APPLICATION NOTE

Allen-Bradley MMI_sample_loop Sample Logic

Purpose

This application note describes the Allen Bradley MMI_sample_loop sample logic operating theory, setup and options in general terms. More detailed explanations of each routine are provided as comments in the sample code. This logic is applicable to QuickStick, QuickStick High Thrust and MagneMover LITE systems.

Introduction

The provided MMI_sample_loop logic uses two state machines that work together, station manager and vehicle manager, to implement a station to station based approach to vehicle control. These two state machines also provide the feedback required for the user to implement and trigger their station processes. The logic also contains supporting routines for initializing and communicating with a system.

Sample Logic Routines

Main Routine (Main_Routine)

The Main_Routine calls all subsequent routines in the program. It also allows the user to restart the program by triggering the restart_demo bit. When triggered, a value of ten is placed in the step variable used to iterate through the cold_start_service routine. The cold_start_service, msg_service, hlc_link_monitor, and station_processing routines are called from the main. The for-loops that run the station manager and vehicle manager are also called from the main. These iterate from one up to the max_station_id and max_vehicle_id.

Cold Start Service (cold_start_service)

This routine uses the cold_start_service.step variable to iterate through all the steps required to start-up a MagneMotion system from an initial power up including the initialization of variables, verifying controller status, reset, startup, and sending the initial vehicle order. The sequential logic allows for the correct order of events to occur, and makes for simple diagnostics. The routine will call the init_constants routine and the init_stations helper routines during the first step. This startup sequence is also described in detail in the Application Development Recommendations Application Note (990000610).

Initialize Constants (init_constants)

init_constants sets PLC variables to a numerical value to make the logic easier to follow. Included in this routine are the max_path_id, max_station_id, max_vehicle_id and max_nc_id. These must be set by the user to the maximum number of paths, stations, vehicles, and Node Controllers in the system respectively.

Initialize Stations (init_stations)

init_stations fills the stations_array that is used by the station manager. First the array is zeroed out. After this, there are several example instances of the MMI_Init_station Add-On Instruction (AOI). These need to be filled in by the user in order to define the station position and the parameters the vehicle will use when departing the station. The station_entry parameter will be the stations_array[x] where the array element is the station ID to be associated with the data that follows.

Message Service (msg_service)

The msg_service routine is a convenient way to handle the explicit messages required to run a MagneMotion system. This mitigates the need for having many instances of message instructions throughout the code, which is often undesirable. This routine can be triggered from other routines, and for each message type it will send the message, wait for confirmation that the message was received, and resend the message if necessary. Each message type has an associated step variable. Setting the associated step variable to 20 triggers the process to send a message. The service will then reset a timeout timer, send the message and look for a response before the timer expires. If there is no response when the timer finishes the process is restarted and the message is resent.

Next Command Count (next_command_count)

When called, next_command_count returns a unique number. MagneMotion allows for an order number with every command sent. This order number is the host controller’s reference to know if the command was received or not. Before a command is sent, this routine is called to get a new number for the command count. Once the count reaches 1000000 it begins at 1 again.

Station Manager (station_mgr)

station_mgr is the state machine controlling the stations that the user defined in the init_stations routine. This routine works with the vehicle_mgr to control all vehicle movement in the system. The station_mgr has four states; idle, dwell, processing, and depart. The routine iterates over a variable "s". This is the iterator that is defined when the station_mgr for-loop is called from the Main_Routine, and iterates from one to max_station_id set in the init_constants routine. The state machine uses the following states:

> **STATION_STATE_IDLE:** In this state if the station hold flag is set for this station, do nothing. If it is not set, and if there is an active vehicle at the station, check to see if there is a dwell set for the station (as set by the init_stations routine). If so, move to the dwell state. If not, go to the processing state.
>
> **STATION_STATE_DWELL:** In the dwell state the dwell specified in init_stations is carried out. Once the dwell is complete, the station state moves to STATION_STATE_PROCESSING.
>
> **STATION_STATE_PROCESSING:** This state allows the user to input their own requirements before the vehicle leaves the station. The state change between processing and depart is controlled by the user in stations_processing.
>
> **STATION_STATE_DEPART:** The depart state will take the active vehicle from the station (stations_array[x].active_vehicle_id) and the next_station_id for the station (stations_array[x].next_station_id set in the init_stations routine) and fill in the vehicle manager array entry for this vehicle with the destination station from the current station. This will kick off the vehicle_mgr routine for this vehicle. The station then goes back to station_state_idle to wait for another vehicle.

The state of each station is tracked separately in the stations_array entry for that station.

Stations Processing (stations_processing)

In the station processing routine, the user can create conditions for specific stations to enter the depart state by examining specific values for the station state instead of the variable "s". For example, to specify what to do at station 1 create a check that stations_array[1].state is equal to processing. Fill in the required tasks, then set stations_array[1].state equal to depart. If the vehicle should leave the station immediately, simply set the station to depart as soon as it shows as processing.

