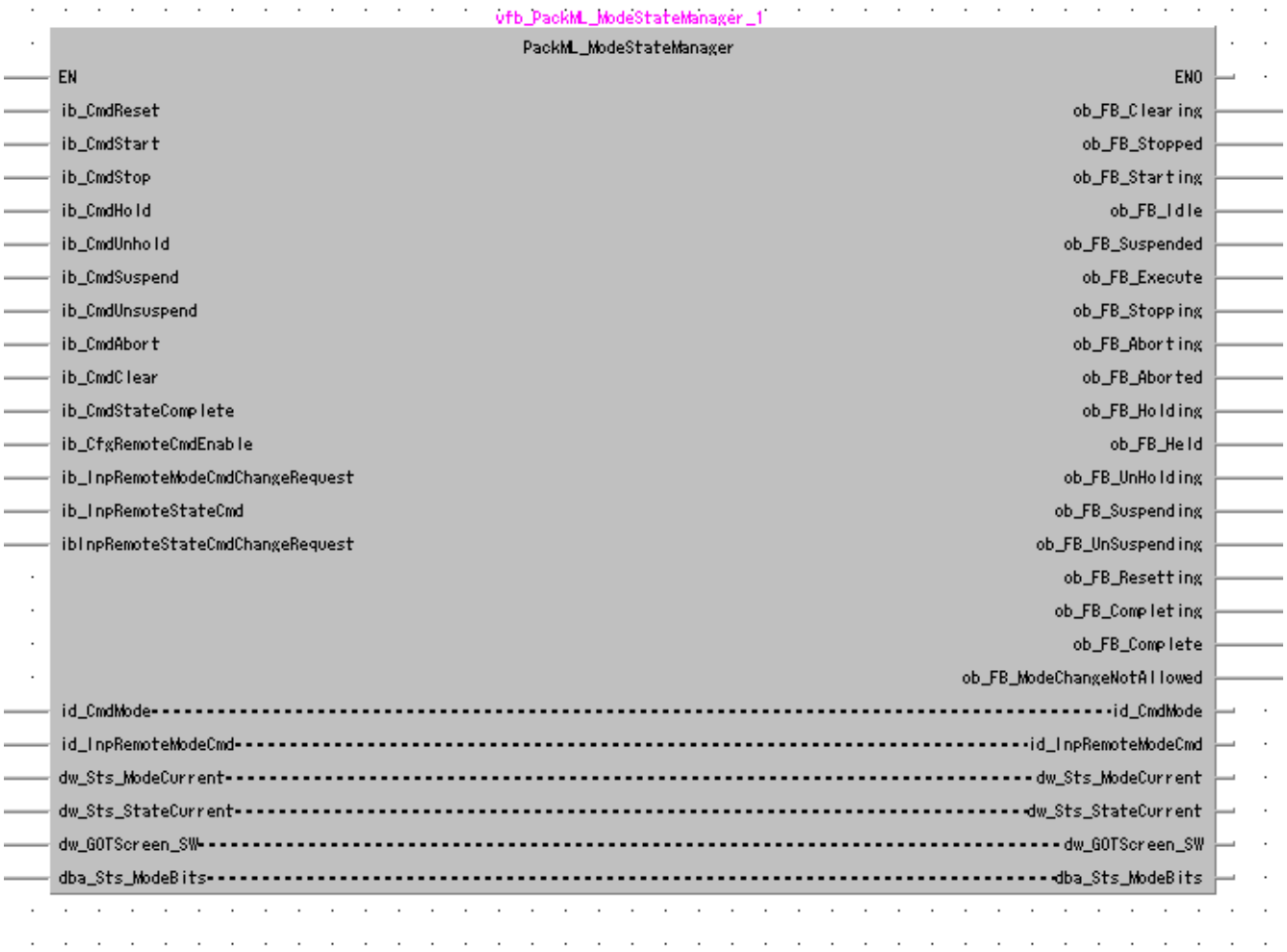


Figure 8 PackML_ModeStateManager Function Block with Inputs and Outputs

4.3 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

Variable Type	Variable Label	Data Type	Description
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.
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.

Variable Type	Variable Label	Data Type	Description
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	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_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	ib_InpRemoteModeCmdChangeRequest	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_InpRemoteStateCmd	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.
VAR_INPUT	ibInpRemoteStateCmdChangeRequest	Bit	Set this bit to allow changing states from remote command

Variable Type	Variable Label	Data Type	Description
VAR_IN_OUT	dw_Sts_ModeCurrent	Double Word	This label contains the current PackML mode
VAR_IN_OUT	dw_Sts_StateCurrent	Double Word	This label contains the current state
VAR_IN_OUT	dw_GOTScreen_SW	Word[Signed]	This is the GOT screen selection label
VAR_IN_OUT	dba_Sts_ModeBits	Bit(0..31)	This is the label that contains the current mode
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.

5 Function Block: PackML_ModeStateTimes

5.1 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.

5.2 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 9 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	od_Current_Timer_Value_FB	Double Word	Contains the current timer value.
VAR_INPUT	id_Start_Timer_Value_FB	Double Word	Contains the initial value of the timer before it starts accumulating.
VAR_INPUT	ib_Timer_Enable_FB	Bit	Enables the timer to start accumulating.
VAR_INPUT	ib_Timer_Reset_FB	Bit	Resets the timer value to the Start Timer Value. The reset is effective only when the timer is enabled.

5.3 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:

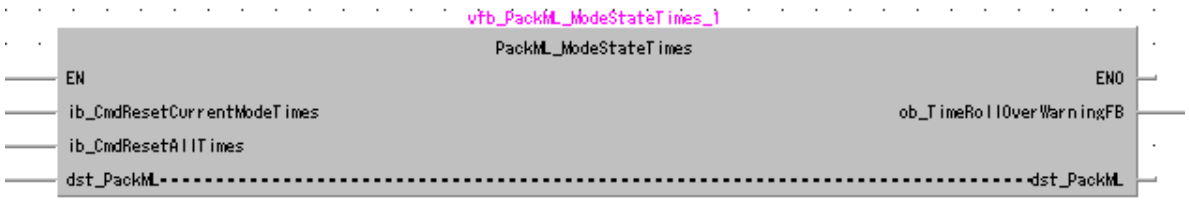


Figure 10 PackML_ModeStateTimes Function Block

5.4 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	Bit	This label contains all the current and cumulative timer values for each mode and state

5 Function Block: PackML_StateDisable_Set

5.1 Description

The PackML_StateDisable_Set FB is used to enable or disable states for each mode. Set the inputs to “FALSE” to enable and “TRUE” to disable that state.

The following PackML states can be disabled or enabled using this FB.

- Clearing
- Starting
- Suspended
- Stopping

- Aborting
- Holding
- Suspending
- Unsuspending
- Resetting
- Completing
- Complete

The following core PackML states cannot be disabled as they are part of the "minimum" defined states and must be present in a PackML compliant system.

- Aborted
- Stopped
- Execute
- Idle

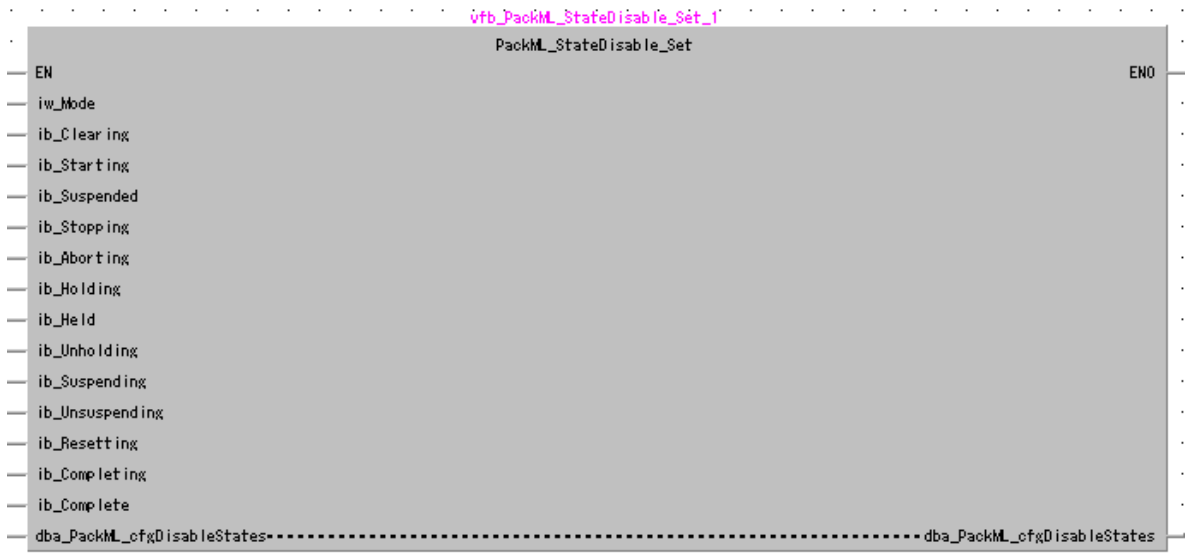


Figure 11 PackML_StateDisable_Set Function Block

5.2 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_IN/OUT	dba_FB_PackMLcfgDisableStates	Bit (0..31,0..17)	Contains the PackML disabled states configuration
VAR_INPUT	ib_Clearing	Bit	Disable/enable the Clearing state
VAR_INPUT	ib_Starting	Bit	Disable/enable the Starting state
VAR_INPUT	ib_Suspending	Bit	Disable/enable the Suspending state
VAR_INPUT	ib_Stopping	Bit	Disable/enable the Stopping state
VAR_INPUT	ib_Aborting	Bit	Disable/enable the Aborting state
VAR_INPUT	ib_Holding	Bit	Disable/enable the Holding state
VAR_INPUT	ib_Held	Bit	Disable/enable the Held state
VAR_INPUT	ib_Unholding	Bit	Disable/enable the Unholding state

VAR_INPUT	ib_Suspending	Bit	Disable/enable the Suspending state
VAR_INPUT	ib_Unsuspending	Bit	Disable/enable the Unsuspending state
VAR_INPUT	ib_Resetting	Bit	Disable/enable the Resetting state
VAR_INPUT	ib_Completing	Bit	Disable/enable the Completing state
VAR_INPUT	ib_Complete	Bit	Disable/enable the Complete state

6 Function Block: PackML_StateTransition_Set

6.1 Description

The PackML_StateTransition_Set FB is used to enable or disable transitions between states. Set the inputs to "FALSE" to disable and "TRUE" to enable that transition.

The following transitions between these states can be disabled or enabled using this FB.

- Stopped
- Idle
- Suspended
- Aborted
- Held
- Complete

Figure 12 PackML_StateTransition_Set Function Block

6.2 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_IN/OUT	dba_PackMLcfgModeTransition	Bit (0..31,0..17)	Contains the PackML Mode transition configuration
VAR_INPUT	iw_Mode	Word	Current mode input
VAR_INPUT	ib_Stopped	Bit	Disable/enable the Clearing state
VAR_INPUT	ib_Idle	Bit	Disable/enable the Starting state
VAR_INPUT	ib_Suspended	Bit	Disable/enable the Suspending state
VAR_INPUT	ib_Aborted	Bit	Disable/enable the Stopping state
VAR_INPUT	ib_Held	Bit	Disable/enable the Aborting state
VAR_INPUT	ib_Complete	Bit	Disable/enable the Holding state

7 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.

7.1 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:

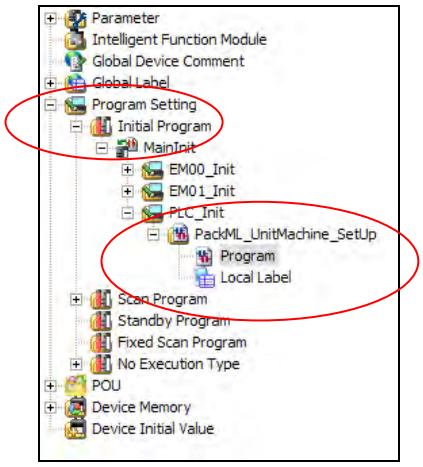


Figure 13 –Example of Setting the Program for Initial Scan Only

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.

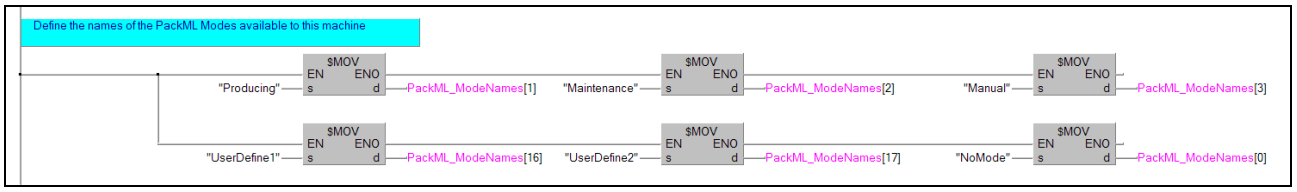


Figure 14 – Example of Setting PackML Modes for a Unit Machine

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

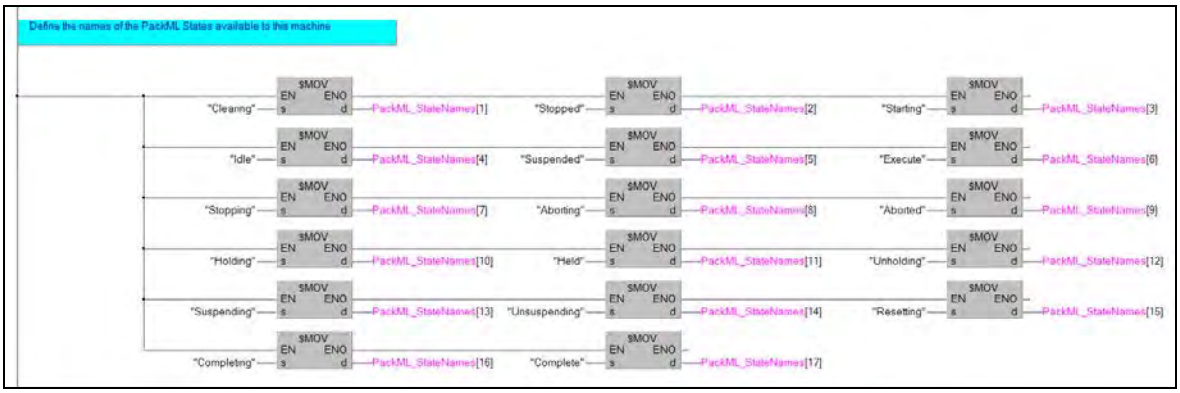


Figure 15 - Example of Setting PackML States for a Unit Machine

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

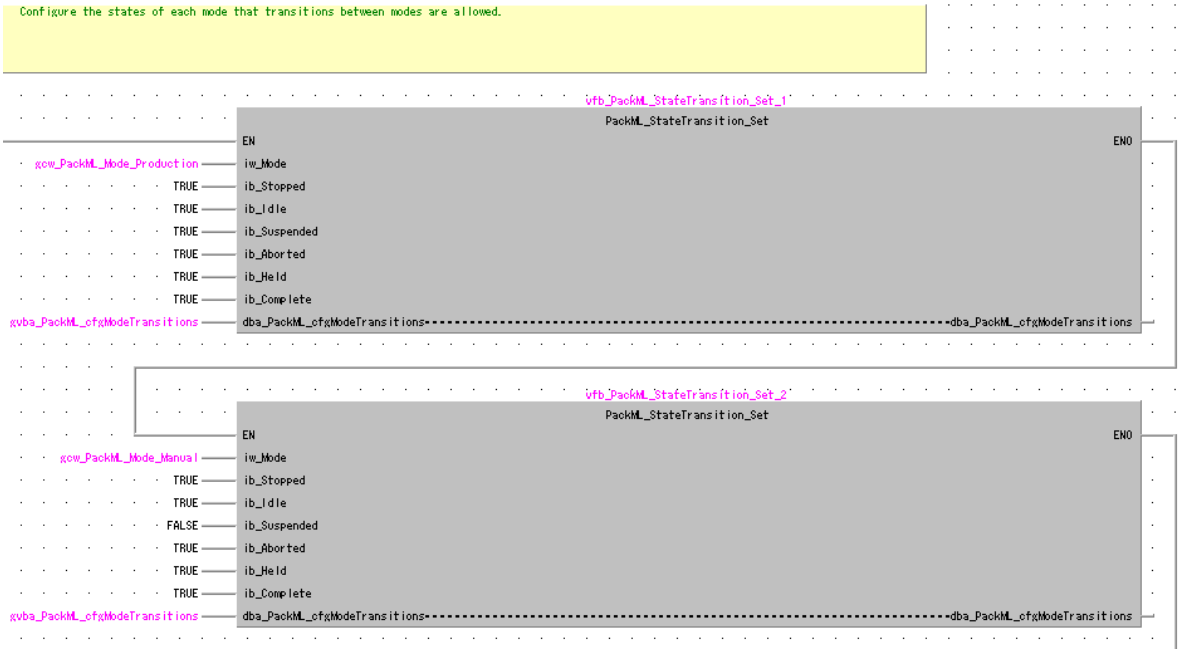


Figure 16 - Example of Setting States that Mode Transition are Allowed

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

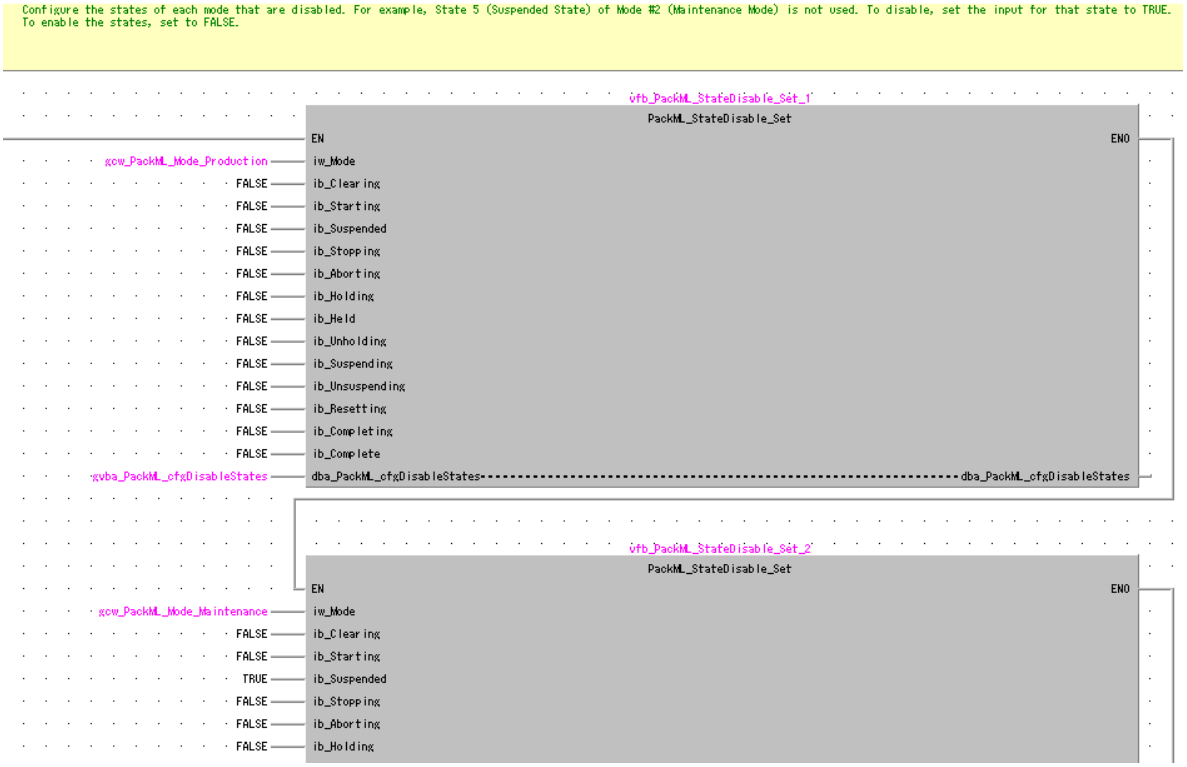


Figure 17 - Example of Setting States that are Disabled in Each Mode

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.

