Rockwell Automation Application Content

Rockwell Automation Robotics Libraries

Reference Manual

Load Path - Robot

raM_Robot_Opr_LoadPath v2.0

December, 2023

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

Table of Contents **3**

1 Overview **4**

1.1 Prerequisites *4*

1.2 Functional Description *5*

1.3 Path planner *7*

1.4 Motion planner *14*

1.5 Execution *17*

2 Instruction **20**

2.1 Input Data *20*

2.2 Output Data *27*

2.3 Error Codes *27*

3 Application Code Manager **29**

3.1 Definition Object: raM_Robot_Opr_LoadPath *29*

3.2 Implementation Object: raM_LD_Robot_LoadPath *29*

3.3 Attachments *29*

4 Application **30**

4.1 Using raM_Robot_Opr_LoadPath *30*

5 Appendix **31**

General *31*

Common Information for All Instructions *31*

Conventions and Related Terms *31*

1 Overview

raM_Robot_Opr_LoadPath:

This instruction loads a single path point or sequence of path points to the Device Handler motion planner.

Use when:

- Using a Device Handler for Robot Management
- Programmatically creating move sequences for a robot
- Loading pre-defined path points from an array of Cartesian poses or joint positions

Do NOT use when:

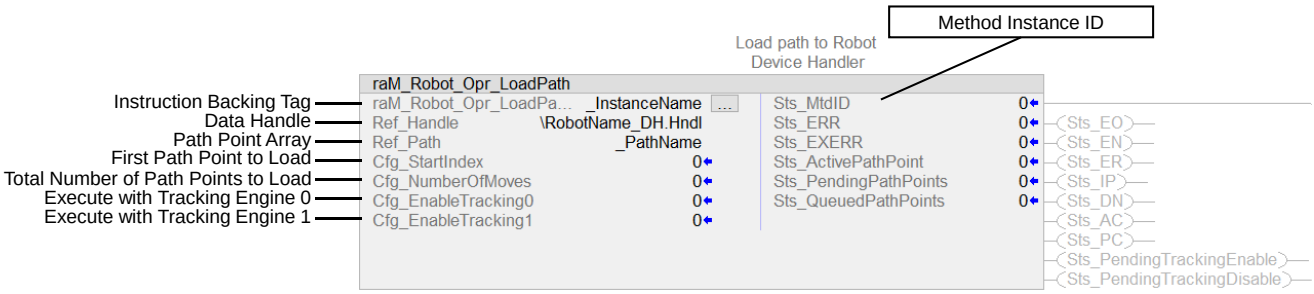
- Not using a Device Handler for Robot Management

1.1 Prerequisites

- Device Handler for Robot
  - Rockwell Automation Robotic Libraries v2.0 →
- Studio 5000 – Logix Designer
  - v35.0 →
- Studio 5000 – Application Code Manager
  - v4.03.00 →

1.2 Functional Description

The LoadPath object is an instruction that enables path planning by loading new path points to the Robot Device Handler's path planner. The Device Handler then executes robot move sequences from an array of path points provided by the LoadPath instruction.



The LoadPath instruction can be queued and if there are enough available buffer points in the Device Handler (5) the instruction will load all the selected path points in one scan. As new buffer points become available, remaining pending path points will be loaded into the executing buffer. Prior to successfully loading a path, the Device Handler must be available, not faulted, and the robot must be energized.

A path is a collection of path points stored in the tag Ref_Path. Each path point (e.g. Ref_Path[0..n]) is represented by the UDT raM_UDT_Robot_Opr_PathPoint, whose fields are further detailed in Section 2.1.

The inputs Cfg_StartIndex and Cfg_NumberOfMoves specify the subset of the path point array to load, using zero-based indexing. When the instruction is called with Cfg_EnableTracking0 or Cfg_EnableTracking1, the path array will not start executing until the respective tracking objects is enabled. Therefor the tracking objects must be configured for operation and master axes linked before path planning is executed with tracking enabled. If the tracking objects are configured for automatic execution, the tracking objects will automatically be switched on when the last path point loaded without tracking is executing.

If on the other hand the instructions is executed without Cfg_EnableTrackingX enabled, the path points will not execute until the tracking objects have stopped. If the tracking objects are configured for automatic execution, the tracking objects will automatically be switched off when the last path point loaded with tracking is executing.

The execution of path points can be monitored on the instruction outputs, e.g., the currently executing point, how many points are remaining, and when the move is complete. For more information, see raM_Opr_ConfigureTracking.

In the raM_UDT_Robot_Opr_PathPoint UDT, the fields TargetPointType and InterpolationType are decoupled. This means a Cartesian target can be reached independent of interpolation type, Point-to-Point (PTP) or Cartesian Linear (CP-L) moves, with the same rule applicable for joint targets. However, if a Cartesian target is specified for a PTP move, it will be first converted to a joint target, as PTP moves are always calculated in joint space. In a similar fashion, if a joint target is specified for a CP-L move, it will be first converted to a Cartesian target, since CP-L moves are always calculated in Cartesian space.

Any referenced user frames or tool frames must have been configured prior to executing the Load path instruction and will not be permitted to change during execution.

General Status Bit Behavior:

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

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

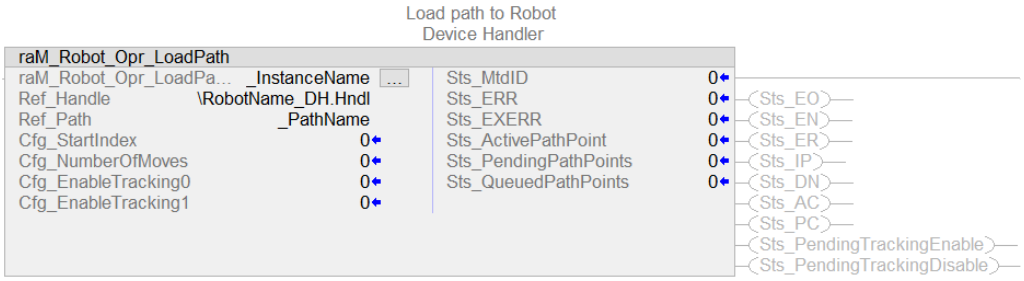
1.3 Path planner

This section refers to the following concepts, defined for all supported robot types in the Rockwell Automation Publication MOTION-UM002-EN-P:

- coordinates of the joints
- robot frame used as a reference frame for the Coordinates in Cartesian space
- coordinates in Cartesian space of the flange of the robot with respect to the robot frame (X,Y,Z,Rx,Ry,Rz)
- the configuration of the robot as a set of flags:
  - Lefty (1) / Righty (0)
  - Above (1) / Below (0)
  - Flip (1) / No Flip (0)

The trajectory performed by the robot must be provided by the user by means of an ordered sequence of movement objects, each defining a target position. Such a sequence is referred to as a path.
When moving along a path, the robot will then move continuously from one target position to the next starting with the first movement of the path till to the last movement.
Besides its target coordinates, each movement object contains a set of properties that the user must input to define the trajectory with which the robot will move to the correspondent target point from the previous one. This section describes these properties which must be set to define the geometry of the trajectory and its kinematics.

