AXYSTEMS Ltd.		Document No.
Department:	***Engineering***	Revision No:1
Document Title:		Page 1 of 22
Written By:	Authorized By	Authorization Date:

Generic Points

TABLE OF CONTENTS

Revision History

Revision Number	Description
	•

Non-generic points	Currently existing joint- and location-type points, with predefined and unchangeable robot-types, determined during point declaration.
Generic points	Joint- and location-type points, with changeable robot-types.
Lists of coordinates	Constant joint- and location-type point values, without robot-types. Appear as a list of double-type arguments, separated by commas, within curly brackets. In location-type lists, curly brackets are preceded by '#'.
	Joint-type list: {<coordinate>, <coordinate>, …}
	Location-type list: #{<coordinate>, <coordinate>, …}

1. Declaration

As in non-generic points, generic points also have two distinct and types: joint and location. The type is determined during declaration and cannot be changed afterwards.
However, unlike non-generic points, generic points are not linked to specific robot-types during declaration. Therefore, robot-types of generic points are not fixed, and can be changed several times throughout application. As a result, generic points have no robot-types in translation phase, and their current robot-types can be calculated only during run-time, much like values of variables and handles of generic axes and groups.
Declaration syntax of generic points was designed to be consistent with declaration syntax of non-generic points:

```
Declaration of generic joints:

{Common Shared| Dim Shared| Dim} <name>{[size]…} As Generic Joint

Declaration of generic locations:

{Common Shared| Dim Shared| Dim} <name>{[size]…} As Generic Location

Declaration of generic joint and location fields:

Type <struct_name>
      <name>{[size]…} As Generic joint
      <name>{[size]…} As Generic Location
End Type
```

Immediately after declaration generic points have no robot-types and zero size. Therefore, they cannot be used before initialization. Initialization can be performed through casting (see section 2), or through assignment of another point variable with a previously defined robot-type (see section 3).

2. Casting

```
? CASTPOINT(<list_of_coordinates>, <robot_type_number>)

<joint_point> = CASTJOINT(<double_or_long_scalar>, <robot_type_number>)

<location_point>= CASTLOCATION(<double_or_long_scalar>, <robot_type_number>)

<joint_point> = CASTJOINT(<double_or_long_whole_array>, <robot_type_number>)

? CASTLOCATION(<double_or_long_whole_array>, <robot_type_number>)
```

- The points created and returned by all casting functions are generic, i.e., their robot-type match to other points is checked during run-time and not during translation.
- **CASTPOINT** can accept only a list-of-coordinates as point parameter. Using point variables or properties instead will raise a translation syntax error.
- The type (i.e., joint or location) of the point created and returned by **CASTPOINT** is determined by the type of the list-of-coordinates parameter.
- In **CASTPOINT**, the robot-type input must match the size of the list-of-coordinates parameter.
- Values of point coordinates created and returned by **CASTPOINT** are taken from the list-of-coordinates argument.

```
Common shared GenJoint as Generic Joint         /* Generic point */
Common shared GenLoc as Generic Location        /* Generic point */

Common shared JointXYZ as joint of XYZ    /* Non-generic point */
Common shared LocXYZ as Location of XYZ   /* Non-generic point */

GenJoint = CASTPOINT({0.0, 10.0, 20.0}, TYPE_XYZ)→ OK
GenLoc = CASTPOINT(#{0.0, 0.0, 0.0}, TYPE_XYZ)    → OK
/* Casting of a non-generic point – robot-types must match */
JointXYZ = CASTPOINT({0, 1, 1}, TYPE_XYZ)→ OK, TYPE_XYZ is redundant

Translation errors:
/* First parameter must be a list-of-coordinates */
GenLoc = CASTPOINT(LocXYZ, TYPE_XYZ)      → Syntax error

/* Casting of a joint generic point, using a location-type list */
GenJoint = CASTPOINT(#{0.0, 10.0, 20.0}, TYPE_XYZ)→ Wrong input type

/* Casting of a location generic point, using a joint-type list */
GenLoc = CASTPOINT({0.0, 0.0, 0.0}, TYPE_XYZ)   → Wrong input type

Run-time errors:
/* Size of list-of-coordinates does not match robot-type */
GenJoint = CASTPOINT({1, 0, 0}, TYPE_XYZR )   → Size mismatch

/* Invalid robot-type value */
GenLoc = CASTPOINT(#{0.0, 10.0, 20.0}, 100)     → Invalid robot-type

/* Casting of a non-generic point, using a different robot-type */
JointXYZ = CASTPOINT({0,0,0,0}, TYPE_XYZR)    → Robot-type mismatch
RobotXYZR.TOOL = CASTPOINT(#{260,0,0},TYPE_XYZ)→Robot-type mismatch
```

