Rockwell Automation Application Content

Rockwell Automation Robotics Libraries

Reference Manual

Robot – Device Handler

raM_Robot_Dvc_DeviceHandler	**v2**
raM_Robot _Dvc_DHLP	**v1**
raM_Dvc_DH_SysIni	**v1**

January, 2024

Rockwell Automation

Important User Information

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

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

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

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

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

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

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

Table of Contents

1 Overview

Device Handler – Robotics

- Provides the basic instruction set for commissioning and enabling a robot.
- Provides a set of instructions for basic operation of robotics system(s): Energize, De-Energize, Clear faults, Path commands, etc.
- Provides an architecture for virtual or physical operation of the motors associated with the robot.
- Provides an enhanced management of robot as well as individual joints, cartesian, and motors in the form of axes with
    - Text-based status and diagnostics
    - Faceplate
    - Error management

Use when:

- Utilizing Rockwell Automation Robotics Libraries methods for robot control, including Robot_rOS HMI interface.
- Easy switch between virtual and physical operation is desired.

Do NOT use when:

- Device is not a robot supported by Logix Coordinate Systems.
- Rockwell Automation Robotics Libraries methods for robot control are not being utilized.

1.1 Prerequisites

- PAC
    - ControlLogix 5580 / CompactLogix 5380 or newer with 3.5MB or greater memory
- Studio 5000 – Logix Designer
    - v35.0 →
- FactoryTalk View ME Station
    - v12.00.00 →·
- FactoryTalk View Studio ME
    - v13.00.00 →·
- Studio 5000 – Application Code Manager
    - V4.03.00 →

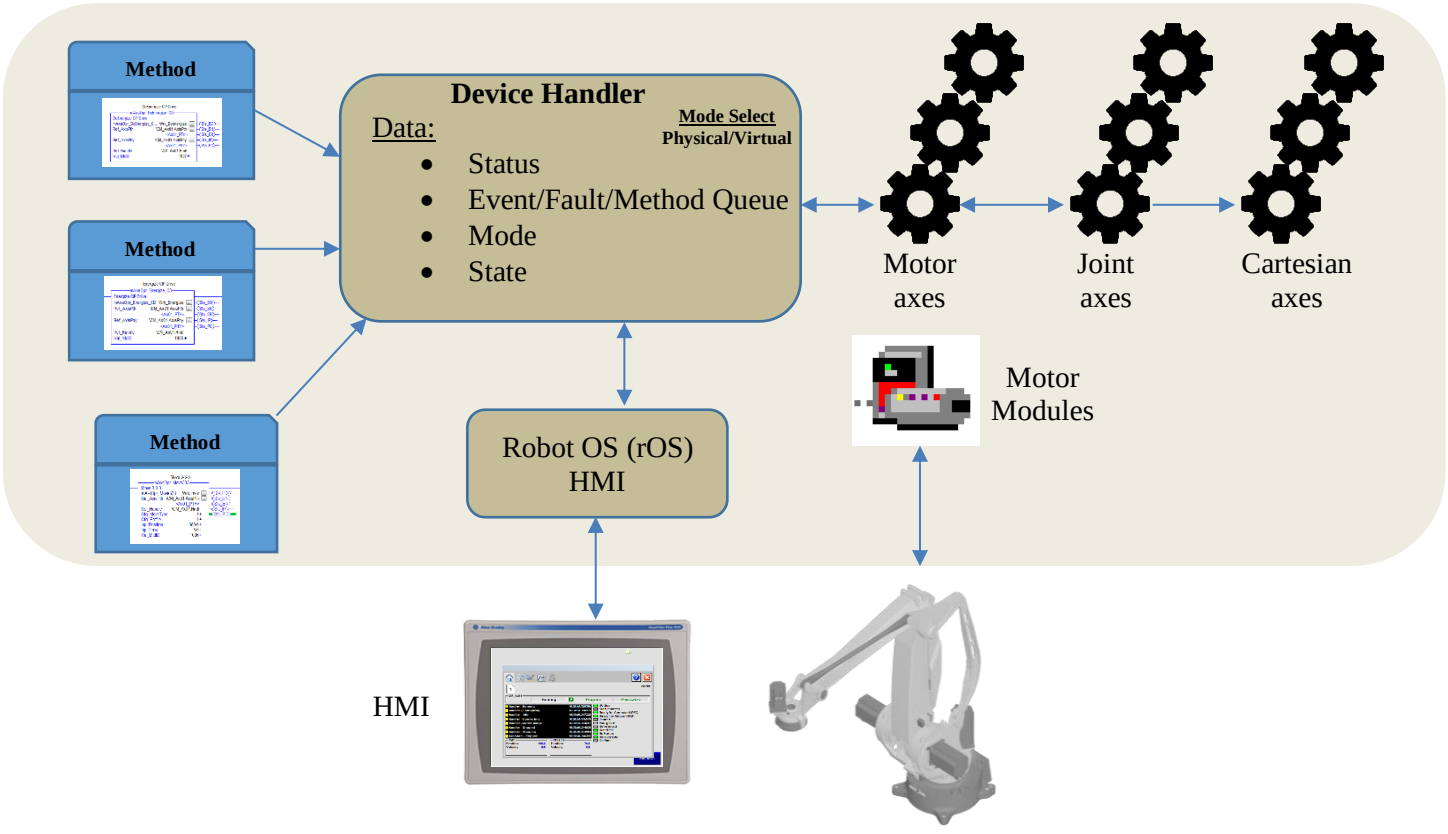
1.2 Functional Description

The Robot Device Handler is a library set developed for Logix that can be used with supported robot geometries/actuators and provides powerful tools for diagnostics and hardware management. At its core, it consists of a program paired with associated data structures which synchronizes physical motors with path axes in the PLC and provides diagnostics and status of your cartesian, joint, and motor axes. The libraries also provide path planning capabilities to execute coordinated moves in cartesian and joint space, with the ability to blend between different targets.

The Device Handler provides:

- Focused and enhanced set of status tags allow for easy troubleshooting.
- Hardware abstraction by easily switching off communication to the hardware and allowing virtual operation with all robot commands and instructions, allowing the user to develop much of the application code virtually without any hardware present.
- Faceplates for visualization.
- At-a-glance indication of status, methods, faults, state, and commands.
- Integrated path planner that supports various robot move types such as Point-to-point and Cartesian Linear moves.
- Path planning that can speed up or slow down the robot using a set of instructions and ability to stop on a path.

All these features combined provide the user with tools that offers quick feedback, shortened recovery time, and a simplified implementation.



During the Configuring state, the Path Axis automatically takes on the configuration of the Physical Motors.

Enhances user experience by:

- Consolidating relevant information
  - Essential robot and hardware status
  - Text-based diagnostic messages for faults and events
- Combining and simplifying essential robot status
  - Consolidates status bits for use in application programming
    - All robot hardware is connected
    - Path planning is available
    - Robot is energized
  - Text-based first out fault for fault aggregation
  - Accessible Event Queue
- Virtualization at runtime
  - Manages coupling of joints to motors
  - Inhibits all physical axes and modules
  - Motion executed on path axes and status is reflected on and from physical motors
  - Allows decoupling of physical motors and hardware for/during testing, debugging, and simulation
- Robot path planning capabilities
  - Incremental and absolute moves
  - Blending for cartesian and joint moves
  - Supported motion profiles for path planning:
    - Cubic
    - Poly-5
    - Sin^2
    - Modified Sine

1.3 Execution

- Handler
  - Motion Group Execution / high priority
- Language Pack
  - 512 ms (periodic task) / low priority
- System Initialization
  - PAC Power-Up Handler

Each Handler program will require 2ms of execution time. Note that the Motion Group Coarse Update Rate must be adjusted accordingly to ensure that each robot instance has enough time to complete execution. Higher Coarse Update Rates may have an impact on the overall performance of the application.

1.4 Footprint

Characteristic	Description	Value		Unit
Instance +Definition	Estimated memory required to store the object definition, including all dependents (including 1 Instance).	6 DoF Robot	2,850,000	Memory blocks
Instance +Definition	Estimated memory required to store the object definition, including all dependents (including 1 Instance).	4 DoF Robot	2,700,000	Memory blocks
Instance +Definition of all* Robotics Library objects	Estimated memory required to store the object definition, including all dependents (including 1 Instance).	6 DoF Robot	5,250,000	Memory blocks
Instance +Definition of all* Robotics Library objects	Estimated memory required to store the object definition, including all dependents (including 1 Instance).	4 DoF Robot	4,400,000	Memory blocks

Characteristic	Description	Value		Unit
Additional Instance		6 DoF	1,130,000	Memory blocks
Additional Instance		4 DoF Robot	980,000	Memory blocks

* Includes instances of every library object included in this release, e.g. rOS, DesignPatterns, Implements as well as assets

NOTE: Footprint estimations INCLUDE Cartesian, joint, and motor axes as well as power supply axes

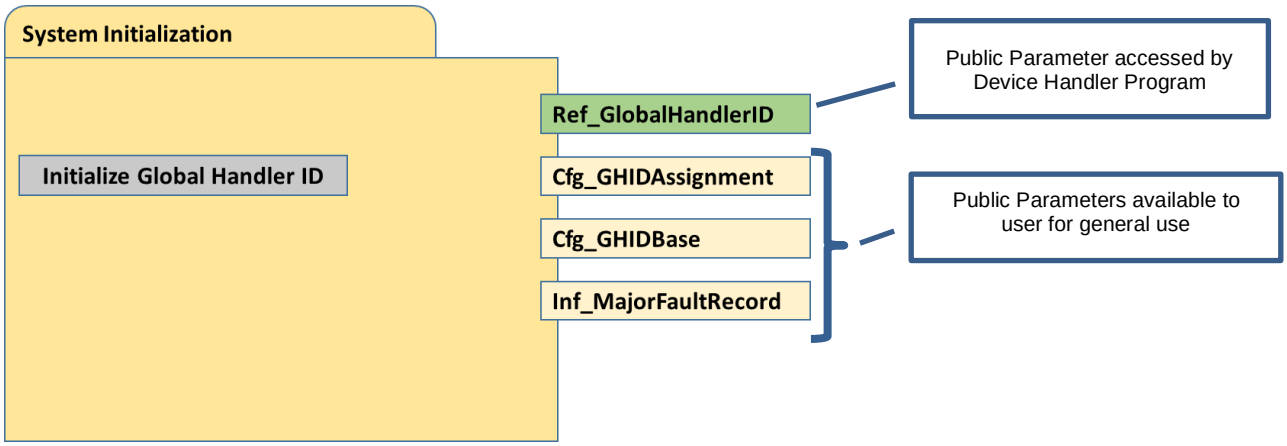
2 Handler Architecture

1.1 Understanding Device Handler Components

The Robot Handler requires three components to exist in the PAC:

2.1.1 System Initialization (raM_Dvc_DH_SysIni)

- Program Folder
- Schedule
  - Must be scheduled under Controller Power-Up Handler
- Parent
  - Controller Power-Up Handler
- Instances
  - One instance per PAC required
- Provides management of the Global Handler ID number at PAC power-up
- Global Handler ID Assignment supports unique ID in multiprocessor applications.



