OMRON

V+ Module

Reference Manual

I668-E-01

NOTE

Trademarks

Copyrights

Introduction

This manual is OMRON's original instructions describing the use of V+ Modules.
These modules can only be used when the ACE server is running or a connection to ACE is present.
Please read this manual and make sure you understand the functionality and performance of the system before attempting to use these modules.
Keep this manual in a safe place where it will be available for reference during programming and configuration.

Sections in this Manual

CONTENTS

Section 3 V+ End-effector Module

Section 4 V+ Process Manager Module

Section 5 V+ Remote Library Module

Terms and Conditions Agreement

Warranty and Limitations of Liability

Warranty

- **Exclusive Warranty**
  Omron's exclusive warranty is that the Products will be free from defects in materials and workmanship for a period of twelve months from the date of sale by Omron (or such other period expressed in writing by Omron). Omron disclaims all other warranties, expressed or implied.
- **Limitations**
  OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCTS. BUYER ACKNOWLEDGES THAT IT ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE.
  Omron further disclaims all warranties and responsibility of any type for claims or expenses based on infringement by the Products or otherwise of any intellectual property right.
- **Buyer Remedy**
  Omron's sole obligation hereunder shall be, at Omron's election, to (i) replace (in the form originally shipped with Buyer responsible for labor charges for removal or replacement thereof) the non-complying Product, (ii) repair the non-complying Product, or (iii) repay or credit Buyer an amount equal to the purchase price of the non-complying Product; provided that in no event shall Omron be responsible for warranty, repair, indemnity or any other claims or expenses regarding the Products unless Omron's analysis confirms that the Products were properly handled, stored, installed and maintained and not subject to contamination, abuse, misuse or inappropriate modification. Return of any Products by Buyer must be approved in writing by Omron before shipment. Omron Companies shall not be liable for the suitability or unsuitability or the results from the use of Products in combination with any electrical or electronic components, circuits, system assemblies or any other materials or substances or environments. Any advice, recommendations or information given orally or in writing, are not to be construed as an amendment or addition to the above warranty.

See http://www.omron.com/global/ or contact your Omron representative for published information.

Limitations of Liability

OMRON COMPANIES SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE OR STRICT LIABILITY. Further, in no event shall liability of Omron Companies exceed the individual price of the Product on which liability is asserted.

Application Considerations

Suitability for Use

Omron Companies shall not be responsible for conformity with any standards, codes or regulations which apply to the combination of the Product in the Buyer's application or use of the Product. At Buyer's request, Omron will provide applicable third party certification documents identifying ratings and limitations of use which apply to the Product. This information by itself is not sufficient for a complete determination of the suitability of the Product in combination with the end product, machine, system, or other application or use. Buyer shall be solely responsible for determining appropriateness of the particular Product with respect to Buyer's application, product or system. Buyer shall take application responsibility in all cases.
NEVER USE THE PRODUCT FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCT(S) IS PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

Programmable Products

- Omron Companies shall not be responsible for the user's programming of a programmable Product, or any consequence thereof.
- Omron Companies shall not be responsible for the operation of the user accessible operating system (e.g. Windows, Linux), or any consequence thereof.

Disclaimers

Performance Data

Data presented in Omron Company websites, catalogs and other materials is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of Omron's test conditions, and the user must correlate it to actual application requirements. Actual performance is subject to the Omron's Warranty and Limitations of Liability.

Change in Specifications

Product specifications and accessories may be changed at any time based on improvements and other reasons. It is our practice to change part numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the Product may be changed without any notice. When in doubt, special part numbers may be assigned to fix or establish key specifications for your application. Please consult with your Omron's representative at any time to confirm actual specifications of purchased Product.

Errors and Omissions

Information presented by Omron Companies has been checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical or proofreading errors or omissions.

1

V+ AceServer Module

The information in this section documents the V+ programs that a user might use in the process of customizing an application leveraging ACE.

1-1 sv.create_msg

Creates a new message, initializing the header for writing.

Syntax

sv.create_msg(msg_num, handle, ptr)

Input Parameters

Parameter	Description
msg_num	Message number to create

Outputs Parameters

Parameter	Description
handle	Handle to new message
ptr	Pointer into the new message

1-2 sv.read_adouble

Read an array of double-precision floats from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_adouble(handle, ptr, offset, value[], n_items)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at
offset	Offset into output array for first item. 0 is default

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value[]	Array filled with data read
n_items	Number of items read

1-3 sv.read_ashort

Read an array of 16-bit integers from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_ashort(handle, ptr, offset, value[], n_items)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at
offset	Offset into output array for first item. 0 is default

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value[]	Array filled with data read
n_items	Number of items read

1-4 sv.read_asingle

Read an array of single-precision float variables from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_asingle(handle, ptr, offset, value[], n_items)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at
offset	Offset into output array for first item. 0 is default

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value[]	Array filled with data read
n_items	Number of items read

1-5 sv.read_astring

Read an array of strings from the message buffer. The message pointer is incremented after the read

Syntax

sv.read_astring(handle, ptr, offset, $value[], n_items)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at
offset	Offset into output array for first item. 0 is default

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
$value[]	Array filled with data read
n_items	Number of items read

1-6 sv.read_bool

Read a boolean from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_bool(handle, ptr, value)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-7 sv.read_byte

Reads a byte from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_byte(handle, ptr, value)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-8 sv.read_double

Read a double-precision float from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_double(handle, ptr, value)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
```

```
CALL sv.write_string("Text", handle, ptr)
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-9 sv.read_short

Read a 16-bit integer from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_short(handle, ptr, value)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-10 sv.read_single

Read a single-precision float from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_single(handle, ptr, value)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
```

```
CALL sv.write_string("Text", handle, ptr)
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-11 sv.read_string

Read a string from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_string(handle, ptr, $string)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
$string	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-12 sv.read_trans

Read a transform from the message buffer. The message pointer is incremented after the read.

Syntax

sv.read_trans(handle, ptr, trans)

Input Parameters

Parameter	Description
handle	Index of the message to read from
ptr	Position in the message buffer to read at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message
value	Value of the parameter read from the message stream

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-13 sv.sig.is.equal

Compares a signal to an expected value. This method is employed when a program checks if a signal is at the expected state. It encapsulates the comparison process but it also manages whether a program is running on an emulator or not. In the case of an emulator, reading inputs can be a problem as in emulator mode an external system generating the input signals is not available. In emulator mode, this method will always consider the signal to be at the expected state and then always return true.

Syntax

sv.sig.is.equal(sig.num, expected.val, is.equal.res)

Input Parameters

Parameter	Description
sig.num	The signal number we want to check.
expected.val	The expected signal value.

Output Parameters

Parameter	Description
is.equal.res	The result value, indicating if signal is at the expected value.

Example

```
.PROGRAM ex1()
; ABSTRACT: Demonstrate comparing a signal to an expected value
;;
INPUTS:
AUTO REAL my.signal1 = 1001
AUTO REAL my.signal2 = 1002
AUTO REAL is.off, is.on
; Check a signal is not at an incorrect value. This is checking an error case
; In emulator mode, is.off will always be true and program will not generate errors
CALL sv.sig.is.equal(my.signal1, FALSE, is.off)
IF NOT is.off THEN
TYPE "input is not at the correct state : it is on and should be off."
END
; wait a signal is at the expected value. If emulation mode is active, true will be
 returned and program can continue.
; In normal mode, the signal is read periodically and is compared to expected value
.
DO
```

```
CALL sv.sig.is.equal(my.signal2, TRUE, is.on)
UNTIL is.on
.END
```

1-14 sv.stop_collect

Stop the data collection operations on the specified robot.

Syntax

sv.stop_collect(robot.num)

Input Parameters

Parameter	Description
robot.num	The robot number to stop data collection on.

Output Parameters

None

1-15 sv.write

Write a message to the client.

Syntax

sv.write(handle, ptr)

Input Parameters

Parameter	Description
handle	Index of the message to write to
ptr	Position in the message buffer to write at
offset	Offset into output array for first item. 0 is default

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-16 sv.write_abyte

Write an array of bytes to the output message at the specified offset.

Syntax

sv.write_abyte($string[], offset, count, handle, ptr)

Input Parameters

Parameter	Description
$string	Array of strings to write
offset	1-based starting byte offset in thestring array
count	Number of bytes to transfer
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-17 sv.write_adoubl

Write an array of doubles to the output message at the specified offset.

Syntax

sv.write_adoubl(first_index, value[], n_items, handle, ptr)

Input Parameters

Parameter	Description
first_index	First index in the array to write
value[]	Array of data values to write
n_items	Number of items in value[] to write into the buffer
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-18 sv.write_ashort

Write an array of shorts to the output message at the specified offset.

Syntax

sv.write_ashort(first_index, value[], n_items, handle, ptr)

Input Parameters

Parameter	Description
first_index	First index in the array to write
value[]	Array of data values to write
n_items	Number of items in value[] to write into the buffer
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-19 sv.write_asingl

Write an array of floats to the output message at the specified offset.

Syntax

sv.write_asingl(first_index, value[], n_items, handle, ptr)

Input Parameters

Parameter	Description
first_index	First index in the array to write
value[]	Array of data values to write
n_items	Number of items in value[] to write into the buffer
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-20 sv.write_astrin

Write an array of strings to the output message at the specified offset.