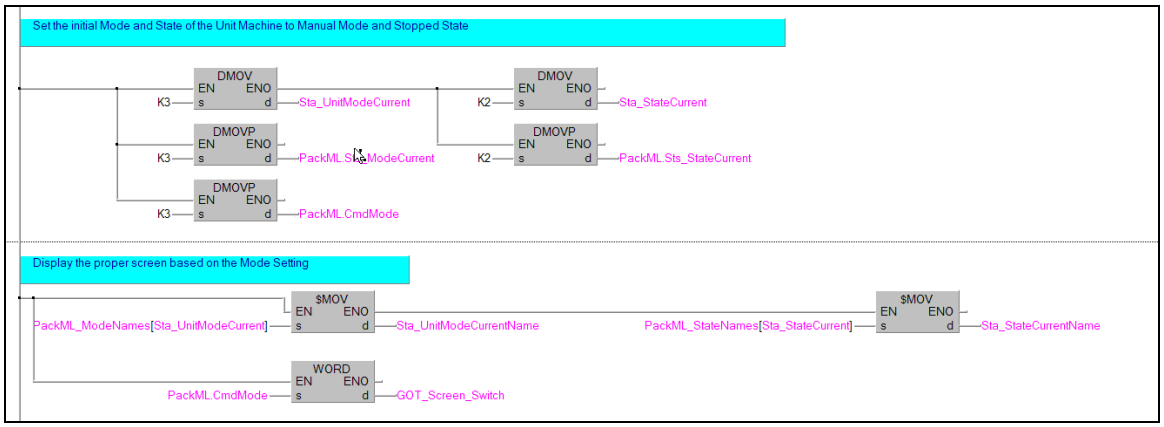


Figure 18 - Example of Setting Initial State and Mode of the Unit Machine

7.2 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.

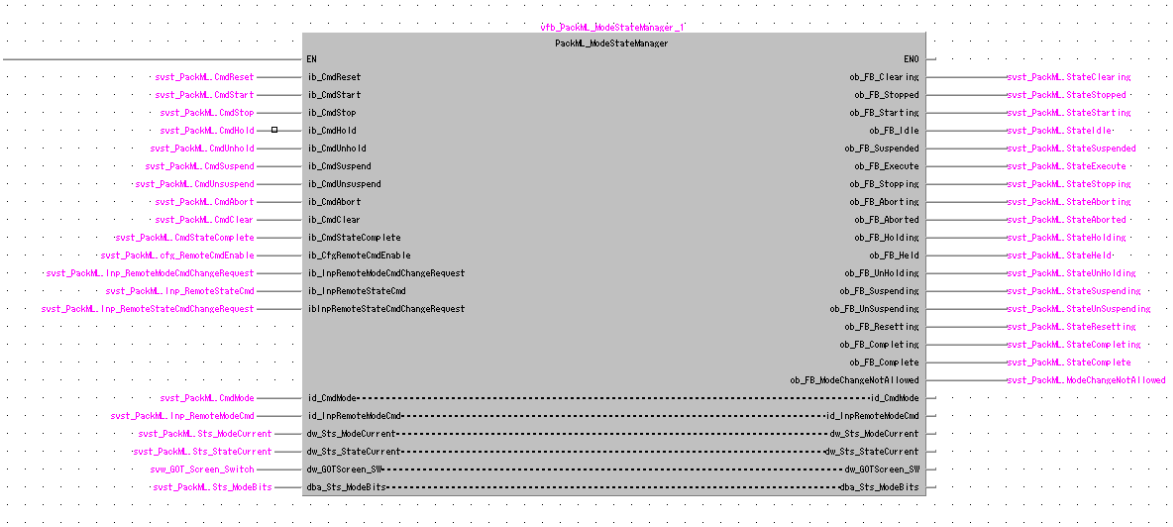


Figure 19 – Calling the PackML_ModeStateManager Function Block

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

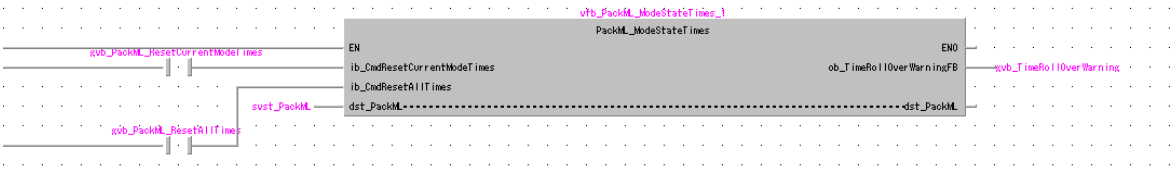


Figure 20 – Calling the PackML_ModeStateTimes Function Block

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 5 –Event Handling FBs

Release 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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:

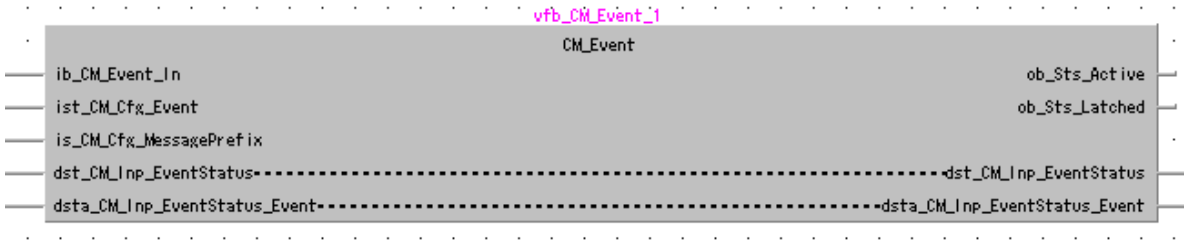


Figure 1 – CM_Event Function Block with Inputs and Outputs

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.
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.

Label	Data Type	Description
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.
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:

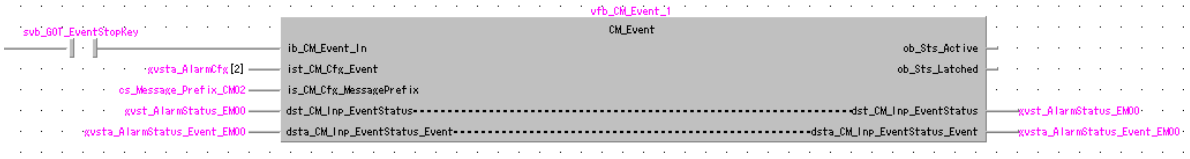


Figure 2 – Example of CM_Event Function Block

- By connecting “EN” to the power rail, this instance of the function block is always enabled.
- 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)
d_Sts_NumEvents	0	1
d_Sts_NumAllEvents	0	1
b_Sts_Category_0_Latched	0	0
b_Sts_Category_1_Latched	0	0
b_Sts_Category_2_Latched	0	1
b_Sts_Category_3_Latched	0	0
b_Sts_Category_4_Latched	0	0
b_Sts_Category_5_Latched	0	0
b_Sts_Category_6_Latched	0	0
b_Sts_Category_7_Latched	0	0
b_Sts_Category_8_Latched	0	0
b_Sts_Category_9_Latched	0	0
b_Sts_Category_0_NotLatched	0	0
b_Sts_Category_1_NotLatched	0	0
b_Sts_Category_2_NotLatched	0	1
b_Sts_Category_3_NotLatched	0	0
b_Sts_Category_4_NotLatched	0	0
b_Sts_Category_5_NotLatched	0	0
b_Sts_Category_6_NotLatched	0	0
b_Sts_Category_7_NotLatched	0	0
b_Sts_Category_8_NotLatched	0	0
b_Sts_Category_9_NotLatched	0	0
b_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]		
d_ID	0	65
d_Value	0	0
s_Message		HMI STOP PB Pressed
wa_TimeEventArray		
[0]	0	2010
[1]	0	6
[2]	0	28
[3]	0	9
[4]	0	15
[5]	0	7
[6]	0	1
wa_TimeAckArray		
[0]	0	0
[1]	0	0
[2]	0	0
[3]	0	0
[4]	0	0
[5]	0	0
[6]	0	0
d_Category	0	2
b_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
d_ID	0	0
d_Value	EM00 HMI STOP PB Pressed	HMI STOP PB Pressed
s_Message		
wa_TimeEventArray	2010	2010
[0]	6	6
[1]	28	28
[2]	9	9
[3]	15	15
[4]	7	7
[5]	1	1
[6]		
wa_TimeAckArray	0	2010
[0]	0	6
[1]	0	28
[2]	0	9
[3]	0	26
[4]	0	6
[5]	0	1
[6]	2	2
d_Category	1	0
b_Trigger		
[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.

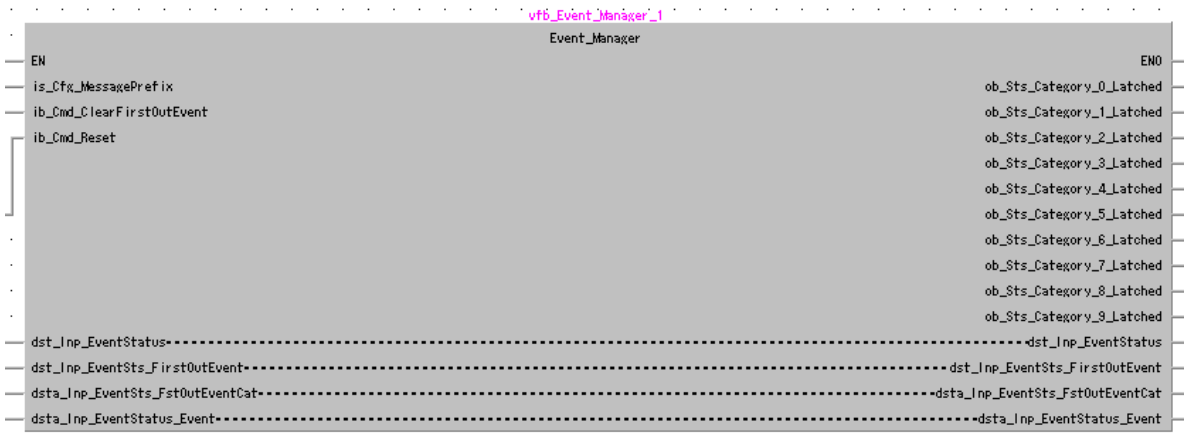


Figure 3 – Event_Manager Function Block

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	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

Variable Type	Variable Label	Data Type	Description
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.

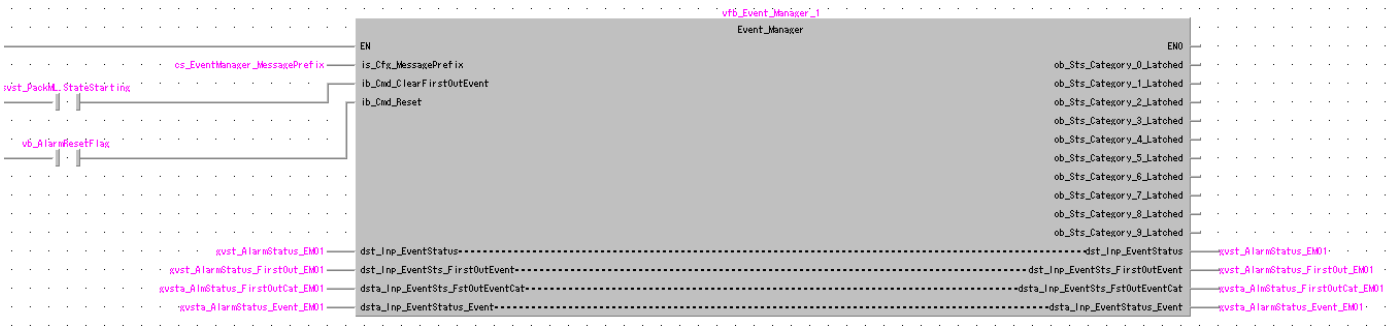


Figure 4 – Example of Event_Manager Function Block

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.

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 5, Figure 6, and Figure 7 respectively.

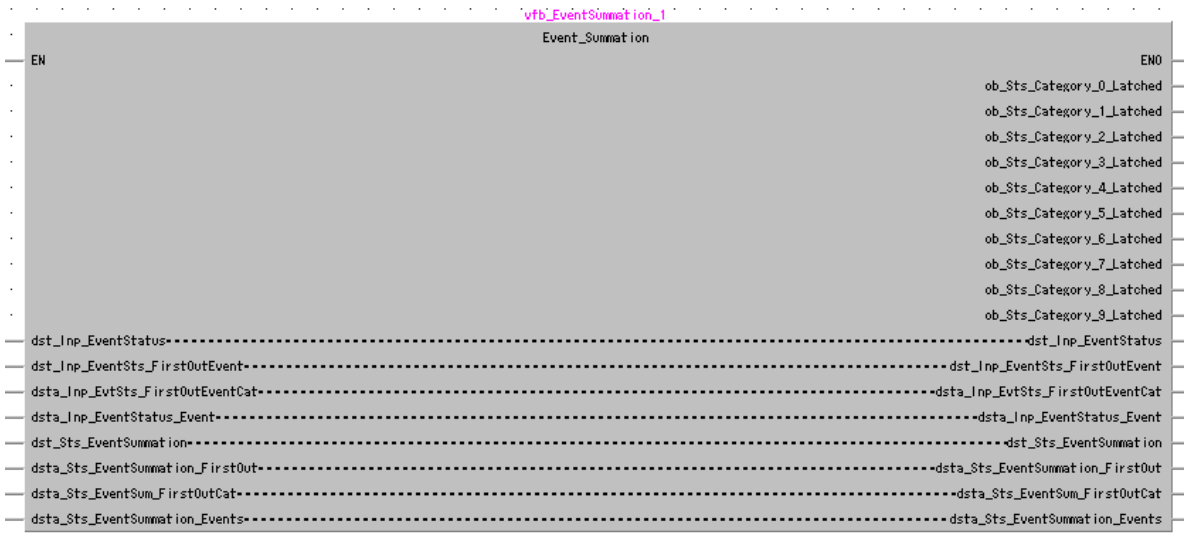


Figure 5 – Event_Summation Function Block

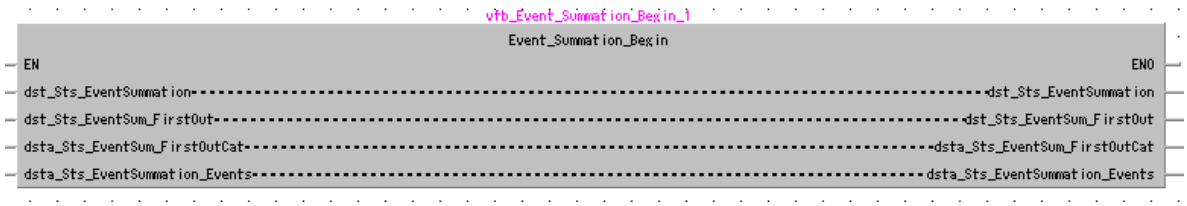


Figure 6 – Event_SummationBegin Function Block

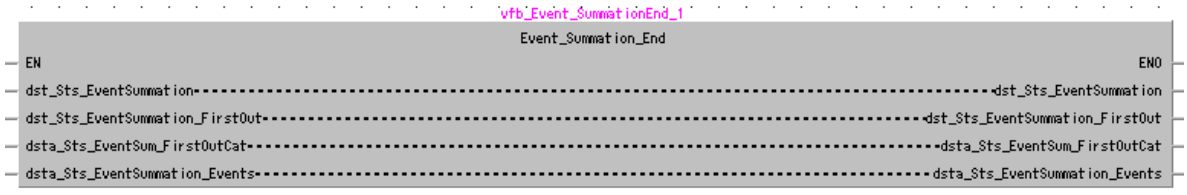


Figure 7 – Event_SummationEnd Function Block

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
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_Sts_EventSum_FirstOutCat	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
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

Label	Data Type	Description
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.
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.