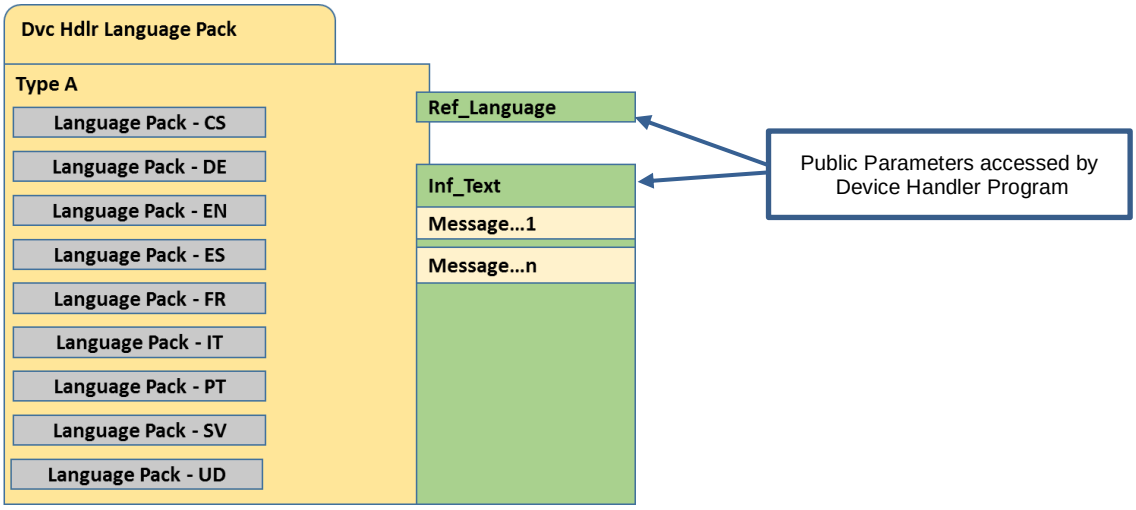
2.1.1.1 Public Program Parameters Tags

Tag Name	Usage	Data Type	Connection	Description
Ref_GlobalHandlerID	Public	DINT	Alias connection is made at the Device Handler instance	Global Device Handler ID
Cfg_GHIDAssignment	Public	DINT	No connections assigned	Global Device Handler ID Assignment Configuration 0 = System 1 = User
Cfg_GHIDBase	Public	DINT	No connections assigned	Global Device Handler ID Numbering Start GHID Assignment Type 1
Inf_MajorFaultRecord	Public	raC_UDT_ControllerFaultRecord	No connections assigned	MajorFaultRecord attribute or MinorFault Record of the PROGRAM object

2.1.2 Device Handler Language Pack (raM_Dvc_DHLP_Robot)

- Program Folder
- Schedule
  - Background operations: Language Pack only operates during handler language changes
- Parent
  - Logical Parent is user defined
- Instances
  - One instance per Device Handler Type.
- Provides language management for all handler event messages
- English is provided as the default language
- User Defined Language allows user to add their desired language
  - Language must be compatible with ASCII characters
- For more information please see Language Package Add-Ins Reference Manual

Note that currently only English is supported.

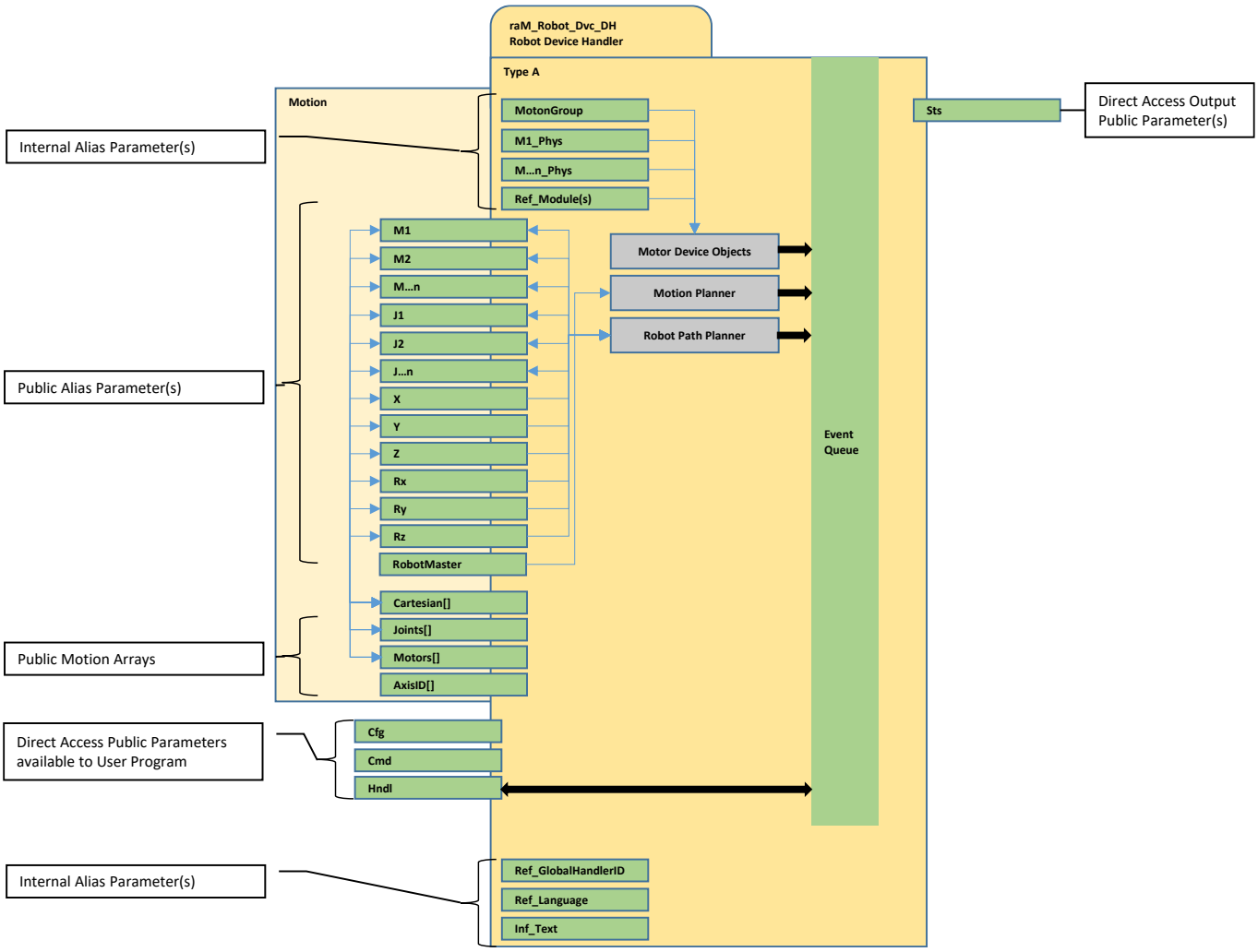


2.1.2.1 Public Program Parameters Tags

Tag Name	Usage	Data Type	Connection	Description
Ref_Language	Public	DINT	Alias connection is made at the Device Handler instance	Language in Use
Inf_Text	Public	raM_UDT_ Robot_Dvc_LP	Alias connection is made at the Device Handler instance	Events Message in the Selected Language • Handler State • Handler Status • CIP Axis Faults • CIP Axis State • Motion Status

2.1.3 Device Handler (raM_Robot_Dvc_DeviceHandler)

- Program Folder
- Schedule
  - Schedule all Robot Handlers at Motion Group Execution
- Parent
  - Logical Parent is Motion Group Execution Event
- Instances
  - One instance per robot
- Provides Device Management for robot and associated virtual axes and CIP axes



2.1.3.1 Local Program Tags "Alias For" System Initialization and Language Package Tags

Tag Name	Usage	Data Type	Connection	Description
Ref_GlobalHandlerID	Local	DINT	VIA: Alias TO: \[SystemInitializationProgramName] .Ref_GlobalHandlerID	Global Device Handler ID
Ref_Language	Local	DINT	VIA: Alias TO: \[LanguagePackProgramName]. Ref_Language	Language in Use
Inf_Text	Local	raM_UDT Robot_Dvc_LP	VIA: Alias TO: \[LanguagePackProgramName]. Inf_Text	Events Message in the Selected Language • Axis Faults • Axis State • Axis Status • Handler State • Handler Status
_MGrp	Local	MOTION_GROUP	Motion group tag	Motion group tag in the project

2.1.3.2 Public Program Parameters Tags available to user programs.

Tag Name	Usage	Data Type	Connection	Description
Cartesian[]	Public	raM_UDT_Robot_Dvc_Motion[6]	No connections assigned	Cartesian motion interface array
Cfg	Public	raM_UDT_Robot_Dvc_Configuration	No connections assigned	Handler Configuration
Cmd	Public	raM_UDT_Robot_Dvc_Command	No connections assigned	Handler Command
Cfg	Public	raM_UDT_Robot_Dvc_Configuration	No connections assigned	Handler Configuration
Hndl	Public	raM_UDT_Robot_Dvc_DataHndl	No connections assigned	Handler Interface
Info	Public	raR_UDT_Hndl_Info	No connections assigned	Handler Information
J1…Jn	Public	AXIS_VIRTUAL	Connections to joint path axes	Joint path axes
Joints[]	Public	raM_UDT_Robot_Dvc_Motion[6]	No connections assigned	Joint motion interface array
M1…Mn	Public	AXIS_VIRTUAL	Connections to motor path axes	Motor path axes
Motors[]	Public	raM_UDT_Robot_Dvc_Motion[n]	No connections assigned	Motor motion interface array
X, Y, Z, Rx, Ry, Rz	Public	AXIS_VIRTUAL	Connections to cartesian path axes	Cartesian path axes
Sts	Public	raM_UDT_Robot_Dvc_Status	No connections assigned	Handler Status

2.1.3.3 Controller scope Tags

Tag Name	Usage	Data Type	Connection
{MotionGroupName}	Controller scope	MOTION_GROUP	Motion Group Name (Controller Scope)
CIP Axes	Controller scope	AXIS_CIP_DRIVE	Physical Motor axes DFE axes
RobotMaster	Controller scope	AXIS_VIRTUAL	Master timing axis
Virtual joint axes	Controller scope	AXIS_VIRTUAL	Path Joint axes
Cartesian axes	Controller scope	AXIS_VIRTUAL	Path Cartesian axes
Virtual Motor axes	Controller scope	AXIS_VIRTUAL	Path Motor axes