Vehicle Manager (vehicle_mgr)

The vehicle manager routine works with the station_mgr routine to control the vehicles in the system. The vehicle_mgr routine has three states; idle, place order, and wait for arrival. The routine iterates over a variable "v". This is the iterator that is defined when the for-loop that runs the vehicle_mgr is called from the Main_Routine, and iterates from one to the max_vehicle_id set in the init_constants routine. The state machine uses the following states:

VEHICLE_MGR_STATE_IDLE: The idle state checks to see if the specified vehicle has a destination station. This is set by the stations_mgr when it is in the depart state. If there is a vehicle with a destination station, the vehicle moves to the VEHICLE_MGR_STATE_PLACE_ORDER state.

VEHICLE_MGR_STATE_PLACE_ORDER: In the place order state the program formulates a vehicle order based on the vehicle's destination station. The order will use the information from the init_stations routine to determine the velocity, acceleration, path ID, and direction to use. After the order is placed, go to the VEHICLE_MGR_STATE_WAIT_FOR_ARRIVAL state.

VEHICLE_MGR_STATE_WAIT_FOR_ARRIVAL: This state will check that that the vehicle received the order and the order completed. Once it has, the vehicle being examined will be put in the stations_array[dest_station_id].active_vehicle_id. The vehicle manager uses the vehicle's destination station to know which active_vehicle entry to put this vehicle in. This is how the station manager and vehicle manager interact. Once the order is complete and the data has been copied the state of the vehicle transitions to VEHICLE_MGR_STATE_IDLE.

The state of each vehicle is tracked separately in the vehicle_mrg_array entry for that vehicle.

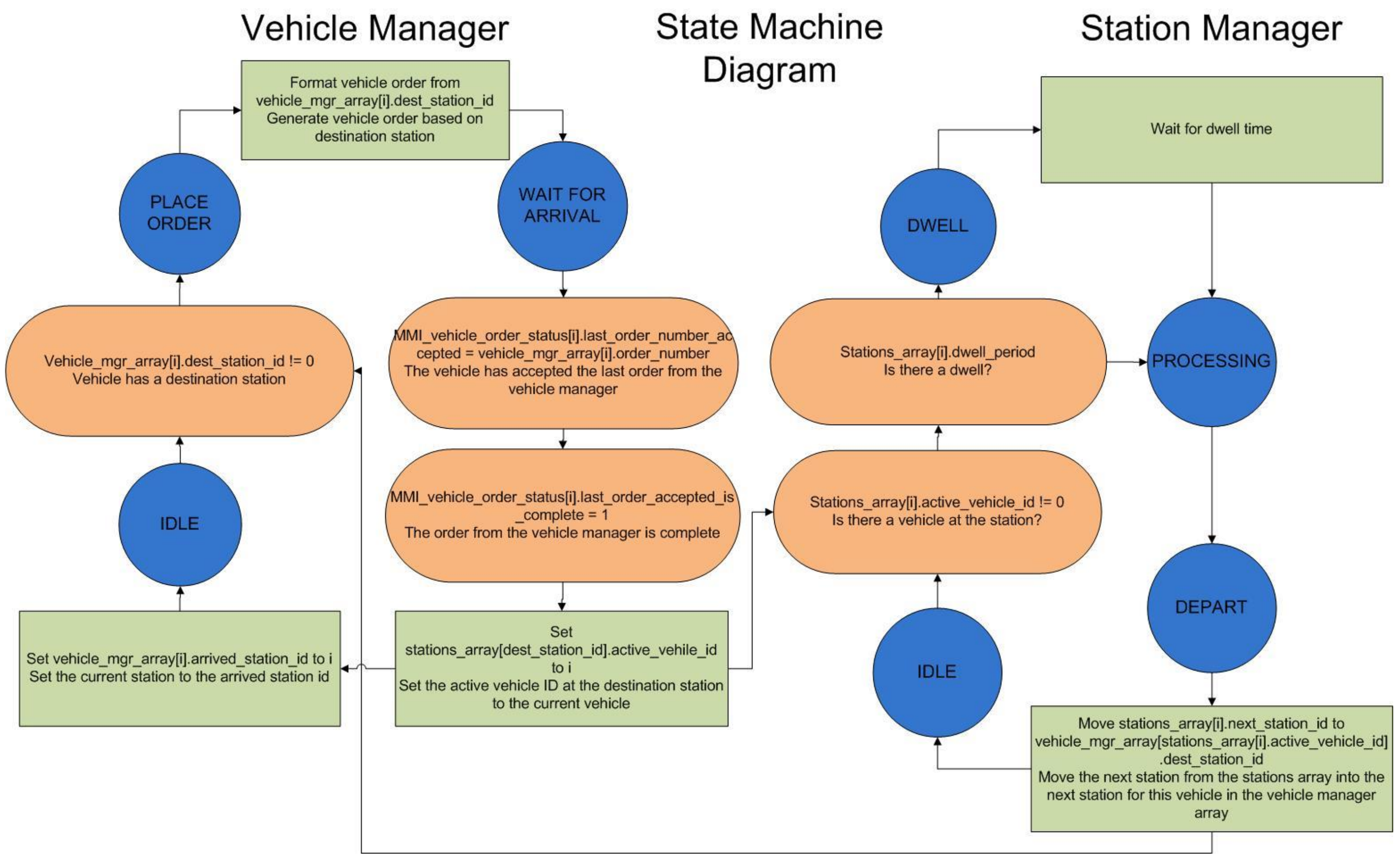
Vehicle Manager
State Machine Diagram
Station Manager
Format vehicle order from vehicle_mgr_array[i].dest_station_id Generate vehicle order based on destination station
PLACE ORDER
WAIT FOR ARRIVAL
Vehicle_mgr_array[i].dest_station_id != 0 Vehicle has a destination station
MMI_vehicle_order_status[i].last_order_number_accepted = vehicle_mgr_array[i].order_number The vehicle has accepted the last order from the vehicle manager
MMI_vehicle_order_status[i].last_order_accepted_is_complete = 1 The order from the vehicle manager is complete
IDLE
Set vehicle_mgr_array[i].arrived_station_id to i Set the current station to the arrived station id
Set stations_array[dest_station_id].active_vehile_id to i Set the active vehicle ID at the destination station to the current vehicle
Wait for dwell time
DWELL
Stations_array[i].dwell_period Is there a dwell?
PROCESSING
Stations_array[i].active_vehicle_id != 0 Is there a vehicle at the station?
DEPART
IDLE
Move stations_array[i].next_station_id to vehicle_mgr_array[stations_array[i].active_vehicle_id].dest_station_id Move the next station from the stations array into the next station for this vehicle in the vehicle manager array

