Table A–15. IO_STATUS Errors (Cont'd)

12328	File is not opened
12329	Cannot write the variable
12330	Write file failed
12331	Cannot read the variable
12332	Read data is too short
12333	Illegal ASCII string for read
12334	Read file failed
12335	Cannot open pre_defined file
12336	Cannot close pre_defined file
12338	Close file failed
12347	Read I/O value failed
12348	Write I/O value failed
12358	Timeout at read request
12359	Read request is nested
12367	Bad base in format

- Use READ file_id(cr) to clear any IO_STATUS error.
- If *file_id* does not correspond to an opened file or one of the pre-defined “files” opened to the respective CRT/KB, teach pendant, and vision windows, the program is aborted with an error.

Example: Refer to Section B.12 , "Displaying a List From a Dictionary File" (DCLIST_EX.KL), for a detailed program example.

A.11 - J - KAREL LANGUAGE DESCRIPTION

A.11.1 J_IN_RANGE Built-In Function

Purpose: Returns a BOOLEAN value indicating whether or not the specified joint position argument can be reached by a group of axes

Syntax : J_IN_RANGE(posn)

Function Return Type :BOOLEAN

Input/Output Parameters:

[in] posn :JOINTPOS

%ENVIRONMENT Group :SYSTEM

Details:

- The returned value is TRUE if *posn* is within the work envelope; otherwise, FALSE is returned.

See Also: IN_RANGE Built-in Function, CHECK_EPROS Built-in procedure

A.11.2 JOINTPOS Data Type

Purpose: Defines a variable, function return type, or routine parameter as JOINTPOS data type.

Syntax : JOINTPOS<n> <IN GROUP[m]>

Details:

- A JOINTPOS consists of a REAL representation of the position of each axis of the group, expressed in degrees or millimeters (mm).
- *n* specifies the number of axes, with 9 as the default. The size in bytes is 4 + 4 * n.
- A JOINTPOS may be followed by IN GROUP[m], where m indicates the motion group with which the data is to be used. The default is the group specified by the %DEFGROUP directive or 1.
- CNV_REL_JPOS and CNV_JPOS_REL Built-ins can be used to access the real values.
- A JOINTPOS can be assigned to other positional types. Note that some motion groups, for example single axis positioners, have no XYZWPR representation. If you attempt to assign a JOINTPOS to a XYZWPR or POSITION type for such a group, a run-time error will result.

Example: Refer to the following sections for detailed program examples:

Section B.5 ,"Using Register Built-ins" (REG_EX.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.11.3 JOINT2POS Built-In Function

Purpose: This routine is used to convert joint angles (in_jnt) to a Cartesian position (out_pos) by calling the forward kinematics routine.

Syntax : JOINT2POS (in_jnt - Joint angles can be converted to Cartesian, uframe, utool, config_ref, out_pos, wjnt_cfg, ext_ang, and status).

Input/Output Parameters:

[in] in_jnt :Jointpos

[in] uframe :POSITION

[in] utool :POSITION

[in] config_ref :INTEGER

[out] out_pos :POSITION

[out] wjnt_cfg :CONFIG

[out] ext_ang :ARRAY OF REAL

[out] status :INTEGER

%ENVIRONMENT Group :MOTN

Details:

- The input *in_jnt* is defined as the joint angles to be converted to the Cartesian position.
- The input *uframe* is the user frame for the Cartesian position.
- The input *utool* is defined as the corresponding tool frame.
- The input *config_ref* is an integer representing the type of solution desired. The values listed below are valid. Also, the pre-defined constants in the parentheses can be used and the values can be added as required. One example includes: config_ref = HALF_SOLN + CONFIG_TCP.
  - 0 :(FULL_SOLN) = Default
  - 1 : (HALF_SOLN) = Wrist joint (xyz456). This value does not calculate/use wpr.
  - 2 :(CONFIG_TCP) = The Wrist Joint Config (up/down) is based on the fixed wrist.
  - 4 :(APPROX_SOLN) = Approximate solution. This value reduce calculation time for some robots.
  - 8 :(NO_TURNS) = Ignore wrist turn numbers. Use the closest path for joints 4, 5 and 6 (uses ref_jnt).
  - 16 :(NO_M_TURNS) = Ignore major axis (J1 only) turn number. Use the closest path.
- The output *out_pos* is the Cartesian position corresponding to the input joint angles.
- The output *wjnt_cfg* is the wrist joint configuration. The value will be output when *config_ref* corresponds to HALF_SOLN.

Draft

- The output *ext_ang* contains the values of the joint angles for the extended axes if they exist.
- The output *status* explains the status of the attempted operation. If it is not equal to 0, then an error has occurred.

A.12 - K - KAREL LANGUAGE DESCRIPTION

A.12.1 KCL Built-In Procedure

Purpose: Sends the KCL command specified by the STRING argument to KCL for execution.

Syntax : KCL (command, status)

Input/Output Parameters :

[in] command :STRING

[out] status :INTEGER

%ENVIRONMENT Group :kclop

Details:

- *command* must contain a valid KCL command.
- *command* cannot exceed 126 characters.
- Program execution waits until execution of the KCL command is completed or until an error is detected.
- All KCL commands are performed as if they were entered at the command level, with the exception of destructive commands, such as CLEAR ALL, for which no confirmation is required.
- *status* indicates whether the command was executed successfully.
- If a KCL command file is being executed and $STOP_ON_ERR is FALSE, the KCL built-in will continue to run to completion. The first error detected will be returned or a 0 if no errors occurred.

See Also: KCL_NO_WAIT, KCL_STATUS Built-In Procedures

Example: The following example will show programs and wait until finished. Status will be the outcome of this operation.

KCL Built-In Procedure

```
PROGRAM kcl_test
VAR
 command :STRING[20]
 status :INTEGER
BEGIN
```

Draft

```
 command = 'SHOW PROGRAMS'
 KCL (command, status)
END kcl_test
```

Example: Refer to Example Program for Display Only Data Items for another example.

A.12.2 KCL_NO_WAIT Built-In Procedure

Purpose: Sends the KCL command specified by the STRING argument to KCL for execution, but does not wait for completion of the command before continuing program execution.

Syntax : KCL_NO_WAIT (command, status)

Input/Output Parameters :

[in] command :STRING

[out] status :INTEGER

%ENVIRONMENT Group :kclop

Details:

- *command* must contain a valid KCL command.
- *status* indicates whether KCL accepted the command.
- Program execution waits until KCL accepts the command or an error is detected.

See Also: KCL, KCL_STATUS Built-In Procedures

Example: The following example will load a program, but will not wait for the program to be loaded before returning. Status will indicate if the command was accepted or not.

KCL_NO_WAIT Built-In Procedure

```
PROGRAM kcl_test
VAR
 command :STRING[20]
 status :INTEGER
BEGIN
 command = 'Load prog test_1'
 KCL_NO_WAIT (command, status)
 delay 5000
 status = KCL_STATUS
END kcl_test
```

Draft

A.12.3 KCL_STATUS Built-In Procedure

Purpose: Returns the status of the last executed command from either KCL or KCL_NO_WAIT built-in procedures.

Syntax : KCL_STATUS

Function Return Type :INTEGER

%ENVIRONMENT Group :kclop

Details:

- Returns the *status* of the last executed command from the KCL or KCL_NO_WAIT built-ins.
- Program execution waits until KCL can return the status.

See Also: KCL_NO_WAIT, KCL Built-In Procedures

A.13 - L - KAREL LANGUAGE DESCRIPTION

A.13.1 LN Built-In Function

Purpose: Returns the natural logarithm of a specified REAL argument

Syntax : LN(x)

Function Return Type :REAL

Input/Output Parameters:

[in] x : REAL

%ENVIRONMENT Group :SYSTEM

Details:

- The returned value is the natural logarithm of *x* .
- *x* must be greater than zero. Otherwise, the program will be aborted with an error.

Example: The following example returns the natural logarithm of the input variable **a** and assigns it to the variable **b** .

LN Built-In Function

```
WRITE(CR, CR, 'enter a number =')
  READ(a,CR)
```

Draft

```
b = LN(a)
```

A.13.2 LOAD Built-In Procedure

Purpose: Loads the specified file

Syntax : LOAD (file_spec, option_sw, status)

Input/Output Parameters:

[in] file_spec :STRING

[in] option_sw :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *file_spec* specifies the device, name, and type of the file to load. The following types are valid:

  .TP Teach pendant program

  .PC KAREL program

  .VR KAREL variables

  .SV KAREL system variables

  .IO I/O configuration data

  no ext KAREL program and variables
- *option_sw* specifies the type of options to be done during loading.

  The following value is valid for .TP files:

  — 1 If the program already exists, then it overwrites the program. If option_sw is not 1 and the program exists, an error will be returned.

  The following value is valid for .SV files:

  — 1 Converts system variables.

- *option_sw* is ignored for all other types.

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

The following note applies to R-30*i*B controllers:

Note The KAREL option must be installed on the robot controller in order to load KAREL programs.

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Draft

A.13.3 LOAD_STATUS Built-In Procedure

Purpose: Determines whether the specified KAREL program and its variables are loaded into memory

Syntax : LOAD_STATUS(prog_name, loaded, initialized)

Input/Output Parameters:

[in] prog_name :STRING

[out] loaded :BOOLEAN

[out] initialized :BOOLEAN

%ENVIRONMENT Group :PBCORE

Details:

- *prog_name* must be a program and cannot be a routine.
- *loaded* returns a value of TRUE if *prog_name* is currently loaded into memory. FALSE is returned if *prog_name* is not loaded.
- *initialized* returns a value of TRUE if any variable within *prog_name* has been initialized. FALSE is returned if all variables within *prog_name* are uninitialized.
- If either *loaded* or *initialized* is FALSE, use the LOAD built-in procedure to load *prog_name* and its variables.

Example: Refer to the following sections for detailed program examples:

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

A.13.4 LOCK_GROUP Built-In Procedure

Purpose: Locks motion control for the specified group of axes

Syntax : LOCK_GROUP(group_mask, status)

Input/Output Parameters:

[in] group_mask : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :MULTI

Details:

- *group_mask* specifies the group of axes to lock for the running task. The group numbers must be in the range of 1 to the total number of groups defined on the controller.
- The *group_mask* is specified by setting the bit(s) for the desired group(s).

Table A–16. Group_mask Setting

GROUP	DECIMAL	BIT
Group 1	1	1
Group 2	2	2
Group 3	4	3

To specify multiple groups select the decimal values, shown in Table A–16 , which correspond to the desired groups. Then connect them together using the OR operator. For example to specify groups 1, 3, enter "1 OR 4".

- Motion control is gained for the specified motion groups.
- If one or more of the groups cannot be locked, then an error is returned, and any available groups will be locked.
- Moving a group automatically locks the group if it has not been previously locked by another task.
- If a task tries to move a group that is already locked by another task, it will be paused.
- *status* explains the status of the attempted operation. If not equal to 0, an error occurred.

Draft

Example: The following example unlocks group 1, 2, and 3, and then locks group 3. Refer to Chapter 15 *MULTI-TASKING* , for more examples.

LOCK_GROUP Built-In Procedure

```
%ENVIRONMENT MOTN
%ENVIRONMENT MULTI
  VAR
    status: INTEGER
  BEGIN
    REPEAT
      -- Unlock groups 1, 2, and 3
      UNLOCK_GROUP(1 OR 2 OR 4, status)
      IF status = 17040 THEN
        CNCL_STP_MTN -- or RESUME
      ENDIF
      DELAY 500
    UNTIL status = 0
    -- Lock only group 3
    LOCK_GROUP(4, status)
  END lock_grp_ex
```

Draft

A.13.5 %LOCKGROUP Translator Directive

Purpose: Specifies the motion group(s) to be locked when calling this program or a routine from this program.

Syntax : %LOCKGROUP = n, n ,...

Details:

- *n* is the number of the motion group to be locked.
- The range of *n* is 1 to the number of groups on the controller.
- When the program or routine is called, the task will attempt to get motion control for all the specified groups if it does not have them locked already. The task will pause if it cannot get motion control.
- If %LOCKGROUP is not specified, all groups will be locked.
- The %NOLOCKGROUP directive can be specified if no groups should be locked.

See Also: %NOLOCKGROUP Directive, LOCK_GROUP, UNLOCK_GROUP Built-In Procedures

A.14 - M - KAREL LANGUAGE DESCRIPTION

A.14.1 MIRROR Built-In Function

Purpose: Determines the mirror image of a specified position variable.

Syntax : MIRROR (old_pos, mirror_frame, orientation_flag)

Function Return Type: XYZWPREXT

Input/Output Parameters :

[in] old_pos :POSITION

[in] mirror_frame :POSITION

[in] orient_flag :BOOLEAN

%ENVIRONMENT Group :MIR

Details:

- *old_pos* and *mirror_frame* must both be defined relative to the same user frame.
- *old_pos* specifies the value whose mirror image is to be generated.
- *mirror_frame* specifies the value across whose xz_plane the image is to be generated.
- If *orient_flag* is TRUE, both the orientation and location component of *old_pos* will be mirrored. If FALSE, only the location is mirrored and the orientation of the new mirror-image position is the same as that of *old_pos.*
- The returned mirrored position is not guaranteed to be a reachable position, since the mirrored position can be outside of the robot's work envelope.

See Also: The appropriate application-specific *FANUC Robotics Setup and Operations Manual* , chapter on "Advanced Functions"

Example: The following example gets the current position of the robot, creates a mirror frame, and generates a mirrored position which is mirrored about the y axis.

MIRROR Built-In Function

```
PROGRAM mir_exam
VAR
 cur_pos:   XYZWPREXT
 org_pos:   POSITION
 mir_pos:   XYZWPREXT
```

Draft

```
  mir_posa:  POSITION
  pos_frame:  XYZWPREXT
  frame:   POSITION
  orient_flag: BOOLEAN
 BEGIN
  cur_pos = curpos(0,0) -- Get the current position of the robot
  cur_pos.x = 1000.00  -- Create a new position which is at(1000,0,300,w,p,r)
  cur_pos.y = 0.0
  cur_pos.z = 300.00
  SET_EPOS_REG(1, cur_pos, status)
  move_to_pr1 -— Call TP program to move to PR[1]
              -- The robot is now at a known position:
              --(1000,0,300,w,p,r) where (w,p,r) have not
              --changed from the original position.
  pos_frame = curpos(0,0) -- Create a frame used to mirror about.
  pos_frame.w = 0   -- By setting (w,p,r) to 0, the x-z plane of
  pos_frame.p = 0   -- pos_frame will be parallel to the world’s x-z
  pos_frame.r = 0    -- plane. pos_frame now set to (1000,0,300,0,0,0)
  frame = pos_frame   -- Convert the mirror frame to a POSITION type.
  cur_pos.y = 200   -- Move 200mm in the y direction.
  SET_EPOS_REG(1, cur_pos, status)
  move_to_pr1    -- Current position is (1000,200,300,w,p,r)
  org_pos = cur_pos   -- Convert org_pos to a POSITION type.
  orient_flag = FALSE  -- Send Mirror current position: (1000, 200, 300,
                       -- w,p,r), and mirror frame: (1000,0,300,0,0,0).
                       -- Mirrors about the y axis without mirroring the
                       -- orientation (w,p,r).
  mir_pos = mirror(org_pos, frame, orient_flag)
          -- mir_pos is the mirrored position: (1000, -200,
          -- 300, w, p, r).
          -- The orientation is the same as org_pos.
  orient_flag = TRUE  -- The mirrored position includes mirroring of
                      -- the tool orientation.
  mir_posa = mirror(org_pos,frame,orient_flag)
          -- mir_posa is the mirrored position where Normal
          -- Orient, & Approach vectors have been mirrored.
 end mir_exam
```

A.14.2 MODIFY_QUEUE Built-In Procedure

Purpose: Replaces the value of an entry of a queue.