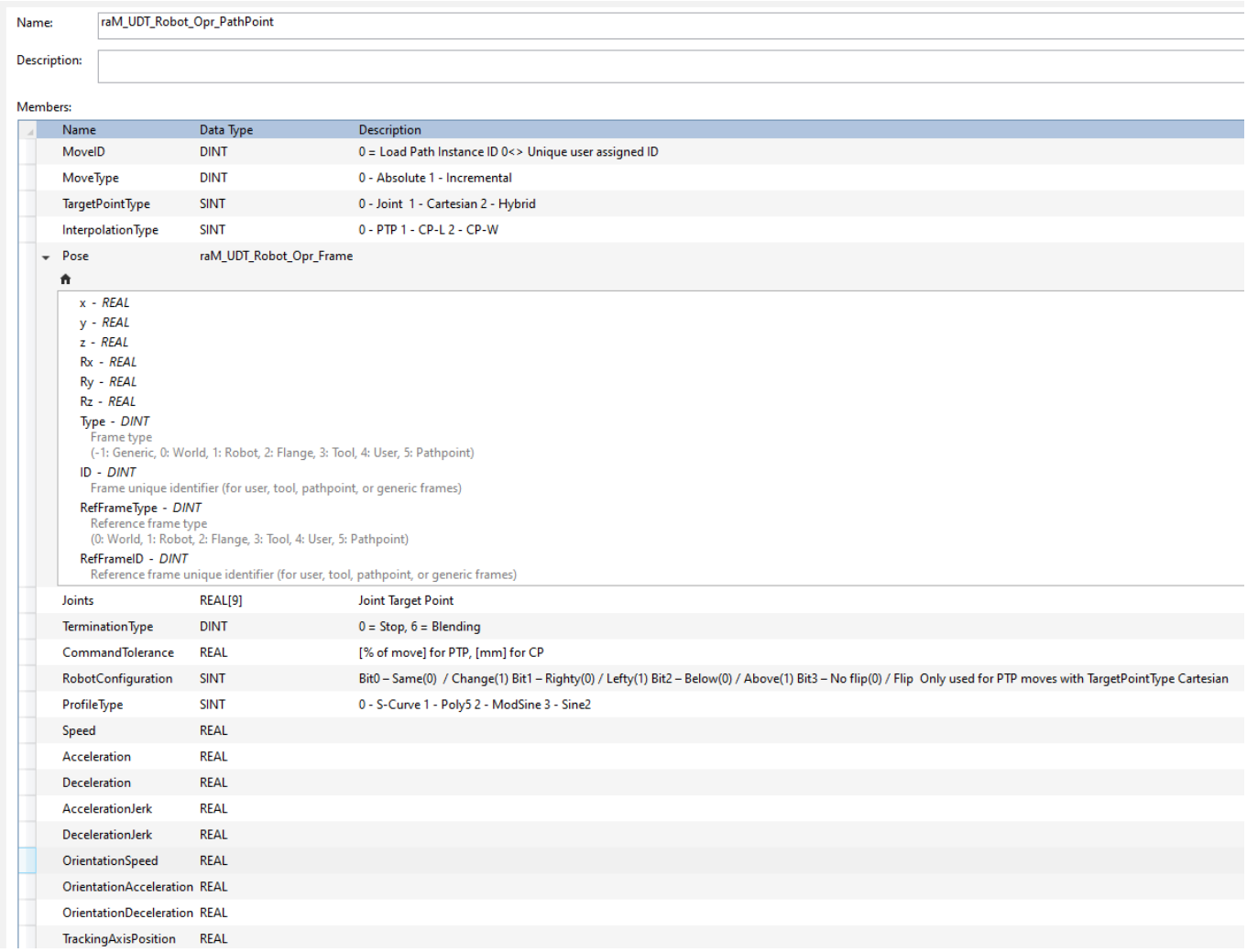
The user loads a path (sequence of target points) into the planner using the raM_Robot_Opr_LoadPath add-on instruction which will actuate the robot:



The Ref_Path parameter of the add-on instruction is an array of members containing movement data, the path is defined by a section of the array selected using the start index, Cfg_StartIndex. And the number of moves, Cfg_NumberOfMoves. In other words, the path is defined by all the movements in the section starting at Ref_Path[Cfg_StartIndex] and ending at Ref_Path[Cfg_StartIndex + Cfg_NumberOfMovements-1].
If Cfg_NumberOfMovements=1, the path is composed of a single movement to the target position defined by Ref_Path[Cfg_StartIndex].

The raM_Robot_Opr_LoadPath add-on instruction can be called even whilst the robot is moving using multiple calls or instances of the AOI. The subsequent loaded movements will be queued seamlessly to the ones previously loaded. This allows the user to create chains or several groups of movements, each corresponding to a call of the AOI executed based on external conditions available at a different point in time. The resulting movement will execute as if all movements were loaded as a single call of the AOI. This workflow allows users to code the logic which makes the robot move depending on external conditions.

The figure below defines the data structure (raM_UDT_Robot_Opr_PathPoint) of elements of Ref_Path, and defines a location and movement definition:



Name: raM_UDT_Robot_Opr_PathPoint

Description:

Members:

Name	Data Type	Description
MoveID	DINT	0 = Load Path Instance ID 0<> Unique user assigned ID
MoveType	DINT	0 - Absolute 1 - Incremental
TargetPointType	SINT	0 - Joint 1 - Cartesian 2 - Hybrid
InterpolationType	SINT	0 - PTP 1 - CP-L 2 - CP-W
Pose	raM_UDT_Robot_Opr_Frame	
Joints	REAL[9]	Joint Target Point
TerminationType	DINT	0 = Stop, 6 = Blending
CommandTolerance	REAL	[% of move] for PTP, [mm] for CP
RobotConfiguration	SINT	Bit0 – Same(0) / Change(1) Bit1 – Righty(0) / Lefty(1) Bit2 – Below(0) / Above(1) Bit3 – No flip(0) / Flip Only used for PTP moves with TargetPointType Cartesian
ProfileType	SINT	0 - S-Curve 1 - Poly5 2 - ModSine 3 - Sine2
Speed	REAL	
Acceleration	REAL	
Deceleration	REAL	
AccelerationJerk	REAL	
DecelerationJerk	REAL	
OrientationSpeed	REAL	
OrientationAcceleration	REAL	
OrientationDeceleration	REAL	
TrackingAxisPosition	REAL	

Pose members:

- x - *REAL*
- y - *REAL*
- z - *REAL*
- Rx - *REAL*
- Ry - *REAL*
- Rz - *REAL*
- Type - *DINT*
  Frame type
  (-1: Generic, 0: World, 1: Robot, 2: Flange, 3: Tool, 4: User, 5: Pathpoint)
- ID - *DINT*
  Frame unique identifier (for user, tool, pathpoint, or generic frames)
- RefFrameType - *DINT*
  Reference frame type
  (0: World, 1: Robot, 2: Flange, 3: Tool, 4: User, 5: Pathpoint)
- RefFrameID - *DINT*
  Reference frame unique identifier (for user, tool, pathpoint, or generic frames)

1.3.1 Frame definition

For any movement in the path, the target position in Cartesian space is defined by the coordinates Ref_Path[…].Pose.X/Y/Z/Rx/Ry/Rz of the *tool frame* (a frame describing the pose of the tool) with respect to *an external reference frame*. These frames are defined by the following parameters:

- Ref_Path[…].Pose.Type
- Ref_Path[…].Pose.ID
- Ref_Path[…].Pose.RefFrameType
- Ref_Path[…].Pose.RefFrameID

Hence an example input of moving Tool Frame 1 to a pose wrt. UserFrame4 with have the following values:

- Ref_Path[…].Pose.Type=3 (tool frame)
- Ref_Path[…].Pose.ID=1 (configured tool frame 1)
- Ref_Path[…].Pose.RefFrameType=4 (user frame)
- Ref_Path[…].Pose.RefFrameID=4 (configured user frame 4 when Ref_Path[…].Pose.RefFrameType =4)

1.3.2 Target position and move type

For each movement of the path:

- if a move type is absolute (Ref_Path[…].MoveType = 0), parameter Ref_Path[…].TargetPointType is used to define target coordinates which can be specified in one of the following ways:
    - Joint space, setting the values for each joint (J1, J2,…) in the Ref_Path[…].Joints[] array
    - Cartesian space, setting Cartesian coordinates in Ref_Path[…].Pose (see previous section 'Frames Definition' for the meaning of these coordinates).