Figure 1: Vehicle Manager and Station Manager State Machines

Sample Logic Setup Procedure:

1. Set the path for all explicit messages to target the HLC being used. This can be done by sorting all controller level tags, finding the tags of type MESSAGE, and configuring them. The communication path will need to be set as Ethernet Card, Port, HLC IP Address, 1, 0 for Controllogix PLCs and Port, HLC IP Address, 1, 0 for CompactLogix PLCs.

   For example, to connect a ControlLogix to an HLC at IP address 192.168.110.91 from port 2 of a card named LocalENB you would need to use communications path:

   "LocalENB, 2, 192.168.110.91, 1, 0"

   If the PLC was a CompactLogix the path would be:

   "2, 192.168.110.91, 1, 0"

2. In the init_constants routine, set the max_nc_id, max_path_id, max_station_id, and max_vehicle_id. max_nc_id and max_path_id need to be set to the largest Node Controller ID and Path ID in the system. max_station_id needs to be set as equal to or larger than the number of stations in the system, and max_vehicle_id needs to be set as equal to or larger than the number of vehicles in the system.

3. In the init_stations routine, create the desired stations for the layout. Under station_entry insert an array element of the stations_array. The array element is the station ID. For example, inserting stations_array[1] creates station ID 1. Fill in the remaining parameters for the station as desired.

4. In the stations_processing routine; for each station in state STATION_STATE_PROCESSING insert a process for any vehicle that arrives at the specified station. If the vehicle should move on immediately, move the station state to STATION_STATE_DEPART as soon as it enters the processing state.

5. In the Main_Routine trigger the restart_demo bit. The system will reset, startup, and begin moving vehicles.

Logic Features:

Several tags are provided that allow the logic to be expanded for the user's application.

stations_array[x].active_vehicle_id: For station ID x, this field will populate with the active vehicle ID at a station.
stations_array[x].station_hold_flag: Setting this bit will hold the next active vehicle at the station until the bit is released.

stations_array[x].next_station_id: Modifying this value allows the user to change which station the station with ID x directs to.

It is best to use the fields provided to write additional logic rather than modifying the vehicle manager and station manager. MagneMotion has tested these routines thoroughly as they are written.

Summary

The MMI_sample_loop logic provides a good base for Allen Bradley PLC controlled station to station based programs. If another approach or platform is chosen by the user, a similar approach may be taken and the included routines may apply to the new solution. This is MagneMotion's preferred approach and the Customer Support team is available to answer questions on this code sample.

Related Documents:

990000437 – MANUAL, Host Controller Ethernet-IP Communications Protocol

800-0100-02 – MMI_Sample_Loop PLC Sample Logic

990000610 – Application Note, Application Development Recommendations

More Information

MagneMotion Website: www.magnemotion.com

Questions & Comments: www.magnemotion.com/about-magnemotion/contact.cfm

Revision History

Rev.	Change Description
A	Initial release