Syntax : MODIFY_QUEUE(value, sequence_no, queue_t, queue_data, status)

Input/Output Parameters:

[in] value :INTEGER

[in] sequence_no :INTEGER

[in,out] queue_t :QUEUE_TYPE

[in,out] queue_data :ARRAY OF INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBQMGR

Details:

- *value* specifies the value to be inserted into the queue.
- *sequence_no* specifies the sequence number of the entry whose value is to be modified
- *queue_t* specifies the queue variable for the queue
- *queue_data* specifies the array used to hold the data in the queue. The length of this array determines the maximum number of entries in the queue.
- *status* is returned with 61003, “Bad sequence no,” if the specified sequence number is not in the queue.

See Also: COPY_QUEUE, GET_QUEUE, DELETE_QUEUE Built-In Procedures Section 15.8 , "Using Queues for Task Communication"

Example: In the following example, the routine update_queue replaces the value of the specified entry (**sequence_no**); of a queue (**queue** and **queue_data** with a new value (**value**).

MODIFY_QUEUE Built-In Procedure

```
PROGRAM mod_queue_x
%ENVIRONMENT PBQMGR
ROUTINE update_queue(value: INTEGER;
                     sequence_no: INTEGER;
                     queue_t: QUEUE_TYPE;
                     queue_data: ARRAY OF INTEGER)
VAR
  status: INTEGER
BEGIN
MODIFY_QUEUE(value, sequence_no, queue_t, queue_data, status)
return
END update_queue
BEGIN
END mod_queue_x
```

A.14.3 MOTION_CTL Built-In Function

Purpose: Determines whether the KAREL program has motion control for the specified group of axes

Syntax : MOTION_CTL<(group_mask)>

Function Return Type :BOOLEAN

Input/Output Parameters:

[in] group_mask :INTEGER

%ENVIRONMENT Group :MOTN

Details:

- If *group_mask* is omitted, the default group mask for the program is assumed.
- The default *group_mask* is determined by the %LOCKGROUP and %NOLOCKGROUP directives.
- The *group_mask* is specified by setting the bit(s) for the desired group(s).

Draft

Table A–17. Group_mask Setting

GROUP	DECIMAL	BIT
Group 1	1	1
Group 2	2	2
Group 3	4	3

To specify multiple groups select the decimal values, shown in Table A–17 , which correspond to the desired groups. Then connect them together using the OR operator. For example to specify groups 1, 3, enter "1 OR 4".

- Returns TRUE if the KAREL program has motion control for the specified group of axes.

A.14.4 MOUNT_DEV Built-In Procedure

Purpose: Mounts the specified device

Syntax : MOUNT_DEV (device, status)

Input/Output Parameters:

[in] device : STRING

[out] status :INTEGER

%ENVIRONMENT Group :FDEV

Details:

- *device* specifies the device to be mounted.
- *status* explains the status of the attempted operation. If it is not equal to 0, then an error occurred.

See Also: DISMOUNT_DEV and FORMAT_DEV

Example: Refer to Section B.9 , "Using the File and Device Built-ins" (FILE_EX.KL), for a detailed program example.

A.14.5 MOVE_FILE Built-In Procedure

Purpose: Moves the specified file from one memory file device to another

Syntax : MOVE_FILE (file_spec, status)

Input/Output Parameters :

[in] file_spec : string

[out] status : integer

%ENVIRONMENT Group :FDEV

Details:

- *file_spec* specifies the device, name, and type of the file to be moved. The file should exist on the FROM or RAM disks.
- If *file_spec* is a file on the FROM disk, the file is moved to the RAM disk, and vice versa.
- The wildcard character (*) can be used to replace the entire file name, the first part of the file name, the last part of the file name, or both the first and last parts of the file name. The file type can also use the wildcard in the same manner. If file_spec specifies multiple files, then they are all moved to the other disk.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Example: In the following example, all .KL files are moved from the RAM disk to the FROM disk.

MOVE_FILE Built-In Procedure

```
PROGRAM move_files
```

Draft

```
%NOLOCKGROUP
%ENVIRONMENT FDEV
VAR
  status: INTEGER
BEGIN
  MOVE_FILE('RD:\*.KL', status)
  IF status <> 0 THEN
    POST_ERR(status, '', 0, 0)
  ENDIF
END move_files
```

A.14.6 MSG_CONNECT Built-In Procedure

Purpose: Connect a client or server port to another computer for use in Socket Messaging.

Syntax : MSG_CONNECT (tag, status)

Input/Output Parameters :

[in] tag :STRING

[out] status :INTEGER

%ENVIRONMENT Group :FLBT

Details:

- Tag is the name of a client port (C1:-C8) or server port (S1:S8).
- Status explains the status of the attempted operation. If it is not equal to 0, then an error occurred.

See Also: Socket Messaging in the *FANUC Robotics Internet Options Setup and Operations Manual.*

Example: The following example connects to S8: and reads messages. The messages are displayed on the teach pendant screen.

MSG_CONNECT Built-In Procedure

```
PROGRAM tcpserv8
VAR
  file_var :  FILE
  tmp_int  :  INTEGER
  tmp_int1 :  INTEGER
  tmp_str  :  string [128]
  tmp_str1 :  string [128]
  status   :  integer
```

```
  entry    :  integer
BEGIN
   SET_FILE_ATR (file_var, ATR_IA)
  -- Set up S8 server tag
  DISMOUNT_DEV('S8:',status)
  MOUNT_DEV('S8:',status)
  write  (' Mount Status = ',status,cr)
  status = 0
  IF status = 0 THEN
    -- Connect the tag
    write ('Connecting ..',cr)
    MSG_CONNECT ('S8:',status)
    write  ('Connect Status = ',status,cr)
    IF status < > 0 THEN
      MSG_DISCO('S8:',status)
      write  (' Connecting..',cr)
      MSG_CONNECT('S8:',status)
      write  (' Connect Status = ',status,cr)
      ENDIF
    IF status = 0 THEN
      -- OPEN S8:
      write ('Opening',cr)
      OPEN FILE file_var ('rw','S8:')
      status = io_status(file_var)
      FOR tmp_int  1 TO 1000 DO
        write  ('Reading',cr)
        BYTES_AHEAD(file_var, entry, status)
        -- Read 10 bytes
        READ file_var (tmp_str::10)
        status = i/o_status(file_var)
       --Write 10 bytes
       write (tmp_str::10,cr)
       status = io_status(file_var)
       ENDFOR
     CLOSE FILE file_var
     write  ('Disconnecting..',cr)
     MSG_DISCO('S8:',status)
     write  ('Done.',cr)
   ENDIF
 ENDIF
END tcpserv8
```

A.14.7 MSG_DISCO Built-In Procedure

Purpose: Disconnect a client or server port from another computer.

Syntax : MSG_DISCO (tag, status)

Input/Output Parameters :

[in] tag :STRING

[out] status :INTEGER

%ENVIRONMENT Group :FLBT

Details:

- Tag is the name of a client port (C1:-C8) or server port (S1:S8).
- Status explains the status of the attempted operation. If it is not equal to 0, then an error occurred.

See Also: Socket Messaging in the *FANUC Robotics Internet Options Setup and Operations Manual* .

Example: Refer to MSG_CONNECT Built-In Procedure for more examples.

Draft

A.14.8 MSG_PING

Syntax : MSG_PING (host name, status)

Input/Output Parameters :

[in] host name :STRING

[out] status :INTEGER

%ENVIRONMENT Group :FLBT

Details:

- Host name is the name of the host to perform the check on. An entry for the host has to be present in the host entry tables (or the DNS option loaded and configured on the robot).
- Status explains the status of the attempted operation. If it is not equal to 0, then an error occurred.

See Also: Ping in the *FANUC Robotics Internet Options Setup and Operations Manual.*

Example: The following example performs a PING check on the hostname "fido". It writes the results on the teach pendant.

MSG_PING Built-In Procedure

```
PROGRAM pingtest
VAR
 Tmp_int  : INTEGER
 Status   : integer
BEGIN
  WRITE('pinging..',cr)
  MSG_PING('fido',status)
  WRITE('ping Status = ',status,cr)
END pingtest
```

A.15 - N - KAREL LANGUAGE DESCRIPTION

A.15.1 NOABORT Action

Purpose: Prevents program execution from aborting when an external error occurs

Details:

- The NOABORT action usually corresponds to an ERROR[n].
- If the program is aborted by itself (i.e., executing an ABORT statement, run time error), the NOABORT action will be ignored and program execution will be aborted.

Example: The following example uses a global condition handler to test for error number 11038, "Pulse Mismatch." If this error occurs, the NOABORT action will prevent program execution from being aborted.

NOABORT Action

```
PROGRAM noabort_ex
%NOLOCKGROUP
BEGIN
  --Pulse Mismatch condition handler
  CONDITION[801]:
    WHEN ERROR[11038] DO
    NOABORT
  ENDCONDITION
  ENABLE CONDITION[801]
END noabort_ex
```

Draft

A.15.2 %NOABORT Translator Directive

Purpose: Specifies a mask for aborting

Syntax : %NOABORT = ERROR + COMMAND

Details:

- ERROR and COMMAND are defined as follows:

```
ERROR : ignore abort error severity
   COMMAND : ignore abort command
```

- Any combination of ERROR and COMMAND can be specified.
- If the program is aborted by itself (for example, executing an ABORT statement, run-time error), the %NOABORT directive will be ignored and program execution will be aborted.
- This directive is only effective for programs with %NOLOCKGROUP. If the program has motion control, the %NOABORT directive will be ignored and program execution will be aborted.

A.15.3 %NOBUSYLAMP Translator Directive

Purpose: Specifies that the busy lamp will be OFF during execution.

Syntax: %NOBUSYLAMP

Details:

- The busy lamp can be set during task execution by the SET_TSK_ATTR built-in.

A.15.4 NODE_SIZE Built-In Function

Purpose: Returns the size (in bytes) of a PATH node

Syntax : NODE_SIZE(path_var)

Function Return Type :INTEGER

Input/Output Parameters:

[in] path_var : PATH

%ENVIRONMENT Group :PATHOP

Details:

Draft

- The returned value is the size of an individual PATH node, including the positional data type size and any associated data.
- The returned value can be used to calculate file positions for random access to nodes in files.

Example: The following example program reads a path, while overlapping reads with preceding moves. The routine **read_header** reads the path header and prepares for reading of nodes. The routine **read_node** reads a path node.

NODE_SIZE Built-In Function

```
PROGRAM read_and_mov
VAR my_path: PATH
    xyz_pos: XYZWPR
    path_base: INTEGER
    node_size: INTEGER
    max_node_no: INTEGER
    i: INTEGER
    file_var:  FILE
--
ROUTINE read_header
BEGIN
  READ file_var(my_path[0])
  IF IO_STATUS(file_var) <> 0 THEN
    WRITE('HEADER READ ERROR:',IO_STATUS(file_var),cr)
    ABORT
  ENDIF
  max_node_no = PATH_LEN(my_path)
  node_size = NODE_SIZE(my_path)
  path_base = GET_FILE_POS(file_var)
END read_header
--
ROUTINE read_node(node_no: INTEGER)
VAR status: INTEGER
BEGIN
  SET_FILE_POS(file_var, path_base+(node_no-1)*node_size, status)
  READ file_var(my_path[node_no])
END read_node
--
BEGIN
  SET_FILE_ATR(file_var, atr_uf)
  OPEN FILE F1('RO','PATHFILE.DT')
  read_header
  FOR i = 1 TO max_node_no DO
    read_node(i)
    xyz_pos = my_path[i]
    SET_POS_REG(1 xyz_pos, status)
    move_to_pr1 —— Call TP program to move to node
  ENDFOR
```

Draft

```
  CLOSE FILE file_var
END read_and_mov
```

A.15.5 %NOLOCKGROUP Translator Directive

Purpose: Specifies that motion groups do not need to be locked when calling this program, or a routine defined in this program.

Syntax : %NOLOCKGROUP

Details:

- When the program or routine is called, the task will not attempt to get motion control.
- If %NOLOCKGROUP is not specified, all groups will be locked when the program or routine is called, and the task will attempt to get motion control. The task will pause if it cannot get motion control.
- The task will keep motion control while it is executing the program or routine. When it exits the program or routine, the task automatically unlocks all the motion groups.
- If the task contains executing or stopped motion, then task execution is held until the motion is completed. Stopped motion must be resumed and completed or cancelled.
- If a program that has motion control calls a program with the %NOLOCKGROUP Directive or a routine defined in such a program, the program will keep motion control even though it is not needed.
- The UNLOCK_GROUP built-in routine can be used to release control.
- If a motion statement is encountered in a program that has the %NOLOCKGROUP Directive, the task will attempt to get motion control for all the required groups if it does not already have it. The task will pause if it cannot get motion control.

See Also: %LOCKGROUP Translator Directive, LOCK_GROUP, UNLOCK_GROUP Built-In Procedures

Example: Refer to the following sections for detailed program examples:

Section B.3 ,"Saving Data to the Default Device" (SAVE_VR.KL)

Section B.5 ,"Using Register Built-ins" (REG_EX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.11 , "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Draft

Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)

A.15.6 NOMESSAGE Action

Purpose: Suppresses the display and logging of error messages

Syntax : NOMESSAGE

Details:

- Display and logging of the error messages are suppressed only for the error number specified in the corresponding condition.
- Use a wildcard (*) to suppress all messages.
- Abort error messages still will be displayed and logged even if NOMESSAGE is used.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.15.7 NOPAUSE Action

Purpose: Resumes program execution if the program was paused, or prevents program execution from pausing

Syntax : NOPAUSE

Details:

- The NOPAUSE action usually corresponds to an ERROR[n] or PAUSE condition.
- The program will be resumed, even if it was paused before the error.
- If the program is paused by itself, the NOPAUSE action will be ignored and program execution will be paused.

Example: The following example uses a global condition handler to test for error number 12311. If this error occurs, the NOPAUSE action will prevent program execution from being paused and the NOMESSAGE action will suppress the error message normally displayed for error number 12311. This will allow the routine **uninit_error** to be executed without interruption.