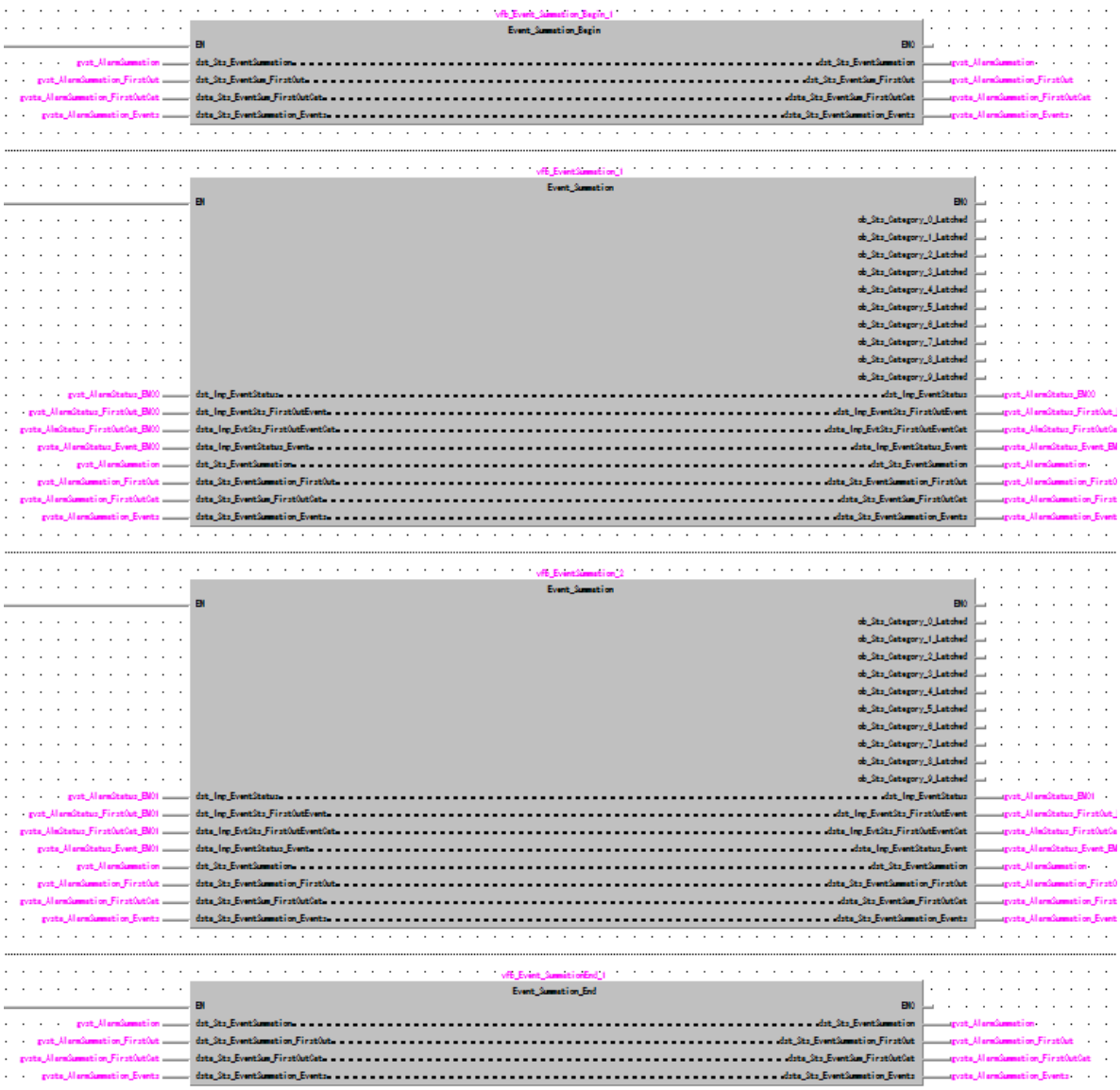


Figure 8 – Example – Event Summation of a Unit Machine

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:

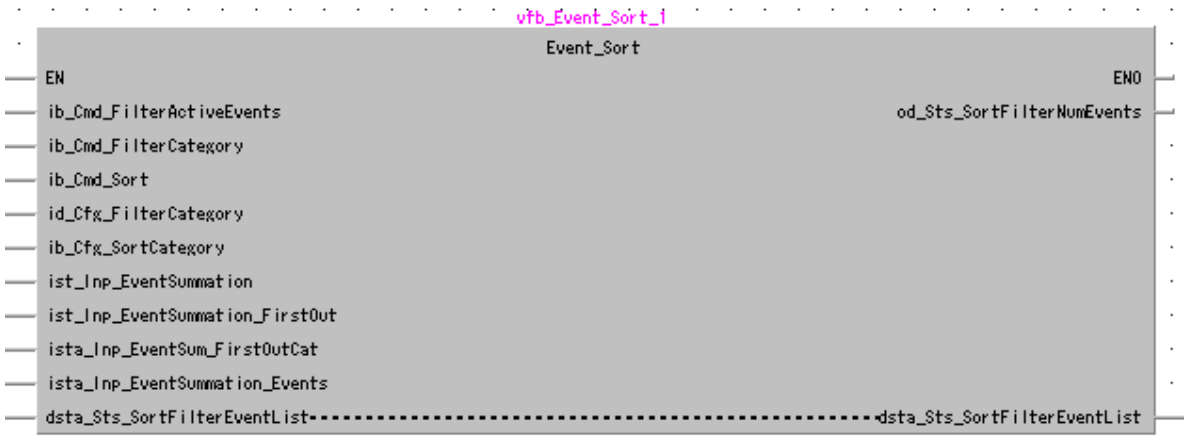


Figure 9 – Event_Sort Function Block

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	dsta_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.

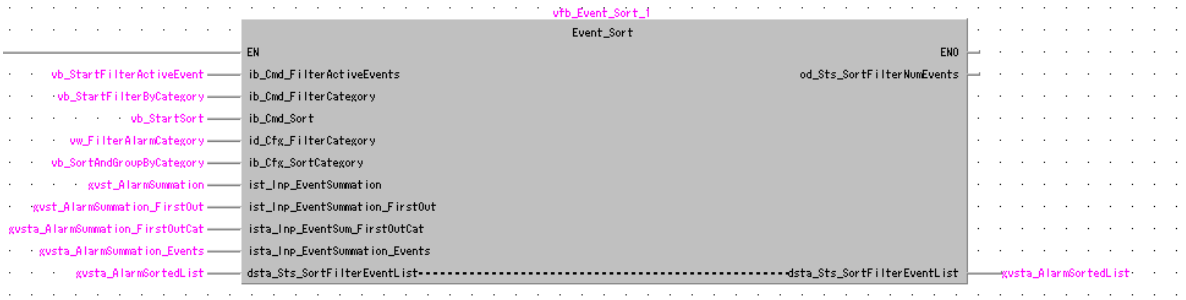


Figure 10 – Example of Event_Sort FB Operations

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 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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 2 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 Mitsubishi PackML templates are designed to run on a system with the minimum of a QnUDEH PLC and a GOT-16 HMI. Optionally the system can contain a Q172D Motion Controller for motion applications that are required for a particular machine. However, no motion programs are included in the PackML templates. Only examples on how to configure Multi-CPU parameters are included in this template release.

The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a Q06UDEHCPU, the motion controller is Q172D, and the GOT is a GT-16s 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 Q06UDEHCPU.

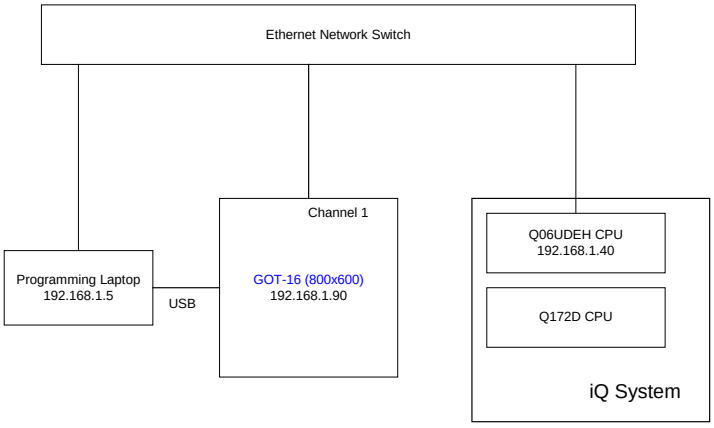


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT using the USB port to download screen information and to the Q06UDEHCPU through the GOT Ethernet Transparent Mode. The Transparent Mode is set up to go through the Ethernet port on the Q06UDEH CPU (with IP Address 192.168.1.40). During the system operation, GOT is configured to work with the iQ PLC through the same Q06UDEH CPU Built-in Ethernet port. The configuration of the motion controller is also through the GOT Ethernet Transparent Mode and targeted to CPU #2 in the system.

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 -> Tasks - > Program Block or Program Organization Unit (POU).

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

- The Project “PackML and Event Implementation” contains many Program Files.

- Program File "MainInit" contains the tasks 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 Mitsubishi PackML Template project is designed to have one Task per Program File other than the main initialization functions.
- The Program File UnitMach contains the Task "Unit_Machine" which 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 task and 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, an Event_Control program block, and an Equipment Module PackML Command and Status Summation Program block together with as many control modules as necessary to control the actual equipment.

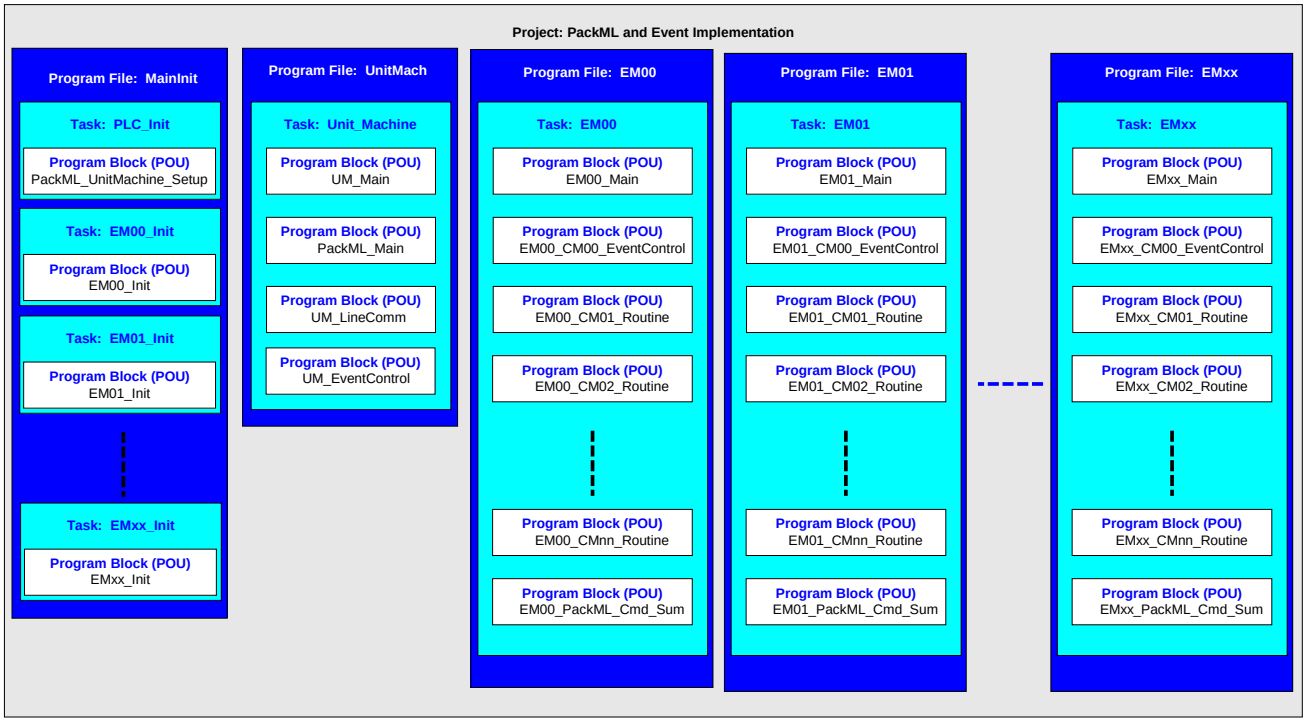


Figure 2 – The Overall Structure of the PackML Template Project

4 Mitsubishi PackML Template Project

The actual Mitsubishi PackML Template Project contains the structure to support a Unit Machine with two Equipment Modules and five Control Modules (including one module 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 8 of this document.

Figure 3 shows the actual project structure in GX Works 2. After all Program Organization Units (POU) are created, they are registered in the proper program setting areas.

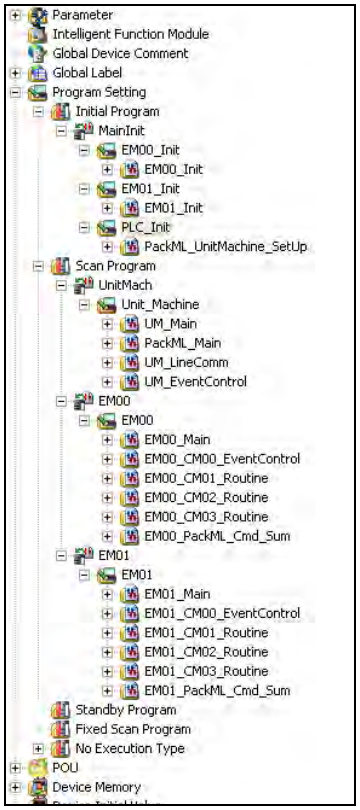


Figure 3 – Project Structure of PackML Implementation Template

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 three tasks EM00_Init, EM01_Init, PLC_Init.
- EM00_Init task contains the POU EM00_Init which has the actual program routine initializing the Equipment Module EM00 and local variables associated with the routine.
- EM01_Init task contains the POU EM01_Init which has the actual program routine initializing the Equipment Module EM00 and local variables associated with the routine.
- PLC_Init task contains the 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 the Unit_Machine Task and there are four POUs: UM_Main, PackML_Main, UM_LineComm, and UM_EventControl within the Task.
- The EM00 Program File contains the EM00 Task and there are five POUs: EM00_Main, EM00_CM00_EventControl, EM00_CM01_Routine, EM00_CM02_Routine, EM00_CM03_Routine, and EM00_PackML_Cmd_Sum.

- The EM01 Program File contains the EM01 Task and there are five POUs: EM01_Main, EM01_CM00_EventControl, EM01_CM01_Routine, EM01_CM02_Routine, EM01_CM03_Routine, and EM01_PackML_Cmd_Sum.

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, gvba_PackML_cfgModeTransitions[1, 2] = 1 means at Mode 1 ("Producing" mode) State 2 ("Stopped") state, the machine is allow to switch mode. However, gvba_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, gvba_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
svst_ PackML	**PackMLFB**	The overall PackML label that contains various values and parameters for the machine states and modes

Variable Label	Data Type	Description
d_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_SDT	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_SDT	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_SDT	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_SDT	The label indicating the PackML commands and status of Control Module EM00_CM00
gvst_ EM00_CM01_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM00_CM01
gvst_ EM00_CM02_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM00_CM02
gvst_ EM00_CM03_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM00_CM03
gvst_ EM01_CM00_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM01_CM00
gvst_ EM01_CM01_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM01_CM01
gvst_ EM01_CM02_PackML_Sts	PackML_Module_Cmd_SDT	The label indicating the PackML commands and status of Control Module EM01_CM02
gvst_ EM01_CM03_PackML_Sts	PackML_Module_Cmd_SDT	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.

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.
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.

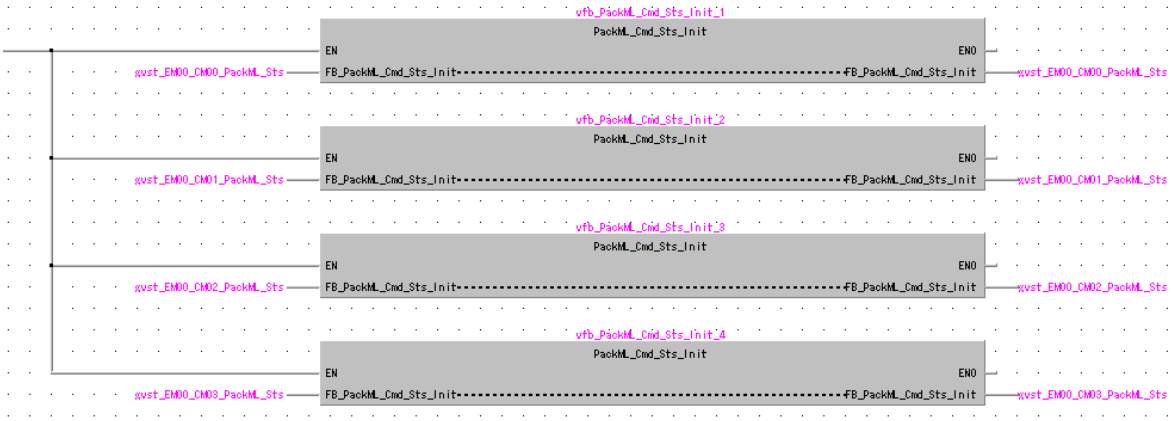


Figure 4 – Equipment Module PackML Initialization

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.

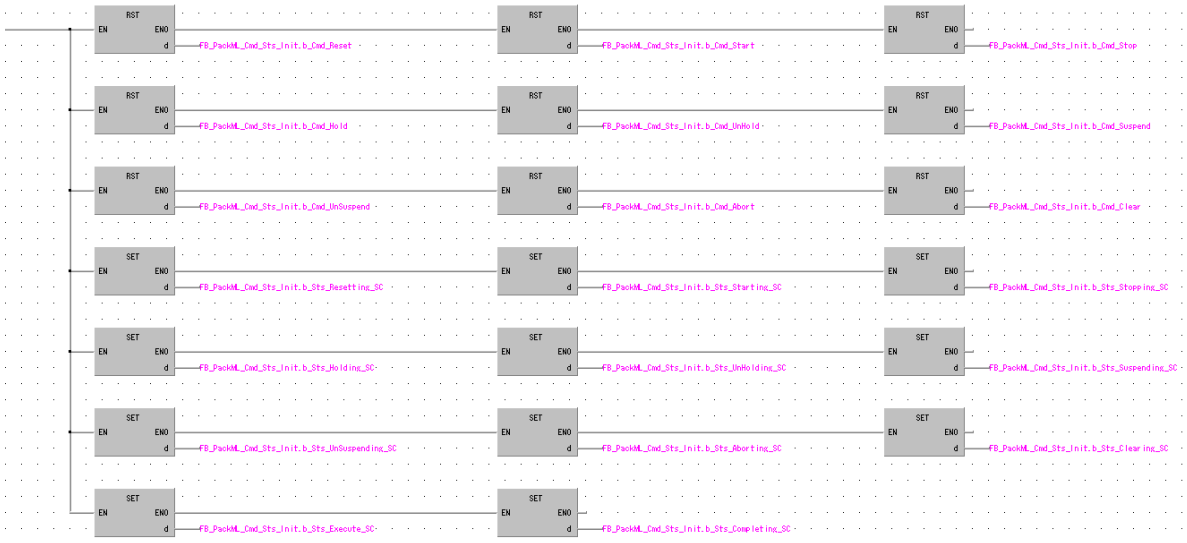


Figure 5 Function Block: PackML_Cmd_Sts_Init

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 Gvba_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 POUs

It is recommended that a user should refer to the actual structured ladder logic code in the Mitsubishi PackML Implementation Template project to get a better understanding of the functions of these POUs.

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

This POU 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

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

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 POUs

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

6.3.2. EMxx_CMnn_Routine

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 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 and EM01_CM02_Routine contain the GOT interface routine for simulating events in the Unit Machine through GOT key presses.

6.3.3. EMxx_EventControl

The purpose of this POU 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.