- **CASTJOINT** creates and returns a joint-type point. Number of coordinates and robot-type are both determined by robot-type input.
- **CASTLOCATION** creates and returns a location-type point. Number of coordinates and robot-type are both determined by robot-type input.
- Both **CASTJOINT** and **CASTLOCATION** can accept double- or long-type scalar expressions, as well as single-dimension double- or long-type whole arrays. However, these casting functions cannot accept point arguments.
- **CASTJOINT** and **CASTLOCATION** accepting a double- or long-type scalar expression will create a point composed of identical coordinate values, taken from the expression's value.
- Coordinate values of **CASTJOINT** and **CASTLOCATION** accepting a whole double- or long-type array will be taken from the array elements in the same order of appearance, i.e., first coordinate will be assigned by first array element, etc. If the

number of array elements exceeds the number of coordinates determined by robot-type argument, redundant values will be ignored. On the other hand, if the number of coordinates exceeds the number of array elements, a run-time error will be raised.

```
Common shared GenJoint as Generic Joint         /* Generic point */
Common shared GenLoc as Generic Location        /* Generic point */

Common shared JointXYZ as joint of XYZ    /* Non-generic point */
Common shared LocXYZ as Location of XYZ   /* Non-generic point */

Common shared DblArr2Dim[2][2] as Double

Common shared DblArr[3] as Double
DblArr[1] = 2.6
DblArr[1] = 1.8
DblArr[3] = 0.5

Dim shared LngArr[4] as Long

LngArr[1] = 1
LngArr[1] = 0
LngArr[3] = 2
LngArr[4] = 10


Dim Shared LngVar as Long = 3
Dim DblVar as Double = 2.5

/* First argument is a long- or double-type scalar */
GenJoint = CASTJOINT(1, TYPE_XYZ)→ {1, 1, 1}
? CASTJOINT(DblVar, TYPE_XYZR)→ {2.5, 2.5, 2.5, 2.5}
GenLoc = CASTLOCATION(LngArr[4], TYPE_XY)→ #{10, 10}

/* First argument is a single-dimension long- or double-type whole
array */
? CASTJOINT(LngArr, TYPE_XY)→ {1, 0} /* Last two array elements are
ignored */
LocXYZ = CASTLOCATION(DblArr, TYPE_XYZ)→ #{2.6, 1.8, 0.5}


/* First argument is a complex expression */
GenLoc = CASTLOCATION(0.5+1, TYPE_XYZR)→ #{1.5, 1.5, 1.5, 1.5}


Translation errors:
/* First argument is not a point */
GenLoc = CASTLOCATION(LocXYZ, TYPE_XYZ)    → Syntax error
GenJoint = CASTJOINT({0.0, 10.0, 20.0}, TYPE_XYZ)→ Syntax error

/* Only a single-dimension array can be used as argument */
LocXYZ = CASTLOCATION(DblArr2Dim, TYPE_XYZ)→ Syntax error

 Run-time errors:
/* Array argument has less elements (3) than coordinates of robot-
type (4) */
? CASTJOINT(DblArr, TYPE_XYZR)  → Size mismatch
```