Syntax

sv.write_astrin(first_index, $value[], n_items, handle, ptr)

Input Parameters

Parameter	Description
first_index	First index in the array to write
$value[]	Array of data values to write
n_items	Number of items in value[] to write into the buffer
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

1-21 sv.write_bool

Write a boolean to the output message at the specified offset.

Syntax

sv.write_bool(value, handle, ptr)

Input Parameters

Parameter	Description
value	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-22 sv.write_byte

Write a byte to the output message at the specified offset.

Syntax

sv.write_byte(value, handle, ptr)

Input Parameters

Parameter	Description
value	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-23 sv.write_hdr

Write the header for an output message.

Syntax

sv.write_hdr(handle, ptr)

Input Parameters

Parameter	Description
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

None

1-24 sv.write_short

Write a 16-bit integer to the output message at the specified offset.

Syntax

sv.write_short(value, handle, ptr)

Input Parameters

Parameter	Description
value	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-25 sv.write_single

Write a float to the output message at the specified offset.

Syntax

sv.write_single(value, handle, ptr)

Input Parameters

Parameter	Description
value	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-26 sv.write_string

Write a string to the output message at the specified offset.

Syntax

sv.write_string($string, handle, ptr)

Input Parameters

Parameter	Description
$string	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

1-27 sv.write_trans

Write a transform to the output message at the specified offset.

Syntax

sv.write_trans(trans, handle, ptr)

Input Parameters

Parameter	Description
trans	Value to write
handle	Index of the message to write to
ptr	Position in the message buffer to write at

Output Parameters

Parameter	Description
ptr	Index pointing to the next data item in the message

Example

```
.PROGRAM ex1.msg(handle, ptr)
; ABSTRACT: Demonstrate writing then reading to a message buffer
;
INPUTS: handle: Index of the message to write into
; ptr: Position in the message buffer

AUTO REAL saved.ptr
AUTO REAL bool.val, byte.val, double.val, short.val
AUTO REAL single.val
AUTO $string.val
AUTO LOC trans.val

; Save the original position of the pointer
saved.ptr = ptr

; Write some data
CALL sv.write_bool(TRUE, handle, ptr)
CALL sv.write_byte(8, handle, ptr)
CALL sv.write_double(100.1, handle, ptr)
CALL sv.write_short(2, handle, ptr)
CALL sv.write_single(121.2, handle, ptr)
CALL sv.write_string("Text", handle, ptr)
```

```
CALL sv.write_trans(NULL, handle, ptr)

; Read the data that was written
CALL sv.read_bool(handle, saved.ptr, bool.val)
CALL sv.read_byte(handle, saved.ptr, byte.val)
CALL sv.read_double(handle, saved.ptr, double.val)
CALL sv.read_short(handle, saved.ptr, short.val)
CALL sv.read_single(handle, saved.ptr, single.val)
CALL sv.read_string(handle, saved.ptr, $string.val)
CALL sv.read_trans(handle, saved.ptr, trans.val)
.END
```

2

V+ Robot Vision Manager Module

The information in this section documents the V+ programs that can be used to interact with the Robot Vision Manager.

2-1 as.save.image

Save the image associated with the vision tool into a file.

Syntax

as.save.image($filename, $ip, seq.idx, tool.idx, status)

Input Parameters

Parameter	Description
$filename	The name of the file to save into
$ip	The IP address of the PC
seq.idx	The index of the sequence
tool.idx	The index of the virtual camera ; or other image tool

Output Parameters

Parameter	Description
status	Status of the operation. 0 = Success

2-2 check_tracking

This function is used to check if the robot is successfully tracking the belt

Syntax

check_tracking(tracking_ok)

Input Parameters

None

Output Parameters

Parameter	Description
tracking_ok	If true, the robot is tracking the belt successfully; otherwise an error occured while tracking the belt

2-3 clear_queue

Clears a Robot Vision Manager queue.

Syntax

clear_queue(queue_index)

Input Parameters

Parameter	Description
queue_index	The index of the queue to clear

Output Parameters

None

2-4 getinstance

This function is the main function called to retrieve the instance.

Syntax

getinstance(queue_index, flags, location, model, encoder, visionx, visiony, visionrot)

Input Parameters

Parameter	Description
queue_index	This is the queue index in which we want to retrieve the instance
flags	Flag bits to control operation of the routine: • **Bit 1**: If set, routine waits for queue element to be defined; otherwise, "fail" if the element is not defined • **Bit 2**: If set, do not remove the instance from the queue; otherwise, remove the instance from the queue. • **Bit 3**: If set, we are sure there is an instance in the queue. When using this bit, the calling program is responsible for checking if an instance is ready in the queue; otherwise, getinstance will check if there is an instance before retrieving it from the queue.

Output Parameters

Parameter	Description
location	Location of the part found by the Robot Vision Manager. The location can be relative to the belt reference frame or the robot reference frame depending on the option selected in the communication tool on the Robot Vision Manager side.model.
model	Model Index (-1 if there was an error)
encoder	Contains the encoder value when the instance was found.
visionx	Countains the part X position in the vision coordinates reference frame
visiony	Countains the part Y position in the vision coordinates reference frame
visionrot	Countains the part Rotation in the vision coordinates reference frame

2-5 reset_seq

Reset a Robot Vision Manager sequence.

Syntax

reset_seq($myip, seq_id)

Input Parameters

Parameter	Description
$myip	String containing the IP Address of the Robot Vision Manager vision server
seq_id	Index of the sequence to reset

Output Parameters

None

2-6 set_as_exec_mod

Set the Robot Vision Manager execution mode for a given sequence.

Syntax

set_as_exec_mod($myip, seq_id, mode)

Input Parameters

Parameter	Description
$myip	String containing the IP Address of the Robot Vision Manager vision server
seq_id	Index of the sequence to reset
mode	Desired Robot Vision Manager execution. 0 = Single execution mode; otherwise: Continuous execution mode

Output Parameters

None

3

V+ End-effector Module

The information in this section documents the V+ programs that can be used to interact with the end-effector. These programs are a subset of the AceServer module.

3-1 ee.chk.er

Checks the status of the end-effector for extend or release input signals. The gripper waits until the end-effector reaches the desired state before moving.

Syntax

ee.chk.er(ee.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of operation

3-2 ee.chk.oc

Checks the status of the gripper for an open or close input signal. The robot waits until the gripper reaches the desired state.

Syntax

ee.chk.oc(tsk.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of operation

3-3 ee.clroff

Clears all refinement offsets associated with the end-effector.

Syntax

ee.clroff(ee.idx, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector

Output Parameters

Parameter	Description
sts	Status of operation

3-4 ee.gr.settle

Waits for the required settling time for the end-effector.

Syntax

ee.gr.settle(ee.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of operation

3-5 ee.gr.tip.er

Changes the extend or retract state of the gripper tip for the specified robot.

Syntax

ee.gr.tip.er(tsk.idx, tip.idx, state)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access
state	State of the gripper

Output Parameters

None

3-6 ee.presence

Check if the part presence sensor is in the desired state. The robot waits until the gripper reaches the desired state.

Syntax

ee.presence(tsk.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the end-task
tip.idx	Zero-based index of the tip to access
state	Desired state of the part presence sensors

Output Parameters

Parameter	Description
sts	Status of operation

3-7 ee.release

Turns the release signals on the end-effector on or off.

Syntax

ee.release(ee.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
tip.idx	Zero-based index of the tip to access: -1 — All tips 0...N — The tip to operate with
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of operation

3-8 ee.select

Performs a tip selection, extending or retracting tips as needed. It also checks to see if a tip selection programis associated with the current robot end-effector. If one is specified it is executed.

Abstract

ee.select(ee.idx, tip.idx)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access: -1 : All tips 0...N : The tip to operate with

Output Parameters

None

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)
```

```
; Get the offset associated with tip #2
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

3-9 ee.select.ck

Checks for any end-effector selection errors.

Syntax

ee.select.ck(ee.idx, tip.idx, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access

Output Parameters

Parameter	Description
sts	Status of operation

3-10 ee.setoff

Associates an offset with a specific end-effector tip.

Syntax

ee.setoff(ee.idx, tip.idx, tool.trans, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access
tool.trans	Gripper tip transform

Output Parameters

Parameter	Description
sts	Status of operation

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
```

```
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

3-11 ee.state

Changes the state of the end-effector tip.

Syntax

ee.state(ee.idx, tip.idx, state, check.inputs, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access
state	State of the gripper
check.inputs	Check to see if the end-effector is waiting for an input

Output Parameters

Parameter	Description
sts	Status of operation

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)
```

```
; Get the offset associated with tip #2
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

3-12 ee.tip.erdw

Performs the required extend or retract dwell of the end-effector tip.

Syntax

ee.tip.erdw(ee.idx, tip.idx)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access

Output Parameters

None

3-13 ee.tip.oc

Changes the open and closed state of the end-effector tip.

Syntax

ee.tip.oc(ee.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector
tip.idx	Zero-based index of the tip to access
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of operation

3-14 ee.unselect

Retracts all of the end-effector tips.

Syntax

ee.unselect(ee.idx)

Input Parameters

Parameter	Description
ee.idx	Index of the end-effector

Output Parameters

None

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

3-15 Error Codes

If a problem is encountered when executing an end-effector library method, the following error codes can be returned.

- -20003: The end-effector will not close.
- -20004: The end-effector will not open.
- -20010: Invalid end-effector referenced.
- -20026: The end-effector will not extend.
- -20027: The end-effector will not retract.
- -20030: A part is detected in the end-effector.
- -20031: No part detected in the end-effector.