6.4 POU Scan Order

The scan order of all the POU's is shown in Figure 6 below:

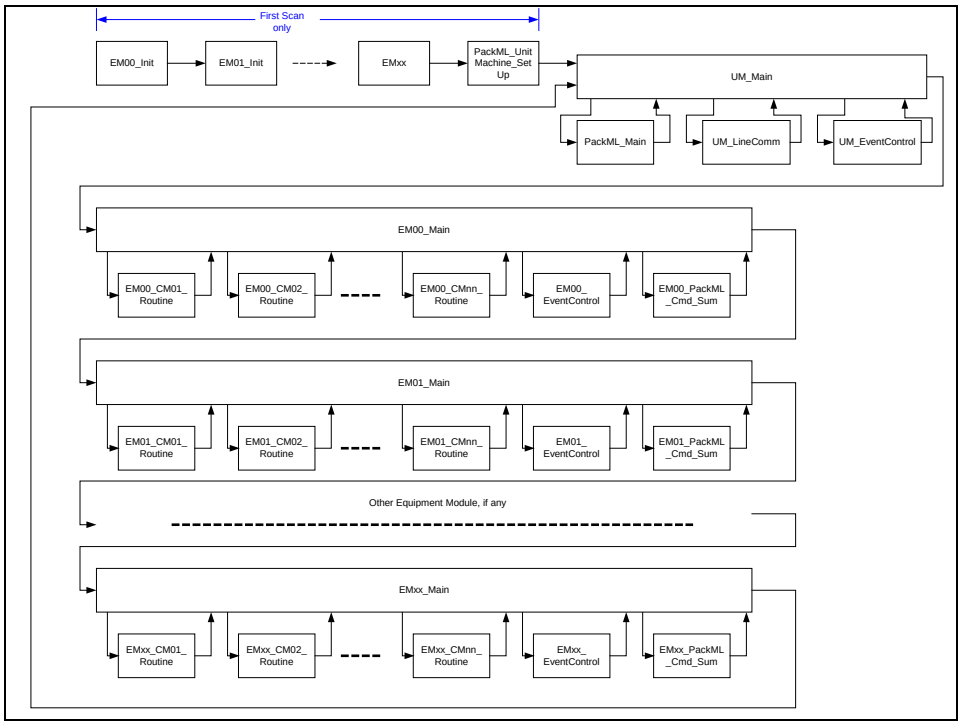


Figure 6 – PackML Implementation Template POU Scan Order

7 PLC CPU Parameters and Settings

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

7.1 PLC System Parameters

Most the parameters on this screen are default parameters except the Common Pointer number that was set at 2000 so that pointer values greater than 2000 are assigned by the project manually and not automatically assigned.

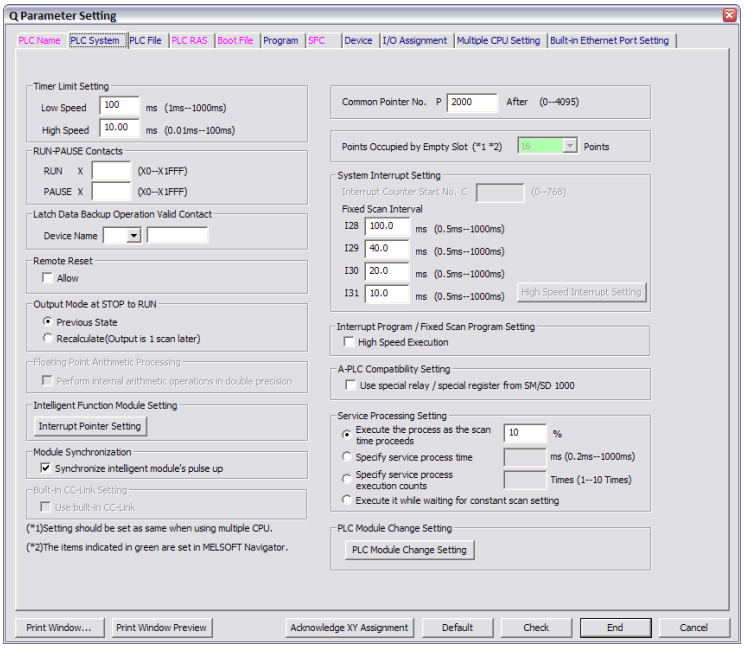


Figure 7 – PLC Parameters, PLC System Settings

7.2 PLC File Parameters

Because of the large number of PackTags and labels that are required to support the PackML and Event Handling functions, a File Register system is configured to support the these labels.

It is critical to assign the File Registers to "Standard RAM (Drive 3)" so the labels are stored in the Standard RAM area of the Q06UDEHCPU. If the File Registers are assigned to the Memory Card (Drive 1), they will have significant impact on the scan time of the project when Event Handling routines are being scanned.

The capacity of the File is set at 300K points but can be adjusted according to the application needs. The limitation will be the amount of Standard RAM memory size of the PLC. Q06UDEHCPU has 1024K Standard RAM.

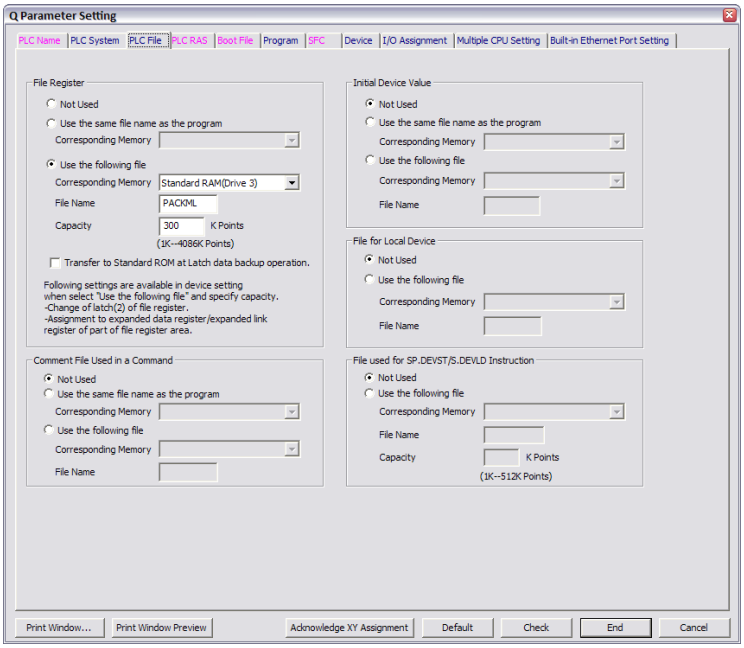


Figure 8 – PLC Parameters, PLC File Settings

7.3 Device Settings

The key settings on this screen are the allocated size for ZR registers (180K points) and Extended Data registers (120K points) to support automatic allocation of labels and pre-defined PackTags in D Registers.

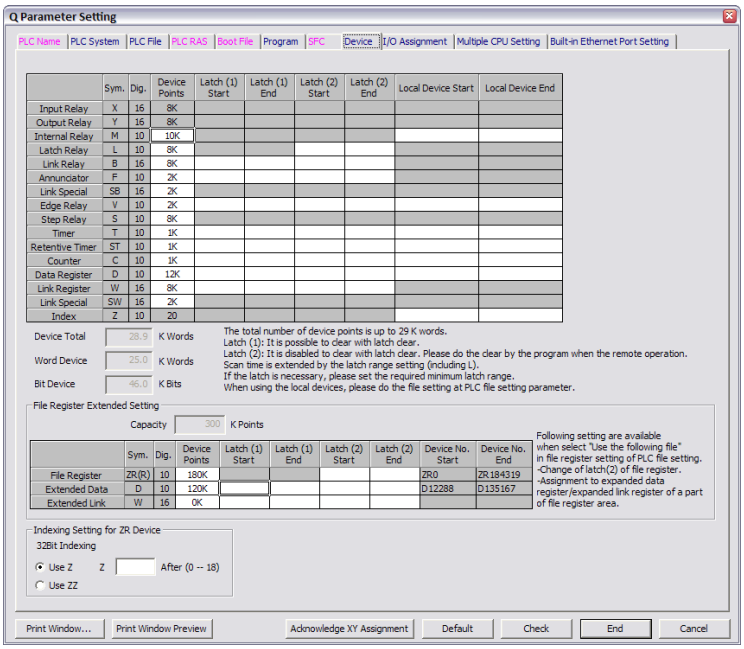


Figure 9 – PLC Parameters, Device Settings

7.4 I/O Assignments

For the template project, there are only two modules in the system and the I/O assignments are defined by iQ Works module configurations. For an actual system with additional CPU, I/O, or Intelligent I/O modules, the configurations of these modules should also be done in the iQ Works MELSOFT Navigator and the I/O assignments will be reflected here. The assignments can be overwritten if the option in GX Works 2 is enabled by selecting the checkbox in Tool-> Options -> iQ Works Interaction.

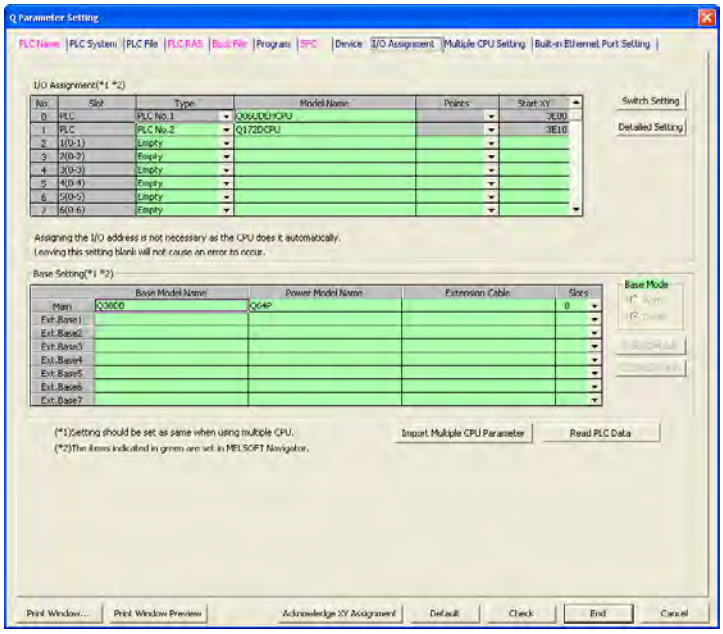


Figure 10 – PLC Parameters, I/O Assignment Settings

7.5 Multiple CPU Setting

The template system only has two CPUs, the PLC and the Motion Controller. In the template project, the motion CPU is not programmed. However, the basic parameter settings are shown here for references.

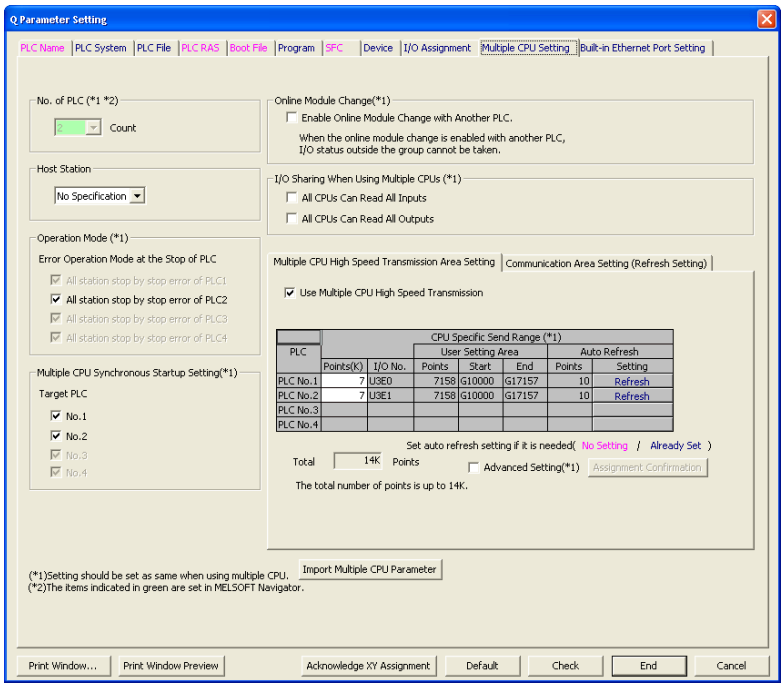


Figure 11 – PLC Parameters, MultiCPU Settings

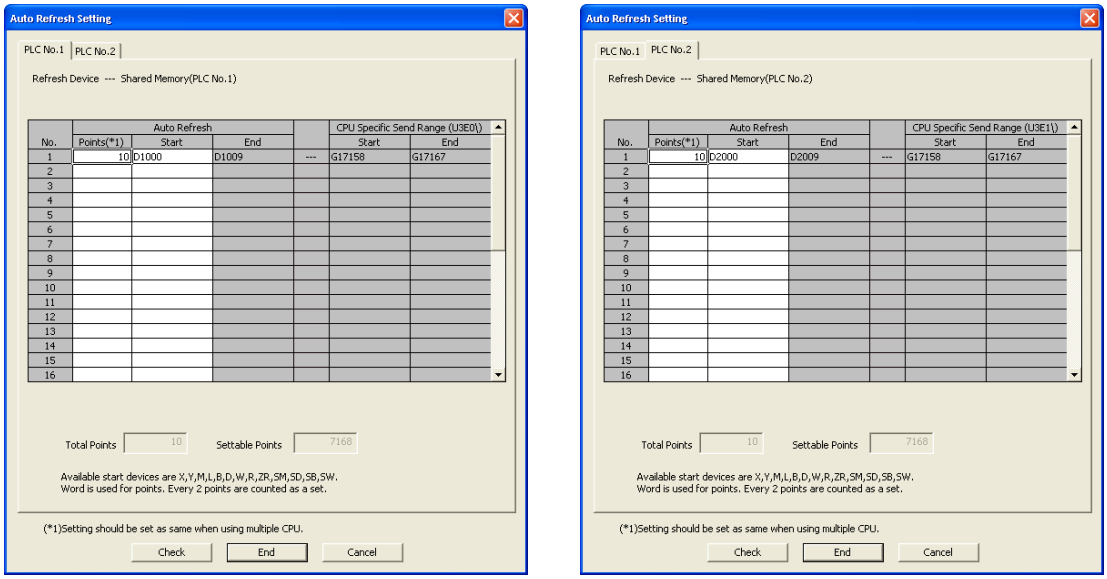


Figure 12 – PLC Parameters, Auto Refresh Settings

The Auto Refresh settings shown here are just for example purposes only. There need to be configured to represent the actual system requirements.

7.6 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.

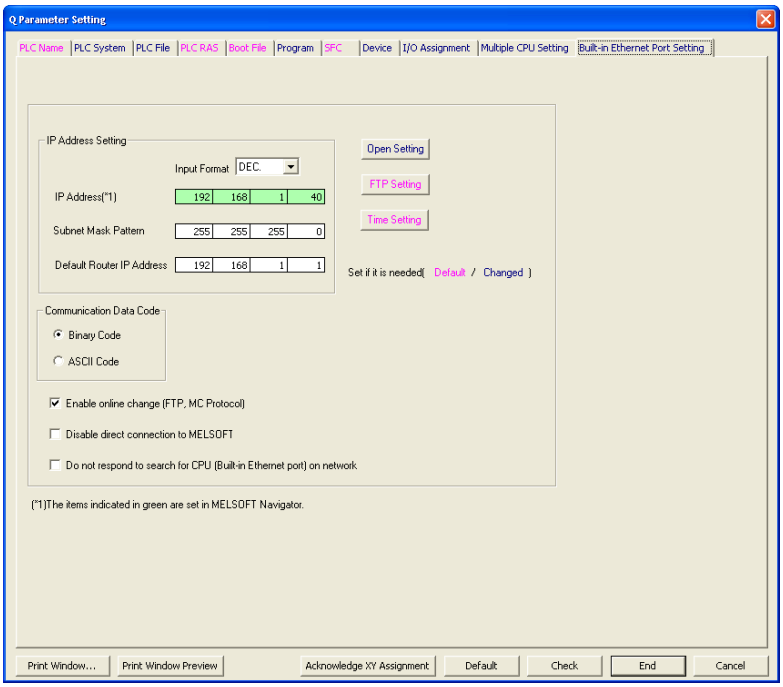


Figure 13 – PLC Parameters, Built-In Ethernet Port Settings

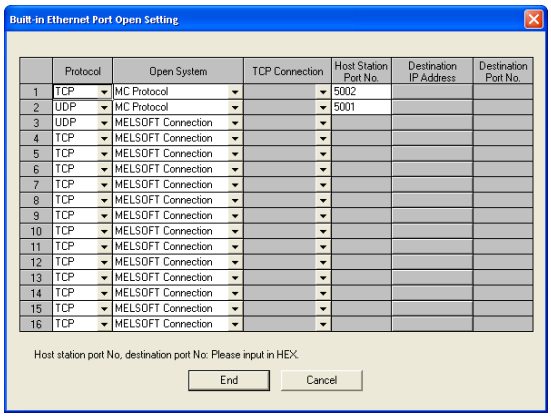


Figure 14 – PLC Parameters, Built-In Ethernet Port Open Settings

7.7 Device Label Automatic Assignments

Use “Tool -> Device/Label Automatic-Assign Setting…” to configure the system where labels should be assigned.

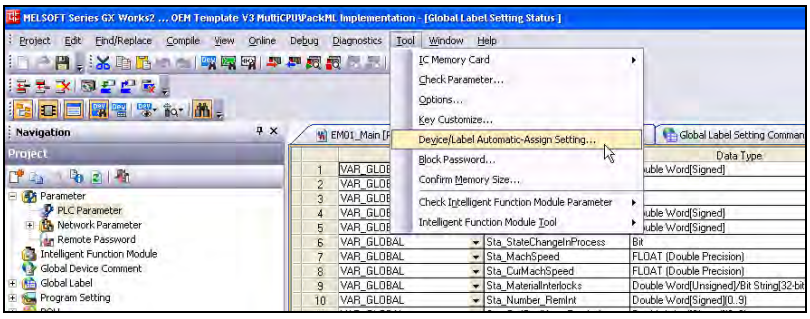


Figure 15 – Configuring Device Label Automatic Assignments

The range of automatic assignment of “Word” labels is allocated to ZR registers from ZR0 to ZR183190. The range of automatic assignment of “Bit” labels is allocated to M bits M3000 to M8000. The rest of the ranges are left with default values.

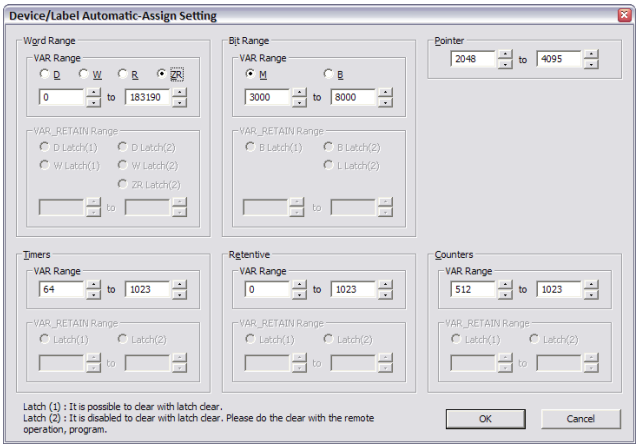


Figure 16 – Label Automatic Assignment Settings

8 Program Memory Space

The full PackML template requires about 245K of program memory to execute. The Q06UD(E) CPU capacity for standard RAM is 245K. This means that the PackML template with the additional OEM code will not fit in the standard program memory location of Q06UD(E) CPU. To overcome this program memory constraint, download the "Symbolic Information" to the "Standard ROM" instead of program memory. This will reduce the amount of the required program memory size down to 138K, leaving 107K for OEM code.