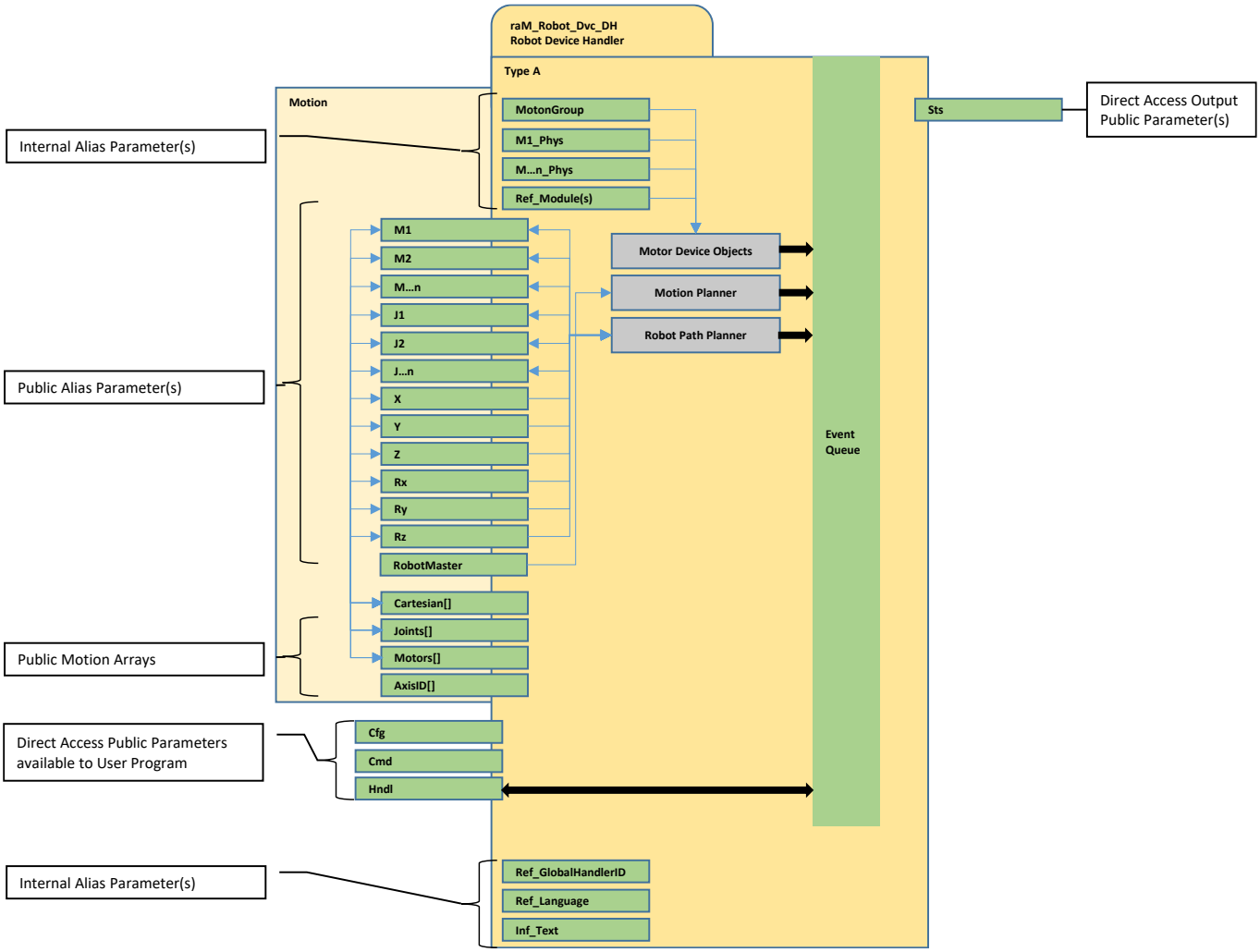
2.2 Connecting Device Handler Components

A Robot Handler must connect to the following:

- Other Device Handler components (Language Pack, System Initialization, etc.)
- Hardware Modules
- Motion Group
- CIP axes for the Motors
- Virtual path axes representing Motors, Joints, and Cartesian axes

Based on how data will be accessed and used by the programmer, connections between components are made in three ways:

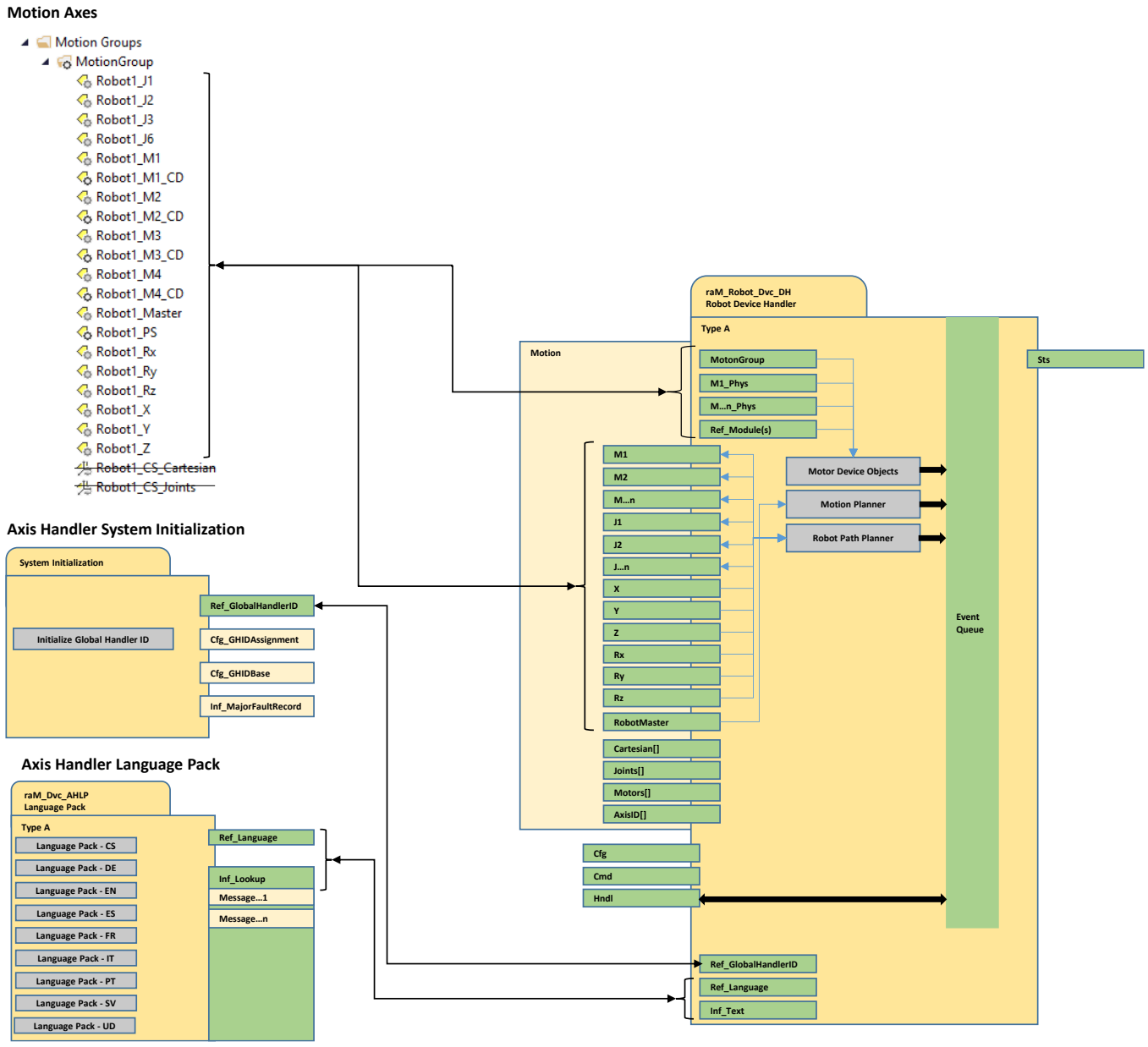
- Connected Parameters
- Direct Access Parameters
- Alias



Connection Type	Program Tag Usage	Used When
Connected Parameter	In/Out	A connection is made acting as an input/output to the Device Handler but access to the tag via direct access is not required
Direct Access Parameter	Public	Direct access to a tag in the program tag database is desired Syntax: \ProgramName.TagName Example: \DeviceHandler.Sts.Energized
Alias	Public	A connection is made acting as an input/output to the Device Handler and access to the tag via direct access is desired. While the programmer will access the data via the direct access parameter, the connection to the handler is made using an alias because the base tag can only exist as a controller scope tag. This is commonly used for motion axes and explicit message data types.
Alias	Local	A connection is made acting as an input/output to the Device Handler. The connection to the handler is made using an alias because the base tag can only exist as a controller scope tag.

2.2.1 Single Instance

Connection	Line Type
Parameter Connection	Dash
Direct Access Parameter (Not Shown)	Dot
Alias	Solid / Thick

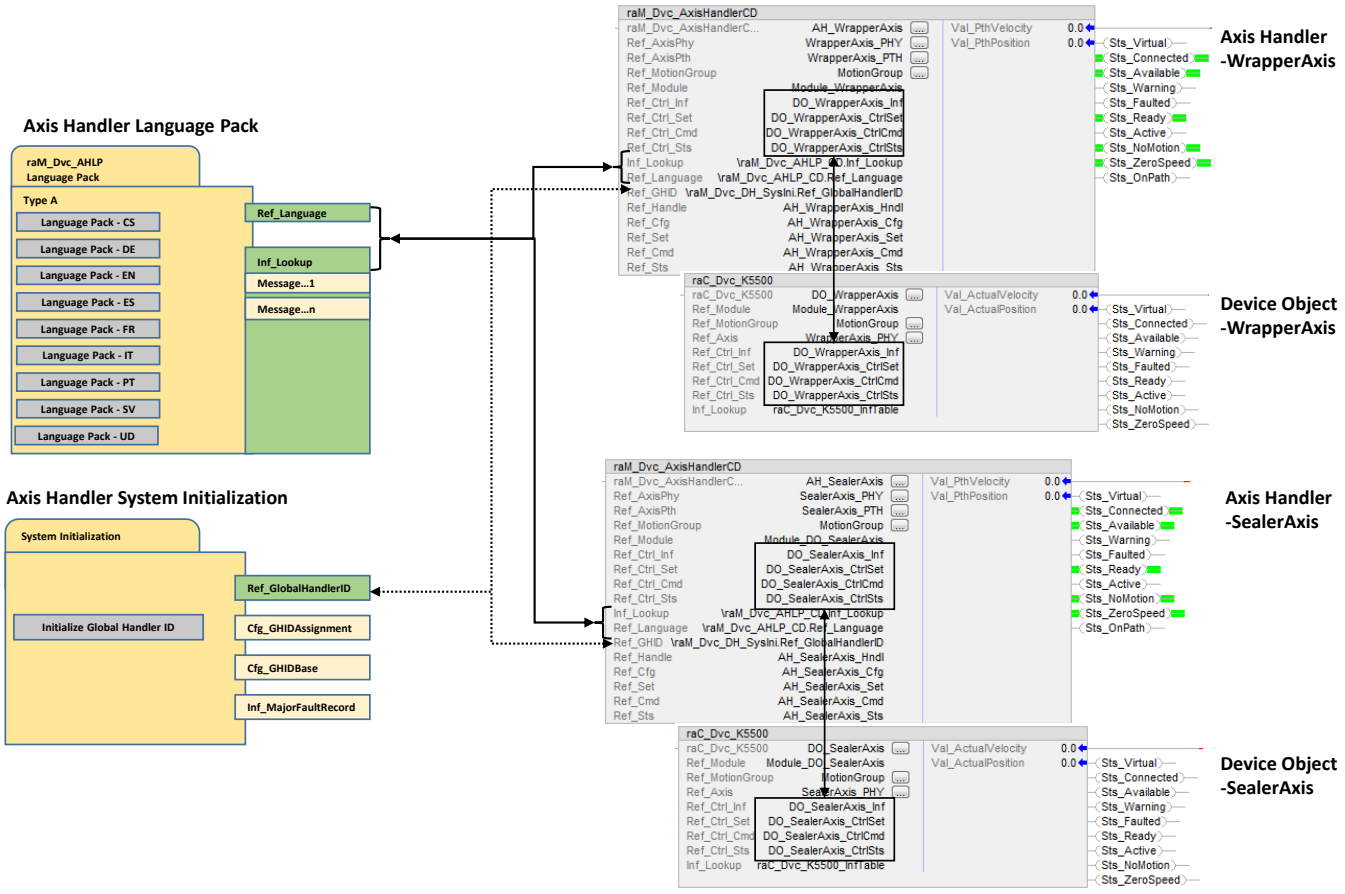


2.3 Connecting to the Application

A Robot Device Handler must also connect to the following:

- Methods designed for use with the Device Handler
- User Application Code

These connections are made using Direct Access Parameters.