4

V+ Process Manager Module

The information in this section documents the V+ programs that a user might use in the process of customizing a Process Manager application.

4-1 pm.ace.blt.cam

Indicate to ACE that a camera needs to be triggered.

Syntax

pm.ace.blt.cam(tsk.idx, $camera, remote, encoder)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
$camera	Name of the camera that needs to be triggered
remote	Is the camera remote or local?
encoder	Encoder reference

Output Parameters

None

Typical Use

This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use.

4-2 pm.ace.blt.ltch

Send a command to the PC that a latch has been detected.

Syntax

pm.ace.blt.ltch(blt.idx, enc, latch, value, $tag)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
$camera	Name of the camera that needs to be triggered
remote	Is the camera remote or local?
encoder	Encoder reference

Output Parameters

None

Typical Use

This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use or refer to the Process Manager documentation under the Belt Monitoring section.
Note that if the user passes in a $tag, all instances created relative to that latch event will be assigned the specified tag.

4-3 pm.ace.blt.pos

Send the current belt position to the PC.

Syntax

pm.ace.blt.pos(blt.idx, encoder, encoder.value, encoder.vel)

Input Parameters

Parameter	Description
tsk.idx	Index of the task that owns the belt being updated
encoder	Index of the encoder being updated
encoder.value	The encoder position
encoder.vel	The encoder velocity

Output Parameters

None

Typical Use

This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use.

4-4 pm.ace.blt.sp

Indicate to ACE that a part/target belt spacing event has occurred.

Syntax

pm.ace.blt.sp(blt.idx, $spacing, value, $tag)

Input Parameters

Parameter	Description
blt.idx	Index of the task that owns the belt being updated
$spacing	Name of the part/target to which the spacing is relative
value	Encoder value defining the location
$tag	Tag to associate with instances created

Output Parameters

None

Typical Use

This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use or refer to the Process Manager documentation under the Belt Monitoring section.
Note that if the user passes in a $tag, all instances created relative to that spacing event will be assigned the specified tag.
.

4-5 pm.ace.pkstate

Change the run state of a process manager. This program is called by the startup program to start/stop a process manager application.

Syntax

pm.ace.pkstate($manager, state, time.out, sts)

Input Parameters

Parameter	Description
$manager	Process manager name
state	The desired state where True = Start False = Stop
time.out	Time (in s) which rm.execute is allowed to run

Output Parameters

Parameter	Description
sts	Status of the operation

4-6 pm.ace.ref

Sends a refinement request to the PC.

Syntax

pm.ace.ref(tsk.idx, $ref.name, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task requiring refinement
$ref.name	Name of the refinement operation

Output Parameters

Parameter	Description
sts	Status of the operation

Typical Use

This program is always used in the context of a refinement program. Create a custom refinement program to see an example.

4-7 pm.ace.ref.wt

Wait for the refinement operation to complete.

Syntax

pm.ace.ref.wt(tsk.idx, time.out, loc, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
time.out	Maximum amount of time to wait for the operation

Output Parameters

Parameter	Description
sts	Status of the operation

Typical Use

This program is always used in the context of a refinement program. Create a custom refinement program to see an example.

4-8 pm.blt.convert

Convert a real world reference location to an encoder value in range between min/max.
This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use.

Syntax

pm.blt.convert(encoder)

Input Parameters

Parameter	Description
encoder	The encoder reference to convert

Output Parameters

Parameter	Description
encoder	The converted position

4-9 pm.blt.travel

Determine the distance the belt has traveled between 2 encoder values.

Syntax

pm.blt.travel(point.1, point.2, distance)

Input Parameters

Parameter	Description
point.1	The first encoder position
point.2	The second encoder position

Output Parameters

Parameter	Description
distance	The distance between the 2 positions

Typical Use

This program is always used in the context of a belt monitoring program. Create a custom belt program to see this in use.

.

4-10 pm.chk.run

Check to see if a process application is running.

Syntax

pm.chk.run(run)

Input Parameters

None

Output Parameters

Parameter	Description
run	Checks to see if a process manager application is running True - ACE and a process manager application are running False - ACE and/or the process manager application are not running.

4-11 pm.chk.stat

Determine if both the process manager application is running and the ACE server is running. Process manager tasks uses this method to check if they should continue running.

Syntax

pm.chk.stat(run)

Input Parameters

None

Output Parameters

Parameter	Description
run	True - ACE and a process manager application are running False - ACE and/or the process manager application are not running

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)
```

```
; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-12 pm.chk.tskerr

Check for error conditions and display an error if one exists on the specified task.

Syntax

pm.chk.tskerr(tsk.idx, resp.mask, code, resp)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
resp.mask	Possible responses to the error

Output Parameters

Parameter	Description
code	Detected error code, or 0 if no error detected
resp	Response to the error

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)
```

```
; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-13 pm.def.btn.txt

Define the text associated with a response button for the next error reported on a given task. This method must be called before calling pm.error.

Syntax

pm.def.btn.txt(tsk.idx, button, $text)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
button	Response code associated with the button
$text	Text associated with the button. "" will clear the text.

Output Parameters

None

4-14 pm.def.err.txt

Define a custom text message associated with an error for the next error reported on the specified task index. This method must be called before calling pm.error.

Syntax

pm.def.err.txt(tsk.idx, $text)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
$text	Text associated with the error. "" will clear the text.

Output Parameters

None

4-15 pm.fdr.state

Updates the state of a feeder on the PC.

Syntax

pm.fdr.state($name, state)

Input Parameters

Parameter	Description
$name	Name of the feeder part or target
state	State of the feeder: pm.fst.avail Is available pm.fst.busy Is not available pm.fst.cycle Is cycling pm.fst.hshk Waiting for handshake

Output Parameters

None

4-16 pm.get.free.tsk

Selects an available task that can be used. Tasks from first.tsk to last.tsk are scanned and the first available one is returned

Syntax

pm.get.free.tsk(first.tsk, last.tsk, tsk.num)

Input Parameters

Parameter	Description
first.task	The task number at which to start scanning
last.task	The task number at which we want to stop scanning

Output Parameters

Parameter	Description
task.num	The task number. If value is -1, no available task was found.

4-17 pm.get.type

Gets the friendly name for a given part / target type name

Syntax

pm.get.type(rob.idx, $type, $f.name)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
$type	The name of the type

Output Parameters

Parameter	Description
$f.name	The friendly name of the type

4-18 pm.gr.clroff

Clears all refinement offsets associated with the gripper.

Syntax

pm.gr.clroff(tsk.idx, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task

Output Parameters

Parameter	Description
sts	Status of the operation

4-19 pm.gr.release

Turn on/off the gripper release signals.

Syntax

pm.gr.release(tsk.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access -1 = All tips 0...N = The tip to operate with
state	State of the gripper

Output Parameters

Parameter	Description
sts	Status of the task

4-20 pm.gr.select

Perform a tip selection, extending or retracting tips as needed. It also checks if a tip selection program is associated with the current robot gripper. If one is specified it will be executed.

Syntax

pm.gr.select(tsk.idx, tip.idx)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access -1 = All tips 0...N = The tip to operate with

Output Parameters

None

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
```

```
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

4-21 pm.gr.select.ck

Check for any gripper selection errors.

Syntax

pm.gr.select.ck(tsk.idx, tip.idx, resp)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access

Output Parameters

Parameter	Description
resp	User response to an error

4-22 pm.gr.setoff

Associates an offset with a specific gripper tip.

Syntax

pm.gr.setoff(tsk.idx, tip.idx, tool.trans, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access
tool.trans	Gripper tip transform

Output Parameters

Parameter	Description
sts	Status of the operation

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
```

```
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

4-23 pm.gr.settle

Wait for the required settling time for the gripper.

Syntax

pm.gr.settle(tsk.idx, tip.idx, state, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access
state	Desired state of the gripper

Output Parameters

Parameter	Description
sts	Status of the operation

4-24 pm.gr.state

Change the state of the gripper tip for the specified robot.

Syntax

pm.gr.state(tsk.idx, tip.idx, state, check.inputs, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access
state	Desired state of the gripper
check.inputs	Do we wait for inputs?

Output Parameters

Parameter	Description
sts	Status of the operation

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)
```

```
; Get the offset associated with tip #2
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

4-25 pm.gr.trans

Associates an offset with a specific gripper tip.

Syntax

pm.gr.trans(tsk.idx, tip.idx, tool.trans, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
tip.idx	Zero-based index of the tip to access

Output Parameters

Parameter	Description
tool.trans	The gripper tip transformation
sts	Status of the operation

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
```

```
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

4-26 pm.gr.unselect

Retract all the tips.

Syntax

pm.gr.unselect(tsk.idx)

Input Parameters

Parameter	Description
tsk.idx	Index of the task

Output Parameters

None

Example

```
.PROGRAM a.ex2.gripper(ee.idx)
;
AUTO REAL sts
AUTO LOC tool.trans

; Select the first tip
CALL ee.select(ee.idx, 0)

; Select all tips
CALL ee.select(ee.idx, -1)

; Unselect all tips
CALL ee.unselect(ee.idx)

; Extend tip #1 and #2
CALL ee.state(ee.idx, 0, TRUE, TRUE, sts)
CALL ee.state(ee.idx, 1, TRUE, TRUE, sts)

; Set the offset associated with tip #2
CALL ee.setoff(ee.idx, 1, NULL, sts)

; Get the offset associated with tip #2
CALL ee.trans(ee.idx, 1, tool.trans, sts)

.END
```

4-27 pm.log

Send text associated with the current task to the AceServer log file.