    Note. In general, more than one set of joint values (J1,J2,…) may correspond to the same position and orientation of the tool in Cartesian space (X,Y,Z,Rx,Ry,Rz), hence for PTP and CPW moves only (see the following section) the Ref_Path[…].RobotConfiguration must also be inputted as either same or new configuration to select the right joint set. The configurations of all supported robot types are described in the Rockwell Automation Publication MOTION-UM002-EN-P.

- if the move type is incremental (Ref_Path[…].MoveType = 1), parameter Ref_Path[…].TargetPointType is used to define that the coordinates are specified in one of the following ways:
    - Joint space, values in the Ref_Path[…].Joints[] array are added to the target joint absolute position of the previous movement to calculate the absolute joint coordinates of the target position;
    - Cartesian space, values in Ref_Path[…].Pose are a Cartesian offset which is added to the target Cartesian absolute position of the previous movement to calculate the target Cartesian absolute position of the actual movement.

1.3.3 Interpolation type

Another concept unrelated to the space used for defining the coordinates of the target, is the interpolation type, which the user must input in the Ref_Path[…].InterpolationType parameter.

- If commands are issued which control how the joints' positions change with respect to time, the robot is moving in Joint space.
- If commands are issued which control how the flange position and orientation changes in Cartesian space with respect to time, we say the robot is moving in Cartesian space.

For each movement Ref_Path[…].InterpolationType defines the interpolation between the starting position and the target position. In other words, it defines how the position of the robot changes with respect to time from the starting position to the target position. The 'how' includes if the joint values (J1,J2,…) or the Cartesian positions (X,Y,Z,Rx,Ry,Rz) are set at any point in time, that is if the movement is in joint space or in Cartesian space.

An important point to reiterate is that Ref_Path[…].TargetPointType is unrelated to Ref_Path[…].InterpolationType:

- A movement in joint space (e.g., Ref_Path[…].InterpolationType = PTP) can be executed with either a target point defined in Cartesian or in joint space;
- A movement in Cartesian space (e.g., Ref_Path[…].InterpolationType = CP-L/CP-W) can be executed with either a target point defined in Cartesian or in joint space

Movements with any set of Ref_Path[…].TargetPointType, Ref_Path[…].InterpolationType, Ref_Path[…].MoveType values can be freely mixed in a path sequence.

1.3.4 Singularities

Knowing the space of the movement is important as movements in joint space allow changes in the configuration of the robot, whilst movement in Cartesian space does not. *If an application requires a change in the configuration of the robot, the interpolation type must be a movement in joint space and in some cases a CPW move for 6-DoF robots.* Currently the PTP type (see below) is the only movement entirely in joint space while the CPW type is a hybrid interpolation of XYZ in cartesian space and J4J5J6 in joint space.

When the robot moves in joint space, the system can always calculate the position in Cartesian space correspondent to the interpolated Joint space position, but the opposite is not always true. For some interpolated Cartesian positions, it is not possible to calculate the correspondent Joint space position and we call these Cartesian positions 'singular positions' or singularities.

For this reason, attempting to move a robot away from a singular position by means of a movement with a Cartesian Linear InterpolationType, will result in an error. The only way to move the robot away from the singularity is issuing PTP move(s) (and in some cases a CPW move for 6 DoF robots if the singularity is around J5).

Singularities are all robot positions correspondent to a change in the robot configuration. Hence, as already stated, a Cartesian interpolated movement cannot change the configuration of the robot, whilst a joint interpolated movement can.

1.3.5 Point-to-Point (PTP) Moves

Point-to-Point (PTP) interpolation type (Ref_Path[…].InterpolationType=0) is a movement in joint space. The movement of the joints (J1,J2,…) is interpolated so that each of them start the movement at the same point in time and end at the same point in time.

The Ref_Path[…].ProfileType parameter is used to define the profile type of the movement over time.

The maximum speed, acceleration, deceleration, and jerk must be set by the user in the following parameters:
Ref_Path[…].Speed (deg/s)
Ref_Path[…].Acceleration (deg/s^2)
Ref_Path[…].Deceleration (deg/s^2)
Ref_Path[…].AccelerationJerk (% of time)
Ref_Path[…].DecelerationJerk (% of time)

Orientation speed, acceleration and deceleration are not used for PTP moves.

Generally, for any given target position, a PTP interpolation will decrease the time needed for a movement with respect any cartesian interpolation counterparts in Cartesian space, but gives less control on the resulting trajectory. Despite knowing the start and end position of the movement, it is hard to predict the shape of the trajectory in for PTP moves. Despite this, PTP interpolation type should be preferred as it is the fastest interpolator (resulting in smaller cycle times). Only where the application requires a precise and predictable movements (e.g., during the picking or placing of an object), a Cartesian InterpolationType should be used.

1.3.6 Continuous Path Linear (CP-L) Moves

Continuous Path Linear (CP-L) interpolation type (Ref_Path[…].InterpolationType=1) is a movement in Cartesian space.
The starting position of the flange frame is represented by the coordinates: Xs,Ys,Zs,Rxs,Rys,Rzs and the target position of the flange frame is represented by the coordinates: Xt,Yt,Zt,Rxt,Ryt,Rzt, (with respect to the robot frame).
The movement is composed of two parts:

- The translational movement where the origin of the flange will move along a straight segment in 3D space connecting the origin of the start position (Xs,Ys,Zs) and the origin of the target position (Xt,Yt,Zt) of the movement.
- The orientation movement where the orientation of the flange will be interpolated by a smooth interpolation in Cartesian space from the orientation of the start position (Rxs,Rys,Rzs) to the orientation of the target position (Rxt,Ryt,Rzt) of the movement.

The two movements are interpolated so that they both start the movement at the same point in time, and end at the same point in time as well.

The Ref_Path[…].ProfileType parameter is used to define the profile type of the movement over time.

The maximum speed, acceleration, deceleration and jerk must be set by the user in the following parameters:

- For the translational movement
  Ref_Path[…].Speed (mm/s)
  Ref_Path[…].Acceleration (mm/s^2)
  Ref_Path[…].Deceleration (mm/s^2)
  Ref_Path[…].AccelerationJerk (% of time)
  Ref_Path[…].DecelerationJerk (% of time)

- For the rotational movement
  Ref_Path[…].OrientationSpeed (deg/s)
  Ref_Path[…].OrientationAcceleration (deg/s^2)
  Ref_Path[…].OrientationDeceleration (deg/s^2)

1.3.7 Continuous Path Wrist (CP-W) Moves

Continuous Path Wrist (CP-W) interpolation type (Ref_Path[…].InterpolationType=2) is a hybrid interpolation type where the linear movement of the tooltip happens in Cartesian space along a segment from the starting point to the target point, whilst the orientation interpolation of the tool happens in joint space.
The starting position of the flange frame is represented by the coordinates: Xs,Ys,Zs, (J4s), (J5s), J6s and the target position of the flange frame is represented by the coordinates: Xt,Yt,Zt, (J4t), (J5t), J6t. The Cartesian coordinates are used for the linear movement and joint coordinates for the orientational movement.
Here:

- Xs, Ys, Zs, Xt, Yt, Zt, are with respect to the robot frame.
- The parenthesis around J4, and J5 mean that not all the joints exist for all kinds of robots: an articulated independent robot has J4, J5 and J6, whilst for a SCARA and articulated dependent robot only J6 is present. For a delta, only J5 and J6 exist at most.

The movement is composed of two parts:

- The translational movement where the origin of the tool will move along a straight segment in 3D space connecting the origin of the start position (Xs,Ys,Zs) and the origin of the target position (Xt,Yt,Zt) of the movement.
- The orientation movement where the orientation of the tool will be interpolated by a smooth interpolation in joint space from the orientation of the start position ((J4s), (J5s), J6s) to the orientation of the target position ((J4t), (J5t), J6t) of the movement.