3 Handler Operation

3.1 Device Handler Modes

The Device Handler and device operations are defined by the following modes:

Syntax: [Handler Operation]-[Device Operation]

- Physical
- Virtual

PAC Power-Up Defaults:

- Physical

Handler Operation:

- If available, commands other than mode commands are accepted through the \Device.Cmd data structure
- Operator commands accepted through the HMI application

Device Operation:

- Physical
    - Handler / Device configured to operate in a 'physical' capacity. Physical I/O modules must be connected and Motion Group synchronized with coordinate system definitions, Cartesian, Joint, and Motor motion axes for the desired geometry.
- Virtual
    - Handler / Device configured to 'virtualize' the physical device. I/O module connection is inhibited.
    - Support for Axis Virtualization requires the Handler to be used in conjunction with axis methods for any function that affects device state or operates directly on the device.
        - For example: Energize, DeEnergize, and Configuration, etc

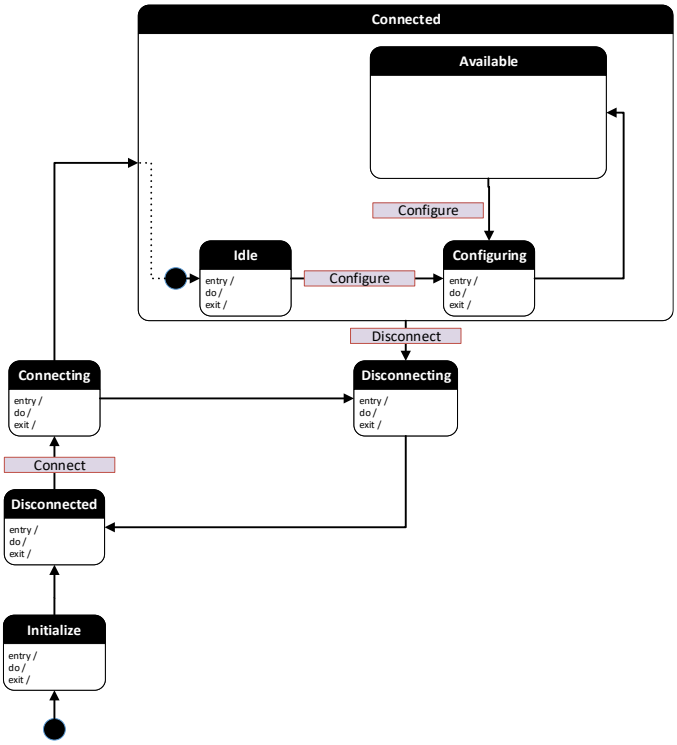
Switching Modes:

- Device operation can be selected through the [DeviceHandleProgram].Cmd data structure
- Requests for handler operation and device operation are made independently
- Mode changes can only be made when the Robot is not Energized

3.2 Device Handler States

The following table describes the states that define the Device Handler behavior:

Name	Value	Description
Initializing	1	Initial state at controller power up. ➔ Acquire Handler ID ➔ Clear Method Registry ➔ Configure Initial Module State ➔ Initialize Handler Data Interface
Disconnected	2	Device Handler is waiting for hardware to be connected ➔ No Action
Disconnecting	3	Device Handler is performing reset operations before checking for a new connection. ➔ No Action
Connecting	4	Device Handler is connecting to the managed device ➔ Verify all motor axes are connected and operational ➔ Verify that the motion group is synchronized
Connected	-	Macro state ➔ Macro state representing state 5 - 7
Idle	5	Device Handler is idle. ➔ No Action
Configuring	6	Device Handler is configuring. ➔ Device Handler verifies and configures all motion axes and coordinate systems for robot operations.
Available	7	Device Handler is running. ➔ Device Handler is configured and ready for commands.



4 User Interface

Reference the Robot_rOS User manual for more information regarding the predefined User Interface displays available.

5 Supported geometries

The Rockwell Automation Robotics Libraries currently support the following geometries:

- Articulated Independent (6 DoF)
- Articulated Dependent (4 DoF)
- SCARA (4 DoF)
- Delta (3 – 5 DoF)

For more information on supported geometries and deployment reference the following user manuals:

- raM_Robot_Opr_ConfigureArtIndependent
- raM_Robot_Opr_ConfigureScara
- raM_Robot_Opr_ConfigureArtDependent
- raM_Robot_Opr_ConfigureDelta

6 Load Protection

The load protection consists of continuously monitoring velocity, acceleration, and torque applied to all motors, transmissions, and joints, limiting the feedrate of the robot when any of these units are exceeded. By controlling the feedrate to protect the robot instead of limiting the speed, acceleration, and torque of each axis independently, the robot stays on path.

The Load Protection Add-On Instruction is applied to any robot geometry with up to six axes and can be found in the open _12_AlwaysOn routine in the Device Handler.

Motor specification, transmission specification, and joint limits from all joints of the robot are required for proper protection with this AOI. For robots supported natively by the RA Robotics Library, their values are loaded automatically. For generic robot or any robot not supported natively by the RA Robotics Library, these values are entered with the raM_Robot_Tec_ConfigureLoadProtection AOI (see documentation of this AOI for more details).

The load protection AOI protects motors, transmissions, and joints by monitoring seven variables from all joints of the robot:

- **Joint velocity**: lowest value between motor peak velocity, transmission input peak velocity, and joint peak velocity.
- **Joint acceleration**
- **Peak air-gap motor torque**
- **Peak shaft motor torque**: lowest value between joint peak torque referenced at motor shaft, and transmission peak output torque calculated at motor shaft.
- **RMS joint velocity**: lowest value between motor rated speed and transmission nominal input speed scaled to RMS.
- **RMS air-gap motor torque**: this is the motor continuous stall torque.
- **RMS joint torque**: this is the transmission output nominal torque scaled to RMS.

The robot protection against excessive velocity, acceleration, and torque is obtained by varying the robot feedrate as follows:

- *Velocity protection*: the feedrate is reduced as the speed of at least one axis is exceeded. Once the velocity of all axes is within limits, the feedrate returns to the value before the speed was exceeded.
- *Acceleration and torque protection*: the robot feedrate is reduced to a lower level as a function of by how much the acceleration or torque is exceeded.
- *RMS velocity and RMS torque*: the robot feedrate is reduced by up to 3% every 2 seconds while the RMS velocity or RMS torque is exceeded.

7 Programmer Interface

7.1 Command

*.Cmd	Function / Description	Data Type
\[ProgramName].Cmd.Reinitialize	Reinitialize the Device Handler and perform a complete reset of internal state	BOOL
\[ProgramName].Cmd.Physical	Physicalize Device Operation Request	BOOL
\[ProgramName].Cmd.Virtual	Virtualize Device Operation Request	BOOL

7.2 Configuration

*.Cfg	Function / Description	Data Type
\[ProgramName].Cfg.MethodError	Method Error Interpretation Enumerated 0 = Method Error generates a Warning Event 1 = Method Error generates a Fault Event	DINT
\[ProgramName].Cfg.bCFG	Configuration (Bit-Overlay)	DINT
\[ProgramName].Cfg.LogHandlerState	1=Enable logging of handler state events into the queue	BOOL
\[ProgramName].Cfg.LogMotionStatus	1=Enable logging of Motion status events into the queue	BOOL
\[ProgramName].Cfg.LogAxisState	1=Enable logging of physical axis state events into the queue	BOOL
\[ProgramName].Cfg.DisableLoadProtection	1=Disable load protection. Resets on transition to energized	BOOL
\[ProgramName].Cfg.ClearFramesOnInitialize	False=Retain frames when reinitializing True=Clear frames when reinitializing	BOOL

7.3 Status

*.Sts	Function / Description	Data Type
\[ProgramName].Sts.Physical	Device Operation – Physical operation	BOOL
\[ProgramName].Sts.Virtual	Device Operation – Virtual operation	BOOL
\[ProgramName].Sts.Initializing	Device Handler in state Initializing	BOOL
\[ProgramName].Sts.Disconnected	Handler to device connections missing	BOOL
\[ProgramName].Sts.Disconnecting	Handler is resetting for a new connection attempt	BOOL
\[ProgramName].Sts.Connecting	Handler is verifying connections to devices	BOOL
\[ProgramName].Sts.Connected	Handler to device connections made	BOOL
\[ProgramName].Sts.Idle	Device Handler is awaiting a configuration	BOOL
\[ProgramName].Sts.Configuring	User has made a configuration request which is being applied to robot axes	BOOL
\[ProgramName].Sts.Available	Devices and axes are connected and configured	BOOL
\[ProgramName].Sts.Ready	All hardware ready for commands	BOOL
\[ProgramName].Sts.SafetyEnabled	Safety circuit enabled by user	BOOL
\[ProgramName].Sts.PowerSupplyReady	DC bus power flow is enabled	BOOL
\[ProgramName].Sts.Energized	Drive power structure closed loop on Motor axes	BOOL
\[ProgramName].Sts.OnPath	Motor axes are synchronized with path axes	BOOL
\[ProgramName].Sts.TransformEnabled	Transformed space enabled, cartesian positions updating	BOOL
\[ProgramName].Sts.LoadProtectionEnabled	Load protection is configured and actively monitoring motion	BOOL