Syntax

pm.log($text)

Input Parameters

Parameter	Description
$text	The text to append to the AceServer log on the PC

Output Parameters

None

Additional Notes

The **pm.log** method should only be called on a process manager task. The program will check to see if logging is enabled before sending the log string to the PC. The variable **pm.log** will enable logging on all V+ process manager tasks. If you want to enable logging for an individual V+ task, the variable **pm.tsk.log [TASK()]** can be set to **TRUE**.

4-28 pm.mv

Perform a move to the location using the specified motion parameters.

Syntax

pm.mv(blt.idx, position, offset, pars[])

Input Parameters

Parameter	Description
blt.idx	Belt number to move relative to. • 0 = Non-belt relative move • > 0 = Belt relative move
position	Position to move to
offset	Starting index in the motion parameters array
pars[]	Motion parameters array

Output Parameters

None

Additional Notes

The **pars[]** parameter uses the following offset values when applying the motion parameters:

Variable	Usage
pm.fv.spd	The speed percentage
pm.fmv.spdmd	The Speed Mode
pm.fmv.accel	The accelaration percentage
pm.fmv.decel	The deceleration percentage
pm.fmv.dur	The move duration
pm.fmv.scv	The S-curve profile
pm.fmv.straigh	Is straight line motion requested?
pm.fmv.motend	Motioin end parameter
pm.fmv.stlpct	The settle percentage
pm.fmv.break	Should it wait until motion is done?
pm.fmv.single	Single/multiple mode
pm.fmv.righty	Right/Lefty mode
pm.fmv.above	Above/below mode
pm.fmv.flip	Flip/No-flip

For a standard motion sequence, the following values are used for the offset parameter.

Variable	Usage
pm.ms.aoffset	Approach motion segment
pm.ms.poffset	Move motion segment

pm.ms.doffset	Depart motion segment

For a standard vision refinement sequence, the following values are used for the offset parameter.

Variable	Usage
pm.vrf.aoffset	Approch motion segment
pm.vrf.poffset	Move motion segment
pm.vrf.doffset	Depart motion segment

4-29 pm.mv.attach

Attach the robot to the current task.

Syntax

pm.mv.attach(tsk.idx, rob.num)

Output Parameters

None

Input Parameters

Parameter	Description
tsk.idx	Index of the task
rob.num	Robot number to attach

Additional Notes

For a robot task, the global variable **pm.tsk.robnum[]** saves the robot number associated with the task.

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO
```

```
; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-30 pm.mv.chkbrk

Check for a break in motion using the specified motion parameters.

Syntax

pm.mv.chkbrk(offset, pars[])

Input Parameters

Parameter	Description
offset	Starting index in the motion parameters array
pars[]	Motion parameters array

Output Parameters

None

4-31 pm.mv.dest

Move to the current destination. Used to stop robot from tracking a part.

Syntax

pm.mv.dest(rob.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot

Output Parameters

None

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)
```

```
; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-32 pm.mv.idle

Move a robot to the idle location. The robot first retracts the Z axis (maintaining the current XY position) to the level of the idle position. Once retracted it moves to idle.

Syntax

pm.mv.idle(rob.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot

Output Parameters

None

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
```

```
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-33 pm.proc.enable

Enable or disable a process based on the process index.

Syntax

pm.proc.enable($manager, proc.idx, state, sts)

Input Parameters

Parameter	Description
$manager	Name of the process manager
proc.idx	Index of the process
state	State of the process • TRUE = Enable the process • FALSE = Disable the process

Output Parameters

Parameter	Description
sts	Status of the operation: • 0 = successs • <0 = error

4-34 pm.prt.avail

Gets the number of available parts for the robot.

Syntax

pm.prt.avail(rob.idx, part.idx, count)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
part.idx	Part index to access

Output Parameters

Parameter	Description
count	Number of items available in the queue

Example

```
.PROGRAM a.ex2.qlookup()

AUTO REAL rob.idx, part.idx, target.idx, count
AUTO $part, $target

; Identify the first robot task
rob.idx = 3

; Look up the queue number for a part in the workspace
$part = "/Process/Part - Spacing 2"
CALL pm.prt.get.idx(rob.idx, $part, part.idx)

Get the number of parts available
CALL pm.prt.avail(rob.idx, part.idx, count)

; Look up the queue number for a target in the workspace
$target = "/Process/Target 2 - Static"
CALL pm.trg.get.idx(rob.idx, $target, target.idx)

; Get the number of targets available
CALL pm.trg.avail(rob.idx, target.idx, count)
.END
```

4-35 pm.prt.done

Indicate a part has been processed.

Syntax

pm.prt.done(rob.idx, part.idx, obj.idx, sts)

Input Parameters

Parameter	Description
rob.idx	Index of the robot
part.idx	Part index to access
obj.idx	Index of the instance/object to mark as done
sts	Was the part processed by the robot: • TRUE = part was processed • FALSE = part was not processed

Output Parameters

None

4-36 pm.prt.get

Gets a part from the buffer.

Syntax

pm.prt.get(rob.idx, part.idx, obj.idx, $id, position, reference, pal.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot
part.idx	Part index to access
obj.idx	Index of the object/instance to extract

Output Parameters

Parameter	Description
$id	Unique identifier of the part
position	Position of the part
reference	Reference position of the part
pal.idx	Pallet index associated with the instance

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
```

```
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

4-37 pm.prt.get.idx

Get the index of the part buffer for a given robot task.

Syntax

pm.prt.get.idx(rob.idx, $part, part.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot
$part	Part name to search for

Output Parameters

Parameter	Description
part.idx	Part index in the robot part buffer. -1 = No matching part was found

Example

```
.PROGRAM a.ex2.qlookup()

AUTO REAL rob.idx, part.idx, target.idx, count
AUTO $part, $target

; Identify the first robot task
rob.idx = 3

; Look up the queue number for a part in the workspace
$part = "/Process/Part - Spacing 2"
CALL pm.prt.get.idx(rob.idx, $part, part.idx)

Get the number of parts available
CALL pm.prt.avail(rob.idx, part.idx, count)

; Look up the queue number for a target in the workspace
$target = "/Process/Target 2 - Static"
CALL pm.trg.get.idx(rob.idx, $target, target.idx)

; Get the number of targets available
CALL pm.trg.avail(rob.idx, target.idx, count)
.END
```

4-38 pm.prt.getloc

Get the location of a given part in the belt window.

Syntax

pm.prt.getloc(rob.idx, part.idx, obj.idx, pct)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
part.idx	Part index to access
obj.idx	Index of the object/instance to access

Output Parameters

Parameter	Description
pct	Percentage of travel along the belt

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
```

```
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

4-39 pm.prt.move.ptr

Increment the pointer to the next slot in the buffer.

Syntax

pm.prt.move.ptr(ptr)

Input Parameters

Parameter	Description
ptr	The pointer to increment

Output Parameters

Parameter	Description
ptr	The incremented pointer

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
```

```
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

4-40 pm.ps.error

Report an error and wait for a response.

Syntax

pm.ps.error(tsk.idx, code, resp.mask, resp)

Input Parameters

Parameter	Description
tsk.idx	Index of the task
code	Error code/status of the operation
resp.mask	Mask of available response options

Output Parameters

Parameter	Description
resp	The user response to the error

4-41 pm.ps.map.idx

Converts a process index from the PC to a robot-relative index.

Syntax

pm.ps.map.idx(proc.idx, map.idx)

Input Parameters

Parameter	Description
proc.idx	Process index from the PC

Output Parameters

Parameter	Description
map.idx	Robot relative index for the process

Example

```
.PROGRAM cu.robot2()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL task.idx, i, sts
AUTO REAL proc.idx.1, proc.idx.2

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE)

; Get the index associated with the first two processes in the process manager
CALL pm.ps.map.idx(0, proc.idx.1)
CALL pm.ps.map.idx(1, proc.idx.2)

; Control loop
DO

; Execute the first process 3 time
FOR i = 1 TO 3
CALL pm.rob.process(tsk.idx, proc.idx.1, sts)
END
```

```
; Execute the second process
CALL pm.rob.process(tsk.idx, proc.idx.2, sts)

; Check process status
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

RETURN
.END
```

4-42 pm.ps.power

Change the power status of the system. If power is enabled, issue a calibration request, if needed. If power cannot be enabled, it will report an error. Because of this, this method is not intended to be used from within a custom error program. It may lead to a recursive call and a stack overflow.

Syntax

pm.ps.power(tsk.idx, enable)

Input Parameters

Parameter	Description
tsk.idx	The task index for error reporting
enable	Enable the power.

Output Parameters

None

4-43 pm.rob.chkidle

Check the status when the robot is idle, including checking for the robot cycle stop condition.

Syntax

pm.rob.chkidle(rob.idx, part.type, grip.idx, excl[], is.reset, sts)

Input Parameters

Parameter	Description
tsk.idx	Index of the robot task
start.time	The reference time when the robot entered idle mode
in.idle	Is the robot in idle mode
at.wait	Is the robot at the waiting position

Output Parameters

Parameter	Description
in.idle	Is the robot in idle mode
at.wait	Is the robot at the waiting position

4-44 pm.rob.clear

Move the robot to the idle position and clear the robot gripper.