- Casting may be used to initialize newly declared generic points by giving them a robot-type, size and coordinate values. It can be used within declaration statement itself, and anywhere throughout application.

```
Common shared GenJoint as Generic Joint = CASTJOINT(1.1, TYPE_XYZ)

Dim shared GenLoc as Generic Location = CASTPOINT(#{1.0,0.0,0.0}, TYPE_XYZ)
```

- The “robot_type” argument of the casting functions can be any long-type expression (a double-type value will be converted to long).
- The robot-type parameter must return a valid robot-type value. For this purpose, a list of constants, representing valid robot-type values, will be added to language. These constants may be used as robot-type parameters in casting functions (see below).
- Assignment of a point returned for casting function into a non-generic point (with a predefined robot-type) wil result in assignment of the point’s values by the values of the list-of-coordinates input, but a matching robot-type input will be ignored. However, if the robot-types do not match, a run-time error will be raised.

<u>List of robot type constants and their values</u>:

? TYPE_NONE	→	0	*/*coordinate lists, uninitialized generic points*/*
? TYPE_X	→	1	
? TYPE_XY	→	2	
? TYPE_XYZ	→	3	
? TYPE_XYZU	→	4	
? TYPE_XYZUV	→	5	
? TYPE_XYZUVW	→	6	
? TYPE_XYR	→	7	
? TYPE_XYRZ	→	8	
? TYPE_XYRZU	→	9	
? TYPE_XYRZUV	→	10	
? TYPE_XYZR	→	11	
? TYPE_XYZRU	→	12	
? TYPE_XYZRUV	→	13	
? TYPE_XYZURV	→	14	
? TYPE_XYZUR	→	15	
? TYPE_XYZRPU	→	16	
? TYPE_XYZRP	→	17	
? TYPE_XYRURV	→	18	
? TYPE_XYRUR	→	19	
? TYPE_XYRP	→	20	
? TYPE_XYRPU	→	21	
? TYPE_XYRPUV	→	22	
? TYPE_XYRPUQ	→	23	
? TYPE_XYZYPR	→	24	
? TYPE_C7	→	25	
? TYPE_C8	→	26	
? TYPE_C9	→	27	
? TYPE_C10	→	28	
? TYPE_CMAX	→	29	
? TYPE_USER1	→	31	
? TYPE_USER2	→	32	
? TYPE_USER3	→	33	
? TYPE_USER4	→	34	
? TYPE_USER5	→	35	
? TYPE_XYZPR	→	36	
? TYPE_XYZA	→	37	
? TYPE_XYZAB	→	38	
? TYPE_C11	→	39	
? TYPE_C12	→	41	
? TYPE_C13	→	42	
? TYPE_C14	→	43	
? TYPE_C15	→	44	
? TYPE_C16	→	45	
? TYPE_C17	→	46	
? TYPE_C18	→	47	
? TYPE_C19	→	48	
? TYPE_C20	→	49	
? TYPE_YPR	→	50	
? TYPE_PR	→	51	
? TYPE_R	→	52	

3. Assignment

3.1 Initialization through assignment

Assignment may be used to initialize newly declared generic points by giving them the robot-type, size and coordinate values of the “right-side” point in the assignment statement.