NOPAUSE Action

```
ROUTINE uninit_error
  BEGIN
    WRITE ('Uninitialized operand',CR)
    WRITE ('Use KCL> SET VAR to initialize operand',CR)
```

```
    WRITE ('Press Resume at Test/Run screen to ',cr)
    WRITE ('continue program',cr)
    PAUSE  --pauses program (undoes NOPAUSE action)
  END uninit_error
CONDITION[1]:
  WHEN ERROR[12311] DO
    NOPAUSE, NOMESSAGE, uninit_error
ENDCONDITION
```

Draft

A.15.8 %NOPAUSE Translator Directive

Purpose: Specifies a mask for pausing

Syntax : %NOPAUSE = ERROR + COMMAND + TPENABLE

Details:

- The bits for the mask are as follows:

```
ERROR : ignore pause error severity
   COMMAND : ignore pause command
   TPENABLE : ignore paused request when TP enabled
```

- Any combination of ERROR, COMMAND, and TPENABLE can be specified.
- If the program is paused by itself, the %NOPAUSE directive will be ignored and program execution will be paused.
- This directive is only effective for programs with %NOLOCKGROUP. If the program has motion control, the %NOPAUSE Directive will be ignored and program execution will be paused.

A.15.9 %NOPAUSESHFT Translator Directive

Purpose: Specifies that the task is not paused if shift key is released.

Syntax : %NOPAUSESHFT

Details:

- This attribute can be set during task execution by the SET_TSK_ATTR built-in routine.

A.16 - O - KAREL LANGUAGE DESCRIPTION

A.16.1 OPEN FILE Statement

Purpose: Associates a data file or communication port with a file variable

Syntax : OPEN FILE file_var (usage_string, file_string)

where:

file_var : FILE

usage_string : a STRING

file_string : a STRING

Details:

- *file_var* must be a static variable not already in use by another OPEN FILE statement.
- The *usage_string* is composed of the following:

  'RO' :Read only

  'RW' :Read write

  'AP' :Append

  'UD' :Update
- The *file_string* identifies a data file name and type, a window or keyboard, or a communication port.
- The SET_FILE_ATR built-in routine can be used to set a file's attributes.
- When a program is aborted or exits normally, any opened files are closed. Files are not closed when a program is paused.
- Use the IO_STATUS built-in function to verify if the open file operation was successful.

See Also: IO_STATUS Built-In Function, SET_FILE_ATR Built-In Procedure, Chapter 7 *FILE INPUT/OUTPUT OPERATIONS* , Chapter 9 *FILE SYSTEM* , Appendix E , "Syntax Diagrams" for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Draft

A.16.2 OPEN HAND Statement

Purpose: Opens a hand on the robot

Syntax : OPEN HAND hand_num

where:

hand_num : an INTEGER expression

Details:

- The actual effect of the statement depends on how the HAND signals are set up. Refer to Chapter 13, ‘‘Input/Output System.’’
- *hand_num* must be a value in the range 1-2. Otherwise, the program is aborted with an error.
- The statement has no effect if the value of *hand_num* is in range but the hand is not connected.
- If the value of **hand_num** is in range but the HAND signal represented by that value has not been assigned, the program is aborted with an error.

See Also: Appendix D, ‘‘Syntax Diagrams’’ for more syntax information

Example: The following example moves the TCP to the position register **PR[2]** and opens the hand of the robot specified by the INTEGER variable **hand_num** .

OPEN HAND Statement

```
move_to_pr2  --  Call TP program to move to PR[2]
OPEN HAND hand_num
```

A.16.3 OPEN_TPE Built-In Procedure

Purpose: Opens the specified teach pendant program

Syntax : OPEN_TPE(prog_name, open_mode, reject_mode, open_id, status)

Input/Output Parameters:

[in] prog_name :STRING

[in] open_mode :INTEGER

[in] reject_mode :INTEGER

[out] open_id :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *prog_name* specifies the name of the teach pendant program to be opened. *prog_name* must be in all capital letters.
- *prog_name* must be closed, using CLOSE_TPE, before *prog_name* can be executed.
- *open_mode* determines the access code to the program. The access codes are defined as follows:

  0 : none

  TPE_RDACC : Read Access

  TPE_RWACC : Read/Write Access
- *reject_mode* determines the reject code to the program. The program that has been with a reject code cannot be opened by another program. The reject codes are defined as follows:

  TPE_NOREJ : none

  TPE_RDREJ : Read Reject

  TPE_WRTREJ : Write Reject

  TPE_RWREJ : Read/Write Reject

  TPE_ALLREJ : All Reject
- *open_id* indicates the id number of the opened program.
- *status* explains the status of the attempted operation. If not equal to 0, then an error has occurred.
- All open teach pendant programs are closed automatically when the KAREL program is aborted or exits normally.

See Also: CREATE_TPE Built-In Procedure, COPY_TPE Built-In Procedure, AVL_POS_NUM Built-In Procedure

Example: Refer to Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL), for a detailed program example.

A.16.4 ORD Built-In Function

Purpose: Returns the numeric ASCII code corresponding to the character in the STRING argument that is referenced by the index argument

Syntax : ORD(str, str_index)

Function Return Type :INTEGER

Input/Output Parameters:

[in] str :STRING

[in] str_index :INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- The returned value represents the ASCII numeric code of the specified character.
- *str_index* specifies the indexed position of a character in the argument *str* . A value of 1 indicates the first character.
- If *str_index* is less than one or greater than the current length of *str* , the program is paused with an error.

See Also: Appendix D , “Character Codes”

Example: Refer to Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL), for a detailed program example.

A.16.5 ORIENT Built-In Function

Purpose: Returns a unit VECTOR representing the y-axis (orient vector) of the specified POSITION argument

Syntax : ORIENT(posn)

Function Return Type :VECTOR

Input/Output Parameters:

[in] posn : POSITION

%ENVIRONMENT Group :VECTR

Details:

- Instead of using this built-in, **you can directly access the Orient Vector of a POSITION.**
- The returned value is the orient vector of *posn* .
- The orient vector is the positive y-direction in the tool coordinate frame.

A.17 - P - KAREL LANGUAGE DESCRIPTION

A.17.1 PATH Data Type

Purpose: Defines a variable or routine parameter as PATH data type

Syntax : PATH

Details:

- A PATH is a varying length list of elements called path nodes, numbered from 1 to the number of nodes in the PATH.
- No valid operators are defined for use with PATH variables.
- A PATH variable is indexed (or subscripted) as if it were an ARRAY variable. For example, *tool_track[1]* refers to the first node of a PATH called *tool_track* .
- An uninitialized PATH has a length of zero.
- PATH variables cannot be declared local to routines and cannot be returned from functions.
- Only PATH expressions can be assigned to PATH variables or passed as arguments to PATH parameters.
- A PATH variable can specify a data structure constituting the data for each path node.
- A PATH variable can specify a data structure constituting the path header. This can be used to specify the UFRAME and/or UTOOL to be used with recording the path. It can also specify an axis group whose current position defines a table-top coordinate frame with respect to which the robot data is recorded.
- A PATH can be declared with either, neither, or both of the following clauses following the word PATH:
  - — NODEDATA = node_struct_name, specifying the data structure constituting a path node.
  - — PATHHEADER = header_struct_name, specifying the structure constituting the path header.

If both fields are present, they can appear in either order and are separated by a comma and optionally a new line.

- If NODEDATA is not specified, it defaults to the STD_PTH_NODE structure described in Appendix A.
- If PATHHEADER is not specified, there is no (user-accessible) path header.
- An element of the PATHHEADER structure can be referenced with the syntax path_var_name.header_field_name.
- An element of a NODEDATA structure can be referenced with the syntax path_var_name[node_no].node_field_name.

- The path header structure can be copied from one path to another with the path_var1 = path_var2 statement.
- The path node structure can be copied from one node to another with the path_var[2] = path_var[1] statement.
- A path can be passed as an argument to a routine as long as the PATHHEADER and NODEDATA types match. A path that is passed as an argument to a built-in routine can be of any type.
- A path node can be passed as an argument to a routine as long as the routine parameter is the same type as the NODEDATA structure.
- A path can be declared with a NODEDATA structure having no position type elements. This can be a useful way of maintaining a list structure.

See Also: APPEND_NODE, DELETE_NODE, INSERT_NODE Built-In Procedures, PATH_LEN, NODE_SIZE Built-In Functions

Example: The following example shows different declarations of PATH variables.

PATH Data Type

```
TYPE
      node_struct = STRUCTURE
        node_posn: XYZWPR IN GROUP[1]
        aux_posn: JOINTPOS IN GROUP[2]
        weld_time: INTEGER
        weld_current: INTEGER
      ENDSTRUCTURE
    hdr_struct = STRUCTURE
        uframe1: POSITION
        utool: POSITION
        speed: REAL
      ENDSTRUCTURE
    VAR
      path_1a: PATH PATHHEADER = hdr_struct, NODEDATA = node_struct
      path_1b: PATH NODEDATA = node_struct, PATHHEADER = hdr_struct
      path_2:  PATH NODEDATA = node_struct -- no header
      path_3:  PATH -- NODEDATA is STD_PTH_NODE
      path_4:  PATH PATHHEADER = hdr_struct -- NODEDATA is STD_PTH_NODE
```

The following example shows how elements of the NODEDATA and PATHHEADER structures can be referenced.

PATH Data Type

```
-- Using declarations for path_1a:
-- Using NODEDATA fields:
 path_1a[1].node_posn = CURPOS(0, 0)
```

```
 cnt_dn_time = path_1a[node_no].weld_time
-- Using PATHHEADER fields:
 path_1a.utool = tool_1
```

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

A.17.2 PATH_LEN Built-In Function

Purpose: Returns the number of nodes in the PATH argument

Syntax : PATH_LEN(path_nam)

Function Return Type : INTEGER

Input/Output Parameters :

[in] path_nam : PATH

%ENVIRONMENT Group :PBCORE

Details:

- The returned value corresponds to the number of nodes in the PATH variable argument.
- Calling PATH_LEN with an uninitialized PATH returns a value of zero.

See Also: COPY_PATH Built-in

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.17.3 PAUSE Action

Purpose: Suspends execution of a running task

Draft

Syntax : PAUSE <PROGRAM[n]>

Details:

- The PAUSE action pauses task execution in the following manner:
  - Any motion already initiated continues until completed.
  - Files are left open.
  - All connected timers continue being incremented.
  - All PULSE statements in execution continue execution.
  - Sensing of conditions specified in condition handlers continues.
  - Any Actions, except routine call actions, are completed. Routine call actions are performed when the program is resumed.
- The PAUSE action can be followed by the clause PROGRAM[n], where n is the task number to be paused.
- Use GET_TSK_INFO to find a task number.

See Also: UNPAUSE Action

Draft

A.17.4 PAUSE Condition

Purpose: Monitors the pausing of program execution

Syntax : PAUSE < PROGRAM [n] >

Details:

- The PAUSE condition is satisfied when a program is paused, for example, by an error, a PAUSE Statement, or the PAUSE Action.
- If one of the actions corresponding to a PAUSE condition is a routine call, it is necessary to specify a NOPAUSE action to allow execution of the routine.

  Also, the routine being called needs to include a PAUSE statement so the system can handle completely the cause of the original pause.
- The PAUSE condition can be followed by the clause PROGRAM[n], where n is the task number to be paused.
- Use GET_TSK_INFO to find a task number.

Example: The following example scans for the PAUSE condition in a global condition handler. If this condition is satisfied, DOUT[1] will be turned on. The CONTINUE action continues program execution; ENABLE reenables the condition handler.

PAUSE Condition

```
CONDITION[1]:
  WHEN PAUSE DO
    DOUT[1] = TRUE
    CONTINUE
    ENABLE CONDITION[1]
ENDCONDITION
```

A.17.5 PAUSE Statement

Purpose: Suspends execution of a KAREL program

Syntax : PAUSE < PROGRAM [n] >

Details:

- The PAUSE statement pauses program execution in the following manner:
  - Any motion already initiated continues until completed.
  - Files are left open.
  - All connected timers continue being incremented.
  - All PULSE statements in execution continue execution.
  - Sensing of conditions specified in condition handlers continues.
  - Any actions, except routine call actions, are completed. Routine call actions are performed when the program is resumed.
- The PAUSE statement can be followed by the clause PROGRAM[n], where n is the task number to be paused.
- Use GET_TSK_INFO to find a task number.

See Also: Appendix E , "Syntax Diagrams," for more syntax information

Example: If DIN[1] is TRUE, the following example pauses the KAREL program using the PAUSE statement. The message, "Program is paused. Press RESUME function key to continue" will be displayed on the CRT/KB screen.

PAUSE Statement

```
PROGRAM p_pause
BEGIN
   IF DIN[1] THEN
      WRITE ('Program is Paused. ')
      WRITE ('Press RESUME function key to continue', CR)
      PAUSE
```

```
    ENDIF
END p_pause
```

Draft

A.17.6 PAUSE_TASK Built-In Procedure

Purpose: Pauses the specified executing task

Syntax : PAUSE_TASK(task_name, force_sw, stop_mtn_sw, status)

Input/Output Parameters:

[in] task_name :STRING

[in] force_sw :BOOLEAN

[in] stop_mtn_sw :BOOLEAN

[out] status :INTEGER

%ENVIRONMENT Group :MULTI

Details:

- *task_name* is the name of the task to be paused. If task name is '*ALL*', all executing tasks are paused except the tasks that have the "ignore pause request" attribute set.
- *force_sw* specifies whether a task should be paused even if the task has the "ignore pause request" attribute set. This parameter is ignored if task_name is '*ALL*'.
- *stop_mtn_sw* specifies whether all motion groups belonging to the specified task are stopped.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: RUN_TASK, CONT_TASK, ABORT_TASK Built-In Procedures, Chapter 15 *MULTI-TASKING*

Example: The following example pauses the user-specified task and stops any motion. Refer to Chapter 15 *MULTI-TASKING* , for more examples.

PAUSE_TASK Built-In Procedure

```
PROGRAM pause_ex
%ENVIRONMENT MULTI
VAR
  task_str: STRING[12]
  status  : INTEGER
BEGIN
  WRITE('Enter task name to pause:')
```

```
  READ(task_str)
  PAUSE_TASK(task_str, TRUE, TRUE, status)
END pause_ex
```

A.17.7 PEND_SEMA Built-In Procedure

Purpose: Suspends execution of the task until either the value of the semaphore is greater than zero or max_time expires

Syntax : PEND_SEMA(semaphore_no, max_time, time_out)

Input/Output Parameters:

[in] semaphore_no :INTEGER

[in] max_time :INTEGER

[out] time_out :BOOLEAN

%ENVIRONMENT Group :MULTI

Details:

- PEND_SEMA decrements the value of the semaphore.
- *semaphore_no* specifies the semaphore number to use.
- *semaphore_no* must be in the range of 1 to the number of semaphores defined on the controller.
- *max_time* specifies the expiration time, in milliseconds. A max_time value of -1 indicates to wait forever, if necessary.
- On continuation, *time_out* is set TRUE if max_time expired without the semaphore becoming nonzero, otherwise it is set FALSE.