Syntax

pm.rob.clear(rob.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot

Output Parameters

None

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)
```

```
; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-45 pm.rob.gettag

Returns the tag associated with the part or target instance currently being operated on by the robot. Typically, this would be called from within a custom motion sequence program to access information that was associated with a VisionTransform Tag associated with a custom vision tool or the Tag associated with a LocatedInstance set by a custom allocation script.

Syntax

pm.rob.gettag(rob.idx, $tag)

Input Parameters

Parameter	Description
tsk.idx	Index of the robot

Output Parameters

Parameter	Description
$tag	Tag associated with the instance being processed

Example

```
.PROGRAM cu.mv.sequence(tsk.idx, $type, blt.idx, reference, pal.idx, grip.idx, grip
.state, pos, vals[], sts)

AUTO REAL at.wait, belt.win.idx, code, distance, offset
AUTO REAL resp, rob.num
AUTO $cust.program, $tag
AUTO LOC appro.pos, depart.pos, grip.trans, position

sts = pm.tsk.success

CALL pm.rob.gettag(tsk.idx, $tag)
TYPE "Tag = "+$tag

; Calculate the position, factoring in the gripper transformations

; Initiate the gripper selection
CALL pm.gr.select(tsk.idx, grip.idx)

; Get the gripper transformation
CALL pm.gr.trans(tsk.idx, grip.idx, grip.trans, sts)
```

4-46 pm.rob.idlemd

Change the idle status of the robot. This program sets the status in global memory. It expects the process strategy background program to perform the sending of the status to the PC.

Syntax

pm.rob.idlemd(rob.tsk, in.idle, code, $info)

Input Parameters

Parameter	Description
rob.tsk	Index of the robot
in.idle	Is the robot in idle mode? • True = Robot is idle • False = Robot is running
code	Error code associated with the idle mode state. If non-zero, an error will be reported
$info	Additional information to send with the error code. This is used by 'pm.err.no.inst' to send the name of the part or target the robot is waiting for.

Output Parameters

None

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)
```

```
; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-47 pm.rob.init

Initialize the robot by attaching the robot, clearing the TOOL, moving to the idle position, and turning off all gripper signals.

Syntax

pm.rob.init(rob.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot

Output Parameters

None

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
```

```
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-48 pm.rob.monque

Process the active part and targets queues and mark any out-of-range instances as done. This method should only be called on a robot task when the robot is waiting for some operation to complete. This ensures any instances that go out of range of the robot are properly released and sent back to the PC.
If the robot is currently processing a part or target, the index of the associated queue should be passed as a parameter to the method.

Syntax

pm.rob.monque(rob.tsk, part.idx, trg.idx)

Input Parameters

Parameter	Description
rob.tsk	Index of the robot
part.idx	Index of a part queue to not check. -1 indicates all queues will be checked
trg.idx	Index of a target queue to not check. -1 indicates all queues will be checked

Output Parameters

None

Example

```
.CALL pm.rob.monque(robtsk, -1 -1)
```

4-49 pm.rob.pick

Perform a single pick operation with the robot in the current task.

Syntax

pm.rob.pick(rob.idx, part.type, grip.idx, excl[], is.reset, sts)

Input Parameters

Parameter	Description
rob.tsk	Index of the robot
part.idx	Type of part to pick
grip.idx	The index of the gripper tip to use -1 = All tips
ms.type	The move segment type: • pm.ms.mv.norm - Normal Move • pm.ms.mv.iad - Pick / place motion with intra approach and intra depart • pm.ms.mv.iand - Pick / place with intra approach and normal depart • pm.ms.mv.naid - Pick / place with normal approach and intra depart
excl[]	The exclusions used when picking

Output Parameters

Parameter	Description
is.reset	Was the queue reset during the operation
sts	Status code: • pm.tsk.success = Successful operation • pm.tsk.abort = Abort the operation

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
```

```
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-50 pm.rob.place

Perform a single place operation with the robot in the current task.

Syntax

pm.rob.place(rob.idx, target.type, part.type, grip.idx, ms.type, excl[], is.reset, sts)

Input Parameters

Parameter	Description
rob.tsk	Index of the robot
target.type	Type of target to place
part.type	The type of part being placed
grip.idx	The index of the gripper tip to use -1 = All tips
ms.type	The move segment type: • pm.ms.mv.norm - Normal Move • pm.ms.mv.iad - Pick / place motion with intra approach and intra depart • pm.ms.mv.iand - Pick / place with intra approach and normal depart • pm.ms.mv.naid - Pick / place with normal approach and intra depart
excl[]	The exclusions used when picking

Output Parameters

Parameter	Description
is.reset	Was the queue reset during the operation
sts	Status code: • pm.tsk.success = Successful operation • pm.tsk.abort = Abort the operation

Example

```
.PROGRAM cu.robot()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL tsk.idx, is.reset, , is.running, sts
AUTO REAL part.idx, target.idx

tsk.idx = TASK()

; Initialize the robot
```

```
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Get the part and target indexes
CALL pm.prt.get.idx(tsk.idx, "/Process/Part - Spacing 2", part.idx)
CALL pm.trg.get.idx(tsk.idx, "/Process/Target 2 - Static", target.idx)

; Control loop
DO

; Perform a pick
CALL pm.rob.pick(tsk.idx, part.idx, -1, , is.reset, sts)

; Perform a place
CALL pm.rob.place(tsk.idx, target.idx, part.idx, -1, , is.reset, sts)

; Indicate we are entering idle mode
CALL pm.rob.idlemd(tsk.idx, TRUE, 0, "")

; Move to the current destination to ensure
; we are no longer tracking a belt.
CALL pm.mv.dest(tsk.idx)

; Wait for a period of time
WAIT.EVENT , 3

; Check to see if any errors occured while we were delaying
CALL pm.chk.tskerr(tsk.idx, pm.tsk.retry, , sts)
IF (sts <> pm.tsk.success) THEN
CALL pm.mv.attach(tsk.idx, pm.tsk.robnum[tsk.idx])
END

; Indicate we are leaving idle mode
CALL pm.rob.idlemd(tsk.idx, FALSE, 0, "")

; Check to see if the process manager is still running
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

CALL pm.rob.clear(tsk.idx, sts)
RETURN
.END
```

4-51 pm.rob.process

Converts a process index from the PC to a robot-relative index.

Syntax

pm.rob.process(rob.idx, proc.idx, sts)

Input Parameters

Parameter	Description
rob.idx	Index of the robot to drive
proc.idx	Process to operate with

Output Parameters

Parameter	Description
sts	Status code: • pm.tsk.success = Successful operation • pm.tsk.skip = Skip the operation • pm.tsk.abort = Abort the operation

Example

```
.PROGRAM cu.robot2()
; ABSTRACT: Custom robot program for the Process Manager
; Process Strategy.

AUTO REAL task.idx, i, sts
AUTO REAL proc.idx.1, proc.idx.2

tsk.idx = TASK()

; Initialize the robot
CALL pm.rob.init(tsk.idx)
CALL pm.rob.idlemd(tsk.idx, TRUE)

; Get the index associated with the first two processes in the process manager
CALL pm.ps.map.idx(0, proc.idx.1)
CALL pm.ps.map.idx(1, proc.idx.2)

; Control loop
DO

; Execute the first process 3 time
```

```
FOR i = 1 TO 3
CALL pm.rob.process(tsk.idx, proc.idx.1, sts)
END

; Execute the second process
CALL pm.rob.process(tsk.idx, proc.idx.2, sts)

; Check process status
CALL pm.chk.stat(is.running)

UNTIL is.running == FALSE

RETURN
.END
```

4-52 pm.rob.refine

Refine the part in the robot gripper using vision.

Syntax

pm.rob.refine(rob.idx, refine.idx, grip.idx, sts)

Input Parameters

Parameter	Description
rob.tsk	Index of the robot
refine.idx	Index of the refinement station
grip.idx	The index of the gripper tip to use. -1 = All tips
excl[]	The exclusions used when picking

Output Parameters

Parameter	Description
sts	Status code: • pm.tsk.success = Successful operation • pm.tsk.retry = Retry the operation • pm.tsk.abort = Abort the operation

4-53 pm.rob.settag

Sets the tag associated with the part or target instance currently being operated on by the robot. Typically, this would be called from within a custom motion sequence program to set a string for the Tag associated with a LocatedInstance set by a custom allocation script.

Syntax

pm.rob.settag(tsk.idx, $tag)

Input Parameters

Parameter	Description
tsk.idx	Index of the robot
$tag	Tag associated with the instance

Output Parameters

None

Example

```
.PROGRAM cu.mv.sequence(tsk.idx, $type, blt.idx, reference, pal.idx, grip.idx, grip
.state, pos, vals[], sts)

AUTO REAL at.wait, belt.win.idx, code, distance, offset
AUTO REAL resp, rob.num
AUTO $cust.program, $tag
AUTO LOC appro.pos, depart.pos, grip.trans, position

sts = pm.tsk.success

CALL pm.rob.gettag(tsk.idx, $tag)
TYPE "Tag = "+$tag

; Calculate the position, factoring in the gripper transformations

; Initiate the gripper selection
CALL pm.gr.select(tsk.idx, grip.idx)

; Get the gripper transformation
CALL pm.gr.trans(tsk.idx, grip.idx, grip.trans, sts)
```

4-54 pm.sig.check

Checks for the status of a digital signal ensuring the state is valid for the specified amount of time. This method does not block execution.