- For initializtion through assignmnet, the “right side” of the assignment statement must have a predefined robot-type as in non-generic point variables, generic points with robot-types, and point properties. Therefore, initialization cannot be performed through assignment of a list of coordinates.
- Initialization through assignment requiers point-type match, i.e., joint generic points must be initialized by joint-type points, and location generic points must be initialized by location-type points.
- Initialization through assignmnet may be performed within declartion statement.

```
Initialization of generic points through assignment:

Common shared GenJoint as Generic Joint
Common shared GenLoc as Generic Location

Common shared XYZJoint as Joint of XYZ
Common shared XYZLoc as Location of XYZ

Dim GenJointArr[10] as Generic Joint

/* Initialization within declaration statemnt */
Dim GenPoint as Generic location = XYZLoc

/* Type mismatch */
GenJoint = XYZLoc                → Translation error
/* Type mismatch */
GenLoc = XYZRobot.VCMD           → Translation error


/* "Right side" is a list of coordinates, with no robot-type */
GenLoc = #{1.0, 0.0, -1.0)       → Run-time error

/* "Right side" generic point is not initialized */
GenJoint = GenJointArr[1]    → OK, but GenJoint remain uninitialized

/* Initialization through a pre-initialized generic point */
GenJointArr[1] = CASTPOINT( {1.0, 0.0, -1.0}, TYPE_XYZ )
GenJoint = GenJointArr[1]        → OK

/* Initialization through a non-generic point */
GenJoint = XYZJoint              → OK

/* Initialization through a point property */
GenLoc = XYZRobot.HERE           → OK
```

3.2 Assignment after initialization

After initialization, the robot-type of a generic point can be changed numerous times through the CASTPOINT function (see section 2), as well as through a regular assignment statement.

- Whenever the left-side of the assignment statement is a generic point, its robot-type (and size) will be run-over by the robot-type (and size) of the right-side point.
- On the other hand, assignment of a non-generic point by a point having a different robot-type will result in a "Robot-type mismatch" run-time error.
- Assignment of a list-of-coordinates (with no robot-type) into a generic is allowed only if the generic point already has a robot-type, and if sizes match.

- Assignment of a preinitialized generic point by a non-initialized generic point is allowed, but will result in nullification of the preinitialized point.

```
Assignment rules of generic points after initialization:

Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYZLoc as Location of XYZ
Common shared XYZJoint as Joint of XYZ

Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim GenLocArr[10] as Generic Location
GenLocArr[1] = XYZRLoc

Dim GenJointArr[2][2] as Generic Joint
GenJointArr[1][1] = CASTPOINT({1,1,1}, TYPE_XYR)

GenLoc = XYZRLoc                    → OK, changed robot-type to XYZR
GenJoint = XYRRobot.PFB             → OK, changed robot-type to XYR
GenJoint = GenJointArr[1][1]        → OK, robot-type remained XYR
GenJoint = {10.0, 0.0, -10.0}       → OK, matching size

/* Point-type mismatch */
GenLoc = GenJoint                   → Translation error
GenLoc = XYZJoint                   → Translation error
GenJoint = #{10.0, 0.0, -10.0}      → Translation error
GenJoint = XYZRobot.BASE            → Translation error

/* Robot-type mismatch */
XYZRJoint = GenJoint                → Run-time error (XYZR vs. XYR)
XYRRobot.BASE = GenLoc              → Run-time error (XYR vs. XYZR)

/* Size mismatch */
GenLoc = #{1.0, 0.0, -1.0)          → Run-time error (size 4 vs. 3)

/* Assignment by a non-initialized generic point */
GenJoint = GenJointArr[2][2]   → OK, but GenJoint will be nullified
```

4. Binary operations

Addition, subtraction and compound operators are operated between two points.