The two movements are interpolated to start the movement at the same point in time, and end at the same point in time as well.

The Ref_Path[…].ProfileType parameter is used to define the profile type of the movement over time.

The maximum speed, acceleration, deceleration and jerk must be set by the user in the following parameters:

- For the translational movement
  Ref_Path[…].Speed (mm/s)
  Ref_Path[…].Acceleration (mm/s^2)
  Ref_Path[…].Deceleration (mm/s^2)
  Ref_Path[…].AccelerationJerk (% of time)
  Ref_Path[…].DecelerationJerk (% of time)

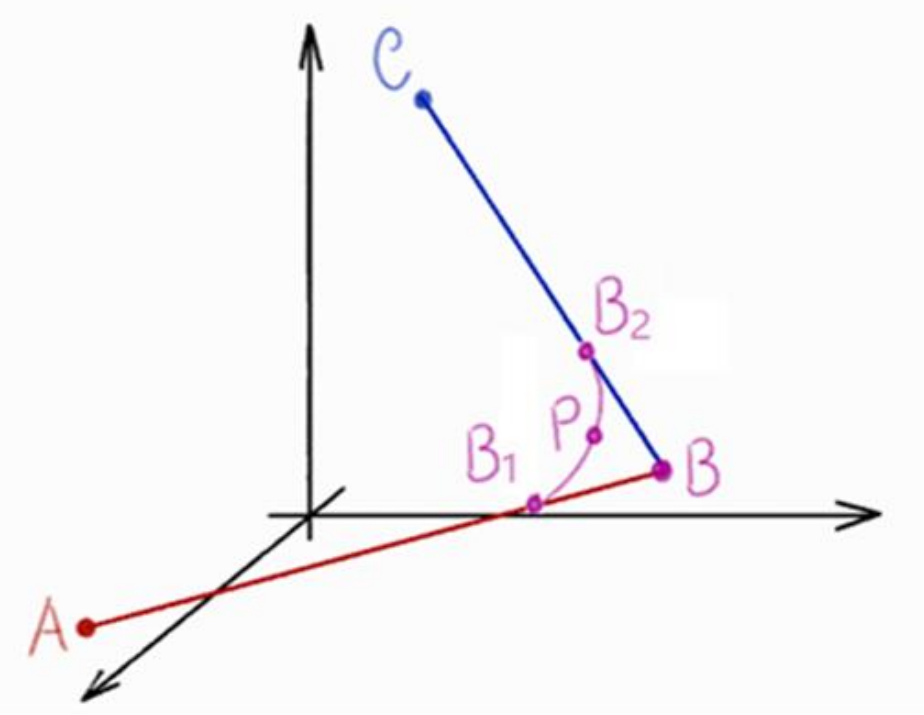
- For the rotational movement
  Ref_Path[…].OrientationSpeed (deg/s)

Ref_Path[…].OrientationAcceleration (deg/s^2)
Ref_Path[…].OrientationDeceleration (deg/s^2)

The CP-W movement still makes the tooltip move in a straight line, but at the same time, being the orientational interpolation in joint space. it has two advantages over the CP_L:

- For the articulated independent robot, the movement isn't affected by the singularity J5=0, meaning that a CP-W allows to change the Flip/Non flip configuration;
- It executes a single movement which makes J6 axis rotate with multiple full rounds.

1.3.8 Blending



Blending is used to connect two consecutive movements which is controlled by the user using the two input parameters Ref_Path[…].TerminationType and Ref_Path[…].CommandTolerance.

The picture represents two consecutive movements (the red first one, from position A to B, and the second blue one, from B to C). For simplicity the two movements are shown as segments in 3D space, but in general each of them can be any of the supported movements (PTP, CP-L, …).

- If Ref_Path[…].TerminationType is set to 0, the resulting motion will be a movement from A to B with a complete stop in point B, hence followed by the movement from B to C which will start from B with zero speed. If Ref_Path[…].TerminationType=0, the value of Ref_Path[…].CommandTolerance is ignored.
- If Ref_Path[…].TerminationType is set to 6, the resulting motion will be a trajectory which doesn't pass through position B, and it won't stop during the transition from the first movement to the next. This results in a total movement (AB + BC) which can be significantly faster.

Using termination type 6 (blending), the resulting trajectory will deviate around point B compared to the trajectory that would have resulted if Ref_Path[…].TerminationType=0. In robotics position B is often called a 'via point' (meaning a point which is used to define the trajectory transition without actually reaching the point) and whereas 'blending' is the modification of the trajectory around point B (shown as the magenta curve in the picture).

In real applications via points are used for fast movements between areas of the work cell. Blending is not recommended for areas where the robot interact with the product (e.g., picking up or placing an object) and should be avoided because as the resulting trajectory is less predictable.

If Ref_Path[…].TerminationType is set to 6, Ref_Path[…].CommandTolerance can be used to control the amount of blending and will reflect how much the trajectory is modified by the blending.

- If the movement from A to B is a PTP interpolation, Ref_Path[…].CommandTolerance must be entered as a percentage of the total length of the AB movement, and the BB1 portion of the movement will always be less than or equal to the specified percentage of the total AB movement;
- If the movement from A to B is in Cartesian space, Ref_Path[…].CommandTolerance must be entered in millimeters, and the BB1 portion of the movement will be always less than or equal to the specified value.

Note that for cases where consecutive invocations of the raM_Robot_Opr_LoadPath add-on instruction are issued where the last movement of the first call has Ref_Path[…].TerminationType=6, the last movement of the first path will be blended with the first movement of the second path, as if all the points were loaded through a single invocation.

1.3.9 MoveID

Each movement can be indicated by the user with a MoveID. The library will not use or interpret the MoveID value in any way. It is merely a value that is returned in verbatim as a read-only value in Sts.ActiveMoveID and Sts.BlendingMoveID (Sts public tag is in the handler management program associated to the robot, in the motion task) when the robot is executing the path point in question. Hence the user will know exactly when the robot is executing a specific movement of the path and may use such information in the application logic.

1.4 Motion planner

The motion planner has two primary components; calculations for the motion planner and the execution for the motion planner.

Currently four different motion profiles are supported:

- Cubic / S-Curve profile
- Poly-5 profile
- Modified Sine profile
- Sine-squared profile

Every motion profile is scaled with a master axis in the execution of the motion planner.
For more information on motion profiles, see the Motion profiles section.

1.4.1 Master axis management

The master axis used for scaling the motion profiles can be set from 0 to 125%. The velocity of this axis is also called Feedrate. At a feedrate of 100% the defined motion profiles are followed with the defined settings. When changing the feed rate to 50%, the calculated profile will be used but scaled by 0.5. This will result in half of the speed, acceleration and deceleration.

The following add-on instruction allows the user to change the feedrate.