See Also: POST_SEMA, CLEAR_SEMA Built-In Procedures, SEMA_COUNT Built-In Function, Chapter 15 *MULTI-TASKING*

Example: See examples in Chapter 15 *MULTI-TASKING*

A.17.8 PIPE_CONFIG Built-In Procedure

Purpose: Configure a pipe for special use.

Syntax : pipe_config(pipe_name, cmos_flag, n_sectors, record_size, form_dict, form_ele, status)

Draft

Input/Output Parameters :

[in] pipe_name :STRING

[in] cmos_flag :BOOLEAN

[in] n_sectors :INTEGER

[in] record_size :INTEGER

[in] form_dict :STRING

[in] form_ele :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :FLBT

Details:

- pipe_name is the name of the pipe file. If the file does not exist it will be created with this operation.
- CMOS_flag if set to TRUE will put the pipe data in CMOS. By default pipe data is in DRAM.
- n_sectors number of 1024 byte sectors to allocate to the pipe. The default is 8.
- record_size the size of a binary record in a pipe. If set to 0 the pipe is treated as ASCII. If a pipe is binary and will be printed as a formatted data then this must be set to the record length.
- form_dict is the name of the dictionary containing formatting information.
- form_ele is the element number in form_dict containing the formatting information.
- status explains the status of the attempted operation. If it is not equal to 0, then an error occurred.

See Also: For further information see "PIP: Device" in Section 9.3.4 .

A.17.9 POP_KEY_RD Built-In Procedure

Purpose: Resumes key input from a keyboard device

Syntax : POP_KEY_RD(key_dev_name, pop_index, status)

Input/Output Parameters:

[in] key_dev_name :STRING

[in] pop_index :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- Resumes all suspended read requests on the specified keyboard device.
- If there were no read requests active when suspended, this operation will not resume any inputs. This is not an error.
- *key_dev_name* must be one of the keyboard devices already defined:

  'TPKB' :Teach Pendant Keyboard Device

  'CRKB' :CRT Keyboard Device
- *pop_id* is returned from PUSH_KEY_RD and should be used to re-activate the read requests.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: PUSH_KEY_RD, READ_KB Built-in Procedures

Example: Refer to the example for the READ_KB Built-In Routine.

Draft

A.17.10 Port_Id Action

Purpose: Sets the value of a port array element to the result of an expression

Syntax : port_id[n] = expression

where:

port_id :an output port array

n :an INTEGER

expression :a variable, constant, or EVAL clause

Details:

- The value of *expression* is assigned to the port array element referenced by *n* .
- The port array must be an output port array that can be written to by a KAREL program. Refer to Chapter 2, "Language Elements."
- *expression* can be a user-defined, static variable, a system variable that can be read by a KAREL program, any constant, or an EVAL clause.

- If *expression* is a variable, the value used is the current value of the variable at the time the action is taken, not when the condition handler is defined.
- If *expression* is an EVAL clause, it is evaluated when the condition handler is defined and that value is assigned when the action is taken.
- The expression must be of the same type as *port_id* .
- You cannot assign a port array element to a port array element directly.
- If the expression is a variable that is uninitialized when the condition handler is enabled, the program will be aborted with an error.

See Also: Chapter 6 *CONDITION HANDLERS* , Chapter 7 *FILE INPUT/OUTPUT OPERATIONS* , Relational Conditions, Appendix A, “KAREL Language Alphabetical Description”

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.17.11 Port_Id Condition

Purpose: Monitors a digital port signal

Syntax : <NOT> port_id[n] < + | - >

where:

port_id :a port array

n :an INTEGER

Details:

- *n* specifies the port array signal to be monitored.
- *port_id* must be one of the predefined BOOLEAN port array identifiers with read access. Refer to Chapter 2, “Language Elements.”
- For event conditions, only the + or - alternatives are used.
- For state conditions, only the NOT alternative is used.

See Also: Chapter 6 *CONDITION HANDLERS* , Chapter 7 *FILE INPUT/OUTPUT OPERATIONS* , Relational Conditions, Appendix A, “KAREL Language Alphabetical Description”

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.17.12 POS Built-In Function

Purpose: Returns an XYZWPR composed of the specified location arguments (x,y,z), orientation arguments (w,p,r), and configuration argument (c)

Syntax : POS(x, y, z, w, p, r, c)

Function Return Type : XYZWPR

Input/Output Parameters:

[in] x, y, z, w, p, and r :REAL

[in] c :CONFIG

%ENVIRONMENT Group :SYStem

Details:

- *c* must be a valid configuration for the robot attached to the controller. CNV_STR_CONF can be used to convert a string to a CONFIG variable.
- *x* , *y* , and *z* are the Cartesian values of the location (in millimeters). Each argument must be in the range±10000000 mm (±10 km). Otherwise, the program is paused with an error.
- *w* , *p* , and *r* are the yaw, pitch, and roll values of the orientation (in degrees). Each argument must be in the range ±180 degrees. Otherwise, the program is paused with an error.

See Also: Chapter 8 *POSITION DATA*

Example: The following example uses the POS Built-In to designate numerically the POSITION **next_pos**.

POS Built-In Function

```
CNV_STR_CONF('n', config_var, status)
next_pos = POS(100,-400.25,0.5,10,-45,30,config_var)
```

A.17.13 POS2JOINT Built-In Function

Purpose: This routine is used to convert Cartesian positions (in_pos) to joint angles (out_jnt) by calling the inverse kinematics routine.

Syntax : POS2JOINT (ref_jnt, in_pos, uframe, utool, config_ref, wjnt_cfg, ext_ang, out_jnt, and status).

Input/Output Parameters:

[in] ref_jnt :JOINTPOS

[in] in_pos :POSITION

[in] uframe :POSITION

[in] utool :POSITION

[in] config_ref :INTEGER

[in] wjnt_cfg :CONFIG

[in] ext_ang :ARRAY OF REAL

[out] out_jnt :JOINTPOS

[out] status :INTEGER

%ENVIRONMENT Group :MOTN

Draft

Details:

- The input *ref_jnt* are the reference joint angles that represent the robot's position just before moving to the current position.
- The input *in_pos* is the robot Cartesian position to be converted to joint angles.
- The input *uframe* is the user frame for the Cartesian position.
- The input *utool* is the corresponding tool frame.
- The input *config_ref* is an integer representing the type of solution desired. The values listed below are valid. Also, the pre-defined constants in the parentheses can be used and the values can be added as required. One example includes: config_ref = HALF_SOLN + CONFIG_TCP.
  - 0 :(FULL_SOLN) = Default
  - 1 : (HALF_SOLN) = Wrist joint (XYZ456). This does not calculate/use WPR.
  - 2 :(CONFIG_TCP) = The Wrist Joint Config (up/down) is based on the fixed wrist.
  - 4 :(APPROX_SOLN) = Approximate solution. Reduce calculation time for some robots.
  - 8 :(NO_TURNS) = Ignore wrist turn numbers. Use the closest path for joints 4, 5 and 6 (uses ref_jnt).
  - 16 :(NO_M_TURNS) = Ignore major axis (J1 only) turn number. Use the closest path.
- The input *wjnt_cfg* is the wrist joint configuration. This value must be input when *config_ref* corresponds to HALF_SOLN.
- The input *ext_ang* contains the values of the joint angles for the extended axes if they exist.
- The output *out_jnt* are the joint angles that correspond to the Cartesian position

- The output *status* explains the status of the attempted operation. If it is not equal to 0, then an error has occurred.

A.17.14 POS_REG_TYPE Built-In Procedure

Purpose: Returns the position representation of the specified position register

Syntax : POS_REG_TYPE (register_no, group_no, posn_type, num_axes, status)

Input/Output Parameters :

[in] register : INTEGER

[in] group_no : INTEGER

[out] posn_type : INTEGER

[out] num_axes : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the position register.
- If *group_no* is omitted, the default group for the program is assumed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.
- *posn_type* returns the position type. posn_type is defined as follows:
  - 1 :POSITION
  - 2 :XYZWPR
  - 6 :XYZWPREXT
  - 9 :JOINTPOS
- *num_axes* returns number of axes in the representation if the position type is a JOINTPOS. If the position type is an XYZWPREXT, only the number of extended axes is returned by num_axes.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_POS_REG, GET_JPOS_REG, SET_POS_REG, SET_JPOS_REG Built-in Procedures

Draft

Example: The following example determines the position type in the register and uses the appropriate built-in to get data.

POS_REG_TYPE Built-In Procedure

```
PROGRAM get_reg_data
%NOLOCKGROUP
%ENVIRONMENT REGOPE
VAR
  entry: INTEGER
  group_no: INTEGER
  jpos: JOINTPOS
  maxpregnum: integer
  num_axes: INTEGER
  posn_type: INTEGER
  register_no: INTEGER
  status: INTEGER
  xyz: XYZWPR
  xyzext: XYZWPREXTBEGIN
  group_no = 1
  GET_VAR(entry, '*POSREG*' ,'$MAXPREGNUM', maxpregnum, status)
  -- Loop for each register
  FOR register_no = 1 to 10 DO
    -- Get the position register type
    POS_REG_TYPE(register_no, group_no, posn_type, num_axes, status)
    -- Get the position register
    WRITE('PR[', register_no, '] of type ', posn_type, CR)
    SELECT posn_type OF
      CASE (2):
        xyz = GET_POS_REG(register_no, status)
      CASE (6):
        xyzext = GET_POS_REG(register_no, status)
      CASE (9):
        jpos = GET_JPOS_REG(register_no, status)
      ELSE:
    ENDSELECT
  ENDFOR
END get_reg_data
```

A.17.15 POSITION Data Type

Purpose: Defines a variable, function return type, or routine parameter as POSITION data type

Syntax : POSITION <IN GROUP[n]>

Details:

Draft

- A POSITION consists of a matrix defining the normal, orient, approach, and location vectors and a component specifying a configuration string, for a total of 56 bytes.
- The configuration string indicates the joint placements and multiple turns that describe the configuration of the robot when it is at a particular position.
- A POSITION is always referenced with respect to a specific coordinate frame.
- The POSITION data type can be used to represent a frame of reference in which case the configuration component is ignored.
- Coordinate frame transformations can be done using the relative position operator (:).
- A POSITION can be assigned to other positional types.
- Valid POSITION operators are the
  - — Relative position (:) operator
  - — Approximately equal (>=<) operator
- A POSITION can be followed by IN GROUP[n], where n indicates the motion group with which the data is to be used. The default is the group specified by the %DEFGROUP directive, or 1.
- Components of POSITION variables can be accessed or set as if they were defined as follows:

Draft

POSITION Data Type

```
POSITION = STRUCTURE
     NORMAL:  VECTOR             -- read-only
     ORIENT:  VECTOR             -- read-only
     APPROACH:  VECTOR           -- read-only
     LOCATION:  VECTOR           -- read-write
     CONFIG_DATA:  CONFIG        -- read-write
   ENDSTRUCTURE
```

See Also: POS, UNPOS Built-In Functions

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.17.16 POST_ERR Built-In Procedure

Purpose: Posts the error code and reason code to the error reporting system to display and keep history of the errors

Syntax: POST_ERR(error_code, parameter, cause_code, severity)

Input/Output Parameters:

[in] error_code :INTEGER

[in] parameter :STRING

[in] cause_code :INTEGER

[in] severity :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *error_code* is the error to be posted.
- *parameter* will be included in *error_code's* message if %s is specified in the dictionary text. If not necessary, then enter the null string.
- *cause_code* is the reason for the error. 0 can be used if no cause is applicable.
- *error_code* and *cause_code* are in the following format:

  ```
  ffccc (decimal)
  ```

  where

  ff represents the facility code of the error.

  ccc represents the error code within the specified facility.
- *severity* is defined as follows:

  0 : WARNING, no change in task execution

  1 : PAUSE, all tasks and stop all motion

  2 : ABORT, all tasks and cancel

See Also: ERR_DATA Built-In Procedure, the appropriate application-specific *FANUC Robotics Setup and Operations Manual,* "Error Codes"

Example: Refer to Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL), for a detailed program example.

A.17.17 POST_ERR_L Built-In Procedure

Purpose: Posts the error code with local severity to the error reporting system to display and keep history of the errors

Draft

Syntax: POST_ERR_L(error_code, parameter, cause_code, severity)

Input/Output Parameters:

[in] error_code :INTEGER

[in] parameter :STRING

[in] cause_code :INTEGER

[in] severity :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *error_code* is the error to be posted.
- *parameter* will be included in *error_code's* message if %s is specified in the dictionary text. If not necessary, then enter the null string.
- *cause_code* is the reason for the error. 0 can be used if no cause is applicable.
- *error_code* and *cause_code* are in the following format:

```
ffccc (decimal)
```

  where

  ff represents the facility code of the error.

  ccc represents the error code within the specified facility.

- *severity* is defined as follows:

  ERSEV_NONE : No severity

  ERSEV_WARN : WARNING, no change in task execution

  ERSEV_PAUSE : PAUSE Global, pause all tasks and stop all motion after current motion segment

  ERSEV_PAUSEL : PAUSE Local, pause local task and stop all motion for local task after current motion segment

  ERSEV_STOP : STOP Global, pause all tasks and stop all motion

  ERSEV_STOPL : STOP Local, pause local task and stop all motion for local task

  ERSEV_SERVO : SERVO Global, turn off all servo power and pause all tasks

Draft

ERSEV_SERVOL : SERVO Local, turn off servo power for local task motion groups and pause local tasks

ERSEV_ABORT : ABORT Global, abort all tasks and cancel all motion

ERSEV_ABORTL : ABORT Local, abort local task and cancel all motion for local task

ERSEV_SYSTEM : SYSTEM Global, system problem exist and prevent any further operation

See Also: POST_ERR Built-In

A.17.18 POST_SEMA Built-In Procedure

Purpose: Add one to the value of the indicated semaphore

Syntax : POST_SEMA(semaphore_no)

Input/Output Parameters:

[in] semaphore_ no : INTEGER

%ENVIRONMENT Group : MULTI

Details:

- The semaphore indicated by *semaphore_no* is incremented by one.
- *semaphore_no* must be in the range of 1 to the number of semaphores defined on the controller.

See Also: PEND_SEMA, CLEAR_SEMA Built-In Procedures, SEMA_COUNT Built-In Function, Chapter 15 *MULTI-TASKING* ,

Example: See examples in Chapter 15 *MULTI-TASKING*

A.17.19 PRINT_FILE Built-In Procedure

Purpose: Prints the contents of an ASCII file to the default device

Syntax : PRINT_FILE(file_spec, nowait_sw, status)

Input/Output Parameters:

[in] file_spec :STRING

[in] nowait_sw :BOOLEAN

[out] status :INTEGER

%ENVIRONMENT Group : FDEV

Details:

- *file_spec* specifies the device, name, and type of the file to print.
- If *nowait_sw* is TRUE, execution of the program continues while the command is executing. If it is FALSE, the program stops, including condition handlers, until the operation has completed. If you have time critical condition handlers in your program, put them in another program that executes as a separate task.

  Note *nowait_sw* is not available in this release and should be set to FALSE.