- In case one, or both points are generic, robot-type (or size, when the other point is a list-of-coordinates) of the two points must match.

```
Rules for binary operations with generic points:

Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYZLoc as Location of XYZ
Common shared XYZJoint as Joint of XYZ

Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim GenLocArr[10] as Generic Location
GenLocArr[1] = XYZRLoc

Dim GenJointArr[2][2] as Generic Joint
GenJointArr[1][1] = CASTPOINT({1,1,1}, TYPE_XYR)

? GenJoint + GenJointArr[1][1]    → OK, point- and robot-types match
? GenLoc + XYZLoc                 → OK, point- and robot-types match
? GenLoc - XYZRobot.BASE          → OK, point- and robot-types match
? GenJoint : {10.0, 0.0, -10.0}   → OK, point-types and sizes match


/* Point-type mismatch */
? GenLoc + GenJoint                 → Translation error
? GenLoc + XYZJoint                 → Translation error
? GenJoint - XYZRobot.BASE          → Translation error
? GenJoint : #{10.0, 0.0, -10.0}    → Translation error

/* Robot-type mismatch */
? GenLoc + XYZRLoc                  → Run-time error (XYZ vs. XYZR)
? XYRRobot.PCMD : GenJoint          → Run-time error (XYR vs. XYZ)
? GenJoint - GenJointArr[1][1]      → Run-time error (XYZ vs. XYR)
/* Binary operation with a non-initialized generic point */
? GenJointArr[2][2] + GenJoint → Run-time error (NONE vs. XYZ)

/* Size mismatch */
? GenLoc - #{1.0, 0.0, -1.0, 0.0)   → Run-time error (size 4 vs. 3)
```

- The result of a binary operation between two generic points is a generic point.
- The result of a binary operation between a generic point and a list-of-coordinates is also a generic point.
- The result of a binary operation between a generic point and a non-generic point variable is a non-generic point.
- The result of a binary operation between a generic point and a point property is also a non-generic point.

```
Type of points resulting from binary operations with generic points:

<generic_point> + <generic_point> → <generic_point>

<generic_point> - <list_of_coordinates> → <generic_point>

<generic_point> : <non_generic_point_variable> → <non_generic_point>

<generic_point> + <point_property> → <non_generic_point>
```

5. Assisting functions

5.1 ROBOTTYPE

Identification of robot-type is especially important in generic points, since robot-type may be changed numerous times throughout application. The ROBOTTYPE function can be applied for generic points, non-generic points, point properties and lists of coordinates, returning a long-type value corresponding to the robot-type. In case of non-initialized generic points, as well as for lists- of-coordinates, the returned value is 0.

```
? ROBOTTYPE(<point_name | list_of_coordinates | point_property>)

Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)
Common shared GenLocArr[10] as Generic Location

Dim shared LocXYZR as Location of XYZR
Dim shared JointXYZ as Joint of XYZ

GenLocArr[1] = LocXYZR

/* Initialized generic points */
? ROBOTTYPE(GenJoint)              →     3
? ROBOTTYPE(GenLocArr[1])          →     11

/* Non-initialized generic point */
? ROBOTTYPE(GenLocArr[3])          →     0
/* Non-generic point */
? ROBOTTYPE(JointXYZ)              →     3
/* Point property */
? ROBOTTYPE(XYZRRobot.START)       →     11
/* List-of-coordinates */
? ROBOTTYPE(#{1.0, 0.0, 1.0, 0.0}) →     0
```

5.2 ROBOTTYPE$

The ROBOTTYPE$ function can be applied for generic points, non-generic points, point properties and lists of coordinates, returning a string containing the name of the robot-type. In case of non-initialized generic points, as well as for lists-of-coordinates, the returned string will contain “NONE”.

```
? ROBOTTYPE$(<point_name | list_of_coordinates | point_property>)

Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)
Common shared GenLocArr[10] as Generic Location

Dim shared LocXYZR as Location of XYZR
Dim shared JointXYZ as Joint of XYZ

GenLocArr[1] = LocXYZR

/* Initialized generic points */
? ROBOTTYPE(GenJoint)                  →     "XYZ"
? ROBOTTYPE(GenLocArr[1])              →     "XYZR"

/* Non-initialized generic point */
? ROBOTTYPE(GenLocArr[3])              →     "NONE"
/* Non-generic point */
? ROBOTTYPE(JointXYZ)                  →     "XYZ"
/* Point property */
? ROBOTTYPE(XYZRRobot.START)           →     "XYZR"
/* List-of-coordinates */
? ROBOTTYPE(#{1.0, 0.0, 1.0, 0.0})     →     "NONE"
```