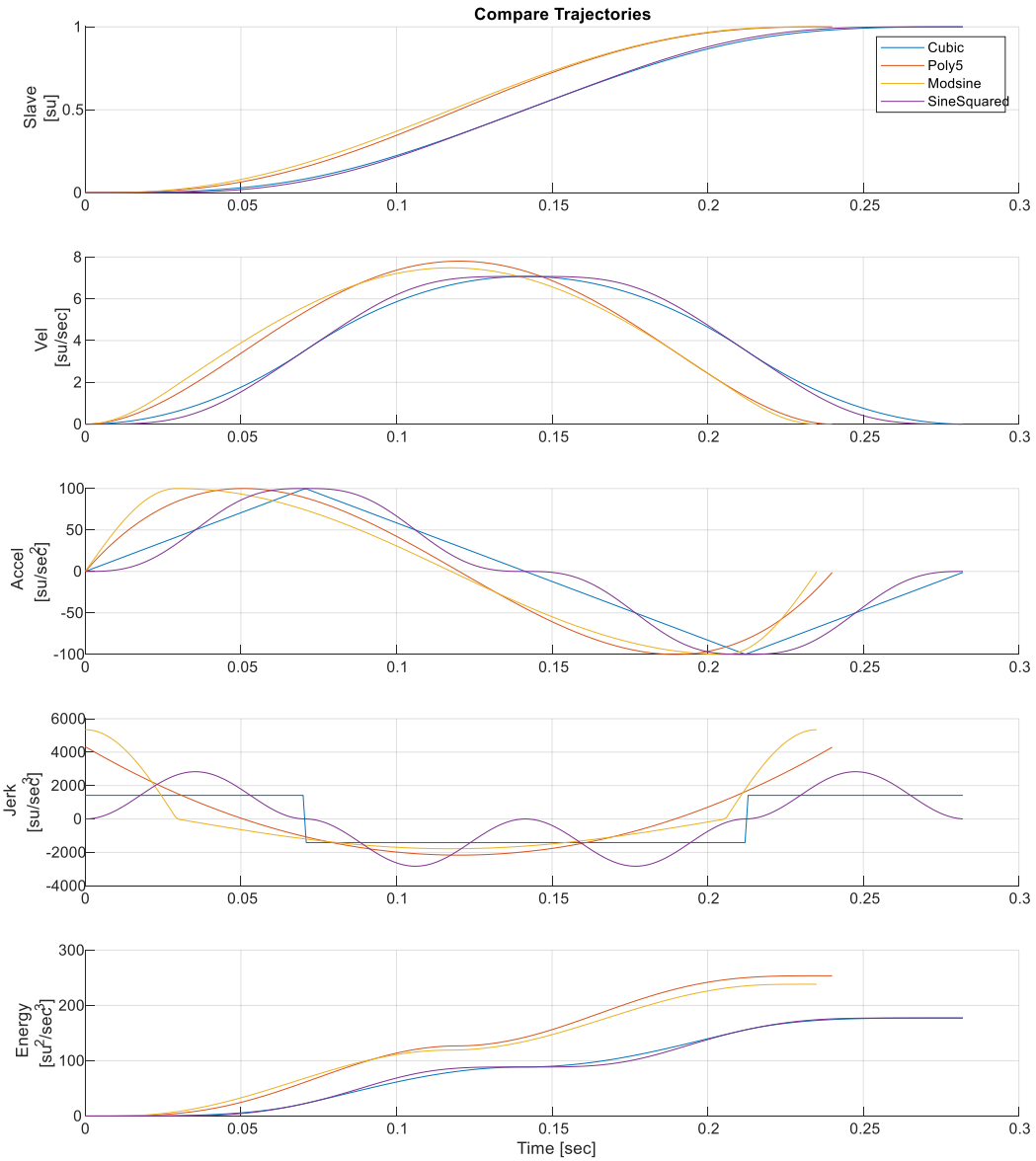
- raM_Robot_Opr_Feedrate
- raM_Robot_Opr_StopOnPath

1.4.2 Motion profiles

When loading a move to the motion planner, a profile with the relevant settings is calculated. Currently it is possible to calculate four different motion profiles:

- Cubic
- Poly5
- Modified Sine
- Sine Squared

The difference between motion profiles is shown below.



Upon review of the figure, the following conclusions are made:

- Cubic – An S-Curve profile that provides basic fast motion but is the least smooth.
- Poly5 – Another basic profile with improved smoothness over S-Curve.
- Modified Sine – Recommended setting for robotics as it generates the shortest move times with the lowest energy while providing the best balance of smoothness and jerk.
- Sine Squared – Smoothest at the cost of longer move times.

Jerk can be configured for any profile from 0-100% of time. Increasing this value trades off slightly slower move times for considerably lower Jerk. Higher is better for smoothness and vibrations.

- 100% Jerk – Default setting (recommended) with continuously varying acceleration.
- 1-99% Jerk – Continuously varying acceleration at the beginning and end of the move with constant acceleration in the middle.
- 0% Jerk – Constant acceleration, generating a Trapezoidal profile. Trapezoidal moves are not recommended as they impose high stress on electrical and mechanical components.

1.5 Execution

- Level

1.5.1 Overview

Rung in condition transition response:

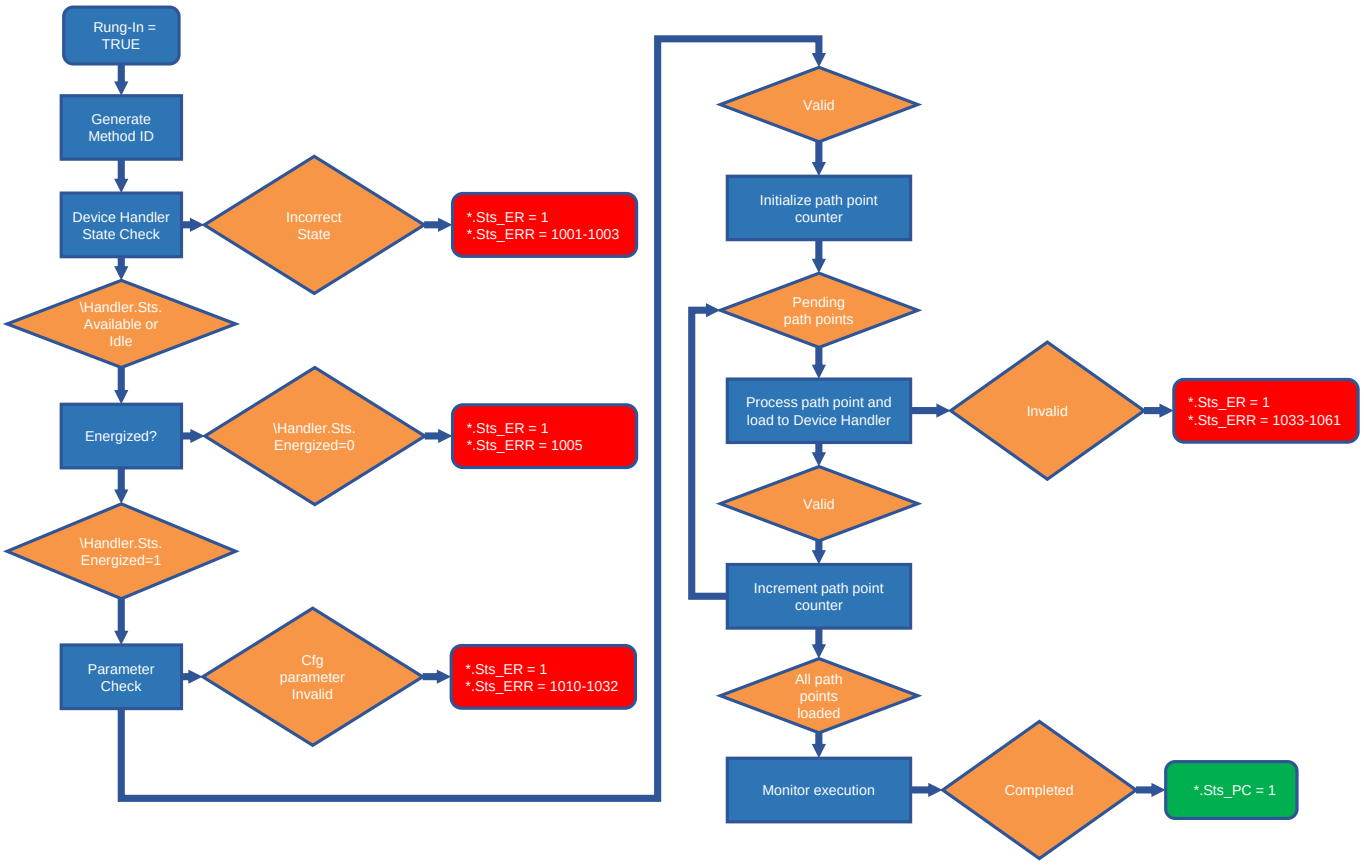
- False → True
  - Initialization
    - *.Sts_EO = 0
    - *.Sts_ER = 0
    - *.Sts_PC = 0
    - *.Sts_IP = 0
  - Running
    - *.Sts_EO = 1
    - *.Sts_EN = 1
    - *.Sts_IP = 1
      - *Check parameters*
      - *Process and load path points to Device Handler buffer*
      - *Monitor path execution*
      - IF: *All path points validated and executed*
        - THEN: *.Sts_PC = 1 and *.Sts_IP = 0
      - IF: Error
        - THEN: *.Sts_IP = 0 and *.Sts_PC = 0 and *.Sts_ER = 1
- True → False
  - *.Sts_EO = 0
  - *.Sts_EN = 0
  - *.Sts_IP = 0
    - IF: Error
      - THEN: *.Sts_ER = 1