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Draft

A.17.20 %PRIORITY Translator Directive

Purpose: Specifies task priority

Syntax : %PRIORITY = n

Details:

- *n* is the priority and is defined as follows:
```
1 to 89 : lower than motion, higher than user interface
90 to 99 : lower than user interface
```

- The lower the value, the higher the task priority.
- The default priority is 50. Refer to Section 15.4 , "Task Scheduling" for more information on how the specified priority is converted into the system priority.
- The priority can be set during task execution by the SET_TSK_ATTR Built-In routine.

Example: Usually an error handling task pends on an error and when an error occurs, it processes the error recovery as soon as possible. In this case, the error handling task might need to have a higher priority than other tasks, so “n” should be less than 50.

%PRIORITY Translator Directive

```
%PRIORITY = 49
```

A.17.21 PROG_BACKUP Built-In Procedure

Purpose: Saves the specified program and all called programs from execution memory to a storage device. If the called programs call other programs they will be saved recursively. You can specify that any associated program variables be saved.

Syntax: PROG_BACKUP (file_spec, prog_type, max_size, write_prot, status)

Input/Output Parameters

[in] file_spec :STRING

[in] prog_type :INTEGER

[in] max_size: INTEGER

[in] write_prot: BOOLEAN

[out] status :INTEGER

%ENVIRONMENT Group: CORE

Details:

- *file_spec* specifies the device and program to save. If a file type is specified, it is ignored.
- *prog_type* specifies the type of programs to be saved. The valid types are:

  PBR_VRTYPE :VR - programs which contain variables

  PBR_MNTYPE :JB, PR, MR, TP

  PBR_JBTYPE :JB - job programs only

  PBR_PRTYPE :PR - process programs only

  PBR_MRTYPE :MR - macro programs only

  PBR_PCTYPE : VR - saves VR files not PC files

  PBR_ALLTYPE :all programs VR, JB, PR, MR, TP

  PBR_NVRTYPE :all programs except VR

  PBR_NMRTYPE :JB, PR, TP (all TPs except Macros)
- *max_size* specifies the maximum size of disk space in kilobytes required to backup the programs. If not enough memory is available on the storage device, no programs will be backed up and status will equal 2002, “FILE-002 Device is Full”.

Draft

- If the required disk space to backup the programs exceeds max_size the backup will continue. The backup might still fail if there is not enough space to save all the programs. The return status will equal 2002, " FILE-002 Device is Full". In this case a partial backup will exist. To prevent this case be sure that max_size is large enough to prevent this error.
- *write_prot* , if true, specifies that write protected programs should be saved. If false, specifies that write protected programs should not be saved.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- The system will stay in the loop and handle as many programs as it can even when it gets an error. For example, if there is one missing program out of eight, the remaining seven programs are saved. In this case the error "Program does not exist" is returned to the user in status. An error is posted in this case with the program name in the error for each program. The cause code is whatever is returned from the save routine.
- If a subdirectory is specified on the storage device, it will be created if it does not already exist. All programs will be saved into the subdirectory.
- If a file already exists but the no changes have occurred, the file is not overwritten.
- If a file already exists but the program has been changed, it will be overwritten and no error is returned.
- A KAREL or teach pendant program of the same name with variables must exist in memory as a called program or else the system will not save the VR.
- The PROG_BACKUP, PROG_CLEAR and PROG_RESTORE builtins consider all references to programs except for macros. This includes any programs referenced in the following statements: CALL, RUN, ERROR_PROG, RESUME_PROG, and MONITOR.

Example: The following example saves ANS00003 with the appropriate extension to GMX_211 subdirectory on FR: device. It will save all programs that are called recursively by ANS00003 regardless of program type. It will not save KAREL variables. It will fail if there is less than 200k of free space on the FR: device.

```
VAR
    status: INTEGER
BEGIN
    PROG_BACKUP('FR:\GMX_211\ANS00003', PBR_NVRTYPE, 200, TRUE, status)
```

Example : The following example saves ANS00003 with the appropriate file extension to GMX_211 subdirectory on FR: device. It will save JB, PR, MR, or TP programs that are called recursively by ANS00003. It will not save write-protected programs. It will fail if there is less than 100k of free space on the FR: device.

Draft

```
VAR
    status: INTEGER
BEGIN
    PROG_BACKUP('FR:\GMX_211\ANS00003', PBR_MNTYPE, 100, FALSE, status)
```

Example : The following example saves MAIN to MC: device with the appropriate file extension. It will save all programs and variables that are called recursively by MAIN. It will fail if there is less than 300k of free space on the MC: device.

```
VAR
    status: INTEGER
BEGIN
    PROG_BACKUP('MC:\MAIN', PBR_ALLTYPE, 300, TRUE, status)
```

Draft

A.17.22 PROG_CLEAR Built-In Procedure

Purpose: Clear the specified program and all called programs from execution memory. If the called programs call other programs they will be cleared recursively. You can specify that any associated program variables also be cleared. Variables which are referenced from other programs will not be cleared.

Syntax: PROG_CLEAR (prog_name, prog_type, status)

Input/Output Parameters:

[in] prog_name :STRING

[in] prog_type :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group: CORE

Details:

- *prog_name* specifies the root program which is to be cleared.
- *prog_type* specifies the type of programs to be cleared. The valid types are:

  PBR_VRTYPE :VR - programs which contain variables

  PBR_MNTYPE :JB, PR, MR, TP

  PBR_JBTYPE :JB - job programs only

  PBR_PRTYPE :PR - process programs only

  PBR_MRTYPE :MR - macro programs only

  PBR_PCTYPE : VR - saves VR files not PC files

  PBR_ALLTYPE :all programs VR, JB, PR, MR, TP

  PBR_NVRTYPE :all programs except VR

  PBR_NMRTYPE :JB, PR, TP (all TPs except Macros)
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- The system will stay in the loop and clear as many programs as it can. If one of the called programs is missing, this is not an error. However, if that missing program calls other programs those other programs will not be found and will not be cleared. Errors are posted with the program name in the error for each program which is not cleared. The cause code is whatever is returned from the clear routine.

Draft

- Clearing of any VR data is subject to being referenced by another program. This error will be ignored for any variable clear operation.
- If a programs which is identified for clearing is the selected program it will not be cleared. The error “Program is in use” is returned in this case. As a countermeasure the use must use SELECT_TPE() built-in to select a program which is not in the clear set.
- The PROG_BACKUP, PROG_CLEAR and PROG_RESTORE builtins consider all references to programs except for macros. This includes any programs referenced in the following statements: CALL, RUN, ERROR_PROG, RESUME_PROG, and MONITOR.

Example: The following example clears ANS00003.TP from memory. It will clear all programs that are called recursively by ANS00003 regardless of program type and clear them from memory. It will not clear write-protected programs. It will not clear any KAREL variables.

```
VAR
    status: INTEGER
BEGIN
    PROG_CLEAR('ANS00003.TP', PBR_NVRTYPE, status)
```

Draft

Example: The following example clears ANS00003.TP program from memory. It will clear only JB, PR, MR, and TP programs that are called recursively by ANS00003. It will not clear write-protected programs.

```
VAR
    status: INTEGER
BEGIN
        PROG_CLEAR('ANS00003.TP', PBR_MNTYPE, status)
```

A.17.23 PROG_RESTORE Built-In Procedure

Purpose: Restores (loads) the specified program and all called programs into execution memory. If the called programs call other programs they will be loaded recursively. Any associated program variables will also be loaded if the VR files exist.

Syntax: PROG_RESTORE (file_spec, status)

Input/Output Parameters:

[in] file_spec :STRING

[out] status :INTEGER

%ENVIRONMENT Group: CORE

Details:

- *file_spec* specifies the storage device and program to restore. If a file type is specified, it is ignored.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- The system will stay in the loop and handles as many programs as it can even when it gets an error. For example, if there is one missing program out of eight, the remaining seven programs are loaded. In this case the error "File does not exist" is returned to the user in status. An error is posted in this case with the program name in the error for each program. The cause code is whatever is returned from the load routine.
- If a subdirectory is specified, the called programs are loaded from that subdirectory. Any extra files in that subdirectory will not automatically be loaded.
- If the program already exists, it will not be restored and no error is returned.
- A KAREL or TP program of the same name must already exist in memory as a called program or else the system will not load the VR.
- VR types will be restored even if the variables already exist. That is the system will overwrite any existing variable values with the values saved in the VR file.
- ASCII programs (.LS) cannot be restored.
- If not enough memory is available, then an error is returned and the restore is incomplete.
- The PROG_BACKUP, PROG_CLEAR and PROG_RESTORE builtins consider all references to programs except for macros. This includes any programs referenced in the following statements: CALL, RUN, ERROR_PROG, RESUME_PROG, and MONITOR.

Example: The following example restores ANS00003.TP from GMX_211 subdirectory on FR: device. It will restore all programs that are called recursively by ANS00003 regardless of program type. It will restore VR files if they are in the restore directory.

Draft

```
VAR
    status: INTEGER
BEGIN
    PROG_RESTORE('FR:\GMX_211\ANS00003.TP', status)
```

Example: The following example restores ANS00003.TP from GMX_211 subdirectory on FR: device. It will restore only TP programs that are called recursively by ANS00003.

```
VAR
    status: INTEGER
BEGIN
    PROG_RESTORE('FR:\GMX_211\ANS00003', status)
```

Example: The following example restores MAIN from MC: device by finding its file type. It will restore all programs and variables that are called recursively by MAIN.

```
  VAR
    status: INTEGER

BEGIN
    PROG_RESTORE('MC:\MAIN', status)
```

Draft

A.17.24 PROG_LIST Built-In Procedure

Purpose: Returns a list of program names.

Syntax : prog_list(prog_name, prog_type, n_skip, format, ary_name, n_progs <,f_index>)

Input/Output Parameters :

[in] prog_name :STRING

[in] prog_type :INTEGER

[in] n_skip :INTEGER

[in] format :INTEGER

[out] ary_name :ARRAY of string

[out] n_progs :INTEGER

[out] status :INTEGER

[in,out] f_index :INTEGER

%ENVIRONMENT Group :BYNAM

Details:

- *prog_name* specifies the name of the program(s) to be returned in *ary_name* . *prog_name* may use the wildcard (*) character, to indicate that all programs matching the *prog_type* should be returned in *ary_name* .
- *prog_type* specifies the type of programs to be retrieved. The valid types are:

  1 :VR - programs which contain only variables

  2 :JB, PR, MR, TP3 :JB - job programs only

  4 :PR - process programs only

  5 :MR - macro programs only

  6 :PC - KAREL programs only

  7 :all programs VR, JB, PR, MR, TP, PC

  8 :all programs except VR

Draft

- *n_skip* This parameter can only be used when using * for the program name and 7 for the program type. Otherwise used the f_index parameter for multiple calls when more programs exist than the declared length of ary_name. Set n_skip to 0 the first time you use PROG_LIST. If ary_name is completely filled with program names, copy the array to another ARRAY of STRINGS and execute PROG_LIST again with n_skip equal to n_skip + n_progs . The call to PROG_LIST will then skip the programs found in the previous passes and locate only the remaining programs..
- *format* specifies the format of the program name. The following values are valid for *format* :

  1 :program name only, no blanks

  2 :'program name program type'

  total length = 15 characters
- *prog_name* = 12 characters followed by a space
- *prog_type* = 2 characters
- *ary_name* is an ARRAY of STRING used to store the program names.
- *n_progs* is the number of variables stored in the *ary_name* .
- *status* will return zero if successful.
- *f_index* is an optional parameter for fast indexing. If you specify prog_name as a complex wildcard (anything other than the straight *), then you should use this parameter. The first call to PROG_LIST set *f_index* and *n_skip* both to zero. *f_index* will then be used internally to quickly find the next prog_name. DO NOT change *f_index* once a listing for a particular *prog_name* has begun.

See Also: VAR_LIST Built-In Procedure

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.17.25 PROGRAM Statement

Purpose: Identifies the program name in a KAREL source program

Syntax : PROGRAM prog_name

where:

prog_name : a valid KAREL identifier

Details:

- It must be the first statement (other than comments) in a program.
- The identifier used to name a program cannot be used in the program for any other purpose, such as to identify a variable or constant.
- *prog_name* must also appear in the END statement that marks the end of the executable section of the program.
- The program name can be used to call the program as a procedure routine from within a program in the same way routine names are used to call procedure routines.

Example: Refer to Appendix B , "KAREL Example Programs," for more detailed examples of how to use the PROGRAM Statement.

A.17.26 PULSE Action

Purpose: Pulses a digital output port for a specified number of milliseconds

Syntax : PULSE DOUT[port_no] FOR time_in_ms

where:

port_no : an INTEGER variable or literal

time_in_ms : an INTEGER

Details:

- *port_no* must be a valid digital output port number.
- *time_in_ms* specifies the duration of the pulse in milliseconds.
- If *time_in_ms* duration is zero, no pulse will occur. Otherwise, the period is rounded up to the next multiple of 8 milliseconds.
- A pulse always turns on the port at the start of the pulse and turns off the port at the end of the pulse.
- If the port is “normally on,” negative pulses can be accomplished by setting the port to reversed polarity, or by executing the following sequence:

```
DOUT[n] = FALSE
DELAY x
DOUT[n] = TRUE
```

- NOWAIT is not allowed in a PULSE action.
- If the program is paused while a pulse is in progress, the pulse will end at the correct time.

Draft

- If the program is aborted while a pulse is in progress, the port stays in whatever state it was in when the abort occurred.
- If *time_in_ms* is negative or greater than 86,400,000 (24 hours), the program is aborted with an error.

See Also: Chapter 6 *CONDITION HANDLERS* , Chapter 7 *FILE INPUT/OUTPUT OPERATIONS*

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.17.27 PULSE Statement

Purpose: Pulses a digital output port for a specified number of milliseconds.

Syntax : PULSE DOUT[port_no] FOR time_in_ms < NOWAIT >

where:

port_no : an INTEGER variable or literal

time_in_ms : an INTEGER

Details:

- *port_no* must be a valid digital output port number.
- *time_in_ms* specifies the duration of the pulse in milliseconds.
- If *time_in_ms* duration is zero, no pulse will occur. Otherwise, the period is rounded up to the next multiple of 8 milliseconds.

  The actual duration of the pulse will be from zero to 8 milliseconds less than the rounded value.

  For example, if 100 is specified, it is rounded up to 104 (the next multiple of 8) milliseconds. The actual duration will be from 96 to 104 milliseconds.
- A pulse always turns on the port at the start of the pulse and turns off the port at the end of the pulse.
- If the port is "normally on," negative pulses can be accomplished by setting the port to reversed polarity, or by executing the following sequence:

```
DOUT[n] = FALSE
DELAY x
DOUT[n] = TRUE
```

- If NOWAIT is specified in a PULSE statement, the next KAREL statement will be executed concurrently with the pulse.
- If NOWAIT is not specified in a PULSE statement, the next KAREL statement will not be executed until the pulse is completed.

See Also: Appendix E , "Syntax Diagrams" for more syntax information

Example: In the following example a digital output is pulsed, followed by the pulsing of a second digital output. Because NOWAIT is specified, **DOUT[start_air]** will be executed before **DOUT[5]** is completed.