The minimum set PackML template only requires 111K of program memory. If the symbolic information is downloaded to the standard ROM location, the size of program can be further reduced to 42K.

The following table shows the amount of available program memory in Q series PLCs.

Model Number	Q03UD(E)CPU	Q04UD(E)CPU	Q06UD(E)CPU	Q13UD(E)CPU	Q26UD(E)CPU
Program Memory	120kB	160kB	240kB	520kB	1040kB
Standard ROM	1024kB			2051kB	4102kB
Model Number	Q03UDVCPU	Q04UDVCPU	Q06UDVCPU	Q13UDVCPU	Q26UDVCPU
Program Memory	120kB	160kB	240kB	520kB	1040kB
Standard ROM	1025.5kB			2048kB	4096kB

Amount of PLC program memory space required for PackML template

Template Version	Full PackML Template	Minimum PackML Template
PLC Program and symbolic information stored in Program memory	292kB	111kB
PLC Program stored in program memory and symbolic info stored in standard ROM	147kB	46kB

*8.1 **Downloading Symbolic Information File to the PLC Standard ROM location to save program memory space***

- By default, GX Works downloads the PLC data (program files, comments and parameters) and the Symbolic Information to the Program Memory/Device Memory as shown in the figure below:

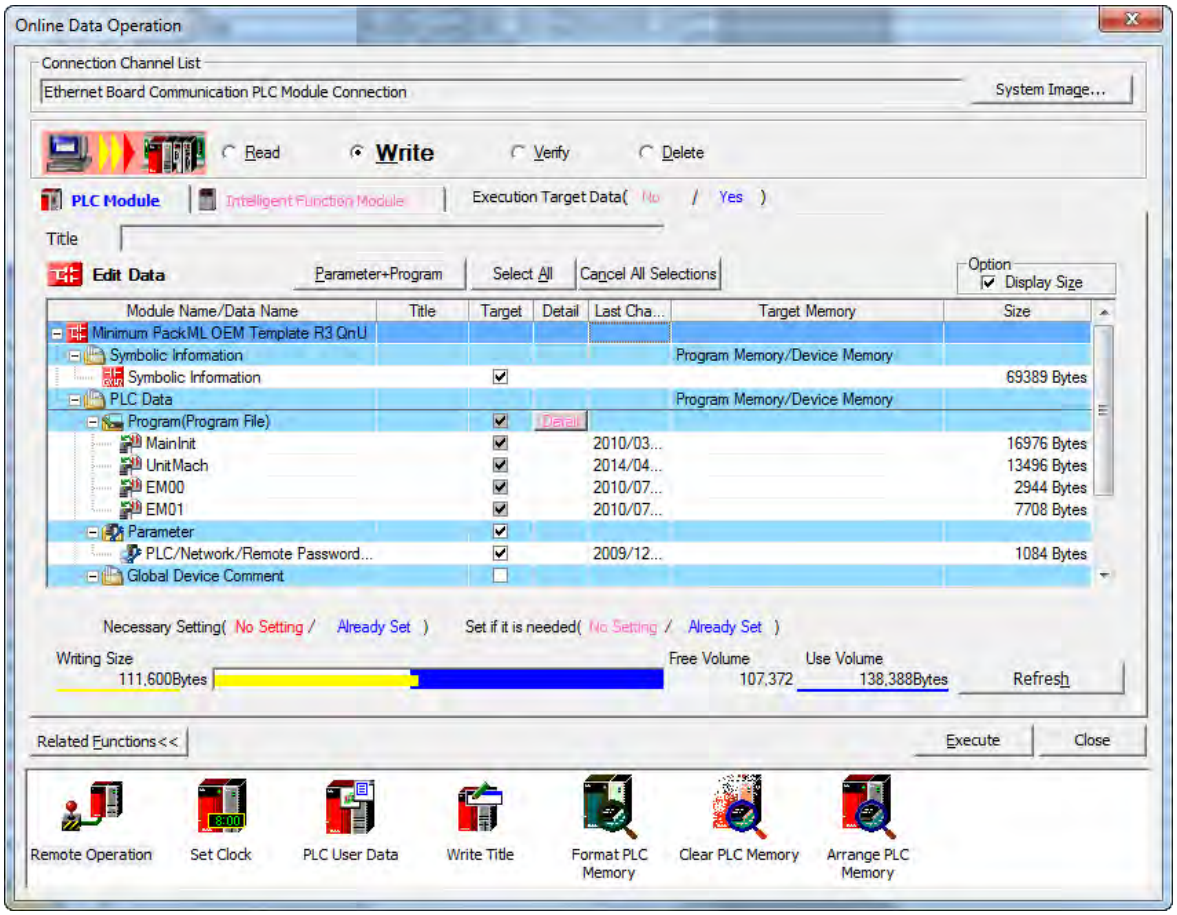


Figure 17 – Downloading to program memory

- Click on the “Program Memory/Device Memory” box to the right of Symbolic Information and select the “Standard ROM” option in the drop down box.

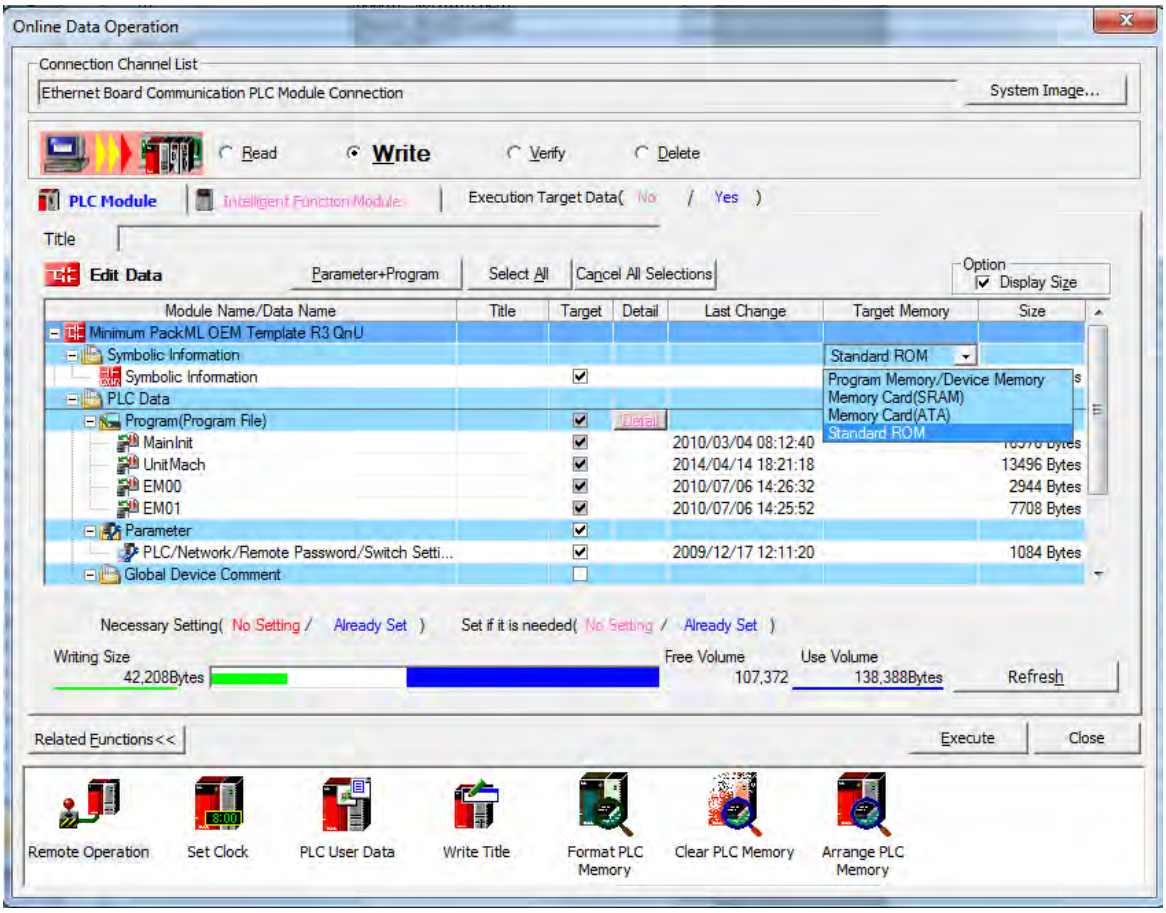


Figure 8 – Downloading the symbolic information to the standard ROM

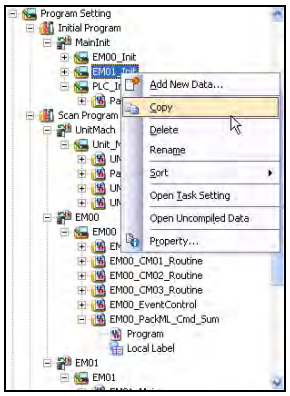
- Notice that the "Writing Size" was reduced to 42,208 Bytes from 111,600 Bytes.
- Download the program to the PLC.

9 This section documents the steps that an OEM will take to add the POU's Example of Adding an Equipment Module to the Template System

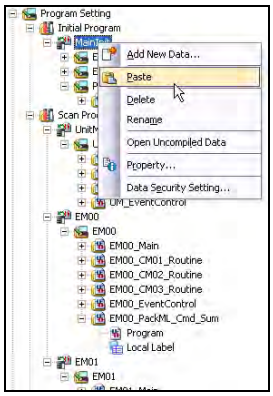
This section documents the steps that an OEM will take to add the POU's of an additional equipment module (e.g. EM02) in the PackML Implementation Template Project.

9.1 Adding EM02_Init POU

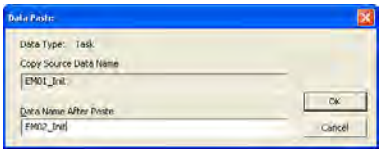
- Right Click on EM01_Init Task in the "Initial Program ->MainInit" tree and select copy.

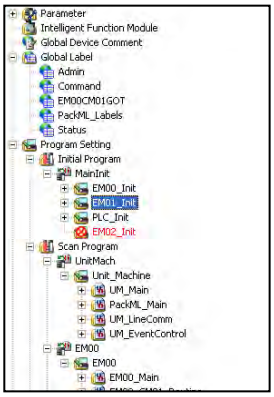


- Right click on MainInit Task and select Paste.

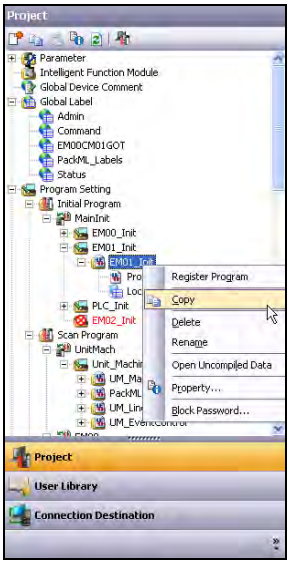


- A pop-up window will be displayed and allow the user to enter a new name “EM02_Init” for the new Task. The EM02_Init Task is then added to the tree.

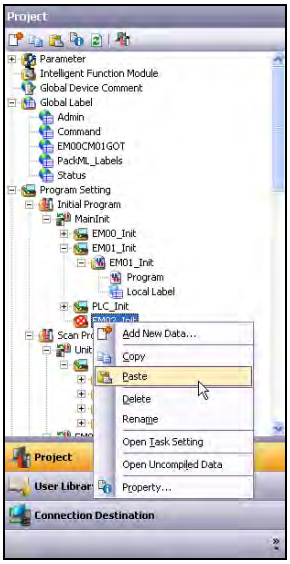




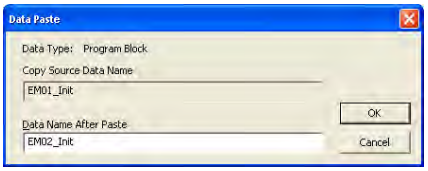
- Right Click on EM01_Init POU in the "Initial Program ->MainInit -> EM01_Init" tree and select copy.

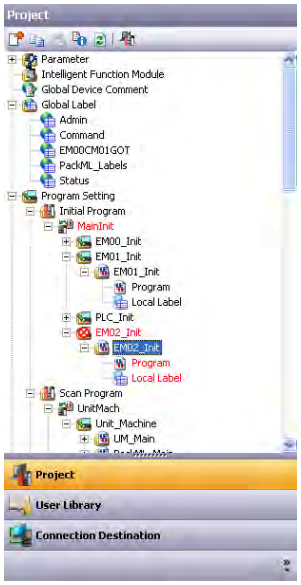


- Right Click on the EM02_Init Task and select Paste to copy the POU

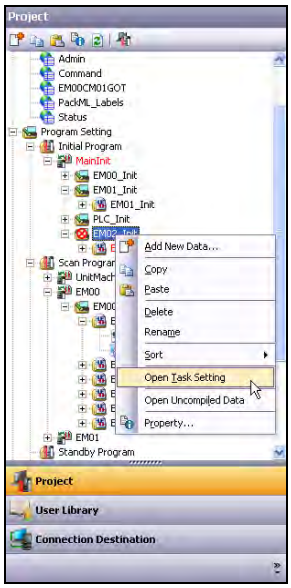


- A pop-up window will be displayed and allow the user to enter a new name “EM02_Init” for the new program block or POU. The EM02_Init program block is then added to the tree.

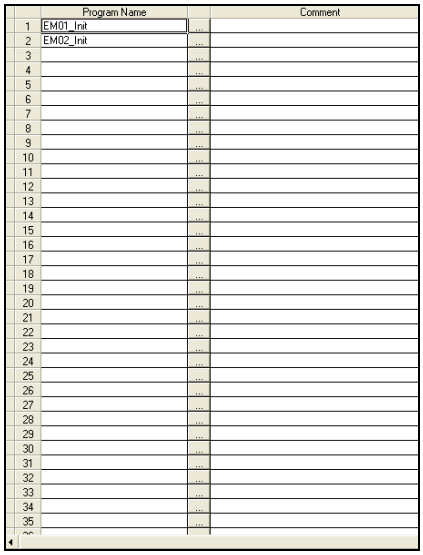




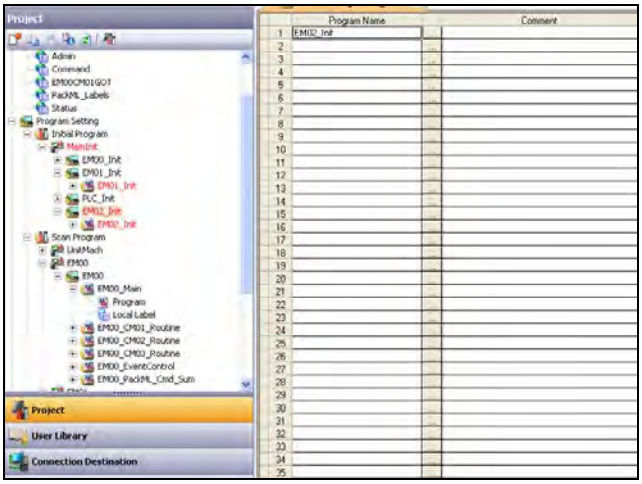
- The red circle X across the EM02_Init task indicates that there are errors associated with this new Task that was created. Right click on the EM02_Init Task and select “Open Task Setting”.



- The Task setting shows that both EM01_Init and EM02_Init are under this task since the EM02_Init task was copied from the EM01_Init Task which has program file EM01_Init.



- Simply delete EM01_Init from the Task Setting of EM02_Init Task and the red circle with the x should disappear from EM02_Init Task



- Double Click on the "Program" in the new EM02_Init Program Block and it will display the logic which is the exact copy of the logic in EM01_Init POU.

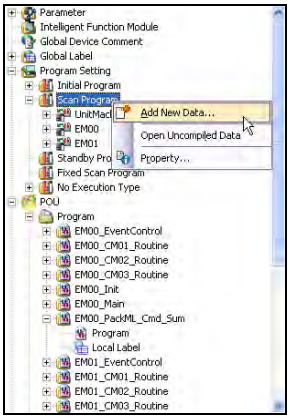
- Create new global variables EM02_CM00_PackML_Sts, EM02_CM01_PackML_Sts, EM02_CM02_PackML_Sts, and EM02_CM03_PackML_Sts with structured data type "PackML_Module_Cmd" in the Group OEM_Template_PackML_Labels.

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	EM00_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
2	VAR_GLOBAL	EM01_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
3	VAR_GLOBAL	UN_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
4	VAR_GLOBAL	EM00_CM00_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
5	VAR_GLOBAL	EM00_CM01_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
6	VAR_GLOBAL	EM00_CM02_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
7	VAR_GLOBAL	EM00_CM03_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
8	VAR_GLOBAL	EM01_CM00_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
9	VAR_GLOBAL	EM01_CM01_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
10	VAR_GLOBAL	EM01_CM02_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
11	VAR_GLOBAL	EM01_CM03_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
12	VAR_GLOBAL	RemoteCmd_ResetAllTimes	Bit	...			
13	VAR_GLOBAL	EM02_CM00_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
14	VAR_GLOBAL	EM02_CM01_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
15	VAR_GLOBAL	EM02_CM02_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
16	VAR_GLOBAL	EM02_CM03_PackML_Sts	PackML_Module_Cmd	...		Detail Setting	Detail Setting
17				...			

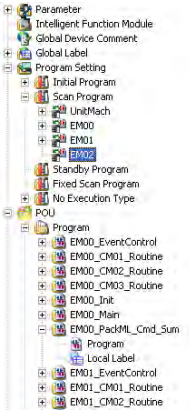
- Edit the EM02_Init logic to use the proper labels and correct instances of the initialization function block.

*9.2 **Adding EM02 Program File***

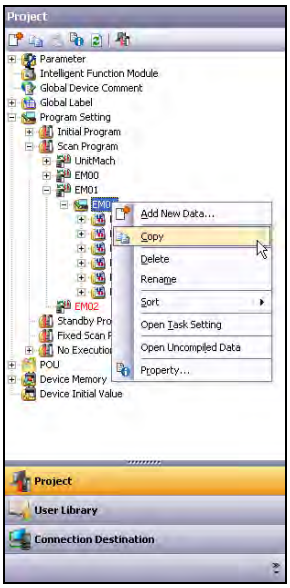
- Right Click on the Scan Program in the project tree and select "Add New Data…"



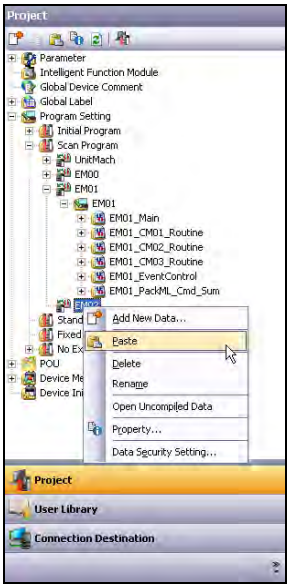
- A pop-up window will appear and enter the new name of the Program File “EM02”and the new program file will be added to the project tree.