1.5.2 Affected Device Handler Status

Status	Value
*.Sts.PathPlannerActive	1
*.Sts.ActiveMoveID	MtdID or ID from path point
*.Sts.ActiveInterpolation	Interpolation type from path point
*.Sts.ActiveType	Flange or tool selection type from path point
*.Sts.ActiveTypeID	Tool ID of selection tool selection from path point
*.Sts.ActiveRefFrameType	Reference frame selected from path point
*.Sts.ActiveRefFrameID	Tool ID or UserID of reference selection from path point
*.Sts.ActiveMoveID	MtdID or ID from path point
*.Sts.ActiveMovePerc	Percentage of completion of path point trajectory
*.Sts.BlendingMoveID	MtdID or ID from path point
*.Sts.BlendingMovePerc	Percentage of completion of path point trajectory while blending
*.Sts.LinearVelocity	Linear flange velocity
*.Sts.LinearAccleration	Linear flange acceleration

Status	Value
*.Sts.AngularVelocity	Angular flange velocity
*.Sts.AngularAcceleration	Angular flange acceleration

1.5.3 Execution Table



2 Instruction

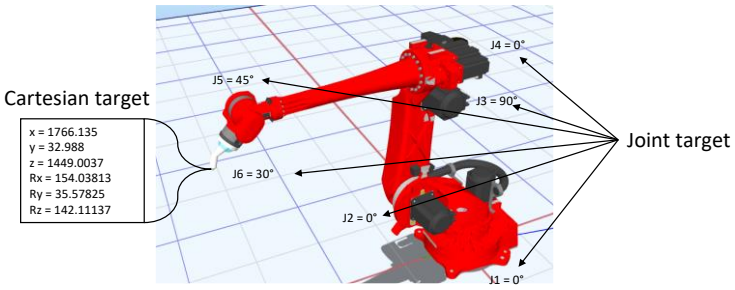
2.1 Input Data

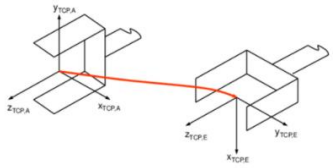
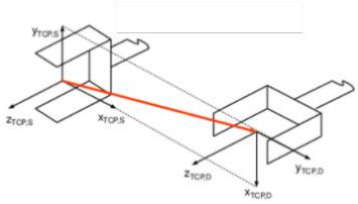
Input	Function / Description	DataType
Ref_Handle	Device Handler data structure	raM_UDT_Robot_Dvc_DataHndl
Ref_Path	Sequence of path points to load	raM_UDT_Robot_Opr_PathPoint
Cfg_StartIndex	First path point to load from Ref_Path	DINT
Cfg_NumberOfMoves	Number of path points to execute	DINT
Cfg_EnableTracking0	Enable Tracking Objects 0	BOOL
Cfg_EnableTracking1	Enable Tracking Objects 1	BOOL

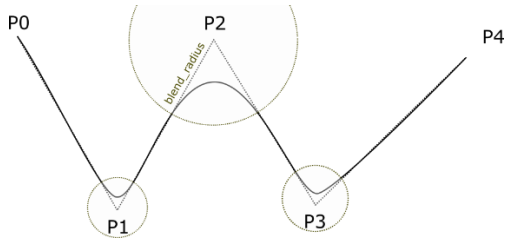
2.1.1 raM_UDT_Robot_Opr_PathPoint

The raM_UDT_Robot_Opr_PathPoint UDT contains the relevant data for each path point along the desired trajectory when using the LoadPath instruction.

Member	Function / Description	DataType
MoveID	Optional numeric identifier for the path point that will be indicated by the Device Handler during travel. 0: Instruction instance ID will be used to identify the path point. <>0: a unique user assigned ID will be used to identify the path point. The library will not use or interpret the MoveID value in any way. It is merely a value that is returned verbatim as a read-only value via the “Sts.ActiveMoveID” Device Handler tag, when the robot starts executing the path point in question. Hence, the user will know exactly when the robot is executing a specific movement of the path and may use such information in the application logic.	DINT
MoveType	Selects the desired interpretation for the target coordinates. 0: Absolute. The move coordinates will be interpreted as displacements from an absolute (zero) reference. In joint space, the absolute reference is the joint home position; in Cartesian space, the absolute reference is the user-defined reference frame specified in the member “Pose”. 1: Incremental. The move coordinates will be interpreted as relative displacements from the current robot position or previous target.	DINT

Member	Function / Description	DataType
TargetPointType	Selects whether the target is a Cartesian target or a Joint target.  0: Joint. The target will be defined by a joint position array (see member “Joints”). 1: Cartesian. The target will be defined by the pose of the end-effector relative to a user-selectable reference frame (see member “Pose”). 1: Hybrid. The target XYZ will be defined by the pose of the end-effector relative to a user-selectable reference frame (see member “Pose”) while the orientation is defined by the J4J5J6 (Joints[3] - Joints[5])	SINT

Member	Function / Description	DataType
InterpolationType	Selects the method for interpolating the path from the current robot pose or previous target to the new target. 0: PTP (Point-to-Point). Used for general movements in which the destination is relevant, but the path shape is irrelevant. Motion planner will try to generate the fastest move to target in joint space. The resulting path in Cartesian movement is unpredictable and generally not the shortest. Special attention must be paid to avoid collisions when moving in narrow spaces. All the joints will begin moving and stop moving at the same time, such that all axes adapt to the slowest axis. One advantage of PTP moves is their ability to move through singularities without issues. Only PTP moves are allowed when transforms are disabled.  1: CP-L (Cartesian Linear). Used for fine movements in which both the destination and path shape are relevant. Robot will move to target following a predictable straight-line path in Cartesian space, while also controlling the end-effector orientation. The position and orientation are interpolated such that the final compound movement starts and ends at the same time. The resulting path is generally the shortest, but total move duration tends to be greater than PTP. Cartesian linear moves are usually appropriate for high precision tasks and limited spaces susceptible to collisions, but special attention must be paid to avoid moving through singularities.  2: CP-W (Cartesian Wrist). Continuous Path Wrist (CP-W) interpolation type (Ref_Path[…].InterpolationType=2) is a hybrid interpolation type where the linear movement of the tooltip happens in Cartesian space along a segment from the starting point to the target point, whilst the orientation interpolation of the tool happens in joint space. CP-W moves the tooltip move in a straight line. it has two advantages over the CP_L: - For the articulated independent robot, the movement isn't affected by the singularity J5=0, meaning that a CP-W allows to change the Flip/Non flip configuration. - It allows to create a single movement which makes J6 axis rotate with multiple full turn counts. 3: CP-C (Cartesian Circular). Currently not supported.	SINT
Pose	Target pose, defined in Cartesian space. Used when TargetPointType = Cartesian. Contains the desired position and orientation of the end-effector relative to a given reference frame. The position must be specified in mm and the orientation in deg. Orientation is represented by three angles following the Euler XYZ fixed-frame convention. The pose must be currently specified as flange frame w.r.t. Robot frame.	raM_UDT_Robot_Opr_Frame