*.Sts	Function / Description	Data Type
\[ProgramName].Sts.Singularity	Robot is in a singularity, cartesian moves not allowed	BOOL
\[ProgramName].Sts.Stopping	Robot Path Master is decelerating to a stop	BOOL
\[ProgramName].Sts.Stopped	Robot Path Master is stopped - Planner is inactive	BOOL
\[ProgramName].Sts.Standstill	All joints are below standstill threshold	BOOL
\[ProgramName].Sts.PathPlannerActive	Robot path points are being processed and acted upon	BOOL
\[ProgramName].Sts.TrackingActive	Actively tracking external axes	BOOL
\[ProgramName].Sts.LoadProtectionActive	Load protection is actively reducing the robot feedrate	BOOL
\[ProgramName].Sts.MtdRegistryFull	Number of associated methods used exceeds the method queue size. Consequently, all methods will not be shown in HMI	BOOL
\[ProgramName].Sts.Faulted	Device Handler is in a faulted state	BOOL
\[ProgramName].Sts.SafetyError	Safety logic error	BOOL
\[ProgramName].Sts.Warning	Device Handler has a warning	BOOL
\[ProgramName].Sts.ERR	Error Code	DINT
\[ProgramName].Sts.EXERR	Extended error code	DINT
\[ProgramName].Sts.FaultMessage	Fault string	STRING
\[ProgramName].Sts.FirstFault	Device Handler First-Out Fault	raM_UDT_Opr_EventCreate_Members
\[ProgramName].Sts.FirstAlarm	Device Handler First Alarm/Warning	raM_UDT_Opr_EventCreate_Members
\[ProgramName].Sts.Pose	Flange position and orientation relative to robot frame (mm and deg)	raM_UDT_Robot_Opr_Frame
\[ProgramName].Sts.RobotConfiguration	Current robot configuration Bit1 – Righty(0) / Lefty(1) Bit2 – Below(0) / Above(1) Bit3 – No flip(0) / Flip	DINT
\[ProgramName].Sts.Joints[]	Current joint positions	REAL[6]
\[ProgramName].Sts.ActiveInterpolation	0 – PTP, 1 - CP-L, 2 - CP-W	DINT
\[ProgramName].Sts.ActiveType	2: Flange, 3: Tool	DINT
\[ProgramName].Sts.ActiveTypeID	Frame unique identifier (for tool frames)	DINT
\[ProgramName].Sts.ActiveRefFrameType	Reference frame type 0: World, 1: Robot, 2: Flange, 3: Tool, 4: User	DINT
\[ProgramName].Sts.ActiveRefFrameID	Reference frame unique identifier (for tool and user frames)	DINT
\[ProgramName].Sts.ActiveMoveID	Move ID of active path point	DINT
\[ProgramName].Sts.ActiveMovePerc	Percentage along active move	REAL
\[ProgramName].Sts.BlendingMoveID	Move ID of blending path point	DINT
\[ProgramName].Sts.BlendingMovePerc	Percentage along blended move	REAL
\[ProgramName].Sts.FeedrateSetpoint	Robot setpoint feedrate (%)	REAL
\[ProgramName].Sts.FeedrateActual	Robot active feedrate (%)	REAL
\[ProgramName].Sts.LinearVelocity	Linear speed of flange relative to robot frame (mm/s)	REAL
\[ProgramName].Sts.LinearAcceleration	Linear acceleration of flange relative to robot frame (mm/s^2)	REAL
\[ProgramName].Sts.AngularVelocity	Angular speed of flange relative to robot frame (deg/s)	REAL
\[ProgramName].Sts.AngularAcceleration	Angular acceleration of flange relative to robot frame (deg/s^2)	REAL

7.4 Motion interface

*.Cartesian[ ] *.Joints[ ] *.Motors[ ]	Function / Description	Data Type
MAM	Motion Axis Move Interface, see Motion Instruction documentation for details of executions	raM_UDT_Opr_Motion_MAM
MAS	Motion Axis Stop Interface, see Motion Instruction documentation for details of execution	raM_UDT_Opr_Motion_MAS
MRP	Motion Axis Redefine Position Interface, see Motion Instruction documentation for details of execution	raM_UDT_Opr_Motion_MRP
Status	Status Feedback Interface of each individual Axis	raM_UDT_Opr_Motion_Status
Enable	Enable axis management	BOOL
Configure	Initiate configuration of axis	BOOL
ClearFaults	Clear axis faults	BOOL
CoarseUpdatePeriod	Motion coarse update period	DINT
HomePosition	Axis home position	REAL
TargetID	Not used	DINT
AxisID	Instance ID of axis	DINT

*.Cartesian[ ].Status *.Joints[ ].Status *.Motors[ ].Status	Function / Description	Data Type
CommandPosition	Virtual axis command position feedback	REAL
CommandVelocity	Virtual axis command velocity feedback	REAL
CommandAcceleration	Virtual axis command acceleration feedback	REAL
ActualPosition	In Physical mode the Actual Position from the robot. In Virtual mode = Command.	REAL
ActualVelocity	In Physical mode the Actual Velocity from the Robot.. In Virtual mode = Command.	REAL
ActualAcceleration	Not used in Physical mode. In Virtual mode = Command.	REAL
AverageVelocity	Axis average velocity	REAL
PositionError	Motion position error	REAL
CurrentCommand	Current command feedback	REAL
AxisFault	Path axis fault	DINT
StoppingStatus	Set if there is a stopping process currently in progress. Cleared when the stopping process is complete.	BOOL
GearingStatus	Set if the axis is a slave that is currently Gearing to another axis. Cleared when the gearing operation is stopped or is superseded by some other motion operation.	BOOL
AccelStatus	The mover is accelerating	BOOL
DecelStatus	The mover is decelerating	BOOL
JogStatus	Set if a Jog motion profile is currently in progress. Cleared when the Jog is complete or is superseded by some other motion operation.	BOOL

*.Cartesian[ ].Status *.Joints[ ].Status *.Motors[ ].Status	Function / Description	Data Type
JogLockStatus	Set when the master axis satisfies the Lock Direction request of a Motion Axis Jog (MAJ) Instruction. If the Lock Direction is Immediate Forward Only or Immediate Reverse Only the JogLockStatus bit will be set immediately when the MAJ is initiated. If the Lock Direction is Position Forward Only or Position Reverse Only the bit will be set when the Master Axis crosses the Master Lock Position in the specified direction.	BOOL
MasterOffset	The position offset that is currently applied to the master side of the position cam. The Master Offset is returned in master position units. The Master Offset shows the same unwind characteristic as the position of a linear axis.	REAL
MasterOffsetMoveLockStatus	Set when the master axis satisfies the Lock Direction request of a Master Offset Move executed using MAM instruction. If the Lock Direction is Immediate Forward Only or Immediate Reverse Only the MasterOffsetMoveLockStatus bit will be set immediately when the MAM is initiated. If the Lock Direction is Position Forward Only or Position Reverse Only the bit will be set when the Master Axis crosses the Master Lock Position in the specified direction.	BOOL
MasterOffsetMoveStatus	The MasterOffsetMoveStatus bit attribute is set if a Master Offset Move motion profile is currently in progress. It is cleared when the Master Offset Move is complete or superseded by some other motion operation.	BOOL
MoveStatus	Set if a Move motion profile is currently in progress. Cleared when the Move is complete or is superseded by some other motion operation.	BOOL
MoveLockStatus	Set when the master axis satisfies the Lock Direction request of a Motion Axis Move (MAM) Instruction. If the Lock Direction is Immediate Forward Only or Immediate Reverse Only the MoveLockStatus bit will be set immediately when the MAM is initiated. If the Lock Direction is Position Forward Only or Position Reverse Only the bit will be set when the Master Axis crosses the Master Lock Position in the specified direction.	BOOL
PositionCamStatus	Set if a Position Cam motion profile is currently in progress. Cleared when the Position Cam is complete or is superseded by some other motion operation.	BOOL
PositionCamLockStatus	Set whenever the master axis satisfies the starting condition of a currently active Position Cam motion profile. The starting condition is established by the Start Control and Start Position parameters of the MAPC instruction. This bit is bit is cleared when the current position cam profile completes, or is superseded by some other motion operation. In uni-directional master direction mode, the Position Cam Lock Status bit is cleared when moving in the "wrong" direction and sets when moving in the correct direction.	BOOL

*.Cartesian[ ].Status *.Joints[ ].Status *.Motors[ ].Status	Function / Description	Data Type
PositionCamPendingStatus	Set if a Position Cam motion profile is currently pending the completion of a currently executing cam profile. This would be initiated by executing an MAPC instruction with Pending execution selected. This bit is cleared when the current position cam profile completes, initiating the start of the pending cam profile. This bit is also cleared if the position cam profile completes or is superseded by some other motion operation.	BOOL
TimeCamStatus	Set if a Time Cam motion profile is currently in progress. Cleared when the Time Cam is complete or is superseded by some other motion operation.	BOOL
TimeCamLockStatus	Set whenever the master axis satisfies the starting condition of a currently active Time Cam motion profile. The starting condition is established by the Start Control and Start Position parameters of the MATC instruction. This bit is bit is cleared when the current position cam profile completes or is superseded by some other motion operation. In uni-directional master direction mode, the Time Cam Lock Status bit is cleared when moving in the "wrong" direction and sets when moving in the correct direction.	BOOL
TimeCamPendingStatus	Set if a Time Cam motion profile is currently pending the completion of a currently executing cam profile. This would be initiated by executing an MATC instruction with Pending execution selected. This bit is cleared when the current time cam profile completes, initiating the start of the pending cam profile. This bit is also cleared if the time cam profile completes or is superseded by some other motion operation.	BOOL

*.Cartesian[ ].Status *.Joints[ ].Status *.Motors[ ].Status	Function / Description	Data Type
MotionStatus	Bitmapped collection of status conditions associated with the motion planner function. Bitmap: 0 = AccelStatus 1 = DecelStatus 2 = MoveStatus 3 = JogStatus 4 = GearingStatus 5 = HomingStatus 6 = StoppingStatus 7 = AxisHomedStatus 8 = PositionCamStatus 9 = TimeCamStatus 10 = PositionCamPendingStatus 11 = TimeCamPendingStatus 12 = GearingLockStatus 13 = PositionCamLockStatus 14 = TimeCamLockStatus 15 = MasterOffsetMoveStatus 16 = CoordinatedMotionStatus 17 = TransformStateStatus 18 = ControlledByTransformStatus 19 = DirectVelocityControlStatus 20 = DirectTorqueControlStatus 21 = MovePendingStatus 22 = MoveLockStatus 23 = JogPendingStatus 24 = JogLockStatus 25 = MasterOffsetMovePendingStatus 26 = MasterOffsetMoveLockStatus 27 = MaximumSpeedExceeded	SINT
PositionFeedback1	Actual position of the axis based on Feedback 1 (counts)	DINT
HomedStatusLast	Last Known motor AxisHomedStatus	BOOL
PositionLast	Last known motor positions	REAL
PositionCheckReq	Position check required	BOOL
MotionFunctionStatus	Not used	SINT
MotionFunctionCompleted	Not used	BOOL
AxisUnwind	Configured unwind	DINT
Enabled	Virtual axis is enabled	BOOL
ClearFualtsFailed	Clearing faults feedback bit	BOOL

7.5 AxisID interface

*.AxisID	Function / Description	Data Type
X	Index in Cartesian motion interface for cartesian X, -1 = not present	SINT
Y	Index in Cartesian motion interface for cartesian Y, -1 = not present	SINT
Z	Index in Cartesian motion interface for cartesian Z, -1 = not present	SINT
Rx	Index in Cartesian motion interface for cartesian Rx, -1 = not present	SINT
Ry	Index in Cartesian motion interface for cartesian Ry, -1 = not present	SINT
Rz	Index in Cartesian motion interface for cartesian Rz, -1 = not present	SINT
J1	Index in Joint motion interface for J1, -1 = not present	SINT
J2	Index in Joint motion interface for J2, -1 = not present	SINT
J3	Index in Joint motion interface for J3, -1 = not present	SINT
J4	Index in Joint motion interface for J4, -1 = not present	SINT
J5	Index in Joint motion interface for J5, -1 = not present	SINT
J6	Index in Joint motion interface for J6, -1 = not present	SINT

Note. The AxisID interface can be used to find which degrees of freedom (DoF) are available in the Device Handler configuration and which members in the \[ProgramName].Cartesian[] or \[ProgramName].Joints[] interfaces each DoF can be accessed. This can also be utilized in user code to abstract motor location. For example, Cartesian[\ProgramName.AxisID.X]

8 Events

Note: All Event ID's will be displayed with the Handler ID preceding the values listed.