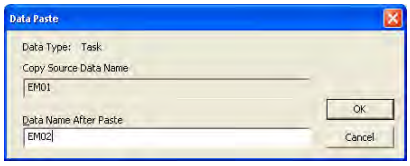
- Right Click on the EM01 Task and select “Copy”.

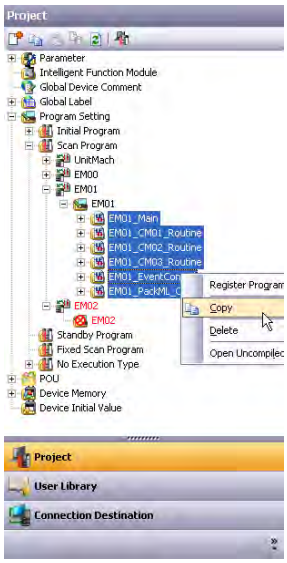


- Right Click on the EM02 Program File and select “Paste”.

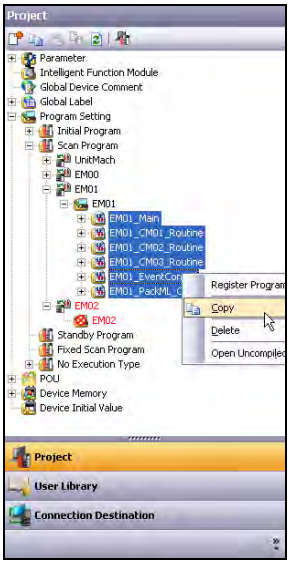


- A pop-up window will appear and that new task name EM02 can be entered.

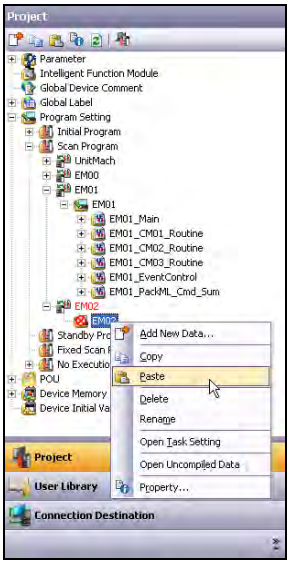




- Select all POU's under the task EM01 and click "Copy"

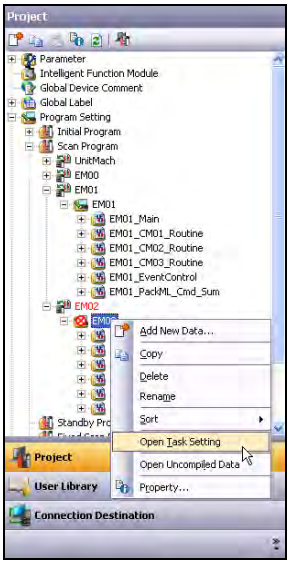


- Right Click on the EM02 Task and select "Paste".

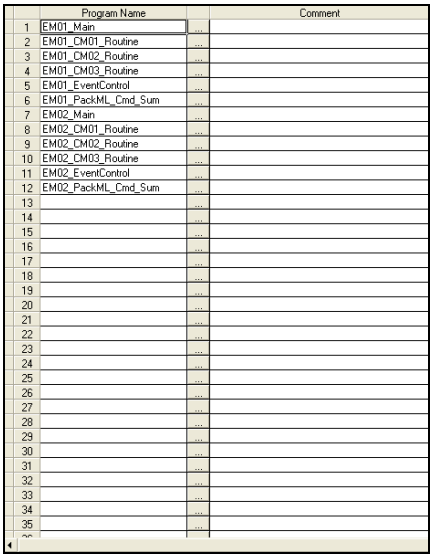


- Pop-Up windows will appear to allow the user to enter the new names of the POUs. Repeat the process until all POU's are renamed.

- The red circle X across the EM02 task indicates that there are errors associated with this new Task that was created. Right click on the EM02 Task and select "Open Task Setting".

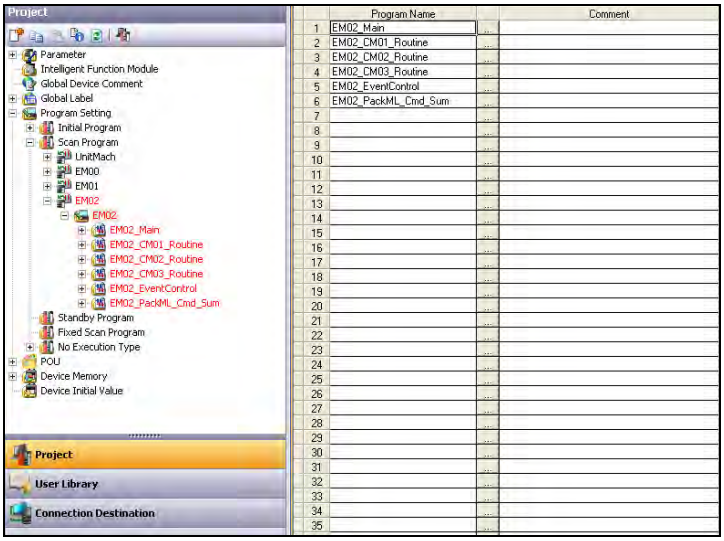


- The Task setting shows that both EM01 POU's and EM02 POU's are under this task since the EM02 task was copied from the EM01 Task which has all program files from EM01



	Program Name		Comment
1	EM01_Main	...	
2	EM01_CM01_Routine	...	
3	EM01_CM02_Routine	...	
4	EM01_CM03_Routine	...	
5	EM01_EventControl	...	
6	EM01_PackML_Cmd_Sum	...	
7	EM02_Main	...	
8	EM02_CM01_Routine	...	
9	EM02_CM02_Routine	...	
10	EM02_CM03_Routine	...	
11	EM02_EventControl	...	
12	EM02_PackML_Cmd_Sum	...	
13–35		...	

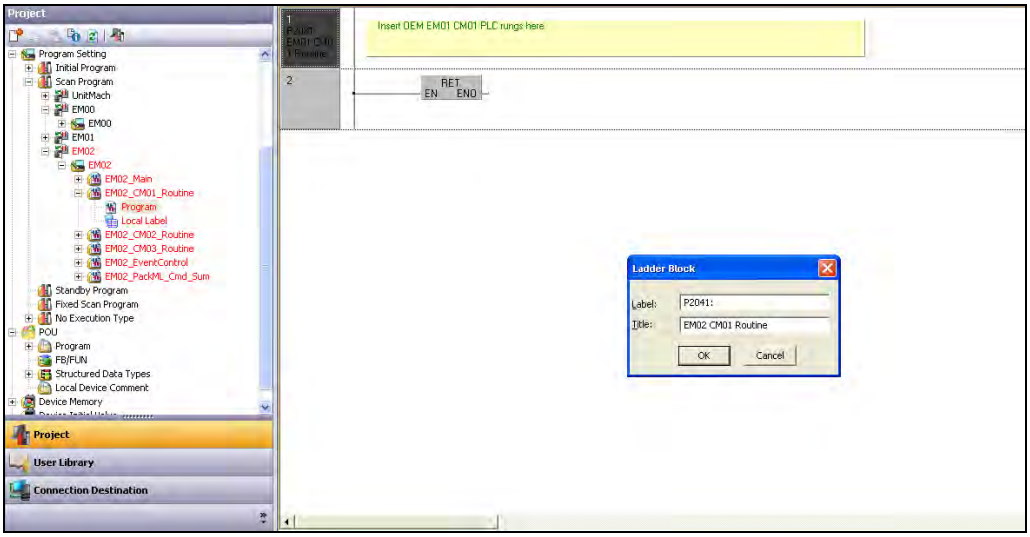
- Simply delete EM01 POU's from the Task Setting of EM02 Task and the red circle with the x should disappear from EM02 Task



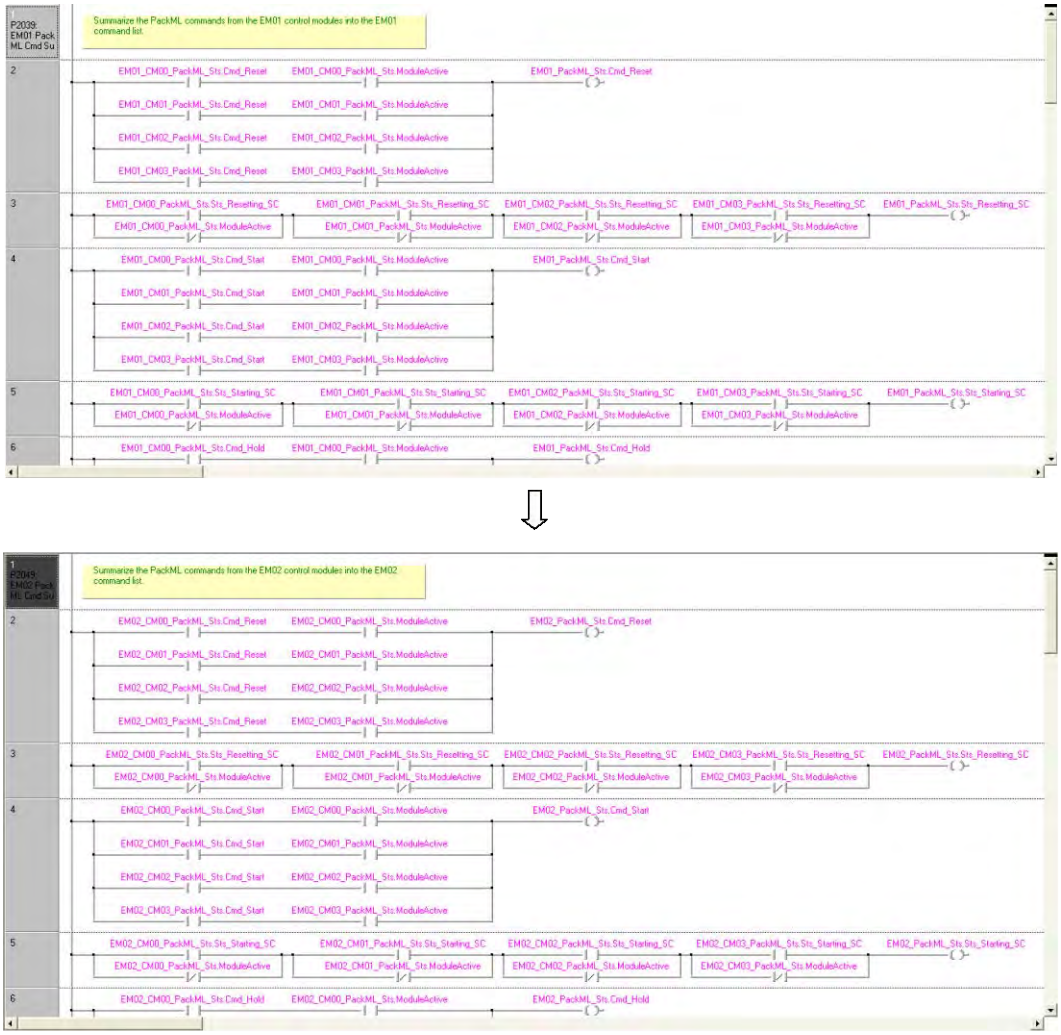
	Program Name		Comment
1	EM02_Main	...	
2	EM02_CM01_Routine	...	
3	EM02_CM02_Routine	...	
4	EM02_CM03_Routine	...	
5	EM02_EventControl	...	
6	EM02_PackML_Cmd_Sum	...	
7–35		...	

- Double Click on the "Program" in the new EM02_Main Program Block and it will display the logic which is the exact copy of the logic in EM01_Main POU. Create new global labels associating with EM02 and then edit the labels in the program to use the new labels. Modify the pointers to point to proper subroutines also. For example, the first subroutine call is using pointer P2040.

- For example, double click the EM02_CM01_Routine and ensure the label block pointer value is revised to the proper pointer label, i.e. P2041. Repeat the steps for all subroutines. OEMs need to add the necessary control code in this POU.



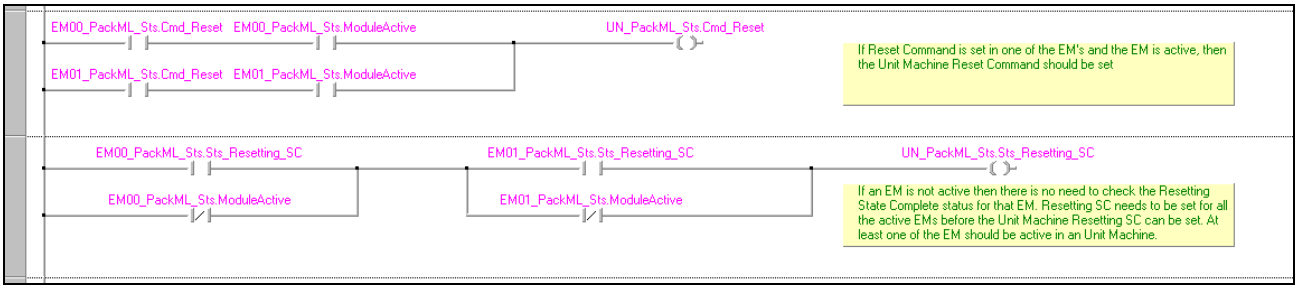
- Edit the EM02_PackML_Cmd_Sum routine to replace all labels from references to EM01 to EM02.



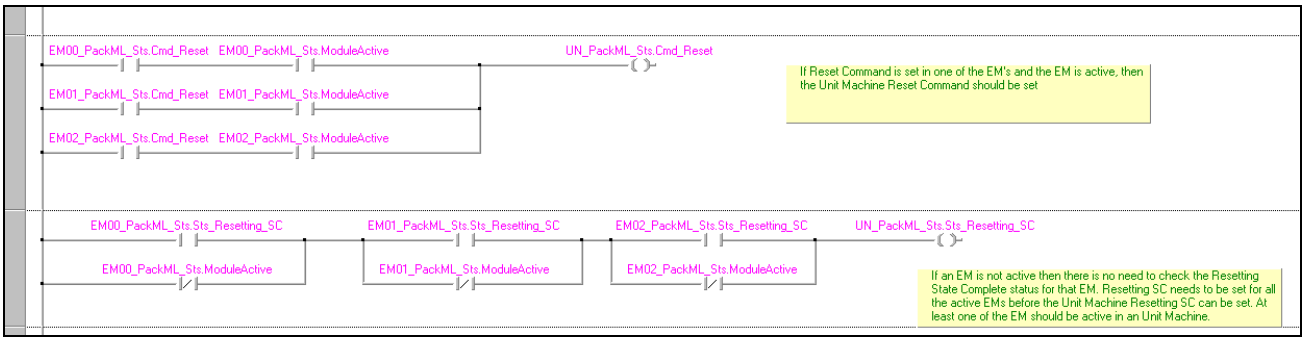
*9.3 **Modifying PackML_Main Routine***

The PackML commands and status from the newly added module EM02 need to be added to the PackML_Main routine so that they can be used to determine the transitions of the state machine properly.

Following is a portion of the PackML_Main routine showing the aggregation of “Reset” command signals and “Resetting” state “State Complete” signals for EM00 and EM01.



These rungs of logic need to be modified to include the signals from the newly added equipment module EM02. The resulting rungs are shown below:

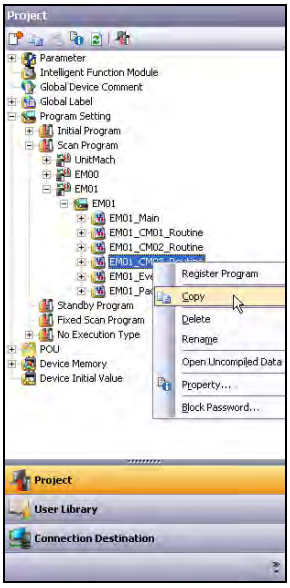


Similar modifications need to be done for all commands and status signals in PackML_Main for all new equipment modules added to the system.

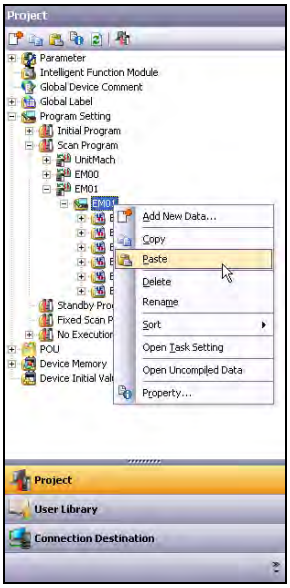
10 Example of Adding a Control Module to the Template System

This section documents the steps that an OEM will take to add the POU's of an additional control module (e.g. CM04) in a particular equipment module (e.g. EM01) in the PackML Implementation Template Project.

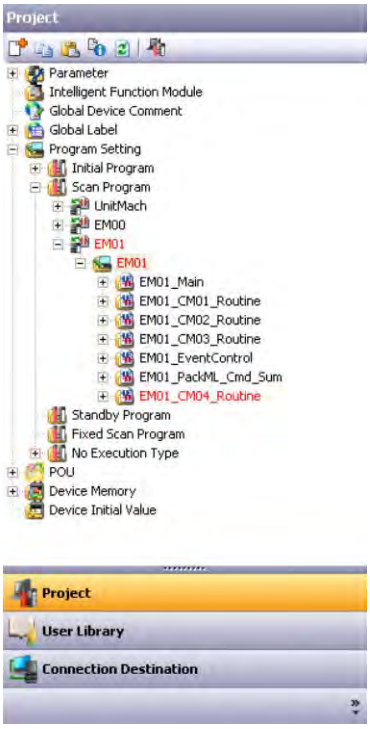
- Right click on one of the CM routine and select "Copy".