Syntax

pm.sig.check(sig, time, is.valid)

Input Parameters

Parameter	Description
sig	Signal to be checked
time	Amount of time the signal is expected to be in the state

Output Parameters

Parameter	Description
is.valid	Is the signal in the specified state

Example

```
.PROGRAM a.ex2.debounce()
AUTO REAL sig, is.on, debounce.time
; Define the signal and the amount of time we want the signal
; to remain on before we recognize it is on
sig = 2100
debounce.time = 2.5
is.on = FALSE
; Clear the tracking structures for the signal
CALL pm.sig.clear(sig)
; Trun the signal on
SIGNAL sig
TYPE $TIME()
; Wait until the signal has been on for the required amount of time
WHILE (is.on == FALSE) DO
CALL pm.sig.check(sig, debounce.time, is.on)
END
TYPE $TIME()
SIGNAL -sig
.END
```

4-55 pm.sig.clear

Clears the variables used to track the status of a signal.

Syntax

pm.sig.clear(sig)

Input Parameters

Parameter	Description
sig	Signal to clear

Output Parameters

None

Example

```
.PROGRAM a.ex2.debounce()
AUTO REAL sig, is.on, debounce.time
; Define the signal and the amount of time we want the signal
; to remain on before we recognize it is on
sig = 2100
debounce.time = 2.5
is.on = FALSE
; Clear the tracking structures for the signal
CALL pm.sig.clear(sig)
; Trun the signal on
SIGNAL sig
TYPE $TIME()
; Wait until the signal has been on for the required amount of time
WHILE (is.on == FALSE) DO
CALL pm.sig.check(sig, debounce.time, is.on)
END
TYPE $TIME()
SIGNAL -sig
.END
```

4-56 pm.src.clear

Clear the specified source

Syntax

pm.src.clear($manager, $source, sts)

Input Parameters

Parameter	Description
$manager	Name of the process manager
$source	The name of the source to clear

Output Parameters

Parameter	Description
sts	Status of operation

Example

```
.PROGRAM a.ex2.srcclr()

AUTO $manager
AUTO REAL sts

$manager = "/Process/Process Manager"
$source = "Belt Handler: /Process/Belt"

; Clear the source with the specified name
CALL pm.src.clear($manager, $source, sts)

; Clear all source
CALL pm.src.clearall($manager, sts)
.END
```

4-57 pm.src.clearall

Clears all sources for the specified process manager

Syntax

pm.src.clearall($manager, sts)

Input Parameters

Parameter	Description
$source	The name of the source to clear

Output Parameters

Parameter	Description
sts	Status of operation

Example

```
.PROGRAM a.ex2.srcclr()

AUTO $manager
AUTO REAL sts

$manager = "/Process/Process Manager"
$source = "Belt Handler: /Process/Belt"

; Clear the source with the specified name
CALL pm.src.clear($manager, $source, sts)

; Clear all source
CALL pm.src.clearall($manager, sts)
.END
```

4-58 pm.trg.avail

Gets the number of available targets for the robot.

Syntax

pm.trg.avail(rob.idx, trg.idx, count)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
trg.idx	Target index to access

Output Parameters

Parameter	Description
count	Number of items available in the queue

Example

```
.PROGRAM a.ex2.qlookup()

AUTO REAL rob.idx, part.idx, target.idx, count
AUTO $part, $target

; Identify the first robot task
rob.idx = 3

; Look up the queue number for a part in the workspace
$part = "/Process/Part - Spacing 2"
CALL pm.prt.get.idx(rob.idx, $part, part.idx)

Get the number of parts available
CALL pm.prt.avail(rob.idx, part.idx, count)

; Look up the queue number for a target in the workspace
$target = "/Process/Target 2 - Static"
CALL pm.trg.get.idx(rob.idx, $target, target.idx)

; Get the number of targets available
CALL pm.trg.avail(rob.idx, target.idx, count)
.END
```

4-59 pm.trg.done

Indicate a target has been processed.

Syntax

pm.trg.done(rob.idx, trg.idx, obj.idx, sts)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
trg.idx	Target index to access
obj.idx	Index of the instance/object to mark as done
sts	Was the instance processed by the robot: • TRUE = part was processed • FALSE = part was not processed

Output Parameters

None

4-60 pm.trg.get

Gets a target from the buffer.

Syntax

pm.trg.get(rob.idx, trg.idx, obj.idx, $id, position, reference, pal.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
trg.idx	Target index to access
obj.idx	Index of the object/instance to extract

Output Parameters

Parameter	Description
$id	Unique identifier of the instance
position	Position of the instance
reference	Reference position of the instance
pal.idx	Pallet index associated with the instance

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
```

```
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

4-61 pm.trg.get.idx

Get the index of the target buffer for a given robot task.

Syntax

pm.trg.get.idx(rob.idx, $target, target.idx)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
$target	Target name to search for

Output Parameters

Parameter	Description
target.idx	Target index in the robot target buffer. -1 = No matching target was found

Example

```
.PROGRAM a.ex2.qlookup()

AUTO REAL rob.idx, part.idx, target.idx, count
AUTO $part, $target

; Identify the first robot task
rob.idx = 3

; Look up the queue number for a part in the workspace
$part = "/Process/Part - Spacing 2"
CALL pm.prt.get.idx(rob.idx, $part, part.idx)

Get the number of parts available
CALL pm.prt.avail(rob.idx, part.idx, count)

; Look up the queue number for a target in the workspace
$target = "/Process/Target 2 - Static"
CALL pm.trg.get.idx(rob.idx, $target, target.idx)

; Get the number of targets available
CALL pm.trg.avail(rob.idx, target.idx, count)
.END
```

4-62 pm.trg.getloc

Get the location of a given target in the belt window.

Syntax

pm.trg.getloc(rob.idx, trg.idx, obj.idx, pct)

Input Parameters

Parameter	Description
rob.idx	Index of the robot task
trg.idx	Target index to access
obj.idx	Index of the object/instance to access

Output Parameters

Parameter	Description
pct	Percentage of travel along the belt

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
```

```
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

4-63 pm.trg.move.ptr

Increment the pointer to the next slot in the buffer.

Syntax

pm.trg.move.ptr(ptr)

Input Parameters

Parameter	Description
ptr	The pointer to increment

Output Parameters

Parameter	Description
ptr	The incremented pointer

Example

```
.PROGRAM a.ex2.queues()
;
AUTO REAL i, rob.idx, inst.idx
AUTO REAL reference, pal.idx, sts, time, pct
AUTO LOC position
AUTO.$id, $source

; identify the first robot task
rob.idx = 3

; Go through all part queues
inst.idx = pm.prt.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.prt.pc.idx[rob.idx,i]) DO

CALL pm.prt.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.prt.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.prt.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
```

```
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.prt.move.ptr(inst.idx)
END
END

; Go through all target queues
FOR i = 0 TO pm.trg.count[rob.idx]-1

TYPE "Target Queue Name: ", $pm.trg.type[rob.idx,i]

; Go through all instances in the queue
inst.idx = pm.trg.rob.idx[rob.idx,i]
WHILE (inst.idx <> pm.trg.pc.idx[rob.idx,i]) DO

CALL pm.trg.get(rob.idx, i, inst.idx, $id, position, reference, pal.idx)
CALL pm.trg.get.stat(rob.idx, i, inst.idx, sts, time, $source)
CALL pm.trg.getloc(rob.idx, i, inst.idx, pct)

TYPE " ID = ", $id
TYPE " Position = ", DX(position), ", ", DY(position), ", ", DZ(position)
TYPE " Reference = ", reference
TYPE " Pallet Index = ", pal.idx
TYPE " Status = ", sts
TYPE " Time = ", time
TYPE " Source = ", $source
TYPE " Location % = ", pct
TYPE " "

CALL pm.trg.move.ptr(inst.idx)

END
END
.END
```

5

V+ Remote Library Module

The information in this section documents the V+ programs that can be used to access functionality on the PC. These programs are a subset of the AceServer module.

5-1 Error Codes

If a problem is encountered when executing a remote library method, the following error codes can be returned.

- -1000: Unable to find the AceObject referenced in the remote command.
- -1001: A general error has occurred while processing the remote command. See the AceServer event log for specific details.
- -1002: The method referenced in a remote library execute command could not be found.
- -1003: A general error has occurred while processing a remote execute invocation or property access. See the AceServer event log for specific details.
- 10000: Server has stopped executing and the connection to the remote PC has been lost.
- 10001: The request has timed out.
- 10002: The referenced object does not exist.
- 10003: The field referenced in the remote object does not exist.
- 10004" The type of the referenced field is not compatible with the requested operation.

5-2 rm.app.event

Generates an application event on the PC associated with this controller.

Syntax

rm.app.event(value, wait.time, status)

Input Parameters

Parameter	Description
value	Numeric code representing the event.
wait.time	The amount of time to wait for the operation to complete.