Event ID = [Handler ID][Event Value]

Example:
Handler ID = 7
Event Value = 501
Event ID = 7501

8.1 Status

Device Handler – State	Event Message	Event Type	Event Value
*.EState = 1	Handler – Initializing	1 (Status)	100
*.EState = 2	Handler – Disconnected	1 (Status)	101
*.EState = 3	Handler – Disconnecting	1 (Status)	102
*.EState = 4	Handler – Connecting	1 (Status)	103
*.EState = 5	Handler - Idle	1 (Status)	104
*.EState = 6	Handler – Configuring	1 (Status)	105
*.EState = 7	Handler - Available	1 (Status)	106

8.2 Fault

The Robot Device Handler fault codes represent faults that can occur on the various components that interact with the Device Handler. The faults fall into the following categories:

1. Joint Path Virtual Axis
2. Motor CIP Axis
3. Motion Planner
4. Path Planner

The Fault Message, Error Code and Extended Error Code can be found in *.Sts.FaultMessage, *.Sts.ERR and *.Sts.EXERR respectively. Extended fault information including the type, category of fault and timestamp can be found in *.Sts.FirstFault and the fault log in *.FaultLog This fault information refers to the first fault recorded by the Device Handler.

Tag Name	Tag Member	Data Type	Description
Sts.FirstFault	Type	DINT	Event Type – 1 = Notification, 2 = Alarm, 3 = Fault
	ID	DINT	Device Handler ID
	Category	DINT	Category of fault. 100 = Axis Handler Motion Instruction/Home/Axis recovery fault 101 = Joint Axis, 102 = Motor CIP Axis, 103 = Motion Planner 104 = Path Planner
	Action	DINT	Error Code
	Value	DINT	Extra Error Code. Depending on Error code it will be Joint number, Motor number or array index of Path point Buffer.
	Message	STRING	Fault String
	EventTime_L	LINT	Timestamp
	EventTime_D	DINT[7]	Timestamp (Y, M, D, h, m, s, us)

Example:

*Sts.FirstFault.Type = 3 - Fault
*Sts.FirstFault.ID = 1 – Handler ID
*Sts.FirstFault.Category = 102 – Motor CIP Axis
*Sts.FirstFault.Action = 101 – Error code
*Sts.FirstFault.Value = 1 – MotorID faulted
*Sts.FirstFault.Message = 'Motor Overcurrent Fault'

NOTE:

1. Axis Handler faults will be Motion Instruction errors for the corresponding axis. Refer to motion-rm002 for information on motion error codes.
2. Virtual/CIP axis faults information can be found in motion-rm003

Consult the following table for Robot Device Handler motion planning and path planning error codes and descriptions.

Error Code	Extended Error Code	Event Message
198		Method Error. Check Fault Log for more information.
1001		Position error limit exceeded, see EXERR for Joint
1002		Brake command failed
1003		Invalid brake command, referenced axis not defined
1004		Error initiating path master
1005		Error stopping path master
1101		Severe motion planner execution error, contact support
1201	-	Blended move error, see extended error code
1202	-	Active move error, see extended error code
1201/1202	1	Forward Geometry: Joint limits exceeded
1201/1202	2	Forward Geometry: Folded robot joint limits error
1201/1202	3	Inverse Geometry: Invalid tool frame
1201/1202	4	Inverse Geometry: Invalid target frame
1201/1202	5	Inverse Geometry: start joints out of range
1201/1202	6	Inverse Geometry: Singularity
1201/1202	7	Inverse Geometry: Robot reach exceeded
1201/1202	8	Inverse Geometry: Joint limits exceeded for target
1201/1202	9	Inverse Geometry: Invalid configuration bits for Art Dependent, SCARA or Delta
1201/1202	10	Inverse Geometry: Invalid target, resulting rotational angles not supported (Rx/Ry)
1201/1202	11	Inverse Geometry: Invalid target, y coordinate must be zero for Delta2
1201/1202	12	Inverse Geometry: Invalid joint input for disabled joint, see EXERR for joint
1201/1202	13	Transform to tool error
1201/1202	14	Transform to flange error
1201/1202	15	Severe path planning error, save and contact RA support
1203	Faulted joint	Axis instruction error during execution, see EXERR for joint
1204	Faulted joint	Move exceeds joint limits, see EXERR for joint
1301		Error initiating tracking
1302		Error when stopping tracking
1303		Error while compensating for tracking
1304		Error transferring accumulated tracking
1305		Error while applying tracking offsets
1306		Joint position error during tracking
1307		Error synchronizing tracking with move

8.3 Alarm

The Robot Device Handler alarm codes represent the CIP axis alarms found in the CIP axis manual motion-rm003. The Robot Device Handler extended alarm information can be found in *.Sts.FirstAlarm and *Sts.EventLog array.

Tag Name	Tag Member	Data Type	Description
Sts.FirstAlarm Sts.EventLog	Type	DINT	Event Type – 1 = Notification, 2 = Alarm, 3 = Fault
	ID	DINT	Device Handler ID
	Category	DINT	Category of alarm/warning: 100 = Axis Handler Motion Instruction/Home/Axis recovery fault 101 = Joint Axis, 102 = Motor CIP Axis, 105 = Message Configuration
	Action	DINT	Error Code
	Value	DINT	Extra Error Code. Depending on Error code it will be Joint number, Motor number or array index of Path point Buffer.
	Message	STRING	Fault String
	EventTime_L	LINT	Timestamp
	EventTime_D	DINT[7]	Timestamp (Y, M, D, h, m, s, us)

Example:

*Sts.FirstAlarm.Type = 2 - Fault
*Sts.FirstAlarm.ID = 1 – Handler ID
*Sts.FirstAlarm.Category = 102 – Motor CIP Axis
*Sts.FirstAlarm.Action = 1008 – Alarm code
*Sts.FirstAlarm.Value = 1 – MotorID with the alarm
*Sts.FirstAlarm.Message = 'Motor Thermal Overload UL Alarm'

8.4 Method

When the event is a method execution, the Method ID will be the Event ID.
All Method ID's will be displayed with the Handler ID preceding the values listed.

Method ID = [Handler ID][Method Registry ID]

Each method registers itself against the handler with which it is associated. The sequence that methods register themselves against the handler will determine the method registry ID. If a method is the third one to register itself against the handler, it will acquire the number three as the method registry ID. Depending on how user code is scanned (optional JSRs especially in Structured Text), MethodID numbers can change from power cycle to power cycle.

Event ID = [MethodID]

Example:
Handler ID = 7
Method is the third one to register itself against handler.
Method Registry Location = 3
Method ID = 7003
Event ID = 7003

Each method is assigned a unique 'type' identifier based on function that is displayed in the Event Value field on method invocation or error. Users may use additional values about 100 for custom methods, if desired.

Method	Event ID	Event Value
raM_Robot_Opr_Clear	Method ID (*.Sts_MtdID)	01
raM_Robot_Opr _Energize	Method ID (*.Sts_MtdID)	02
raM_Robot_Opr_DeEnergize	Method ID (*.Sts_MtdID)	03
raM_Robot_Opr_Feedrate	Method ID (*.Sts_MtdID)	05
raM_Robot_Opr_CancelPath	Method ID (*.Sts_MtdID)	06
raM_Robot_Opr_StopOnPath	Method ID (*.Sts_MtdID)	07
raM_Robot_Opr_LoadPath	Method ID (*.Sts_MtdID)	08
raM_Robot_Opr_BrakeControl	Method ID (*.Sts_MtdID)	09
raM_Robot_Opr_Jog	Method ID (*.Sts_MtdID)	10
raM_Robot_Opr_AssignHome	Method ID (*.Sts_MtdID)	11
raM_Robot_Opr_ConfigureArtDependent	Method ID (*.Sts_MtdID)	12
raM_Robot_Opr_ConfigureArtIndependent	Method ID (*.Sts_MtdID)	13
raM_Robot_Opr_ConfigureDelta	Method ID (*.Sts_MtdID)	14
raM_Robot_Opr_ConfigureScara	Method ID (*.Sts_MtdID)	15
raM_Robot_Opr_ConfigureFrame	Method ID (*.Sts_MtdID)	16
raM_Robot_Opr_ConfigureLoadProtection	Method ID (*.Sts_MtdID)	17
raM_Robot_Opr_ConfigureTracking	Method ID (*.Sts_MtdID)	18
raM_Robot_Opr_MoveJoint	Method ID (*.Sts_MtdID)	19
raM_Robot_Opr_MoveLinear	Method ID (*.Sts_MtdID)	20
raM_Robot_Opr_MovePTP	Method ID (*.Sts_MtdID)	21
raM_Robot_Opr_MoveApproach	Method ID (*.Sts_MtdID)	22
raM_Robot_Opr_MoveDepart	Method ID (*.Sts_MtdID)	23
raM_Robot_Opr_SetConfiguration	Method ID (*.Sts_MtdID)	24
raM_Robot_Opr_SetDynamics	Method ID (*.Sts_MtdID)	25
raM_Robot_Opr_SetFrame	Method ID (*.Sts_MtdID)	26

Method	Event ID	Event Value
raM_Robot_Opr_SetTurnCount	Method ID (*.Sts_MtdID)	27
raM_Robot_Opr_TeachToolFrame	Method ID (*.Sts_MtdID)	28
raM_Robot_Opr_TeachUserFrame	Method ID (*.Sts_MtdID)	29
raM_Robot_Opr_Trigger	Method ID (*.Sts_MtdID)	30
raM_Robot_Tec_CalculatePose	Method ID (*.Sts_MtdID)	31
raM_Robot_Tec_ZoneBox	Method ID (*.Sts_MtdID)	32
raM_Robot_Tec_ZoneCylinder	Method ID (*.Sts_MtdID)	33
raM_Robot_Tec_ZoneSphere	Method ID (*.Sts_MtdID)	34

9 Application Code Manager

9.1 Implementation Object: raM_Robot_Dvc_DeviceHandler

Implementation Language: Various
Content Type: Program

This implement contains an entire program with an instance of the raM_Robot_Dvc_DeviceHandler object