Member	Function / Description	DataType
Joints	Target position, defined in joint space. Used when TargetPointType = Joint or Hybrid. Contains the desired joint positions in zero index (J1 = Joints[0], J6 = Joints [5]), in mm or deg. The values will be validated against the robot's joint position limits.	REAL[9]
TerminationType	Defines whether blending is used to connect two consecutive moves. Considering that the robot is at A, will move from A to target B (move A-B), and finally from B to C (move C-D); the dropdown offers two options to define the blending of moves A-B and C-D: 0: Stop. Blending is disabled. The robot will stop at the target. The resulting motion will be a movement from A to B with a complete stop in point B, hence followed by the movement from B to C which will start from B with zero speed. This termination type is adequate for the first and last targets of a trajectory, and for any intermediate targets that must be accurately reached as part of a process requirement (e.g., picking parts, welding points, etc.). Note that disabling blending for all the intermediate targets without explicit requirements may introduce discontinuous trajectories and unnecessary acceleration/braking cycles. 6: Blend. Blending is enabled. The robot will approach the target without stopping on it. The resulting motion will be a trajectory which doesn't pass through position B, and won't stop during the transition from the first movement to the next. This results in a total movement (AB + BC) which can be significantly faster. To keep a smooth trajectory towards the target, the path planner will apply a blending radius according to a pre-defined tolerance (see member "CommandTolerance"). Not stopping at intermediate targets may reduce the total duration and improve the smoothness of the trajectory. Additionally, smoother moves tend to put less effort on the drives and transmissions, increasing the lifespan of the robot. 	DINT
CommandTolerance	Blending radius from target path point. For CP-L moves, value is in mm and must be greater than zero. For PTP moves, value is in % of total move distance and must be between 0% and 100%.	REAL

Member	Function / Description	DataType
RobotConfiguration	Defines the preferred robot configuration from the set of potential solutions capable of reaching the target position. In general, more than one set of joint positions may correspond to the same position and orientation of the tool in Cartesian space; hence, for PTP moves only, the robot configuration must be entered. Must be zero for Delta robot geometries. Bit0: 1 = change previous configuration; 0 = keep previous configuration. Bit1: 1 = lefty; 0 = righty. Bit2: 1 = above; 0 = below. Bit3: 1 = flip; 0 = no flip. The configurations of all supported robot geometries are described in the Rockwell Automation publication MOTION-UM002-EN-P. More information on robot configurations can also be found in the RM_raM_Robot_Opr_SetConfiguration user manual.	SINT
ProfileType	Motion profile function to be used by the path planner. 0: S-curve. Cubic profile that provides basic / traditional motion. Can provide fast motion but is the least smooth. 1: Poly5 (fifth-order polynomial). Another basic profile. Improves smoothness over S-Curve. 2: ModSine (modified sine). Provides the best balance of move time, smoothness, and energy. Lowest energy consumption. 3: $Sine^2$ (sine squared). Smoothest at the cost of slightly longer move times. NOTE: The above comparison statements were based on the fact that acceleration is an input and kept constant while generating the profiles.	SINT
Speed	For PTP moves, defines the joint speed limit, in mm/s or deg/s. For CP moves, defines the end-effector linear speed limit, in mm/s, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
Acceleration	For PTP moves, defines the joint acceleration, in mm/s^2 or deg/s^2. For CP moves, defines the end-effector linear acceleration, in mm/s^2, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
Deceleration	For PTP moves, defines the joint deceleration, in mm/s^2 or deg/s^2. For CP moves, defines the end-effector linear deceleration, in mm/s^2, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL

Member	Function / Description	DataType
AccelerationJerk	For PTP moves, defines the joint acceleration jerk, in in % of time. For CP moves, defines the end-effector linear acceleration jerk, in % of time, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
DecelerationJerk	For PTP moves, defines the joint deceleration jerk, in % of time. For CP moves, defines the end-effector linear deceleration jerk, in % of time, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
OrientationSpeed	Not used for PTP moves. For CP moves, defines the end-effector angular speed limit, in deg/s, when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
OrientationAcceleration	Not used for PTP moves. For CP moves, defines the end-effector angular acceleration, in deg/s², when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL
OrientationDeceleration	Not used for PTP moves. For CP moves, defines the end-effector angular deceleration, in deg/s², when moving from the start position to the target position. Value must be greater than zero for the first move. For all the remaining points, zero means “keep last valid value”.	REAL

2.1.2 raM_UDT_Robot_Opr_Frame

The raM_UDT_Robot_Opr_Frame data type is used to define the “.Pose” member of raM_UDT_Robot_Opr_PathPoint. It represents a robot target position in Cartesian space, i.e., the desired position and orientation of the end-effector with respect to a given reference frame.

Member	Function / Description	DataType
X	X coordinate value setpoint (mm).	REAL
Y	Y coordinate value setpoint (mm).	REAL
Z	Z coordinate value setpoint (mm).	REAL
Rx	Rotation angle setpoint around X-axis (deg).	REAL
Ry	Rotation angle setpoint around Y-Axis (deg).	REAL
Rz	Rotation angle setpoint around Z-Axis (deg).	REAL
Type	Frame type. 2: Flange 3: Tool	DINT

Member	Function / Description	DataType
ID	0 Frame unique identifier for Type = 3 (tool frame)	DINT
RefFrameType	Reference frame. 0: World 1: Robot 2: Flange 3: Tool 4: User	DINT
RefFrameID	0 Frame unique identifier for RefFrameType = 3 (tool frame) or 4 (user frame)	DINT

2.2 Output Data

Output	Function / Description	DataType
Sts_EO	Instruction has enabled the rung output. Provides a visible indicator of the EnableOut system parameter for use during ladder instantiation	BOOL
Sts_EN	Instruction is Being Scanned - Rung In Condition = TRUE	BOOL
Sts_ER	Instruction is in Error - See Sts_ERR / Sts_EXERR for Additional Error Information	BOOL
Sts_ERR	Instruction Error Code - See Instruction Help for Code Definition	DINT
Sts_EXERR	Instruction Extended Error Code - See Instruction Help for Code Definition	DINT
Sts_MtdID	Method ID	DINT
Sts_IP	Instruction is 'In Process'	BOOL
Sts_DN	Instruction has completed loading all targets	BOOL
Sts_AC	Loaded targets are currently active	BOOL
Sts_PC	Instruction Process is Complete	BOOL
Sts_ActivePathPoint	Current path point being executed	DINT
Sts_PendingPathPoints	Number of path points yet to be executed	DINT

2.3 Error Codes

Sts_ERR	Description
0	No errors present
1000	Method failed to register. Method will not execute until registered. Method Registry Array must be larger.
1001	Device Handler is not in a running state. Verify Ethernet modules and motion group state
1002	Device Handler faulted, if path planning was executing when error occured, see extended error code for last active path point
1003	Device Handler is not in a supported state. Device Handler must be in state Available. Verify Ethernet modules, motion group state and that the Device Handler has been configured
1005	Robot is not energized
1006	User program moves only allowed in automatic external mode
1007	Instance interrupted while loading path points by other instance or stop
1010	Number of moves exceed path point array size
1012	Start Index out of bounds, index must be less than input path array size
1013	Speed, acceleration and deceleration must be > 0.0 for first move
1014	Jerk must be >= 0.0 and <= 100.0 for first move
1015	Cartesian orientation dynamics must be > 0 for first move
1017	Invalid MoveType, see extended error for path point index
1018	Invalid TargetPoint Type, see extended error for path point index
1019	Only absolute cartesian targets are allowed without transform enabled for first move
1020	Invalid InterpolationType, see extended error for path point index
1021	Invalid InterpolationType, only PTP is allowed if transform is not enabled, , see extended error for
1022	Invalid command tolerance < 0.0, see extended error for path point index
1023	Invalid command tolerance, PTP move tolerance is maximum 100.0%, see extended error for path point index
1024	Invalid TerminationType, see extended error for path point index
1025	Invalid ProfileType, see extended error for path point index
1026	Invalid Robot Configuration, see extended error for path point index, see extended error for path point index
1027	Invalid Pose type, select Flange or Tool frame, see extended error for path point index
1028	Invalid Pose reference type, select World, Robot or User frame, see extended error for path point index
1029	Pose ID, Tool ID, out of bounds, input a value <= 0 and <= 15, see extended error for path point index