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.appevt()
AUTO REAL status
CALL rm.app.event(101, 1, status)
IF (status < 0) THEN
TYPE "Unable To Send Event Code: ", status
PAUSE
END
.END
```

5-3 rm.chk.server

Check to see if the ACE server is running

Syntax

rm.chk.server(is.alive)

Input Parameters

None

Output Parameters

Parameter	Description
is.alive	Is the server alive

Example

```
.PROGRAM ex1.rm.ex()

AUTO REAL is.alive
AUTO REAL args[0], status
AUTO $object, $method

; Check to see if communications with the PC is active
CALL rm.chk.server(is.alive)
IF (is.alive == FALSE) THEN
TYPE "Not Communicating"
PAUSE
END

; Execute a script on the server and wait for 3 seconds for it to complete
$object = "/C# Program"
$method = "Execute"
CALL rm.execute($object, $method, 0, $args[], 3, status)
IF (status < 0) THEN
TYPE "Problem executing script: ", status
PAUSE
END

.END
```

5-4 rm.execute

Execute a method on the PC and optionally wait until it has completed. The number of arguments must match the arguments defined by the method signature of the specified method. Additionally, the arguments must be convertable to the type expected in the method signature using a standard Type-Converter on the PC.

Syntax

rm.execute($object, $method, num.args, $args[], wait.time, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$method	The name of the method to execute
num.args	The number of arguments to send
$args[]	The arguments to pass
wait.time	The amount of time to wait for the execution to complete. Units are in seconds. A value less than or equal to 0 indicates no waiting.

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.ex()

AUTO REAL is.alive
AUTO REAL args[0], status
AUTO $object, $method

; Check to see if communications with the PC is active
CALL rm.chk.server(is.alive)
IF (is.alive == FALSE) THEN
TYPE "Not Communicating"
PAUSE
END

; Execute a script on the server and wait for 3 seconds for it to complete
```

```
$object = "/C# Program"
$method = "Execute"
CALL rm.execute($object, $method, 0, $args[], 3, status)
IF (status < 0) THEN
TYPE "Problem executing script: ", status
PAUSE
END

.END
```

5-5 rm.execute3

Execute a method on the PC and optionally wait until it has completed. The number of arguments must match the arguments defined by the method signature of the specified method. Additionally, the arguments must be convertable to the type expected in the method signature using a standard Type-Converter on the PC.

Syntax

rm.execute3($ip, $object, $method, num.args, $args[], wait.time, status)

Input Parameters

Parameter	Description
$ip	The IP address of the PC to send the request to
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$method	The name of the method to execute
num.args	The number of arguments to send
$args[]	The arguments to pass
wait.time	The amount of time to wait for the execution to complete. Units are in seconds. A value less than or equal to 0 indicates no waiting.

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.ex()

AUTO REAL is.alive
AUTO REAL args[0], status
AUTO $object, $method

; Check to see if communications with the PC is active
CALL rm.chk.server(is.alive)
IF (is.alive == FALSE) THEN
TYPE "Not Communicating"
PAUSE
END
```

```
; Execute a script on the server and wait for 3 seconds for it to complete
$object = "/C# Program"
$method = "Execute"
CALL rm.execute($object, $method, 0, $args[], 3, status)
IF (status < 0) THEN
TYPE "Problem executing script: ", status
PAUSE
END

.END
```

5-6 rm.find_ip

Finds the client_handle associated with a specific IP address

Syntax

rm.find_ip($ip, client_handle)

Input Parameters

Parameter	Description
$ip	The IP address to find

Output Parameters

Parameter	Description
client_handle	The client handle associated with the IP address or -1 if not found

5-7 rm.pc.fapp

Append a single line to a file on the PC

Syntax

rm.pc.fapp($file, $line, wait.time, status)

Input Parameters

Parameter	Description
$file	The name of the file to append to on the PC
$line	The line to append to the file
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END

; Append multiple lines in 1 call to the same file
$lines[0] = "Line 1"
$lines[1] = "Line 2"
```

```
$lines[2] = "Line 3"
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-8 rm.pc.fapps

Append an array of lines to a file on the PC

Syntax

rm.pc.fapps($file, num.lines, $lines[], wait.time, status)

Input Parameters

Parameter	Description
$file	The name of the file to append to on the PC
num.lines	The number of lines in the array to copy
$lines[]	The lines to append to the file, starting at index 0
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END

; Append multiple lines in 1 call to the same file
$lines[0] = "Line 1"
```

```
$lines[1] = "Line 2"
$lines[2] = "Line 3"
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-9 rm.pc.fapps2

Append an array of lines to a file on the PC

Syntax

rm.pc.fapps2($file, start.idx, num.lines, $lines[], wait.time, status)

Input Parameters

Parameter	Description
$file	The name of the file to append to on the PC
start.idx	The starting index in the array
num.lines	The number of lines in the array to copy
$lines[]	The lines to append to the file, starting at the start.index
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END

; Append multiple lines in 1 call to the same file
```

```
$lines[0] = "Line 1"
$lines[1] = "Line 2"
$lines[2] = "Line 3"
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-10 rm.pc.fcopy

Copy a file between the controller and the PC.
When specifying the file, one of the files must start with the prefix of 'DISK>'. This is used to determine the direction of the file copy. If neither of the files starts with 'DISK', then it is considered a file copy from 1 PC file to another entirely on the PC.

Syntax

rm.pc.fcopy($from.file, $to.file, wait.time, status)

Input Parameters

Parameter	Description
$from.file	The file to copy from
$to.file	The file to copy to
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END
```

```
; Append multiple lines in 1 call to the same file
$lines[0] = "Line 1"
$lines[1] = "Line 2"
$lines[2] = "Line 3"
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-11 rm.pc.fdel

Delete a file on the PC

Syntax

rm.pc.fdel($file, wait.time, status)

Input Parameters

Parameter	Description
$file	The name of the file to delete on the PC
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END

; Append multiple lines in 1 call to the same file
$lines[0] = "Line 1"
$lines[1] = "Line 2"
$lines[2] = "Line 3"
```

```
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-12 rm.pc.fdir

Perform a file and directory listing on the PC

Syntax

rm.pc.fdir($file.spec, wait.time, num.items, $items[], types[], status)

Input Parameters

Parameter	Description
$file.spec	The file specification to match
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
num.items	The number of items returned
$items[]	The listing of files and directories found
types[]	The types of the items found: 0 = File, 1 = Directory
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.pc()

AUTO REAL i, num.items, status
AUTO $file, $to.file, $lines[3]
AUTO $file.spec
LOCAL REAL types[]
LOCAL $items[]

$file = "C:\log.txt"

; Write 3 lines in a file on the PC
FOR i = 1 TO 3
CALL rm.pc.fapp($file, $TIME(), 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END
END

; Append multiple lines in 1 call to the same file
```

```
$lines[0] = "Line 1"
$lines[1] = "Line 2"
$lines[2] = "Line 3"
$lines[3] = "Line 4"
CALL rm.pc.fapps($file, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Append multiple lines in 1 call specifying the start index
CALL rm.pc.fapps2($file, 0, 4, $lines[], 1, status)
IF (status < 0) THEN
TYPE "Failure to append line on file: ", status
END

; Copy the file from the PC to the controller
$to.file = "DISK>D:\log.txt"
CALL rm.pc.fcopy($file, $to.file, 3, status)
IF (status < 0) THEN
TYPE "Unable to copy the file to the controller: ", status
END

; List all files on the PC
$file.spec = "C:\*.txt"
CALL rm.pc.fdir($file.spec, 3, num.items, $items[], types[], status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END

FOR i = 1 TO num.items

TYPE $items[i-1]

IF (types[i-1] == 0) THEN
TYPE " Is a file"
ELSE
TYPE " Is a directory"
END
END

; Delete the file on the PC
CALL rm.pc.fdel($file, 3, status)
IF (status < 0) THEN
TYPE "Unable to delete the file from the PC: ", status
END
.END
```

5-13 rm.pc.log

Append text to the AceServer log on the PC

Syntax

rm.pc.log($source, $text[], length, wait.time, status)

Input Parameters

Parameter	Description
$source	The source generating the log message
$text	The text to append to the log
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.log()

AUTO REAL status

CALL rm.pc.log("Robot 1", "Message To Log", 1, status)
IF (status < 0) THEN
TYPE "Failure to append to AceServer log: ", status
END
.END
```

5-14 rm.read.anums

Read a list of numeric property variables associated with multiple objects on the PC. The variable referenced must all be numeric types

Syntax

rm.read.anums($objects[], $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$objects[]	The name of the remote objects to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The name of the variables to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
values[]	The values of the variables
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.ns()

AUTO $object, $objects[1], $variables[1]
AUTO REAL values[1], status

; Read multiple values from multiple properties of the same object
$object = "/Numeric Variable"
$variables[0] = "Current Value"
$variables[1] = "IsBoolean"
CALL rm.read.nums($object, $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Read Value: ", status
PAUSE
END

TYPE $variables[0], " = ", values[0]
TYPE $variables[1], " = ", values[1]
```

```
; Read multiple values from multiple objects
$objects[0] = "/Numeric Variable"
$variables[0] = "Current Value"
$objects[1] = "/Numeric Variable 2"
$variables[1] = "Current Value"
CALL rm.read.anums($objects[], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Read Value: ", status
PAUSE
END

TYPE $objects[0], ".", $variables[0], " = ", values[0]
TYPE $objects[0], ".", $variables[1], " = ", values[1]
.END
```

5-15 rm.read.num

Read a numeric property variable associated with an object on the PC. The variable referenced must be a numeric type

Syntax

rm.read.num($object, $variable, wait.time, value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
value	The value of the variables
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.n()