PULSE Statement

```
PULSE DOUT[5] FOR (seconds * 1000) NOWAIT
PULSE DOUT[start_air] FOR 50 NOWAIT
```

Draft

A.17.28 PURGE CONDITION Statement

Purpose: Deletes the definition of a condition handler from the system

Syntax : PURGE CONDITION[cond_hand_no]

where:

cond_hand_no : an INTEGER expression

Details:

- The statement has no effect if there is no condition handler defined with the specified number.
- The PURGE CONDITION Statement is used only to purge global condition handlers.
- The PURGE CONDITION Statement will purge enabled conditions.
- If a condition handler with the specified number was previously defined, it must be purged before it is replaced with a new one.

See Also: ENABLE CONDITION Statement Chapter 6 *CONDITION HANDLERS* , Appendix E , "Syntax Diagrams" for more syntax information

Example: In the following example, if the BOOLEAN variable **ignore_cond** is TRUE, the global condition handler, CONDITION[1], will be purged using the PURGE statement; otherwise CONDITION[1] is enabled.

PURGE CONDITION Statement

```
IF ignore_cond THEN
  PURGE CONDITION[1]
```

```
ELSE
  ENABLE CONDITION[1]
ENDIF
```

A.17.29 PURGE_DEV Built-In Procedure

Purpose: Purges the specified memory file device by freeing any used blocks that are no longer needed

Syntax : PURGE_DEV (device, status)

Input/Output Parameters :

[in] device : STRING

[out] status : INTEGER

%ENVIRONMENT Group :FDEV

Details:

- *device* specifies the memory file device to purge. *device* should be set to ’FR:’ for FROM disk, ’RD:’ for RAM disk, or ’MF:’ for both disks.
- The purge operation is only necessary when the device does not have enough memory to perform an operation. The ’FR:’ device will return 85001 if the FROM disk is full. The ’RD:’ device will return 85020 if the RAM disk is full.
- The purge operation will erase file blocks that were previously used, but no longer needed. These are called garbage blocks. The FROM disk may contain many garbage blocks if files are deleted or overwritten. The RAM disk does not normally contain garbage blocks, but they can occur when power is removed during a file copy.
- The VOL_SPACE built-in can be used to determine the number of garbage blocks on the FROM disk. Hardware limitations may reduce the number of blocks actually freed.
- The device must be mounted and no files can be open on the device or an error will be returned.
- *status* explains the status of the attempted operation. If not equal to 0 then an error occurred. 85023 is returned if no errors occurred, but no blocks were purged.

Example: Return to Section B.9 , "Using the File and Device Built-Ins" (FILE_EX.KL), for a more detailed program example.

Draft

A.17.30 PUSH_KEY_RD Built-In Procedure

Purpose: Suspend key input from a keyboard device

Syntax : PUSH_KEY_RD(key_dev_name, key_mask, pop_index, status)

Input/Output Parameters:

[in] key_dev_name :STRING

[in] key_mask :INTEGER

[out] pop_index :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- Suspends all read requests on the specified keyboard device that uses (either as accept_mask or term_mask) any of the specified key classes.
- If there are no read requests active, a null set of inputs is recorded as suspended. This is not an error.
- *key_dev_name* must be one of the keyboard devices already defined:

  ’TPKB’ :Teach Pendant Keyboard Device

  ’CRKB’ :CRT Keyboard Device
- *key_mask* is a bit-wise mask indicating the classes of characters that will be suspended. This should be an OR of the constants defined in the include file klevkmsk.kl.

  kc_display :Displayable keys

  kc_func_key :Function keys

  kc_keypad :Keypad and Edit keys

  kc_enter_key :Enter and Return keys

  kc_delete :Delete and Backspace keys

  kc_lr_arw :Left and Right Arrow keys

  kc_ud_arw :Up and Down Arrow keys

  kc_other :Other keys (such as Prev)

Draft

- *pop_id* is returned and should be used in a call to POP_KEY_RD to re-activate the read requests.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: POP_KEY_RD Built-In Procedure

Example: Refer to the READ_KB Built-In Procedure for an example.

Draft

A.18 - Q - KAREL LANGUAGE DESCRIPTION

A.18.1 QUEUE_TYPE Data Type

Purpose: Defines the data type for use in QUEUE built-in routines

Syntax : queue_type = STRUCTURE

n_entries : INTEGER

sequence_no : INTEGER

head : INTEGER

tail : INTEGER

ENDSTRUCTURE

Details:

- *queue_type* is used to initialize and maintain queue data for the QUEUE built-in routines. **Do not change this data; it is used internally.**

See Also: APPEND_QUEUE, DELETE_QUEUE, INSERT_QUEUE, COPY_QUEUE, GET_QUEUE, INIT_QUEUE, MODIFY_QUEUE Built-In Procedures

A.19 - R - KAREL LANGUAGE DESCRIPTION

A.19.1 READ Statement

Purpose: Reads data from a serial I/O device or file

Syntax : READ < file_var > (data_item {,data_item})

where:

file_var : a FILE variable

data_item : a variable identifier and its optional format specifiers or the reserved word CR

Details:

- If *file_var* is not specified in a READ statement the default TPDISPLAY is used. %CRTDEVICE directive will change the default to INPUT.
- If *file_var* is specified, it must be one of the input devices (INPUT, CRTPROMPT, TPDISPLAY, TPPROMPT) or a variable that was set in the OPEN FILE statement.
- If *file_var* attribute was set with the UF option, data is transmitted into the specified variables in binary form. Otherwise, data is transmitted as ASCII text.
- *data_item* can be a system variable that has RW access or a user-defined variable.
- When the READ statement is executed, data is read beginning with the next nonblank input character and ending with the last character before the next blank, end of line, or end of file for all input types except STRING.
- If *data_item* is of type ARRAY, a subscript must be provided.
- If *data_item* is of type PATH, you can specify that the entire path be read, a specific node be read ([n]), or a range of nodes be read ([n .. m]).
- Optional format specifiers can be used to control the amount of data read for each *data_item* . The effect of format specifiers depends on the data type of the item being read and on whether the data is in text (ASCII) or binary (unformatted) form.
- The reserved word CR, which can be used as a data item, specifies that any remaining data in the current input line is to be ignored. The next data item will be read from the start of the next input line.
- If reading from a file and any errors occur during input, the variable being read and all subsequent variables up to CR in the data list are set uninitialized.
- If *file_var* is a window device and any errors occur during input, an error message is displayed indicating the bad data item and you are prompted to enter a replacement for the invalid data item and to reenter all subsequent items.
- Use IO_STATUS (file_var) to determine if the read operation was successful.

Note Read CR should never be used in unformatted mode.

See Also: Chapter 7 *FILE INPUT/OUTPUT OPERATIONS* , for more information on the READ format specifiers, IO_STATUS Built-In Function, Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)

Draft

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.19.2 READ_DICT Built-In Procedure

Purpose: Reads information from a dictionary

Syntax : READ_DICT(dict_name, element_no, ksta, first_line, last_line, status)

Input/Output Parameters:

[in] dict_name : STRING

[in] element_no : INTEGER

[out] ksta : ARRAY OF STRING

[in] first_line : INTEGER

[out] last_line : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :PBCORE

Draft

Details:

- *dict_name* specifies the name of the dictionary from which to read.
- *element_no* specifies the element number to read. This element number is designated with a $ in the dictionary file.
- *ksta* is a KAREL STRING ARRAY used to store the information being read from the dictionary text file.
- If *ksta* is too small to store all the data, then the data is truncated and status is set to 33008, "Dictionary Element Truncated."
- *first_line* indicates the array element of *ksta* , at which to begin storing the information.
- *last_line* returns a value indicating the last element used in the *ksta* array.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred reading the element from the dictionary file.
- &new_line is the only reserved attribute code that can be read from dictionary text files using READ_DICT. The READ_DICT Built-In ignores all other reserved attribute codes.

See Also: ADD_DICT, WRITE_DICT, REMOVE_DICT Built-In Procedures. Refer to the program example for the DISCTRL_LIST Built-In Procedure. Chapter 10 *DICTIONARIES AND FORMS*

Example: Refer to Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL), for a detailed program example.

A.19.3 READ_DICT_V Built-In-Procedure

Purpose: Reads information from a dictionary with formatted variables

Syntax : READ_DICT_V(dict_name, element_no, value_array, ksta, status)

Input/Output Parameters:

[in] dict_name : STRING

[in] element_no : INTEGER

[in] value_array : ARRAY OF STRING

[out] ksta : ARRAY OF STRING

[out] status : INTEGER

%ENVIRONMENT Group :UIF

Details:

- *dict_name* specifies the name of the dictionary from which to read.
- *element_no* specifies the element number to read. This number is designated with a $ in the dictionary file.
- *value_array* is an array of variable names that corresponds to each formatted data item in the dictionary text. Each variable name can be specified as '[prog_name]var_name'.
  - *[prog_name]* specifies the name of the program that contains the specified variable. If not specified, then the current program being executed is used.
  - *var_name* must refer to a static variable.
  - *var_name* may contain node numbers, field names, and/or subscripts.
- *ksta* is a KAREL STRING ARRAY used to store the information that is being read from the dictionary text file.
- If *ksta* is too small to store all the data, then the data is truncated and status is set to 33008, "Dictionary Element Truncated."
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred reading the element from the dictionary file.
- &new_line is the only reserved attribute code that can be read from dictionary text files using READ_DICT_V. The READ_DICT_V Built-In ignores all other reserved attribute codes.

See Also: WRITE_DICT_V Built-In Procedure, Chapter 10 *DICTIONARIES AND FORMS*

Example: In the following example, **TPTASKEG.TX** contains dictionary text information which will display a system variable. This information is the first element in the dictionary. Element numbers start at 0. **util_prog** uses READ_DICT_V to read in the text and display it on the teach pendant.

READ_DICT_V Built-In Procedure

```
----------------------------------------------
TPTASKEG.TX
----------------------------------------------
$ "Maximum number of tasks = %d"
----------------------------------------------
UTILITY PROGRAM:
----------------------------------------------
PROGRAM util_prog
  %ENVIRONMENT uif
  VAR
    ksta: ARRAY[1] OF STRING[40]
    status: INTEGER
    value_array: ARRAY[1] OF STRING[30]
  BEGIN
    value_array[1] = '[*system*].$scr.$maxnumtask'
    ADD_DICT('TPTASKEG', 'TASK', dp_default, dp_open, status)
    READ_DICT_V('TASK', 0, value_array, ksta, status)
    WRITE(ksta[i], cr)
END util_prog
```

A.19.4 READ_KB Built-In Procedure

Purpose: Read from a keyboard device and wait for completion

Syntax : READ_KB(file_var, buffer, buffer_size, accept_mask, term_mask, time_out, init_data, n_chars_got, term_char, status)

Input/Output Parameters:

[in] file_var : FILE

[out] buffer : STRING

[in] buffer_size : INTEGER

[in] accept_mask : INTEGER

[in] time_out : INTEGER

[in] term_mask : INTEGER

[in] init_data : STRING

[out] n_chars_got : INTEGER

[out] term _char : INTEGER

[out] stat us : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- Causes data from specified classes of characters to be stored in a user-supplied buffer until a termination condition is met or the buffer is full. Returns to the caller when the read is terminated.
- If you use READ_KB for the CRT/KB, you will get "raw" CRT characters returned. To get teach pendant equivalent key codes, you must perform the following function:

  ```
  tp_key = $CRT_KEY_TBL[crt_key + 1]
  ```

  This mapping allows you to use common software between the CRT/KB and teach pendant devices.
- READ_KB and some other utilities use a variable in your KAREL program called device_stat to establish the association between the KAREL program and user interface display. For example, if you have a task [MAINUIF] which calls READ_KB, the variable which is used to make the association is [MAINUIF]device_stat. If you do not set device_stat, then you can only read characters in single screen mode, or in the left pane.
- device_stat must be set to the paneID in which your application is running. For the standard single mode/monochrome pendant, device_stat =1. To interact in the right pane, set device_stat=2. To interact in the lower right pane, set device_stat=3. External Internet Explorer connections use panes 4-9. For the CRT/KB, set device_stat=255.
- [MAINUIF]device_stat must be set to the correct pane ID before you open the keyboard file that is assoicated with READ_KB. The pane ID for the iPendant can be either 1, 2 or 3.
- The application running in Pane ID 1 is stored in $TP_CURSCRN. Pane ID 2 is stored in $UI_CURSCRN[1], or in general $UI_CURSCRN[device_stat-1]. The CRT applicaton uses $CT_CURSCRN.
- *file_var* must be open to a keyboard-device. If *file_var* is also associated with a window, the characters are echoed to the window.
- The characters are stored in *buffer* , up to a maximum of *buffer_size* or the size of the string, whichever is smaller.
- *accept_mask* is a bit-wise mask indicating the classes of characters that will be accepted as input. This should be an OR of the constants defined in the include file klevkmsk.kl.

Draft

kc_display :Displayable keys

kc_func_key :Function keys

kc_keypad :Key-pad and Edit keys

kc_enter_key :Enter and Return keys

kc_delete :Delete and Backspace keys

kc_lr_arw :Left and Right Arrow keys

kc_ud_arw :Up and Down Arrow keys

kc_other :Other keys (such as Prev)

- It is reasonable for *accept_mask* to be zero; this means that no characters are accepted as input. This is used when waiting for a single key that will be returned as the term_char. In this case, *buffer_size* would be zero.
- If *accept_mask* includes displayable characters, the following characters, if accepted, have the following meanings:
  - — Delete characters - If the cursor is not in the first position of the field, the character to the left of the cursor is deleted.
  - — Left and right arrows - Position the cursor one character to the left or right from its present position, assuming it is not already in the first or last position already.
  - — Up and down arrows - Fetch input previously entered in reads to the same file.
- *term_mask* is a bit-wise mask indicating conditions which will terminate the request. This should be an OR of the constants defined in the include file klevkmsk.kl.

kc_display :Displayable keys

kc_func_key :Function keys

kc_keypad :Key-pad and Edit keys

kc_enter_key :Enter and Return keys

kc_delete :Delete and Backspace keys

kc_lr_arw :Left and Right Arrow keys

kc_ud_arw :Up and Down Arrow keys

kc_other :Other keys (such as Prev)

- *time_out* specifies the time, in milliseconds, after which the input operation will be automatically canceled. A value of -1 implies no timeout.

Draft

- *init_data_p* points to a string which is displayed as the initial value of the input field. This must not be longer than buffer_size.
- *n_chars_got* is set to the number of characters in the input buffer when the read is terminated.
- *term_char* receives a code indicating the character or other condition that terminated the form. The codes for key terminating conditions are defined in the include file klevkeys.kl. Keys normally returned are pre-defined constants as follows:

  ky_up_arw

  ky_dn_arw

  ky_rt_arw

  ky_lf_arw

  ky_enter

  ky_prev

  ky_f1

  ky_f2

  ky_f3

  ky_f4

  ky_f5

  ky_next
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Example: Refer to Section B.2 , "Copying Path Variables" (CPY_PTH.KL), for a detailed program example.