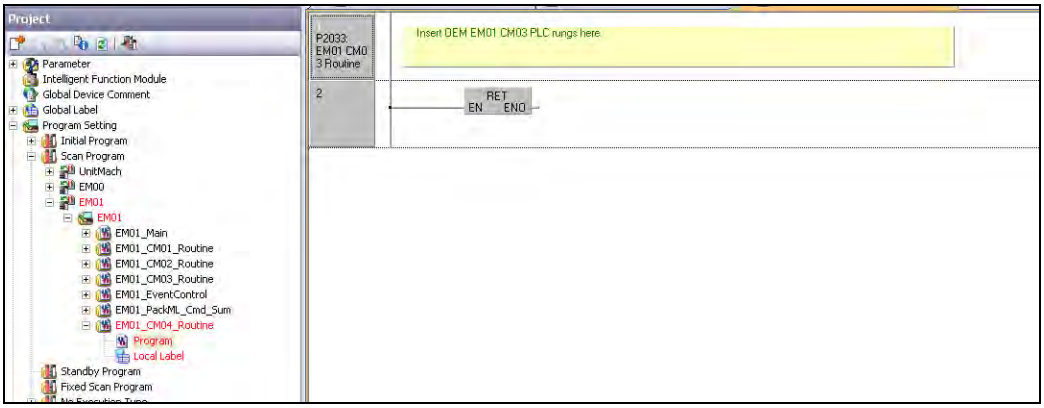
- Right Click on the Task EM01 and select “Paste”.



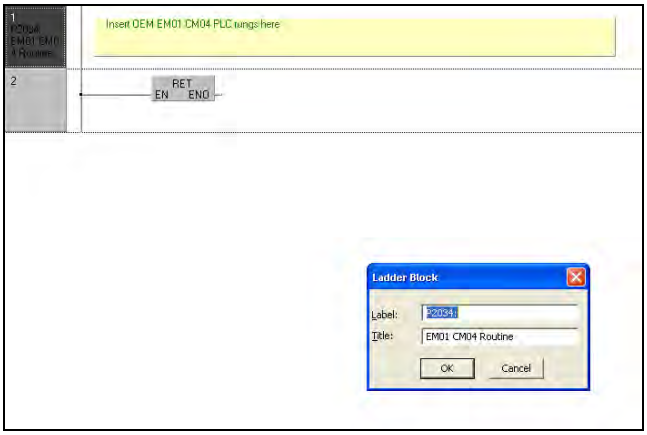
- A pop-up window will appear for the user to enter a new POU name and the new POU is added to the Task.



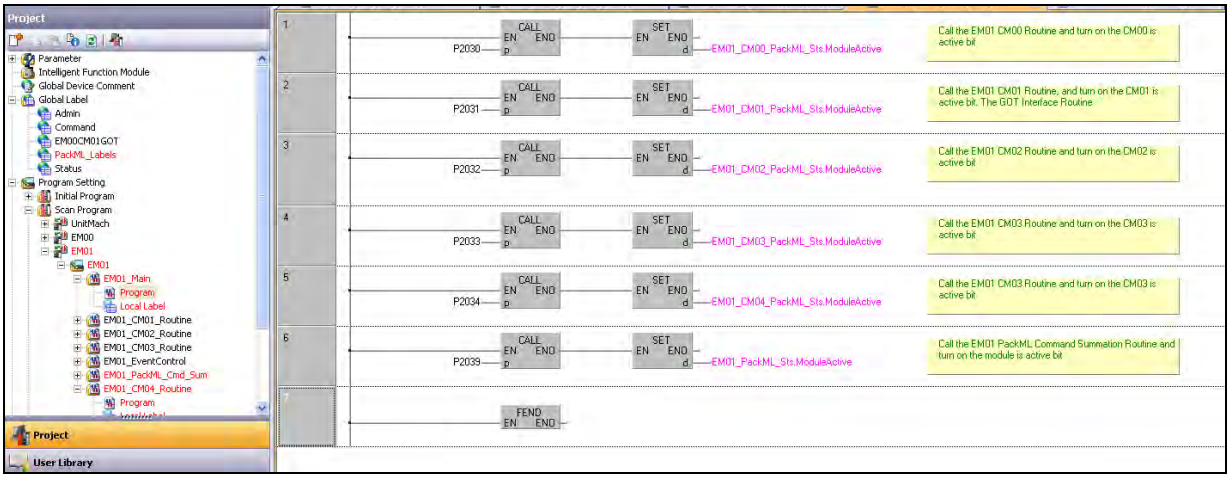
- Double Click the new program routine to open the editing window.



- Make the necessary modification to the Jump label and Title. The OEM needs to add the appropriate control code in this POU.



- In the EM01_Main POU, add the new "Call to Subroutine" instruction to call the new control module EM01_CM04_Routine.



11 Issuing PackML Commands in a Machine Program

The purpose of this section is to show how PackML commands can be issued in OEM's machine control code to cause the Unit Machine State diagram to transition from the current state to the desired next state.

The key steps that need to be programmed regarding a state transition are:

- Reset the command(s) that caused the transition from the previous state to the current state.
- Issue the command that will cause the transition from the current state to the desired next state.
- Clear all commands that may cause the transition away from the desired next state.

The following two examples illustrate the use of these steps to ensure proper state transition.

11.1 Example 1: Transition from Producing Mode Starting State to Execute State

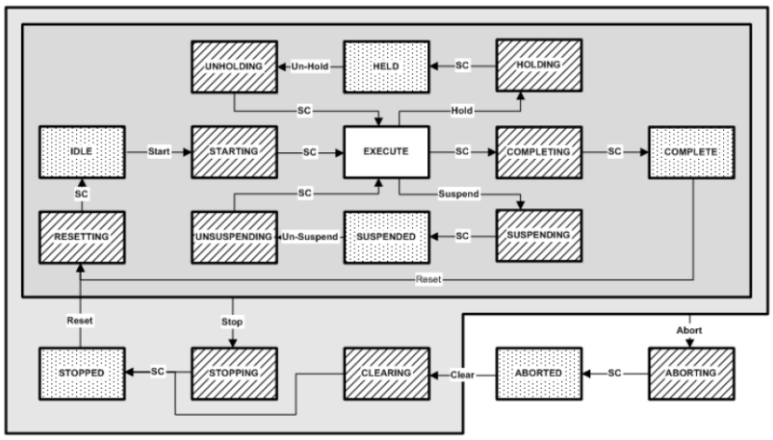


Figure 20 – Producing Mode State Model

If the Unit Machine is in the Producing Mode, Starting state as shown in the State Model in Figure 17, when the execution of the machine control logic of Equipment Module 00 and Control Module 02 in the Starting State is completed, the State Machine should proceed to the "Execute State" when the Starting State "State Complete" (SC) command is issued.

In order to cause this transition of states, the following logic can be used:

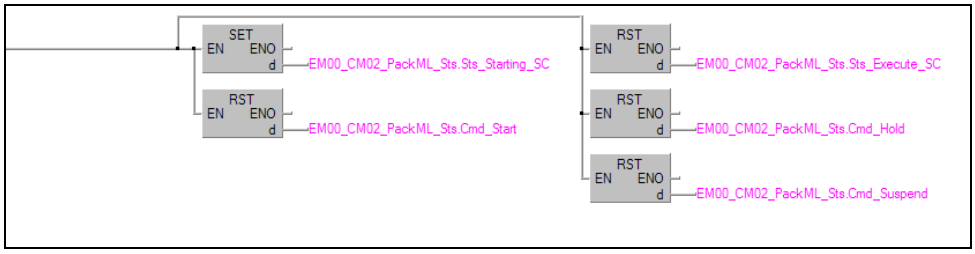


Figure 21 – Sample Ladder Logic for Handling PackML Commands

- The command EM00_CM02_PackML_Sts.Sts_Starting_SC is set to command the transition from the Starting State to the Execute State.
- The command EM00_CM02_PackML_Sts.Cmd_Start should be reset. The Start command must have been issued earlier in the machine control logic to cause the machine to transition to the Starting state. Thus it is a good practice to reset the command when leaving the Starting State to avoid any error by leaving the Start

Command active. The Start Command can actually be reset earlier in the Starting State machine control logic if the user chooses to.

- Referring to the State Model in Figure 17, the Execute State can potentially transition to one of the "Holding", "Completing" and "Suspending" states depending on the PackML command that will be issued when the Execute State logic is complete. It is important that the commands causing the State Model to transition from the Execute State be reset so that the State Model will be transitioned to and remain in the Execute State properly. Thus, referring to Figure 18, the following commands EM00_CM02_PackML_Sts.Sts_Execute_SC, EM00_CM02_PackML_Sts.Cmd_Hold, and EM00_CM02_PackML_Sts.Cmd_Suspend should be reset.

11.2 Example 2: Transition from Manual Mode Execute State to Stopping State

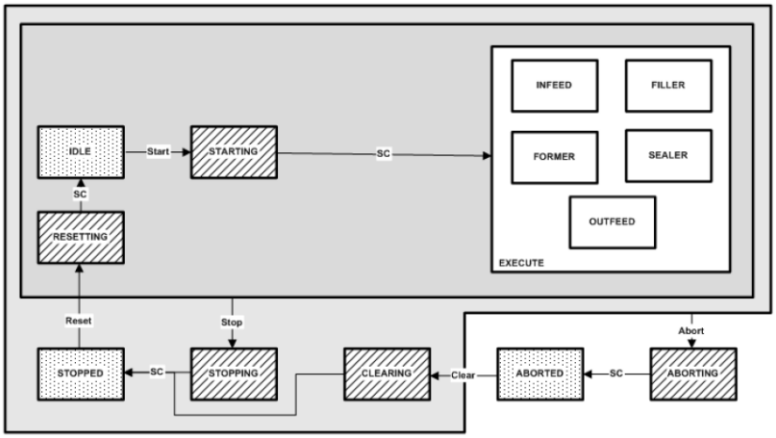


Figure 22 – Manual Mode State Model

Assuming the Unit Machine is in the Manual Mode, Execute state as shown in the State Model Figure 19. During the execution of the Execute State machine control logic in Equipment Module 00 and Control Module 02, it is necessary for the machine to go into "Stop State", the following logic can be used:

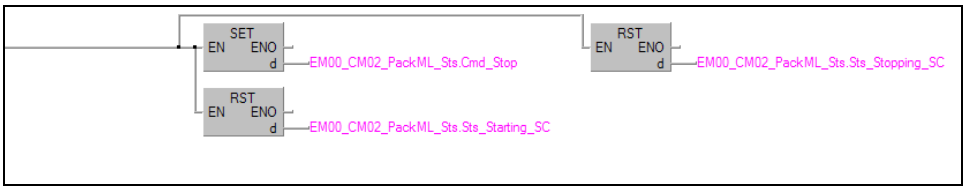


Figure 23 – Sample Ladder Logic for Handling PackML Commands

- The command EM00_CM02_PackML_Sts.Sts_Stopping_SC is set to command the transition from the Execute State to Stopping State.
- The command EM00_CM02_PackML_Sts.Sts_Starting_SC should be reset assuming the machine is in the Execute State from the Starting State. It is a good practice to reset the command when leaving the Execute State to avoid any error by leaving the Sts_Starting_SC Command active.
- Referring to the State Model Figure 19, the Stopping State will transition to the Stopped State when the execution in the Stopping State is complete.
    - It is important to reset the EM00_CM02_PackML_Sts.Sts_Stopping_SC command to ensure the logic of Stopping State is executed properly.

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 7 – GOT Screens

Release 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

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 QnUDEH PLC and a GOT-16 HMI. The system architecture used to create the Mitsubishi PackML is shown in the following block diagram. The PLC is a Q06UDEHCPU and the GOT is a GT-16s with the resolution of 800 x 600. Because of the large number of tags required to support the PackTags specification, an extended memory card is required to be installed in the Q06UDEHCPU.

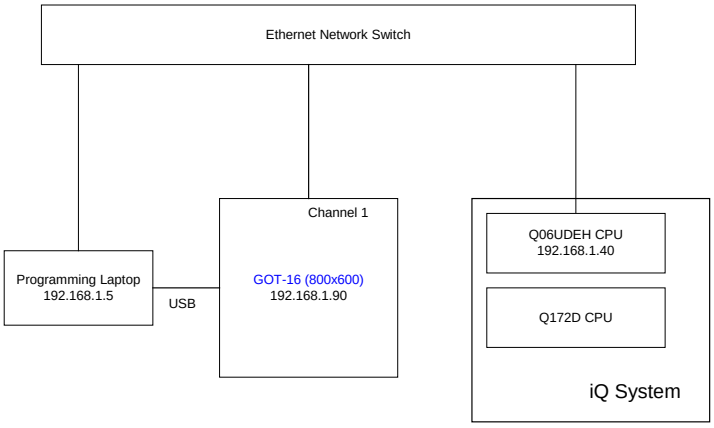


Figure 1 – Mitsubishi PackML Template System

The programming laptop is where the iQ Works is executed. The laptop is connected to the GOT using the USB port to download screen information and to the Q06UDEHCPU through the GOT Ethernet Transparent Mode. The Transparent Mode is set up to go through the Ethernet port on the Q06UDEH CPU (with IP Address 192.168.1.40). During the system operation, GOT is configured to work with the iQ PLC through the same Q06UDEHCPU Built-in Ethernet port.

3 GOT Communication Channel Configuration

The GOT in the Template system uses the USB 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.

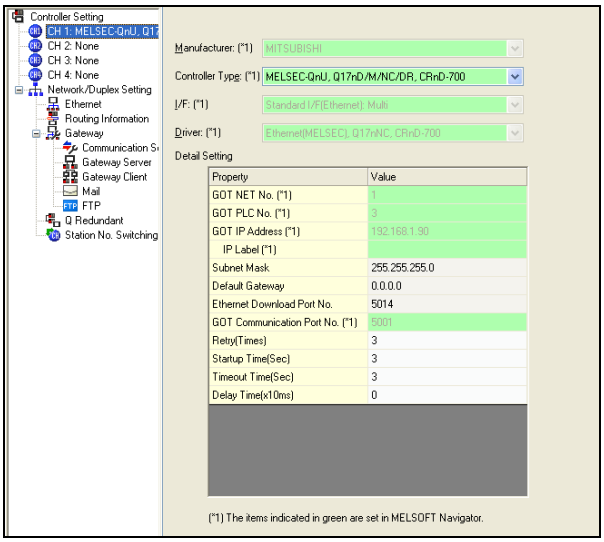


Figure 2 – Communication Channel 1 Configuration

Select the "Ethernet" in Network/Duplex Setting to set the channel 1 as the Host to the PLC.

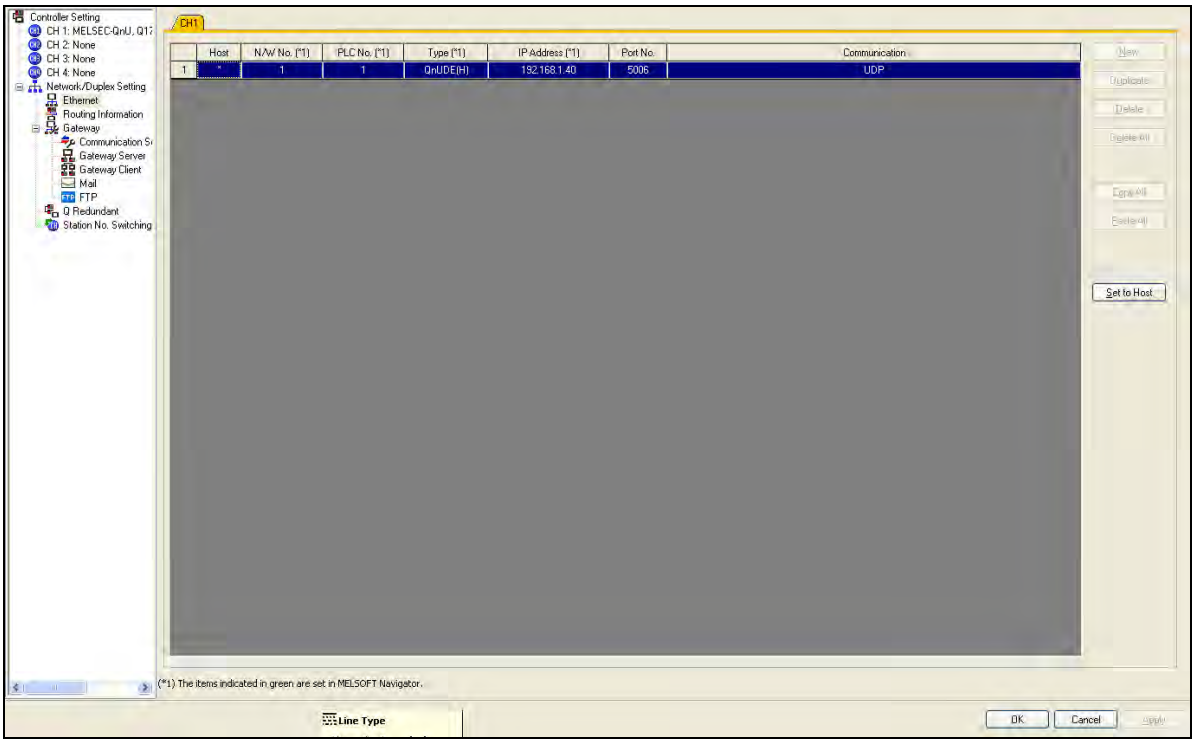


Figure 3 – Setting Channel 1 As Host

And then select the Communication Setting to verify all parameters.

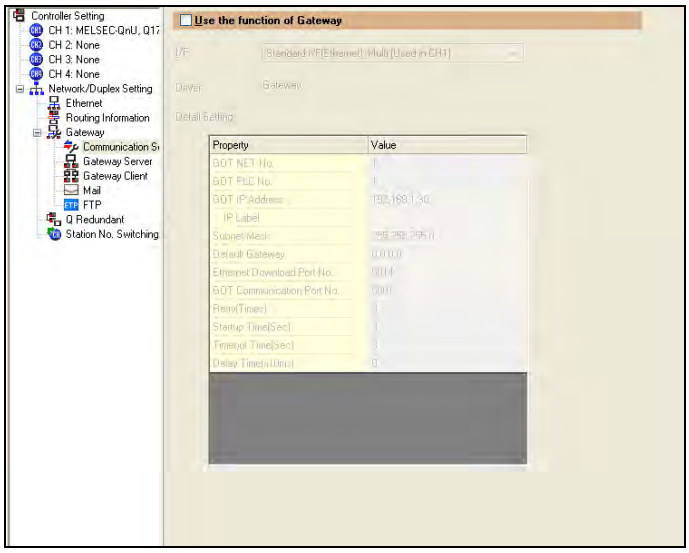


Figure 4 – Verifying Channel 1 Communication Settings

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.