Parameter Name	Default Value	Instance Name	Definition	Description
None				

Linked Library

Link Name	Catalog Number	Revision	Solution	Category
RobotCatalog			Robot Catalogs	
raM_Dvc_AxisHandler_CD	raM_Dvc_AxisHandler_CD	2	(RA-LIB) Machine	Asset-Control
raM_Opr_Home_CD	raM_Opr_Home_CD	2	(RA-LIB) Machine	Asset-Control
raM_Opr_SyncPthPhyAx_CD	raM_Opr_SyncPthPhyAx_CD	2	(RA-LIB) Machine	Asset-Control
raM_Robot_Opr_CancelPath	raM_Robot_Opr_CancelPath	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_StopOnPath	raM_Robot_Opr_StopOnPath	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_LoadPath	raM_Robot_Opr_LoadPath	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_Feedrate	raM_Robot_Opr_Feedrate	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_Configure	raM_Robot_Opr_Configure	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_Clear	raM_Robot_Opr_Clear	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_Energize	raM_Robot_Opr_Energize	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_DeEnergize	raM_Robot_Opr_DeEnergize	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_AssignHome	raM_Robot_Opr_AssignHome	2	(RA-LIB) Robotics	Asset-Control
raC_Dvc_K5700	raC_Dvc_K5700	>=3.2	(RA-LIB) Device	Asset-Control
raM_Robot_Opr_BrakeControl	raM_Robot_Opr_BrakeControl	2	(RA-LIB) Robotics	Asset-Control
raM_LD_DH_SysIni	raM_LD_DH_SysIni	2	(RA-LIB) Machine	Asset-Control
raM_RobotDvc_DeviceStatus	raM_RobotDvc_DeviceStatus	2	(RA-LIB) Robotics	Asset-Control
raM_Robot_Opr_Configure	raM_Robot_Opr_Configure	2	(RA-LIB) Robotics	Asset-Control

9.2 Attachments

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

A Robot Catalog object is required for proper configuratoin. This object contains all of the geometry information and supplementary axes, coordinate systems, and (optionally) hardware associated with a particular robot. At this time, there are a number of Comau specific catalogs and a series of generic geometries that are included in the library. The Comau specific catalogs will completely define all aspects of the robot geometry. The generic catalogs will only provide basic template objects which the user will need to configure after program generation.

10 Application

10.1 Connecting

10.1.1 Device Handler Objects

Set Connections in the Device Handler. (This can be done in the Program Properties window)

Device Handler Tag	Connection Type	Connect To
Ref_GlobalHandlerID	Alias	\raM_Dvc_DH_SysIni.Ref_GlobalHandlerID
Ref_Language	Alias	\raM_Robot_Dvc_DHLP.Ref_Language
Inf_Text	Alias	\raM_Robot_Dvc_DHLP.Inf_Text
X	Alias	{RobotInstanceName}_X axis
Y	Alias	{RobotInstanceName}_Y axis
Z	Alias	{RobotInstanceName}_Z axis
Rx	Alias	{RobotInstanceName}_Rx axis
Ry	Alias	{RobotInstanceName}_Ry axis
Rz	Alias	{RobotInstanceName}_Rz axis
J1	Alias	{RobotInstanceName}_J1 axis
J2	Alias	{RobotInstanceName}_J2 axis
J…n	Alias	{RobotInstanceName}_J…n axis
M1	Alias	{RobotInstanceName}_M1 axis
M2	Alias	{RobotInstanceName}_M2 axis
M..n	Alias	{RobotInstanceName}_M…n axis
M1_Phys	Alias	{RobotInstanceName}_M1_CD CIP Axis
M2_Phys	Alias	{RobotInstanceName}_M2_CD CIP Axis
M…n_Phys	Alias	{RobotInstanceName}_M…n_CD CIP Axis
_MGrp	Alias	{MotionGroupName} Motion Group

10.1.2 Device Handler Internal connections

The internal connections in the Device Handler consist of links to the Virtual Axes, CIP axes, and axis modules.

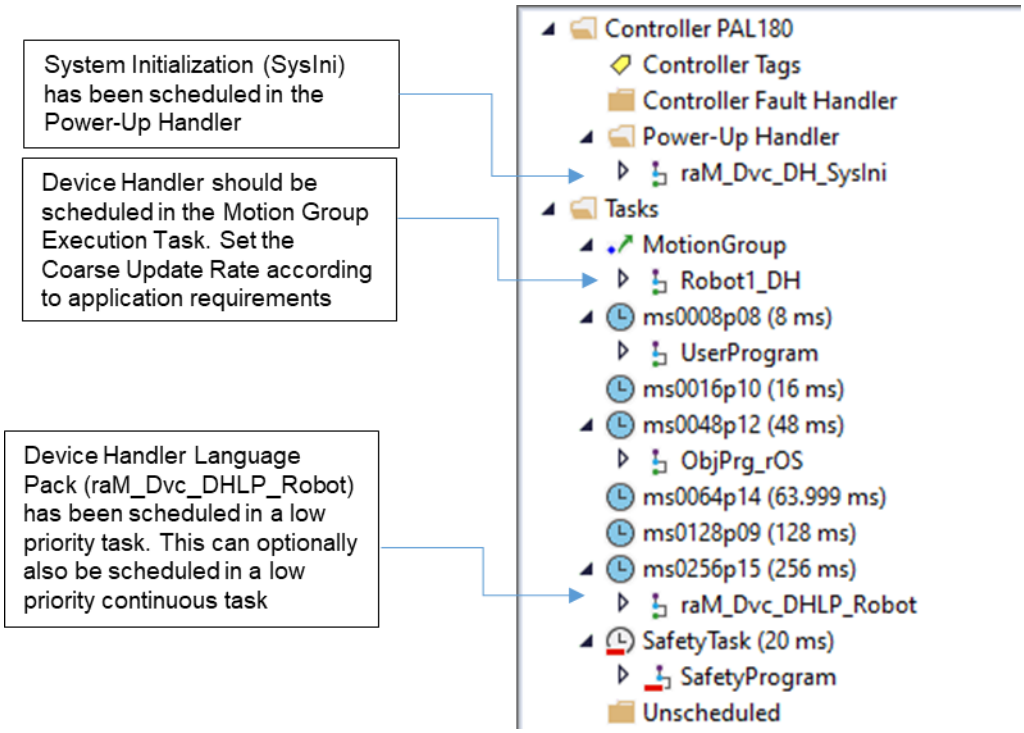
Note: To run the Device Handler with the user configuration and hardware set up, the Device Handler must be scaled accordingly. The Motor instruction array must be managed and have at least enough members as the system's number of motors. These instances must then be linked to the equivalent motion axes. This is done in the _10_MotionCalls routine in the Device Handler, and in the _00_HardwareManagement.

When generating the project through Application Code Manager, all connections are automatically established.

10.2 Scheduling

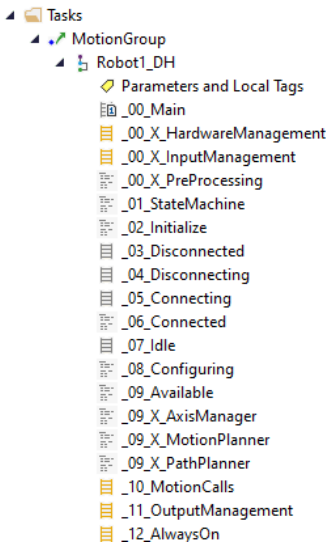
10.2.1 Device Handler Objects

Ensure Device Handler objects have been scheduled for execution either during the import process or manually after the import process.



The Robot Device Handler includes:

- State management of the Robot
- All robot axis management (virtual, physical operation, motion wrapper for arrayed based programming of axes, etc)
- Motion planner and feed rate management through master axis
- Path planner supporting up to 6 Degree of Freedom



10.2.1.1 Hardware Management

The _00_X_HardwareManagement routine is not read or write protected. This routine is used for:

- Management of all Device Objects related to motor axes
- Device Objects “at a glance” output status of each motor
- Manually adding or removing hardware, NOTE. A strict procedure must be followed (array sizes, name conventions, hardware tag connections), therefore manual changes are not recommended. Using ACM to regenerate a new configuration is the recommended workflow.

10.2.1.2 Input Management

The _00_X_InputManagement routine is write protected but not read protected. This routine is used for:

- Status of all inputs and conditions to the Device Handler.
- Actual conditions for mode changes
- Hardware connection conditions and status

10.2.1.3 MotionCalls

The _10_MotionCalls routine is not read or write protected. This routine is used for:

- All motion array management
- Connecting and managing Cartesian DoF
- Connecting and managing Joint DoF
- Connecting and managing Motor axes
- Linking Motor and Joint axes
- Linking Cartesian and Joint axes
- Manually adding or removing hardware and DoF, NOTE. A strict procedure must be followed (array sizes, name conventions, hardware tag connections, coordinate system configuration) therefore manual changes are not recommended. Using ACM to regenerate a new configuration is the recommended workflow.

10.2.1.4 Always On

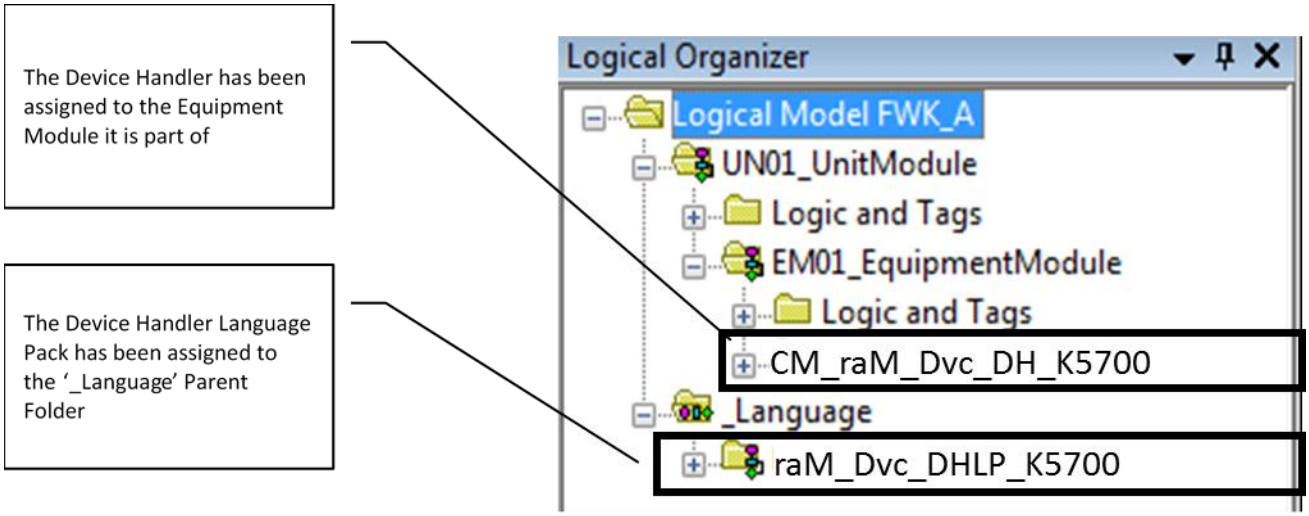
The _12_AlwaysOn routine is not read or write protected. This routine is used for:

- User processing required related to the Robot Device Handler scheduled at the end of execution

10.3 Parenting (optional)

10.3.1 Device Handler Objects

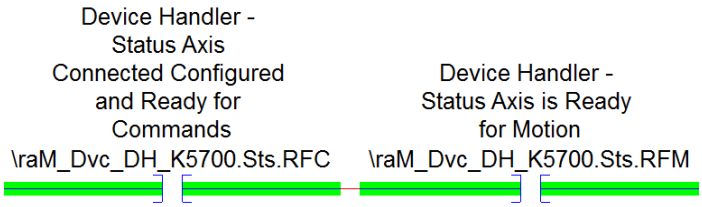
Assign objects to parent folders in the logical organizer. As this user follows an ISA-88 equipment model, the Device Handler has been renamed to indicate it is a control module and placed logically inside the equipment module.