The following example suspends any teach pendant reads, uses READ_KB to read a single key, and then resumes any suspended reads.

READ_KB Built-In Procedure

```
PROGRAM readkb
%NOLOCKGROUP
%ENVIRONMENT flbt
%ENVIRONMENT uif
%INCLUDE FR:eklevkmsk
VAR
  file_var: FILE
  key: INTEGER
```

```
  n_chars_got: INTEGER
  pop_index: INTEGER
  status: INTEGER
  str: STRING[1]

BEGIN
  -- Suspend any outstanding TP Keyboard reads
  PUSH_KEY_RD('TPKB', 255, pop_index, status)
  IF (status = 0) THEN
    WRITE (CR, 'pop_index is ', pop_index)
  ELSE
    WRITE (CR, 'PUSH_KEY_RD status is ', status)
  ENDIF
  -- Open a file to TP Keyboard with PASALL and FIELD attributes
  -- and NOECHO
  SET_FILE_ATR(file_var, ATR_PASSALL)
  SET_FILE_ATR(file_var, ATR_FIELD)
  OPEN FILE file_var ('RW', 'KB:TPKB')
  -- Read a single key from the TP Keyboard
  READ_KB(file_var, str, 1, 0, kc_display+kc_func_key+kc_keypad+
          kc_enter_key+kc_lr_arw+kc_ud_arw+kc_other, 0, '',
          n_chars_got, key, status)
  IF (status = 0) THEN
    WRITE (CR, 'key is ', key, ', n_chars_got = ', n_chars_got)
  ELSE
    WRITE (CR, 'READ_KB status is ', status)
  ENDIF
  CLOSE FILE file_var
  -- Resume any outstanding TP Keyboard reads
  POP_KEY_RD('TPKB', pop_index, status)
  IF (status <> 0) THEN
    WRITE (CR, 'POP_KEY_RD status is ', status)
  ENDIF
END readkb
```

A.19.5 REAL Data Type

Purpose: Defines a variable, function return type, or routine parameter as a REAL data type with a numeric value that includes a decimal point and a fractional part, or numbers expressed in scientific notation

Syntax : REAL

Details:

- REAL variables and expressions can have values in the range of -3.4028236E+38 through -1.175494E-38, 0.0, and from +1.175494E-38 through +3.4028236E+38, with approximately seven decimal digits of significance. Otherwise, the program will be aborted with the “Real overflow” error.
- The decimal point is mandatory when defining a REAL constant or literal (except when using scientific notation). The decimal point is not mandatory when defining a REAL variable as long as it was declared as REAL.
- Scientific notation is allowed and governed by the following rules:
  - — The decimal point is shifted to the left so that only one digit remains in the INTEGER part.
  - — The fractional part is followed by the letter E (upper or lower case) and ±an INTEGER. This part specifies the magnitude of the REAL number. For example, 123.5 is expressed as 1.235E2.
  - — The fractional part and the decimal point can be omitted. For example, 100.0 can be expressed as 1.000E2, as 1.E2, or 1E2.
- All REAL variables with magnitudes between -1.175494E- 38 and +1.175494E-38 are treated as 0.0.
- Only REAL or INTEGER expressions can be assigned to REAL variables, returned from REAL function routines, or passed as arguments to REAL parameters.
- If an INTEGER expression is used in any of these instances, it is treated as a REAL value. If an INTEGER variable is used as an argument to a REAL parameter, it is always passed by value, not by reference.
- Valid REAL operators are (refer to Table A–18):
  - — Arithmetic operators (+, +, *, /)
  - — Relational operators (>, >=, =, < >, <, <=)

Table A–18. Valid and Invalid REAL operators

VALID	INVALID	REASON
1.5	15	Decimal point is required (15 is an INTEGER not a REAL)
1.	.	Must include an INTEGER or a fractional part
+2500.450	+2,500.450	Commas not allowed
1.25E-4	1.25E -4	Spaces not allowed

Example: Refer to the following sections for detailed program examples:

Section B.5 ,"Using Register Built-ins" (REG_EX.KL)

Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.11 , "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.19.6 Relational Condition

Draft

Purpose: Used to test the relationship between two operands

Syntax : variable <[subscript]> rel_op expression

where:

variable : a static INTEGER or REAL variable or a BOOLEAN port array element

subscript : an INTEGER expression (only used with port arrays)

rel_op : a relational operator

expression : a static variable, constant, or EVAL clause

Details:

- Relational conditions are state conditions, meaning the relationship is tested during every scan.
- The following relational operators can be used:

  = :equal

  <> :not equal

  < :less than

  =< :less than or equal

  > :greater than

  >= :greater than or equal
- Both operands must be of the same data type and can only be of type INTEGER, REAL, or BOOLEAN. INTEGER values can be used where REAL values are required, and will be treated as REAL values.
- *variable* can be any of the port array signals, a user-defined static variable, or a system variable that can be read by a KAREL program.
- *expression* can be a user-defined static variable, a system variable that can be read by a KAREL program, any constant, or an EVAL clause.
- Variables used in relational conditions must be initialized before the condition handler is enabled.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.19.7 RELAX HAND Statement

Purpose: Turns off open signal for a tool controlled by one signal or turns off both open and close signals for a tool controlled by a pair of signals.

Syntax : RELAX HAND hand_num

where:

hand_num : an INTEGER expression

Details:

- The actual effect of the statement depends on how the HAND signals are set up. Refer to Chapter 13, “Input/Output System.”
- *hand_num* must be a value in the range 1-2. Otherwise, the program is aborted with an error.
- The statement has no effect if the value of *hand_num* is in range but the hand is not connected.
- If the value of *hand_num* is in range but the HAND signal represented by that value has not been assigned, the program is aborted with an error.

See Also: Chapter 13, “Input/Output System,” Appendix D, “Syntax Diagrams,” for more syntax information

Example: In the following example, the robot hand, specified by **gripper** , is relaxed using the RELAX HAND statement. The robot then moves to the POSITION **pstart** before closing the hand.

RELAX HAND Statement

```
PROGRAM p_release
%NOPAUSE=TPENABLE
%ENVIRONMENT uif
BEGIN
 RELAX HAND gripper
 SET_POS_REG(1, pstart, status) –– Put position in PR[1]
 move_to_pr1 –– Call TP program to move to PR[1]
 CLOSE HAND gripper
END p_release
```

A.19.8 RELEASE Statement

Purpose: Releases all motion control of the robot arm and auxiliary or extended axes from the KAREL program so that they can be controlled by the teach pendant while a KAREL program is running

Syntax : RELEASE

Details:

- Motion stopped prior to execution of the RELEASE statement can only be resumed after the execution of the next ATTACH statement.
- If motion is initiated from the program while in a released state, the program is aborted with the following error, “MCTRL Denied because Released.”
- If RELEASE is executed while motion is in progress or in a HOLD condition, the program is aborted with the following error, "Detach request failed."
- All motion control from all KAREL tasks will be released.

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.19.9 REMOVE_DICT Built-In Procedure

Purpose: Removes the specified dictionary from the specified language or from all existing languages

Syntax : REMOVE_DICT(dict_name, lang_name, status)

Input/Output Parameters:

[in] dict_name : STRING

[in] lang_name : STRING

[out] status : INTEGER

%ENVIRONMENT Group :UIF

Details:

- *dict_name* specifies the name of the dictionary to remove.
- *lang_name* specifies which language the dictionary should be removed from. One of the following pre-defined constants should be used:

dp_default

dp_english

dp_japanese

dp_french

dp_german

dp_spanish

If *lang_name* is '', it will be removed from all languages in which it exists.

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred removing the dictionary file.

See Also: ADD_DICT Built-In Procedure, Chapter 10 *DICTIONARIES AND FORMS*

Example: Refer to Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL), for a detailed program example.

A.19.10 RENAME_FILE Built-In Procedure

Purpose: Renames the specified file name

Syntax : RENAME_FILE(old_file, new_file, nowait_sw, status)

Input/Output Parameters:

[in] old_file : STRING

[in] new_file : STRING

[in] nowait_sw : BOOLEAN

[out] status : INTEGER

%ENVIRONMENT Group :FDEV

Details:

- *old_file* specifies the device, name, and type of the file to rename.
- *new_file* specifies the name and type of the file to rename to.
- If *nowait_sw* is TRUE, execution of the program continues while the command is executing. If it is FALSE, the program stops, including condition handlers, until the operation has completed.

If you have time critical condition handlers in your program, put them in another program that executes as a separate task.

Note *nowait_sw* is not available in this release and should be set to FALSE.

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: COPY_FILE, DELETE_FILE Built-In Procedures

A.19.11 RENAME_VAR Built-In Procedure

Purpose: Renames a specified variable in a specified program to a new variable name

Syntax : RENAME_VAR(prog_nam, old_nam, new_nam, status)

Input/Output Parameters:

[in] prog_nam : STRING

[in] old_nam : STRING

[in] new_nam : STRING

[out] status : INTEGER

%ENVIRONMENT Group :MEMO

Details:

- *prog_nam* is the name of the program that contains the variable to be renamed.
- *old_nam* is the current name of the variable.
- *new_nam* is the new name of the variable.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CREATE_VAR, SET_VAR Built-In Procedures

A.19.12 RENAME_VARS Built-In Procedure

Purpose: Renames all of the variables in a specified program to a new program name

Syntax : RENAME_VARS(old_nam, new_nam, status)

Input/Output Parameters:

[in] old_nam : STRING

[in] new_nam : STRING

[out] status : INTEGER

%ENVIRONMENT Group :MEMO

Details:

- *old_nam* is the current name of the program.
- *new_nam* is the new name of the program.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CREATE_VAR, RENAME_VARS Built-in Procedures

Draft

A.19.13 REPEAT ... UNTIL Statement

Purpose: Repeats statement(s) until a BOOLEAN expression evaluates to TRUE

Syntax : REPEAT

{ statement }

UNTIL boolean_exp

where:

statement : a valid KAREL executable statement

boolean_exp : a BOOLEAN expression

Details:

- *boolean_exp* is evaluated after execution of the statements in the body of the REPEAT loop to determine if the statements should be executed again.
- *statement* continues to be executed and the *boolean_exp* is evaluated until it equals TRUE.
- *statement* will always be executed at least once.

Caution

Make sure your REPEAT statement contains a boolean flag that is modified by some condition, and an UNTIL statement that terminates the loop. If it does not, your program could loop infinitely.

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.9 , "Using the File and Device Built-ins" (FILE_EX.KL)

Section B.11 , "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.19.14 RESET Built-In Procedure

Purpose: Resets the controller

Syntax : RESET(successful)

Input/Output Parameters:

[out] successful : BOOLEAN

%ENVIRONMENT Group :MOTN

Details:

- *successful* will be TRUE even if conditions exist which prevent resetting the controller.
- To determine whether the reset operation was successful, delay 1 second and check OPOUT[3] (FAULT LED). If this is FALSE, the reset operation was successful.
- The statement following the RESET Built-In is not executed until the reset fails or has completed. The status display on the CRT or teach pendant will indicate PAUSED during the reset.
- The controller appears to be in a PAUSED state while a reset is in progress but, during this time, PAUSE condition handlers will not be triggered.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.19.15 RESUME Action

Purpose: Restarts the last stopped motion issued by the task

Syntax : RESUME <GROUP[n{,n}]>

Details:

- A motion set is a group of motions issued but not yet terminated when a STOP statement or action is issued.
- If there are no stopped motion sets, no motion will result from the RESUME.
- If more than one motion set has been stopped, RESUME restarts the most recently stopped, unresumed motion set. Subsequent RESUMEs will start the others in last-in-first-out sequence.
- The motions contained in a stopped motion set are resumed in the same order in which they were originally issued.
- If a motion is in progress when the RESUME action is issued, any resumed motion(s) occur after the current motion is completed.
- If the group clause is not present, all groups for which the task has control (when the condition is defined) will be resumed.
- If the motion that is stopped, resumed, canceled, or held is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are stopped, resumed, canceled, or held.
- Motion cannot be resumed for a different task.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.19.16 RESUME Statement

Purpose: Restarts the last stopped motion issued by the task

Syntax : RESUME <GROUP[n{,n}]>

- A motion set is a group of motions issued but not yet terminated when a STOP statement or action is issued.
- If there are no stopped motion sets, no motion will result from the RESUME.
- If more than one motion set has been stopped, RESUME restarts the most recently stopped, unresumed motion set. Subsequent RESUMEs will start the others in last-in-first-out sequence.
- Those motions in a stopped motion set are resumed in the same order in which they were originally issued.

Draft

- If a motion is in progress when the RESUME statement is issued, any resumed motion(s) occur after the current motion is completed.
- If the group clause is not present, all groups for which the task has control will be resumed.
- If the motion that is stopped, resumed, canceled, or held is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are stopped, resumed, canceled, or held.
- Motion cannot be resumed for a different task.

See Also: Appendix E , “Syntax Diagrams” for more syntax information

Example: In the following example, motion is stopped if **DIN[1]** is ON. It is resumed after F1 is pressed.

RESUME Statement

```
CONDITION[1]:
  WHEN DIN[1] = ON DO
    STOP
  ENDCONDITION
move_to_pr1 -- Call TP program to move to PR[1]
IF DIN[1] THEN
  WRITE(' Motion stopped')
  WRITE(CR, 'Motion and the program will resume')
  WRITE(CR, '    when F1 of teach pendant is pressed')
  WAIT FOR TPIN[129]
  RESUME
ENDIF
```

Draft

A.19.17 RETURN Statement

Purpose: Returns control from a routine/program to the calling routine/program, optionally returning a result

Syntax : RETURN < (value) >

Details:

- *value* is required when returning from functions, but is not permitted when returning from procedures. The data type of *value* must be the same as the type used in the function declaration.
- If a main program executes a RETURN statement, execution is terminated and cannot be resumed. All motions in progress will be completed normally.
- If no RETURN is specified, the END statement serves as the return.

- If a function routine returns with the END statement instead of a RETURN statement, the program is aborted with the 12321 error, “END STMT of a func rtn.”

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

Draft

A.19.18 ROUND Built-In Function

Purpose: Returns the INTEGER value closest to the specified REAL argument

Syntax : ROUND(x)

Function Return Type :INTEGER

Input/Output Parameters:

[in] x :REAL

%ENVIRONMENT Group :SYStem

Details:

- The returned value is the INTEGER value closest to the REAL value x, as demonstrated by the following rules:
  - If x >= 0, let n be a positive INTEGER such that n <= x <= n + 1
  - If x >= n + 0.5, then n + 1 is returned; otherwise, n is returned.
  - If x <= 0, let n be a negative INTEGER such that n >= x >= n - 1
  - If x <= n - 0.5, then n - 1 is returned; otherwise, n is returned.
- *x* must be in the range of -2147483648 to +2147483646. Otherwise, the program will be aborted with an error.

See Also: TRUNC Built-In Function

Example: Refer to Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_.EX.KL), for a detailed program example.