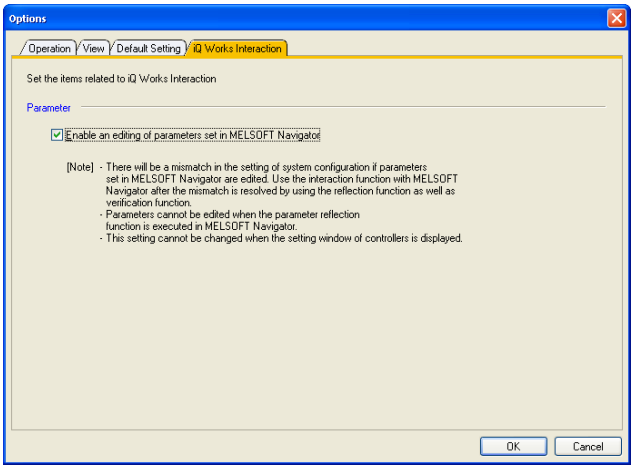


Figure 5 – Option to Modify Parameters Set in iQ Works

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

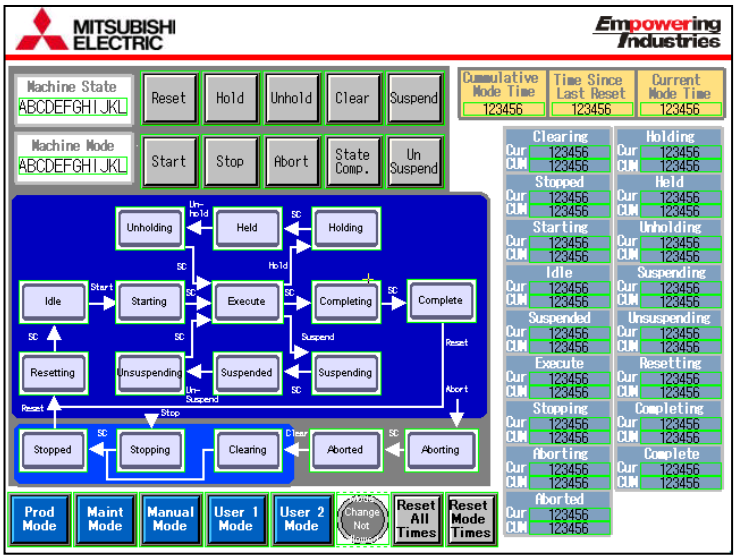


Figure 6 – Producing Mode Screen

4.1.2. Maintenance Mode Screen

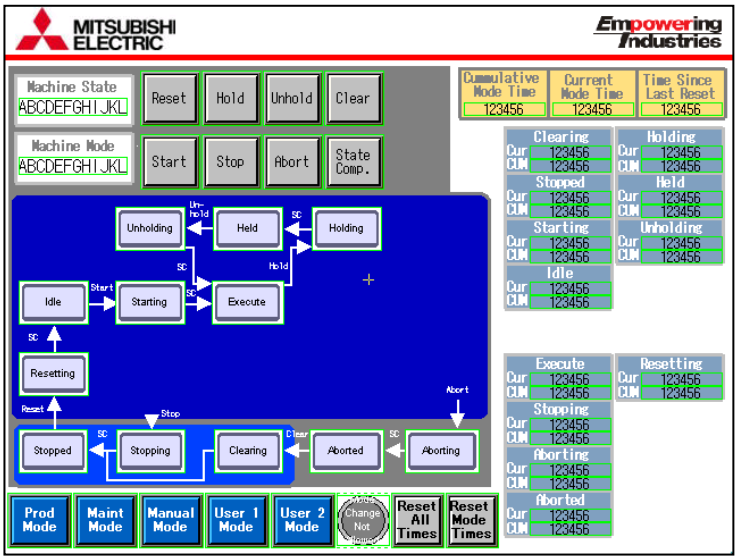


Figure 7 – Maintenance Mode Screen

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.

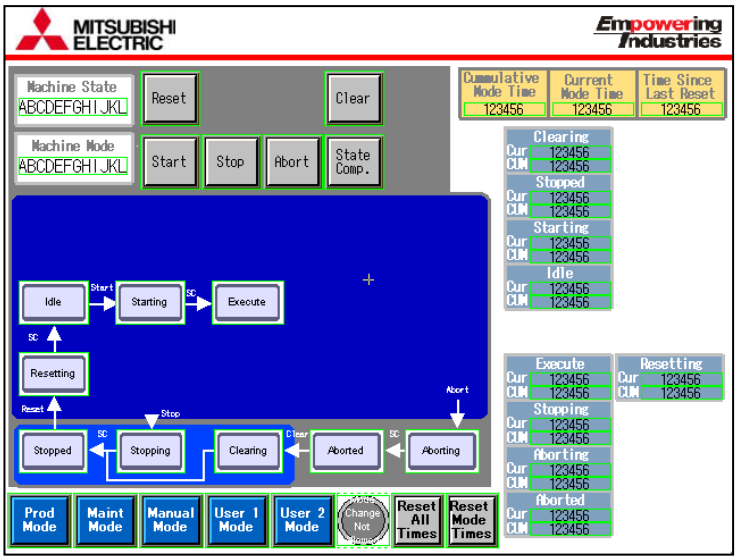


Figure 8 – Manual Mode Screen

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

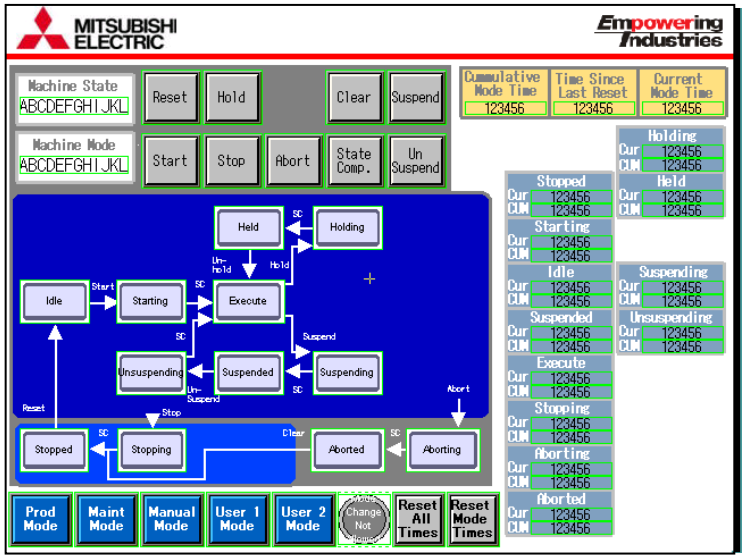


Figure 9 – User Defined Mode Screen

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.

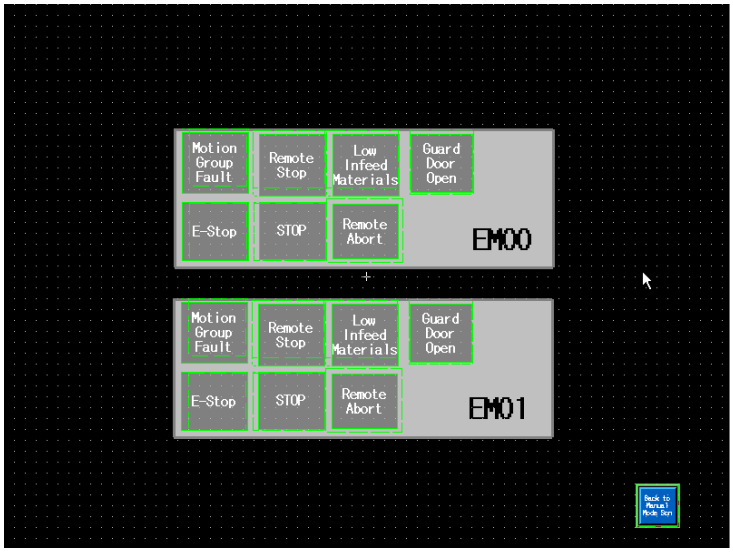


Figure 10 – Event Test Screen

MITSUBISHI ELECTRIC AUTOMATION, INC.

Engineering Group

Users Guide

OEM PackML Implementation Templates

Part 8 – Minimum Set PackML Template

Release 3, Version 5

Content

Revision History

Version	Revision Date	Description
R2 V1.0	July 31, 2010	Initial release of PackML OEM Implementation Templates Release 2
R3 V1.0	June 31, 2014	Release of updated PackML OEM Implementation Templates Release 3
R3 V5	March 9, 2016	Release of updated PackML OEM Implementation Templates Release 3 V5

1 Introduction

This document describes the program structure of the Minimum Set PackML Implementation Template project. The minimum set template is essentially a stripped down version of the full PackML implementation template. The purpose for providing a minimum set template is to reduce the cost of the hardware and the complexity of the program. The minimum template is ideal for simpler machines that do not require all the PackTags and states described in earlier sections of this document.

The main differences are:

- Reduced hardware specifications
- Reduced set of PackTags
- Reduced number of PackML states
- No event handling FB
- Reduced States on the GOT screens

The implementation still uses the same function block library, but not all FBs are used in the sample program.

This document outlines the differences between the full implementation and the minimum implementation. Refer to other parts of the documentation for information on how to use the FB, information on program structure, parameters, label implementation etc.

2 Minimum PackML Template System Architecture

The minimum set PackML templates are designed to run on reduced system specifications when compared to the full template. The PLC is a Q03UDEHCPU and the GOT is a GT-16s with the resolution of 640 x 480. There is also no need for an extended memory card. System architecture with the reduced hardware is shown below.

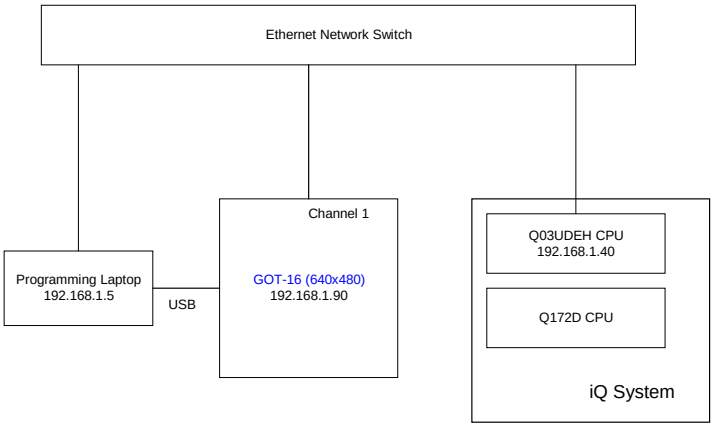


Figure 1 – Mitsubishi PackML Template System

3 Minimum PackTags

The minimum required tags for a PackML compliant system is shown below.

Minimum Admin Tags

PackTag Specification		GX Works 2 Labels	
Tags	Data Type	Label	Data Type
UnitName.Admin.ProdProcessedCount[#].Count	Int(32bit)	gvsta_Adm_ProdProcessedCnt[#].d_Count	Double Word[Signed]
UnitName.Admin.ProdDefectiveCount[#].Count	Int(32bit)	gvsta_Adm_ProdDefectiveCnt[#].d_Count	Double Word[Signed]
UnitName.Admin.StopReason.ID	Int (32bit)	gvst_Adm_AlarmStopReason.d_ID	Double Word[Signed]

Table 1: Minimum Admin Tags

Minimum Command Tags

PackTag Specification		GX Works 2 Labels	
Tags	Data Type	Label	Data Type
UnitName.Command.UnitMode	Int (32bit)	gvd_Cmd_UnitMode	Double Word[Signed]
UnitName.Command.UnitModeChangeRequest	Bool	gvb_Cmd_UnitModeChangeRequest	Bit
UnitName.Command.MachSpeed	Real	gvl_Cmd_MachSpeed	FLOAT (Double Precision)
UnitName.Command.MaterialInterlock	Bool Structure	gvdu_Cmd_MaterialInterlocks	Double Word[Unsigned]/Bit String[32-bit]
UnitName.Command.CntrlCmd	Int (32bit)	gvd_Cmd_CntrlCmd	Double Word[Signed]
UnitName.Command.CmdChangeRequest	Bool	gvb_Cmd_CmdChangeRequest	Bit

Table 2: Minimum Command Tags

Minimum Status Tags

PackTag Specification		GX Works 2 Labels	
Tags	Data Type	Label	Data Type
UnitName.Status.UnitModeCurrent	Int (32bit)	gvd_Sta_UnitModeCurrent	Double Word[Signed]
UnitName.Status.UnitModeRequested	Bool	gvb_Sta_UnitModeChangeRequested	Bit
UnitName.Status.UnitModeChangeInProcess	Bool	gvb_Sta_UnitModeChangeInProcess	Bit
UnitName.Status.StateCurrent	Int (32bit)	gvd_Sta_StateCurrent	Double Word[Signed]

UnitName.Status.MachSpeed	Real	gvl_Sta_MachSpeed	FLOAT (Double Precision)
UnitName.Status.CurMachSpeed	Real	gvl_Sta_CurMachSpeed	FLOAT (Double Precision)
UnitName.Status.EquipmentInterlock	Bool Structure	Sta_EquipmentInterlock	SDT_Sta_EquipInterlock
UnitName.Status.EquipmentInterlock.Blocked	Bool	Sta_EquipmentInterlock.Blocked	Bit
UnitName.Status.EquipmentInterlock.Starved	Bool	Sta_EquipmentInterlock.Starved	Bit

Table 3: Minimum Status Tags

4 Minimum States

The full implementation of the PackML has 17 states. The minimum states required to be PackML compliant are 1 (Stopped), 3 (Idle), 7 (Execute) and 10 (Aborted).

State	Name
1	**STOPPED**
2	**STARTING**
3	**IDLE**
4	**SUSPENDING**
5	**SUSPENDED**
6	**UNSUSPENDING**
7	**EXECUTE**
8	**STOPPING**
9	**ABORTING**
10	**ABORTED**
11	**HOLDING**
12	**HELD**
13	**UNHOLDING**
14	**COMPLETING**
15	**COMPLETE**
16	**RESETTING**
17	**CLEARING**

Table 4: Minimum Required Tags

It is up to the OEM programmer to decide how many states are needed for a particular application. The application may use up to 17 states, but must implement the four base minimum states in order to be compliant with PackML. The next section describes how to enable/disable states to match the application requirements.

5 Template PLC Program

The sample minimum set PLC program is basically a stripped down version of the full implementation template. The main difference is that the template does not include logic for event control and uses only the minimum four PackML states instead of the full 17.

5.1 Configuring States to be used

To configure the states, modify the input parameters in the "PackMLStateDisable_Set(n)" FB instances in the "PackML_UnitMachine_Setup" POU. By default, the inputs are set to "TRUE" which means that all the states except for the minimum required states are set to disabled.

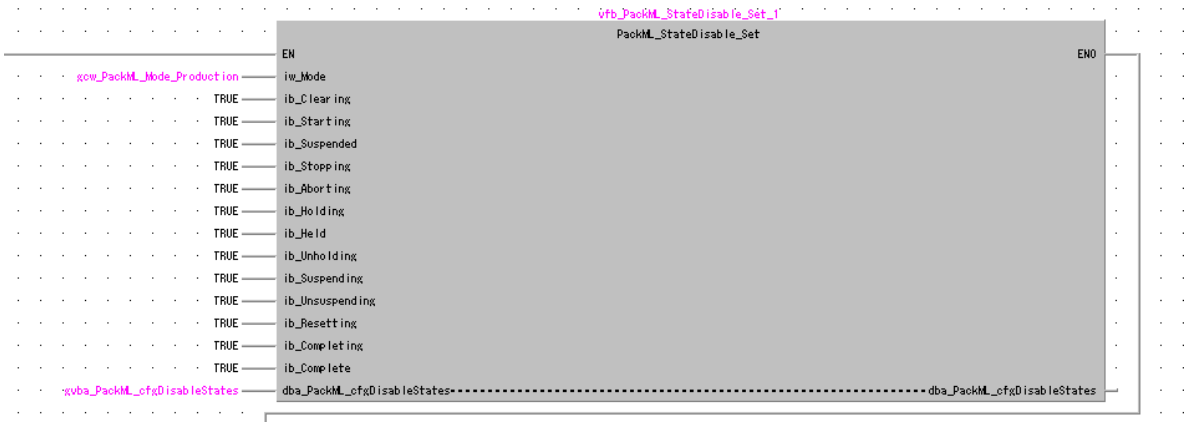


Figure 2: PackML_StatesDisable_Set Function Block

Now configure the "PackML_ModeTransition_Init_(n)" Fb instances to allow for transitions between states

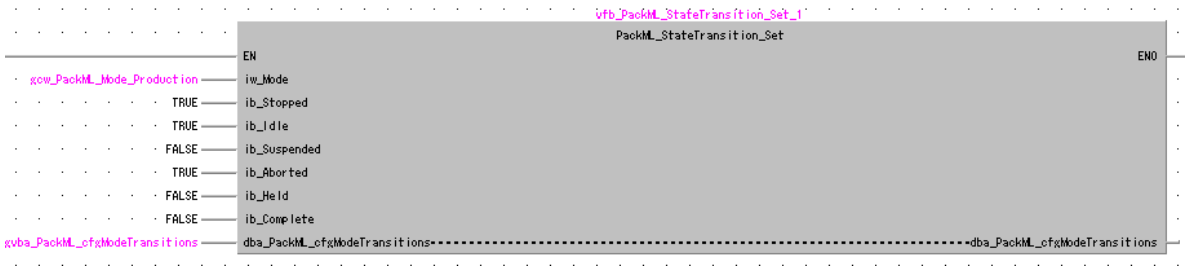


Figure 3: PackML_ModeTransition Function Block

Now add logic to utilize the added states to the "EM00_PackML_Cmd_Sum" and "PackML_Main" to make use of the states (copy logic from the full template).

6 GOT Sample Screens

Five sample screens, consisting of five PackML Mode Screens are included in the Minimum Set 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 minimum four states and their respective timers are shown on the screen.

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

6.1 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. Only the four minimum required states and action buttons are displayed on the screen.

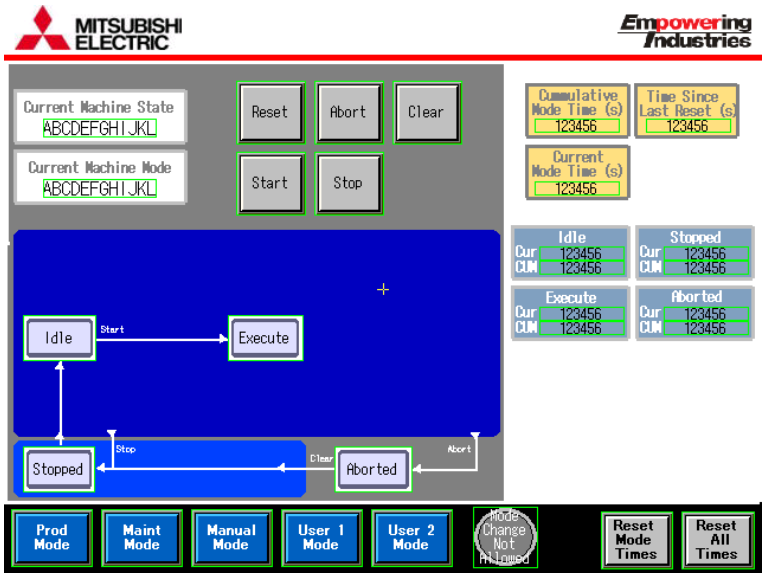


Figure 4 – Producing Mode Screen