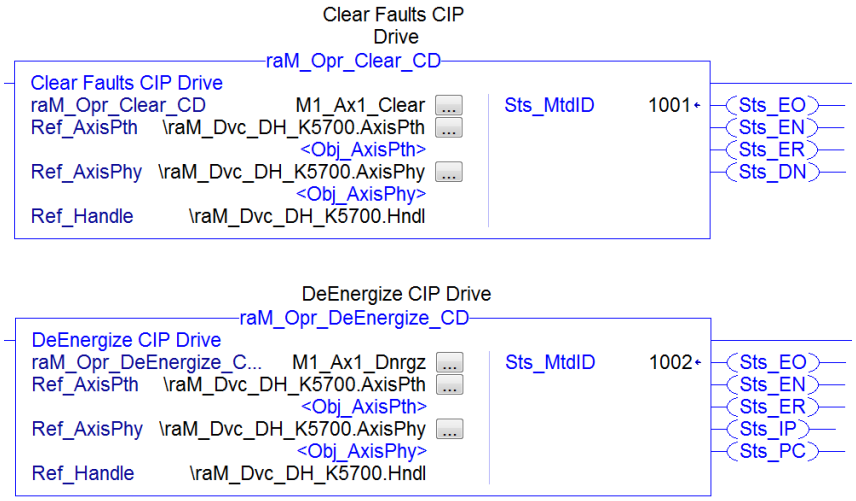
10.4 Interfacing from Application Code

When interfacing with the handler from the user application code, handler commands and status can be accessed using direct access parameters.

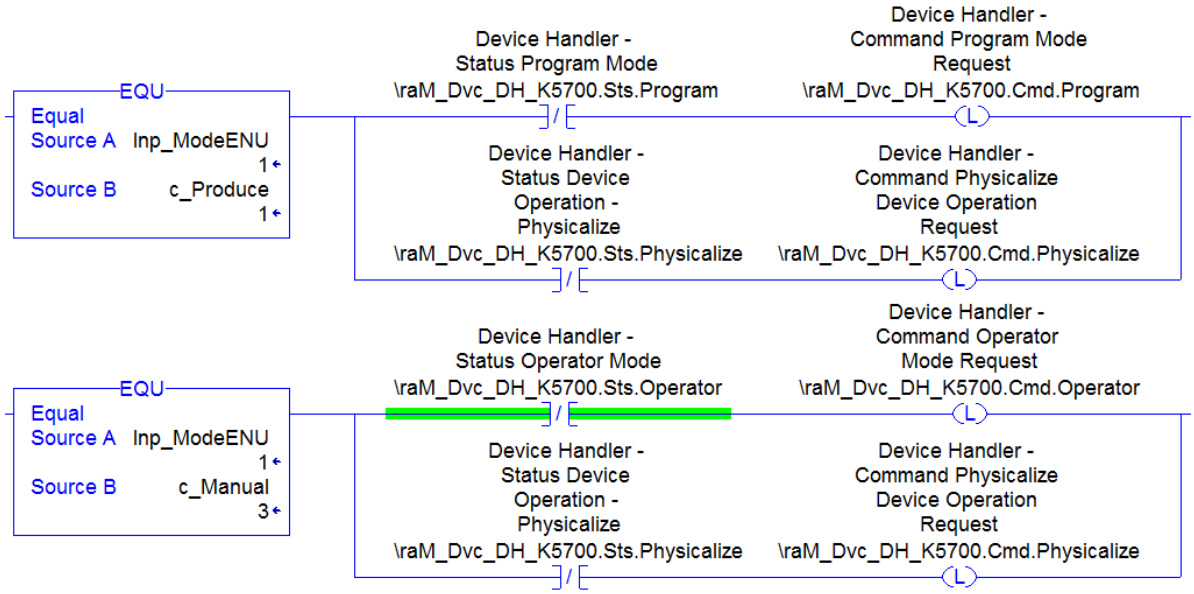
Example: Boolean evaluation of handler status



Example: Method linked to the data handle



Example: Setting Handler Operation Modes based on Machine Mode



10.5 Method Error Configuration

This section illustrates the behavior of the handler when the configuration for the method error is changed from Warning Event to Fault Event.

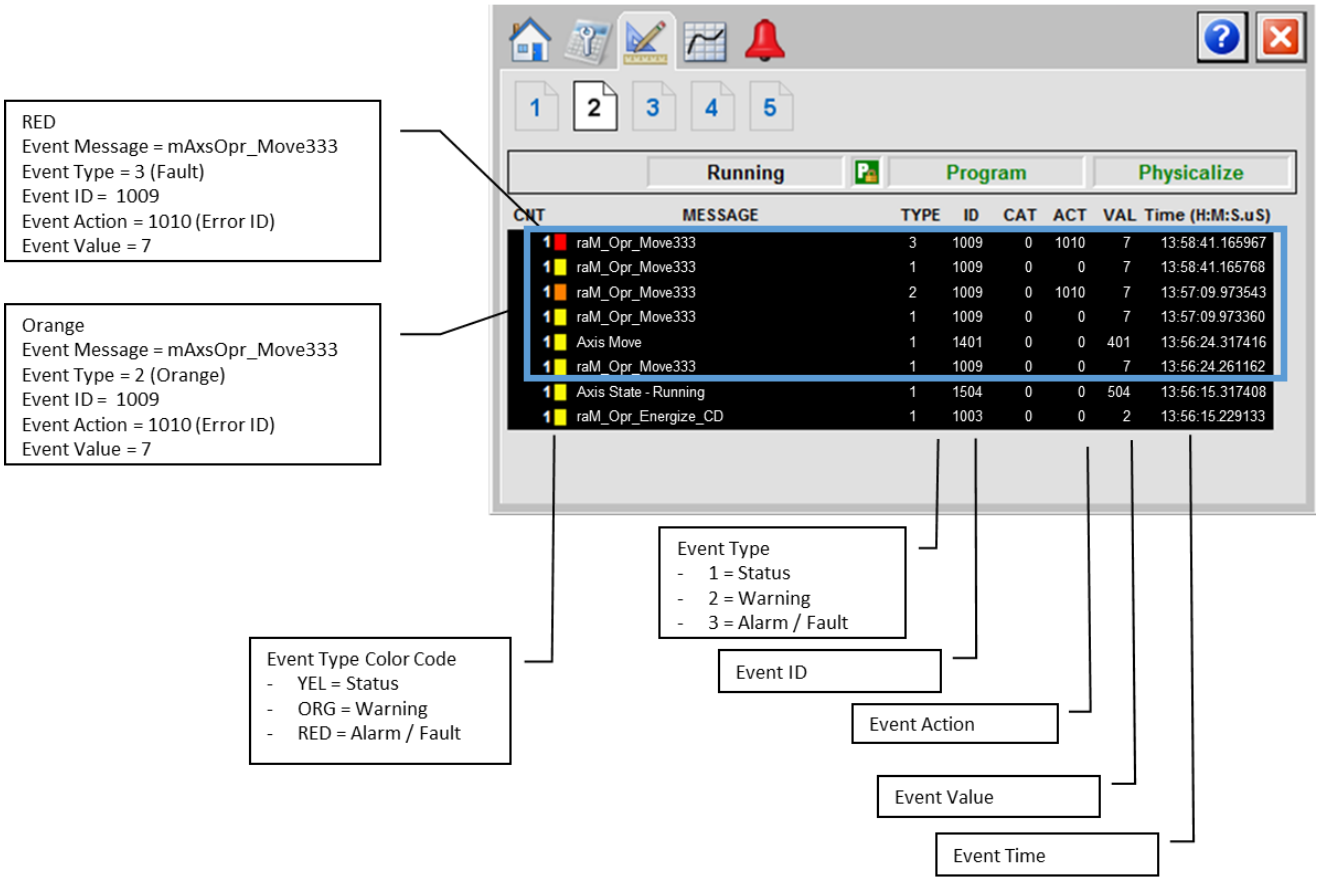
The method error configuration can be changed programmatically any time. User can set the standard desired behavior (warning or fault) for method error and treat individually specific methods error that have a desired behavior different from the standard.

*.Cfg	Function / Description
\[ProgramName].Cfg.MethodError	Method Error Interpretation Enumerated 0 = Warning Event 1 = Fault Event

In this example the method raM_Opr_Move333 (method value assignment = 7) is considered.
An invalid parameter was entered in the method configuration to cause the Error 1010 when instruction is executed.

Error 1010 - Cfg_MoveType is not valid

Event Message (Method)	Event Type	Event ID	Event Action	Event Value
raM_Opr_Move333	3 (Fault)	1009	1010 (Error ID)	7
raM_Opr_Move333	1 (Status)	1009	0	7
raM_Opr_Move333	2 (Warning)	1009	1010 (Error ID)	7



11 Appendix

11.1 General

This document provides a programmer with details on this OEM Building Block instruction for a Logix-based controller. You should already be familiar with how the Logix-based controller stores and processes data.

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

IMPORTANT This OEM Building Block Instruction includes an Add-On Instruction for use with Version 24 or later of Studio 5000 Logix Designer.

11.2 Common Information for All Instructions

Rockwell Automation Building Blocks contain many common attributes or objects. Refer to the following reference materials for more information:

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

11.3 Conventions and Related Terms

Data - Set and Clear

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

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

Signal Processing - Edge and Level

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

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

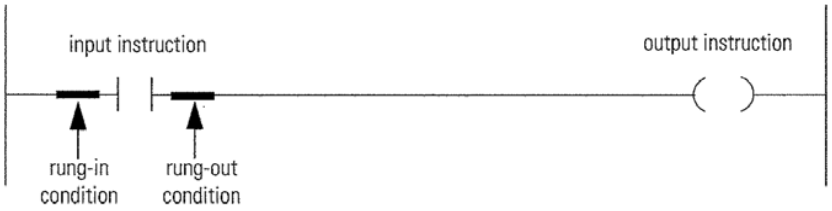
Instruction Execution - Edge and Continuous

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

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

Relay Ladder Rung Condition

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



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

IMPORTANT

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

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

Pre-scan

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

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

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

Pre-scan does not occur for a program when:

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

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