5.3 NOOFCOORDINATES

The ability to detect the point’s size (number of coordinates) is also important in generic point, since size may vary throughout application, due to changes in robot type. The NOOFCOORDINATES function can be applied for generic points, non-generic points, lists of coordinates and point properties. If a generic point is not initialized – the function will return zero.

```
? NOOFCOORDINATES(<point_name>)

Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)
Common shared GenLocArr[10] as Generic Location

Dim shared LocXYZR as Location of XYZR
Dim shared JointXYZ as Joint of XYZ

GenLocArr[1] = LocXYZR

/* Initialized generic points */
? NOOFCOORDINATES(GenJoint)                  →     3
? NOOFCOORDINATES(GenLocArr[1])              →     4

/* Non-initialized generic point */
? NOOFCOORDINATES(GenLocArr{2])              →     0

/* Non generic point */
? NOOFCOORDINATES(JointXYZ)                  →     3
/* Point property */
? NOOFCOORDINATES(XYZRRobot.START)           →     4
/* List of coordinates */
? NOOFCOORDINATES(#{1.0, 0.0, 1.0, 0.0})     →     4
```

6. Functions and subroutines

6.1 By-Value Parameters

Generic points can be used as by-value parameters in function and subroutine prototypes. On the other hand, they can also be passed by-value to both generic and non-generic point parameters. Generic points passed by-value to non-generic point parameters must be initialized first.

- Generic points used as by-value parameters in function and subroutine prototypes can accept generic points, non-generic points and point properties. The only limitation is that point-types (i.e., joint vs. location) must match. Passage of a non-initialized generic point will result in an uninitialized point parameter, which might cause run-time errors when used inside the function's block. Passing lists of coordinates, which do not have robot-types, is forbidden. Thereby, passing a point by-value to a generic point parameter resembles initialization through assignment (see section 3).

```
Generic points as by-value parameters in function \ subroutine
prototypes:

Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim GenJointArr[10] as Generic Joint

Sub MySub1(ByVal GenParamJoint as Generic Joint)
…
End Sub

Passing points with robot-types:
Call MySub1(GenJoint)                → OK, initialized generic point
Call MySub1(XYZRJoint)               → OK, non generic point
Call MySub1(XYZRobot.DEST_JOINT)     → OK, point property

Point-type mismatch:
Call MySub1(GenLoc)                  → Translation error
Call MySub1(XYZRLoc)                 → Translation error
Call MySub1(XYZRobot.DEST)           → Translation error

Passing points without robot-types:
/* A list of coordinates */
Call MySub1({10.0, 0.0, -10.0})     → Run-time error

/* Non-initialized generic point */
Call MySub1(GenJointArr[1])  → OK, but parameter is uninitialized

Sub MySub1(ByVal GenParamJoint as Generic Joint)
   Move XYZRobot GenParamJoint      → Run-time error
End Sub
```

- On the other hand, generic points passed by-value to non-generic point parameters of functions and subroutines must also match prototype in robot-type, as well as in point-type.

```
Generic points passed by-value to non-generic point parameters:

Common shared GenLoc as Generic Location = CASTPOINT(#{0,0,0,0), TYPE_XYZR)
Common shared GenJoint as Generic Joint = CASTPOINT({0,0,0,0), TYPE_XYZR)

Common shared XYZLoc as Location of XYZ
Common shared XYZRJoint as Joint of XYZR

Dim GenLocArr[10] as Generic Location
GenLocArr[1] = XYZLoc


Sub MySub2(ByVal ParamXYZRLoc as Location of XYZR)
…
End Sub

Call MySub2(GenLoc)             → OK (robot-type of GenLoc is XYZR)

/* Point-type mismatch */
Call MySub2(GenJoint)           → Translation error

/* Robot-type mismatch */
Call MySub2(GenLocArr[1])       → Run-time error

/* Non-initialized generic point */
Call MySub2(GenLocArr[4])       → Run-time error (robot-type mismatch)
```

6.2 By-Reference Parameters

Generic points can be used as by-reference parameters in function and subroutine prototypes. On the other hand, they can also be passed by-reference to both generic and non-generic point parameters.