Sts_ERR	Description
1030	Ref Frame ID, User ID, out of bounds, input a value <= 0 and <= 15, see extended error for path point index
1031	Pose ID is not configured, please configure a valid tool fram, see extended error for path point index
1032	Reference frame ID is not configured, please configure a valid user frame, see extended error for path point index
1033	Forward Geometry: Joint limits exceeded
1034	Forward Geometry: Folded robot joint limits error
1035	MoveType of a trajectory point wrt Frame or Tool cannot be incrementa
1036	Start pose to Robot frame transform error
1037	Validation of target pose failed
1038	Number of target references cannot exceed 3
1039	Start pose to target frame transform error
1040	Target pose to Robot frame transform error
1041	Inverse Geometry: Invalid tool frame
1042	Inverse Geometry: Invalid target frame
1043	Inverse Geometry: Start joints out of range
1044	Inverse Geometry: Could not find a solution
1045	Inverse Geometry: Singularity, see extended error for joint
1046	Inverse Geometry: Robot reach exceeded
1047	Inverse Geometry: Joint limits exceeded for target
1048	Cartesian interpolation type cannot change to desired configuration
1049	Severe forward geometry error, contact support
1050	Severe inverse geometry error, contact support
1051	Start pose to Robot frame transform error
1052	Start pose to Flange and Robot frame transform error
1053	Target pose to Flange and Robot frame transform error
1055	Invalid configuration bits for Scara, only lefty/right is supported
1056	Invalid target, resulting rotational angles not supported (Rx, Ry)
1057	Inverse Geometry: Invalid target, y coordinate must be =0 for Delta2
1058	Inverse Geometry: Joint input must be =0 for disabled joint, see EXERR for joint
1059	Internal fault, contact support
1060	Sever motion planner fault, contact support
1061	Output start pose to Flange and Robot frame transform error

Sts_EXERR	Description
>= 0	Path point in the Path input array where error occured, joint reference, refer to error code

3 Application Code Manager

3.1 Definition Object: raM_Robot_Opr_LoadPath

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

3.2 Implementation Object: raM_LD_Robot_LoadPath

Implementation Language: Ladder
Content Type: Routine

This implement contains only a rung with an instance of the raM_Robot_Opr_LoadPath object.

Parameter Name	Default Value	Instance Name	Definition	Description
RoutineName	ObjectName}	{RoutineName}	Routine	Name of the routine where the object will be placed
TagName	_{ObjectName}	{TagName}	Tag	Instruction backing tag
StartBitTagName	Cmd_{ObjectName}	{StartBitTagName}	Local Tag	Tag name for start command enabling bit
PathName	_{ObjectName}Path	{PathName}	Local Tag	Tag name for Path Array
PathSize	20	{PathSize} 1..1000	Array size	Array size for Path Array

Linked Library

Link Name	Catalog Number	Revision	Solution	Category
RobotHandler	raM_Robot_Dvc_DeviceHandler	2	(RA-LIB) Robotics	Robot Handler
raM_Robot_Opr_LoadPath	raM_Robot_Opr_LoadPath	2	(RA-LIB) Robotics	Asset-Control

3.3 Attachments

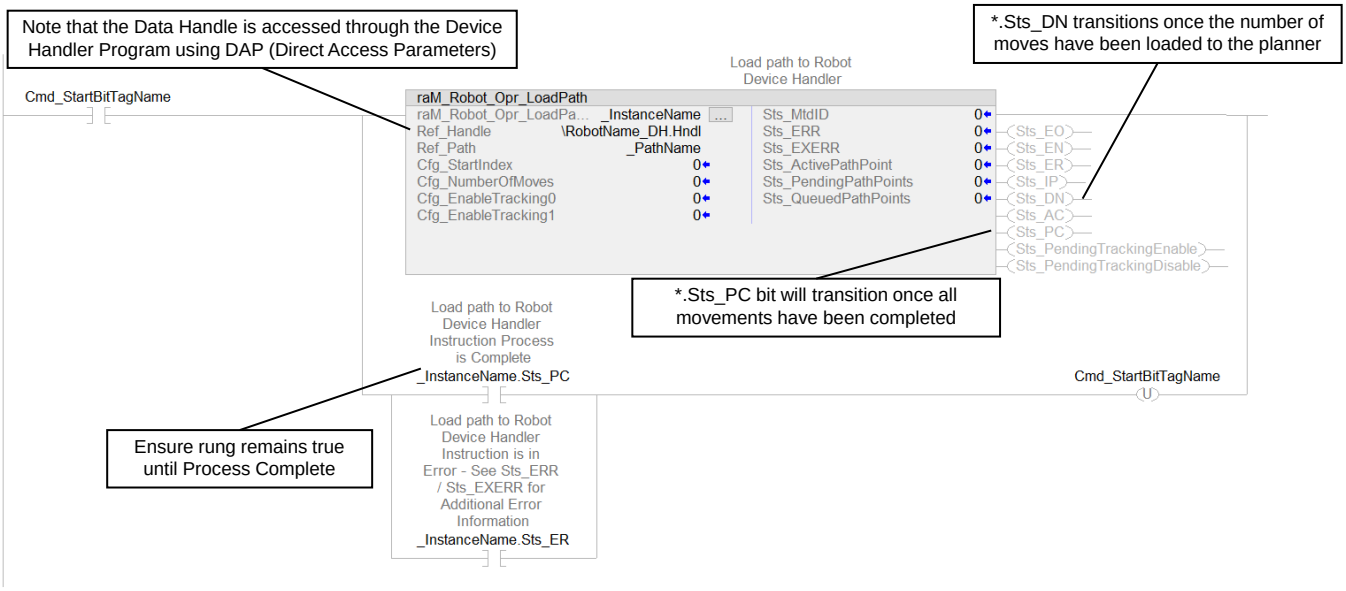
Name	Description	File Name	Extraction path
V2_{LibraryName}	Reference Manual	RM-{LibraryName}.pdf	{ProjectName}\Documentation

4 Application

4.1 Using raM_Robot_Opr_LoadPath

Prior to calling raM_Robot_Opr_LoadPath, make sure that Ref_Path is populated with all the desired path points, and that each path point is properly configured. Use Cfg_StartIndex and Cfg_NumberOfMoves to select the subset of Ref_Path to be executed. Monitor the execution via Sts_ActivePathPoints and Sts_PendingPathPoints.

Ensure that the rung-in condition is maintained throughout the entire path execution, i.e. until the Sts_PC bit is set.



5 Appendix

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.

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**

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

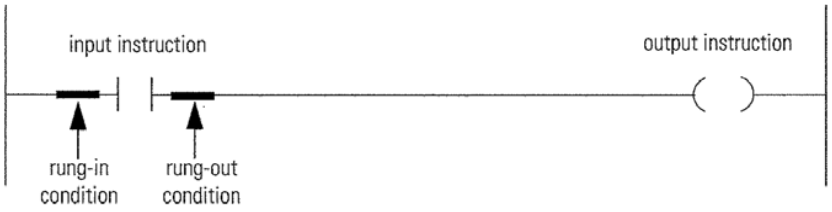
Instruction Execution - Edge and Continuous

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.