AUTO $object, $variable
AUTO REAL value, status

; Write a value of 10 to the numeric variable in the workspace
$object = "/Numeric Variable"
$variable = "CurrentValue"
value = 10
CALL rm.write.num($object, $variable, 1, value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.num($object, $variable, 1, value, status)
IF (status < 0) THEN
```

```
TYPE "Unable To Read the Value: ", status
PAUSE
END

IF (value <> 10) THEN
TYPE "The update did not succeed"
PAUSE
END
.END
```

5-16 rm.read.nums

Read a list of numeric property variables associated with an object on the PC. The variable referenced must all be numeric types

Syntax

rm.read.nums($object, $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The name of the variables to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
values[]	The values of the variables
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.ns()

AUTO $object, $objects[1], $variables[1]
AUTO REAL values[1], status

; Read multiple values from multiple properties of the same object
$object = "/Numeric Variable"
$variables[0] = "Current Value"
$variables[1] = "IsBoolean"
CALL rm.read.nums($object, $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Read Value: ", status
PAUSE
END

TYPE $variables[0], " = ", values[0]
TYPE $variables[1], " = ", values[1]
```

```
; Read multiple values from multiple objects
$objects[0] = "/Numeric Variable"
$variables[0] = "Current Value"
$objects[1] = "/Numeric Variable 2"
$variables[1] = "Current Value"
CALL rm.read.anums($objects[], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Read Value: ", status
PAUSE
END

TYPE $objects[0], ".", $variables[0], " = ", values[0]
TYPE $objects[0], ".", $variables[1], " = ", values[1]
.END
```

5-17 rm.read.str

Read a string property variable associated with an object on the PC. The variable referenced must be a string type

Syntax

rm.read.str($object, $variable, wait.time, $value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
$values	The value of the variable
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.s()

AUTO $object, $variable, $value
AUTO REAL status

; Write a value of "Text" to the string variable in the workspace
$object = "/String Variable"
$variable = "CurrentValue"
$value = "Text"
CALL rm.write.str($object, $variable, 1, $value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.str($object, $variable, 1, $value, status)
IF (status < 0) THEN
```

```
TYPE "Unable To Read the Value: ", status
PAUSE
END

IF $value <> "Text" THEN
TYPE "The update did not succeed"
PAUSE
END
.END
```

5-18 rm.read.strs

Read a list of string property variables associated with an object on the PC. The variable referenced must all be string types

Syntax

rm.read.strs($object, $variables[], wait.time, $values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The names of the variables to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
$values[]	The values of the variables
status	Status of the operation: 0 = success, < 0 = error

5-19 rm.read.trns

Read a transform property variable associated with an object on the PC. The variable referenced must be a Transform3D type

Syntax

rm.read.trns($object, $variable, wait.time, value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
value	The value of the variable
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.t()

AUTO $object, $variable
AUTO REAL status
AUTO LOC t.value

; Write a value of 10 to the transform variable in the workspace
$object = "/Box"
$variable = "OffsetFromParent"
SET t.value = TRANS(10,10,10,0,0,180)
CALL rm.write.trns($object, $variable, 1, t.value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.trns($object, $variable, 1, t.value, status)
```

```
IF (status < 0) THEN
TYPE "Unable To Read the Value: ", status
PAUSE
END
.END
```

5-20 rm.read.trnss

Read a list of transform property variables associated with an object on the PC. The variable referenced must all be Transform3D types

Syntax

rm.read.trnss($object, $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The names of the variables to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
values[]	The values of the variables
status	Status of the operation: 0 = success, < 0 = error

5-21 rm.save

Saves the workspace file on the PC

Syntax

rm.save(wait.time, status)

Input Parameters

Parameter	Description
$file	Name of the file on the PC to save. If empty, save to the current file
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.save()
AUTO REAL status
AUTO $file

$file = "C:\workspace.awp"
CALL rm.save($file, 10, status)
IF (status < 0) THEN
TYPE "Unable To Save Workspace: ", status
PAUSE
END
.END
```

5-22 rm.write.anums

Set the values of an array of numeric properties associated with a collection of objects on the PC. The property variables referenced must all be of a numeric type

Syntax

rm.write.anums($objects[], $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$objects[]	The name of the remote objects to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The name of the variables to access
wait.time	The amount of time to wait for the operation to complete
values[]	The values of the variables

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.nws()

AUTO $objects[0], $variables[0]
AUTO REAL values[0], status
; Write multiple values from multiple properties of the same object
$objects[0] = "/Numeric Variable"
$variables[0] = "CurrentValue"
values[0] = 110
CALL rm.write.nums($objects[0], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Write Values: ", status
PAUSE
END

; Perform a multiple object write
CALL rm.write.anums($objects[], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Write Values: ", status
```

```
PAUSE
END
.END
```

5-23 rm.write.num

Set the value of a numeric property associated with an object on the PC. The property variable referenced must be of a numeric type

Syntax

rm.write.num($object, $variable, wait.time, value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete
value	The value of the variable

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.n()

AUTO $object, $variable
AUTO REAL value, status

; Write a value of 10 to the numeric variable in the workspace
$object = "/Numeric Variable"
$variable = "CurrentValue"
value = 10
CALL rm.write.num($object, $variable, 1, value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.num($object, $variable, 1, value, status)
IF (status < 0) THEN
```

```
TYPE "Unable To Read the Value: ", status
PAUSE
END

IF (value <> 10) THEN
TYPE "The update did not succeed"
PAUSE
END
.END
```

5-24 rm.write.nums

Set the values of an array of numeric properties associated with an object on the PC. The property variables referenced must all be of a numeric type.

Syntax

rm.write.nums($object, $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The name of the variables to access
wait.time	The amount of time to wait for the operation to complete
values[]	The values of the variables

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.nws()

AUTO $objects[0], $variables[0]
AUTO REAL values[0], status
; Write multiple values from multiple properties of the same object
$objects[0] = "/Numeric Variable"
$variables[0] = "CurrentValue"
values[0] = 110
CALL rm.write.nums($objects[0], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Write Values: ", status
PAUSE
END

; Perform a multiple object write
CALL rm.write.anums($objects[], $variables[], 1, values[], status)
IF (status < 0) THEN
TYPE "Unable To Write Values: ", status
```

```
PAUSE
END
.END
```

5-25 rm.write.str

Set the value of a string property associated with an object on the PC. The property variable referenced must be of a string type

Syntax

rm.write.str($object, $variable, wait.time, $value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete
value	The value of the variable

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.s()

AUTO $object, $variable, $value
AUTO REAL status

; Write a value of "Text" to the string variable in the workspace
$object = "/String Variable"
$variable = "CurrentValue"
$value = "Text"
CALL rm.write.str($object, $variable, 1, $value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.str($object, $variable, 1, $value, status)
IF (status < 0) THEN
```

```
TYPE "Unable To Read the Value: ", status
PAUSE
END

IF $value <> "Text" THEN
TYPE "The update did not succeed"
PAUSE
END
.END
```

5-26 rm.write.strs

Set the values of an array of string properties associated with an object on the PC. The property variables referenced must all be of a string type

Syntax

rm.write.strs($object, $variables[], wait.time, $values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable[]	The names of the variables to access
wait.time	The amount of time to wait for the operation to complete
$values[]	The values of the variables

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

5-27 rm.write.trns

Set the value of a transformation property associated with an object on the PC. The property variable referenced must be of a Transform3D type

Syntax

rm.write.trns($object, $variable, wait.time, value, status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variable	The name of the variable to access
wait.time	The amount of time to wait for the operation to complete
values	The value of the variable

Output Parameters

Parameter	Description
status	Status of the operation: 0 = success, < 0 = error

Example

```
.PROGRAM ex1.rm.vars.t()

AUTO $object, $variable
AUTO REAL status
AUTO LOC t.value

; Write a value of 10 to the transform variable in the workspace
$object = "/Box"
$variable = "OffsetFromParent"
SET t.value = TRANS(10,10,10,0,0,180)
CALL rm.write.trns($object, $variable, 1, t.value, status)
IF (status < 0) THEN
TYPE "Unable To Write Value: ", status
PAUSE
END

; Read the variable and ensure it is the same value
CALL rm.read.trns($object, $variable, 1, t.value, status)
```

```
IF (status < 0) THEN
TYPE "Unable To Read the Value: ", status
PAUSE
END
.END
```

5-28 rm.write.trnss

Read a list of transform property variables associated with an object on the PC. The variable referenced must all be Transform3D types

Syntax

rm.write.trnss($object, $variables[], wait.time, values[], status)

Input Parameters

Parameter	Description
$object	The name of the remote object to access. This must include the full path name of the object to access. e.g. "/directory/Robot 1"
$variables[]	The names of the variables to access
wait.time	The amount of time to wait for the operation to complete

Output Parameters

Parameter	Description
values[]	The values of the variables
status	Status of the operation: 0 = success, < 0 = error

OMRON Corporation **Industrial Automation Company**
Kyoto, JAPAN **Contact: www.ia.omron.com**

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, 2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ASIA PACIFIC PTE. LTD.
No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON ELECTRONICS LLC
2895 Greenspoint Parkway, Suite 200
Hoffman Estates, IL 60169 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ROBOTICS AND SAFETY TECHNOLOGIES, INC.
4225 Hacienda Drive, Pleasanton, CA 94588
U.S.A.

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor:

Cat. No. I668-E-01 1221

24000-020 A