- Generic points used as by-reference parameters in function and subroutine prototypes can accept generic points and non-generic point variables, but cannot accept lists of coordinates, and point properties. The only limitation is that point-types (i.e., joint vs. location) must match. Non-initialized generic points can be passed by-reference, and may be initialized inside function \ subroutine block. However, usage without initialization within function \ subroutine block might raise a run-time error.

Generic points as by-reference parameters in function \ subrotine prototypes:

```
Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYRLoc as Location of XYR
Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim GenLocArr[10] as Generic Location

Function MyFunc1(GenParamLoc as Generic Location) as long
…
End Function

? MyFunc1(GenLoc)              → OK, generic point variable
? MyFunc1(XYRLoc)              → OK, non-generic point variable

/* Passing "values" by-reference is not allowed */
? MyFunc1(#{10.0, 0.0, -10.0})                  → Translation error
? MyFunc1(CASTPOINT(#{10.0,0.0,-10.0},TYPE_XYZ))→ Translation error
? MyFunc1(XYZRobot.DEST)                        → Translation error

/* Point-type mismatch */
? MyFunc1(GenJoint)                  → Translation error
? MyFunc1(XYZRJoint)                 → Translation error

/* Changing robot-type inside the function's block */
Function MyFunc1(GenParamLoc as Generic Location) as long
      GenParamLoc = XYZRLoc
End Function
? MyFunc1(GenLoc)→ OK. Robot-type of GenLoc was changed to XYZR.
? MyFunc1(XYRLoc)→ Error. Robot-types of XYRLoc and XYZRLoc differ.

/* Non-initialized generic points: no error for function call, but
run-time error when trying to use the non-initialized parameter
inside the function's block */
? MyFunc1(GenLocArr[1])              → No error for function call

Function MyFunc1(GenParamLoc as Generic Location) as long
      ? GenParamLoc + #{1,0,1}       → Run-time error
/* Initialization inside the function block */
      GenParamLoc = XYRLoc
End Function
```

- On the other hand, generic points can be passed by-reference as non-generic point parameters of functions and subroutines. Although the only limitation is that point-types (i.e., joint vs. location) must match, it is the user's responsibility to also match the robot-type of the non-generic to prototype. Otherwise, robot-type declared in prototype becomes inapplicable. Therefore, passing a generic point by-reference to a non-generic prototype will raise a translation note.

```
Generic points passed by-reference to non-generic point parameters:

Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)
Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)

Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim GenJointArr[10] as Generic Joint
GenJointArr[1] = XYZRJoint

Function MyFunc2(ParamXYZJoint as Joint of XYZ) as String
/* Query the robot-type of the non-generic parameter */
  MyFunc2 = ROBOTTYPE$(ParamXYZJoint)
End Function

/* Initialized generic point with a matching robot-type */
? MyFunc2(GenJoint)                  → "XYZ", Translation note
/* Initialized generic point with a non-matching robot-type */
? MyFunc2(GenJointArr[1])            → "XYZR", Translation note
/* Non-Initialized generic point with no robot-type */
? MyFunc2(GenJointArr[2])            → "NONE", Translation note


/* Point-type mismatch */
? MyFunc2(GenLoc)                    → Translation error
```

6.3 Returned-Values

Generic points can be used as returned-values of functions. On the other hand, they can also be assigned to both generic and non-generic point returned-values. For assignment into non-generic point returned-values - generic points must be initialized first.

- Generic points used as returned-values of functions can be assigned by generic points, non-generic points and point properties. The only limitation is that point-types (i.e., joint vs. location) must match. Assignment of a non-initialized generic point will result in an uninitialized returned-value, which might cause run-time errors when used inside or outside the function's block. Assignment of lists of coordinates, which do not have robot -types, is forbidden. Thereby, assignment of a generic returned-value resembles initialization through assignment (see section 3).

```
Generic points as returned-values in functions:

Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYZRLoc as Location of XYZR
Common shared XYZRJoint as Joint of XYZR

Dim shared GenLocArr[10] as Generic Joint

Function GenLocFunc(…) as Generic Location

      GenLocFunc = <location_point>

End Function

Assignments with robot-types:
GenLocFunc = GenLoc                  → OK, initialized generic point
GenLocFun = XYZRLoc                  → OK, non-generic point
GenLocFunc = XYZRobot.DEST           → OK, point property

Point-type mismatch:
GenLocFunc = GenJoint                   → Translation error
GenLocFunc = XYZRJoint                  → Translation error
GenLocFunc = XYZRobot.DEST_JOINT        → Translation error

Assignments without robot-types:
/* A list of coordinates */
GenLocFunc = #{10.0, 0.0, -10.0}        → Run-time error
/* Non-initialized generic point */
GenLocFunc = GenLocArr[1] → OK, but returned-value is uninitialized
/*Using a non-initialized returned value will cause run-time error*/
Move XYZRobot GenLocFunc(…)
```

- On the other hand, generic points assigned into non-generic point returned-values of functions must also match prototype in robot-type, as well as in point-type.

```
Generic points assigned into non-generic point returned-values:

Common shared GenLoc as Generic Location=CASTPOINT(#{0,0,0},TYPE_XYZ)
Common shared GenJoint as Generic Joint=CASTPOINT({0,0,0},TYPE_XYZ)

Common shared XYZLoc as Location of XYZ
Common shared XYZJoint as Joint of XYZ

Dim shared XYZRJoint as Joint of XYZR

Dim GenJointArr[10] as Generic Joint
GenJointArr[1] = XYZRJoint


Function JointXYZFunc as Joint of XYZ

      JointXYZFunc = <joint_point>

End Function

JointXYZFunc = GenJoint        → OK (robot-type of GenJoint is XYZ)

/* Point-type mismatch */
JointXYZFunc = GenLoc          → Translation error

/* Robot-type mismatch */
JointXYZFunc = GenJointArr[1])→ Run-time error

/* Non-initialized generic point */
JointXYZFunc = GenJointArr[2] → Run-time error (robot-type mismatch)
```

Motion Issues:

Until now all motion variables ware used with a pre-defined robot-type, with the introduction of generic-points we will add flexibilities to the motion variables too:

- Arguments of movement commands (target point) will be possible to assign in any robot-type. Internal conversions will be done inside motion module.

- Robot location properties (BASE, TOOL, ….) will be possible to assign in any robot-type. Internal conversions will be done inside motion module.

New motion property returning robot type will be added:
<robot>.robottype – returns integer, read-only.

So genpnt = CASTPOINT(0.0, robot.robottype) will generate a location with coordinates of 0 of the default robot-type of the given robot.

7. Tests

- Tests must include generic points from every scope available: global, static and local.

- Tests must include both scalars and array elements. Multi-dimensional arrays should also be tested.
- Tests must include structure elements: scalar and array structure elements, elements from scalar structures and form arrays of structures.
- Assignment statements should include generic points in each side of the statement, as well as in both sides of the statement.
- Binary operations should also include generic points in each side of the operator, as well as in both sides of the operator.
- Generic points should appear as by-value and by-reference parameters in prototypes of functions and subroutines.
- Generic points should be used as returned-values of functions.
- Generic points should be passed by-value and by-reference to functions and subroutines, to both generic and non-generic parameters.
- Casting functions should be used in assignment statements and binary operations. They should also be passed by-value to functions and subroutines.