A.19.19 ROUTINE Statement

Purpose: Specifies a routine name, with parameters and types, and a returned value data type for function routines

Syntax : ROUTINE name < param_list > <: return_type >

where:

name : a valid KAREL identifier

param_list : described below

return_type : any data type that can be returned by a function, that is, any type except FILE, PATH, and vision types

Details:

- *name* specifies the routine name.
- *param_list* is of the form (*name_group { ; name_group* })
  - *name_group* is of the form *param_name : param_type*
  - *param_name* is a parameter which can be used within the routine body as a variable of data type *param_type*.
  - If a *param_type* or *return_type* is an ARRAY, the size is excluded. If the *param_type* is a STRING, the string length is excluded.
- When the routine body follows the ROUTINE statement, the names in *param_list* are used to associate arguments passed in with references to parameters within the routine body.
- When a routine is from another program, the names in the parameter list are of no significance but must be present in order to specify the number and data types of parameters.
- If the ROUTINE statement contains a *return_type,* the routine is a function routine and returns a value. Otherwise, it is a procedure routine.
- The ROUTINE statement must be followed by a routine body or a FROM clause.

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.4 ,"Standard Routines" (ROUT_EX.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)

Draft

Section B.9 , "Using the File and Device Built-ins" (FILE_EX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.19.20 RUN_TASK Built-In Procedure

Purpose: Runs the specified program as a child task

Syntax : RUN_TASK (task_name, line_number, pause_on_sft, tp_motion, lock_mask, status)

Input/Output Parameters:

[in] task_name : STRING

[in] line_number : INTEGER

[in] pause_on_sft : BOOLEAN

[in] tp_motion : BOOLEAN

[in] lock_mask : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :MULTI

Details:

- *task_name* is the name of the task to be run. This creates a child task. The task that executes this built-in is called the parent task.
- If the task already exists and is paused, it will be continued. A new task is not created.
- *line_number* specifies the line from which execution starts. Use 0 to start from the beginning of the program. This is only valid for teach pendant programs.
- If *pause_on_sft* is TRUE, the task is paused when the teach pendant shift key is released.
- If *tp_motion* is TRUE, the task can execute motion while the teach pendant is enabled. The TP must be enabled if *tp_motion* is TRUE.
- The control of the motion groups specified in *lock_mask* will be transferred from parent task to child task, if *tp_motion* is TRUE and the teach pendant is enabled. The group numbers must be in

Draft

the range of 1 to the total number of groups defined on the controller. Bit 1 specifies group 1, bit 2 specifies group 2, and so forth.

Table A–19. Group_mask setting

GROUP	DECIMAL	BIT
Group 1	1	1
Group 2	2	2
Group 3	4	3

To specify multiple groups select the decimal values, shown in Table A–19 , which correspond to the desired groups. Then connect them together using the OR operator. For example to specify groups 1 and 3, enter "1 OR 4".

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CONT_TASK, PAUSE_TASK, ABORT_TASK Built-In Procedures, Chapter 15 *MULTI-TASKING*

Example: Refer to Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL), for a detailed program example.

Draft

A.20 - S - KAREL LANGUAGE DESCRIPTION

A.20.1 SAVE Built-In Procedure

Purpose: Saves the program or variables into the specified file

Syntax : SAVE (prog_nam, file_spec, status)

Input/Output Parameters :

[in] prog_nam :STRING

[in] file_spec :STRING

[out] status :INTEGER

%ENVIRONMENT Group :MEMO

Details:

- *prog_nam* specifies the program name. If program name is '*', all programs or variables of the specified type are saved. prog_name must be set to "*SYSTEM*" in order to save all system variables.

- *file_spec* specifies the device, name, and type of the file being saved to. The type also implies whether programs or variables are being saved.

The following types are valid:

.TP : Teach pendant program

.VR : KAREL variables

.SV : KAREL system variables

.IO : I/O configuration data

- If *file_spec* already exists on the specified device, then an error is returned the save does not occur.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CLEAR, LOAD Built-In Procedures

Example: Refer to Section B.3 ,"Saving Data to the Default Device" (SAVE_VRS.KL), for a detailed program example.

A.20.2 SAVE_DRAM Built-In Procedure

Purpose: Saves the RAM variable content to FlashROM.

Syntax: SAVE_DRAM (prog_nam, status)

Input/Output Parameters:

[in] prog_nam: STRING

[out] status: INTEGER

%ENVIRONMENT Group: MEMO

Details:

- prog_nam specifies the program name. This operation will save the current values of any variables in DRAM to FlashROM for the specified program. At power up these saved values will automatically be loaded into DRAM.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

A.20.3 SELECT ... ENDSELECT Statement

Purpose: Permits execution of one out of a series of statement sequences, depending on the value of an INTEGER expression.

Syntax: SELECT case_val OF

CASE(value{,value}):

{statement}

{ CASE(value{, value}):

{statement} }

<ELSE:

{ statement }>

ENDSELECT

where:

case_val : an INTEGER expression

value : an INTEGER constant or literal

statement : a valid KAREL executable statement

Details:

- *case_val* is compared with each of the values following the CASE in each clause. If it is equal to any of these, the statements between the CASE and the next clause are executed.
- Up to 1000 CASE clauses can be used in a SELECT statement.
- If the same INTEGER value is listed in more than one CASE, only the statement sequence following the first matching CASE will be executed.
- If the ELSE clause is used and the expression *case_val* does not match any of the values in the CASE clauses, the statements between the keywords ELSE and ENDSELECT are executed.
- If no ELSE clause is used and the expression *case_val* does not match any of the values in the CASE clauses, the program is aborted with the “No match in CASE” error.

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.3 , "Saving Data to the Default Device" (SAVE_VR.KL)

Draft

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

A.20.4 SELECT_TPE Built-In Procedure

Purpose: Selects the program of the specified name

Syntax : SELECT_TPE(prog_name, status)

Input/Output Parameters :

[in] prog_name :STRING

[out] status : :INTEGER

%ENVIRONMENT Group :TPE

Details:

- *prog_name* specifies the name of the program to be selected as the teach pendant default. This is the program that is "in use" by the teach pendant. It is also the program that will be executed if the CYCLE START button is pressed or the teach pendant FWD key is pressed.
- *status* explains the status of the attempted operation. If it is not equal to 0, then an error has occurred.

See Also: OPEN_TPE Built-in Procedure

Example: Refer to Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL), for a detailed program example.

A.20.5 SEMA_COUNT Built-In Function

Purpose: Returns the current value of the specified semaphore

Syntax : SEMA_COUNT (semaphore_no)

Function Return Type :INTEGER

Input/Output Parameters :

[in] semaphore_no : INTEGER

%ENVIRONMENT Group :MULTI

Draft

Details:

- The value of the semaphore indicated by *semaphore_no* is returned.
- This value is incremented by every POST_SEMA call and SIGNAL SEMAPHORE Action specifying the same *semaphore_no* . It is decremented by every PEND_SEMA call.
- If SEMA_COUNT is greater than zero, a PEND_SEMA call will "fall through" immediately. If it is *-n* (minus n), then there are *n* tasks pending on this semaphore.

See Also: POST_SEMA, PEND_SEMA, CLEAR_SEMA Built-In Procedures, Chapter 15 *MULTI-TASKING*

Example: See examples in Chapter 15 *MULTI-TASKING*

A.20.6 SEMAPHORE Condition

Purpose: Monitors the value of the specified semaphore

Syntax : SEMAPHORE[semaphore_no]

Details:

- *semaphore_no* specifies the semaphore number to use.
- *semaphore_no* must be in the range of 1 to the number of semaphores defined on the controller.
- When the value of the indicated semaphore is greater than zero, the condition is satisfied (TRUE).

A.20.7 SEND_DATAPC Built-In Procedure

Purpose: To send an event message and other data to the PC.

Syntax : SEND_DATAPC(event_no, dat_buffer, status)

Input/Output Parameters :

[in] event_no :INTEGER

[in] dat_buffer :ARRAY OF BYTE

[out] status :INTEGER

%ENVIRONMENT Group :PC

Details:

Draft

- *event_no* - a GEMM event number. Valid values are 0 to 255.
- *dat_buffer* - an array of up to 244 bytes. The KAREL built-ins ADD_BYNAMEPC, ADD_INTPC, ADD_REALPC, and ADD_STRINGPC can be used to format a KAREL byte buffer. The actual data buffer format depends on the needs of the PC. There is no error checking of the dat_buffer format on the controller.
- *status* - the status of the attempted operation. If not 0, then an error occurred and the event request was not sent to the PC.

See Also: ADD_BYNAMEPC, ADD_INTPC, ADD_REALPC, ADD_STRINGPC

Example: The following example sends event 12 to the PC with a data buffer.

SEND_DATAPC Built-In Procedure

```
PROGRAM TESTDATA
%ENVIRONMENT PC
CONST
  er_abort = 2
VAR
  dat_buffer:   ARRAY[100] OF BYTE
  index:        INTEGER
  status:       INTEGER
BEGIN
  index = 1
  ADD_INTPC(dat_buffer,index,55,status)
  ADD_REALPC(dat_buffer,index,123.5,status)
  ADD_STRINGPC(dat_buffer,index,'YES',status)

  -- send event 12 and data buffer to PC
  SEND_DATAPC(12,dat_buffer,status)
  IF status<>0 THEN
    POST_ERR(status,'',0,er_abort)
  ENDIF
END testdata
```

A.20.8 SEND_EVENTPC Built-In Procedure

Purpose: To send an event message to the PC.

Syntax : SEND_EVENTPC(event_no, status)

Input/Output Parameters :

[in] event_no :INTEGER

Draft

[out] status :INTEGER

%ENVIRONMENT Group :PC

Details:

- *event_no* - a GEMM event number. Valid values are 0 through 255.
- *status* - the status of the attempted operation. If not 0, then an error occurred and the event request was not sent to the PC.

Example: The following example sends event 12 to the PC.

SEND_EVENTPC Built-In Procedure

```
PROGRAM TESTEVT
%ENVIRONMENT PC
CONST
  er_abort = 2
VAR
  status:  INTEGER
BEGIN
  -- send event 12 to PC
  SEND_EVENTPC(12,status)     -- call built-in here
  IF status<>0 THEN
    POST_ERR(status,'',0,er_abort)
  ENDIF
END testevt
```

A.20.9 SET_ATTR_PRG Built-In Procedure

Purpose: Sets attribute data of the specified teach pendant or KAREL program

Syntax : SET_ATTR_PRG(program_name, attr_number, int_value, string_value, status)

Input/Output Parameters :

[in] program_name : STRING

[in] attr_number : INTEGER

[in] int_value : INTEGER

[in] string_value : STRING

[out] status : INTEGER

Draft

%ENVIRONMENT Group :TPE

Details:

- *program_name* specifies the program to which attribute data is set.
- *attr_number* is the attribute whose value is to be set. The following attributes are valid:

  AT_PROG_TYPE : (#) Program type

  AT_PROG_NAME : Program name (String[12])

  AT_OWNER : Owner (String[8])

  AT_COMMENT : Comment (String[16])

  AT_PROG_SIZE : (#) Size of program

  AT_ALLC_SIZE : (#) Size of allocated memory

  AT_NUM_LINE : (#) Number of lines

  AT_CRE_TIME : (#) Created (loaded) time

  AT_MDFY_TIME : (#) Modified time

  AT_SRC_NAME : Source file (or original file) name (String[128])

  AT_SRC_VRSN : Source file version

  AT_DEF_GROUP : Default motion groups (for task attribute)

  AT_PROTECT : Protection code; 1 :protection OFF; 2 : protection ON

  AT_STORAGE : Storage type; TPSTOR_CMOS; TPSTOR_SHADOW; TPSTOR_FILE; TPSTOR_SHOD

  AT_STK_SIZE : Stack size (for task attribute)

  AT_TASK_PRI : Task priority (for task attribute)

  AT_DURATION : Time slice duration (for task attribute)

  AT_BUSY_OFF : Busy lamp off (for task attribute)

  AT_IGNR_ABRT : Ignore abort request (for task attribute)

  AT_IGNR_PAUS : Ignore pause request (for task attribute)

Draft

AT_CONTROL : Control code (for task attribute)

(#) --- Cannot be set.

- If the attribute data is a number, it is set to *int_value* and *string_value* is ignored.
- If the attribute data is a string, it is set to *string_value* and *int_value* is ignored.
- *status* explains the status of the attempted operation. If it is not equal to 0, then an error has occurred. Some of the errors which could occur are:

  7073 The program specified in *program_name* does not exist

  7093 The attribute of a program cannot be set while it is running

  17033 *attr_number* has an illegal value or cannot be set

Draft

A.20.10 SET_CURSOR Built-In Procedure

Purpose: Set the cursor position in the window

Syntax : SET_CURSOR(file_var, row, col, status)

Input/Output Parameters :

[in] file_var : FILE

[in] row : INTEGER

[in] col : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- Sets the current cursor of the specified file that is open to a window so subsequent writes will start in the specified position.
- *file_var* must be open to a window.
- A *row* value of 1 indicates the top row of the window. A *col* value of 1 indicates the left-most column of the window.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: DEF_WINDOW Built-In Procedure

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)

Draft

A.20.11 SET_EPOS_REG Built-In Procedure

Purpose: Stores an XYZWPREXT value in the specified register

Syntax : SET_EPOS_REG(register_no, posn, status <, group_no>)

Input/Output Parameters :

[in] register_no : INTEGER

[in] posn : XYZWPREXT

[out] status : INTEGER

[in] group_no :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the position register in which to store the value.
- The *position* data is set in XYZWPREXT representation.
- *status* explains the status of the attempted operation. If it is not equal to 0, then an error occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

See Also: SET_POS_REG, SET_JPOS_REG Built-In Procedures, GET_POS_REG, GET_JPOS_REG Built-In Functions

Example: The following example sets the extended position for the specified register.

SET_EPOS_REG Built-In Procedure

```
PROGRAM spe
%environment REGOPE
VAR
  cur_pos: XYZWPREXT
```

```
  posget: XYZWPREXT
  status: INTEGER
  v_mask, g_mask: INTEGER
  reg_no: INTEGER
BEGIN
  reg_no = 1
  cur_pos = CURPOS(v_mask,g_mask)
  SET_EPOS_REG(reg_no,cur_pos,status)
  posget = GET_POS_REG(reg_no,status)
END spe
```

Draft

A.20.12 SET_EPOS_TPE Built-In Procedure

Purpose: Stores an XYZWPREXT value in the specified position in the specified teach pendant program

Syntax: SET_EPOS_TPE (open_id, position_no, posn, status <,group_no>)

Input/Output Parameters :

[in] open_id : INTEGER

[in] position_no : INTEGER

[in] posn : XYZWPREXT

[out] status : INTEGER

[in] group_no : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *open_id* specifies the opened teach pendant program. A program must be opened before calling this built-in.
- *position_no* specifies the position in the program in which to store the value.
- A motion instruction must already exist that uses the *position_no* or the position will not be used by the teach pendant program.
- The position data is set in XYZWPREXT representation with no conversion.
- *status* explains the status of the attempted operation. If not equal to 0, then an error has occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.

- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

A.20.13 SET_FILE_ATR Built-In Procedure

Purpose: Sets the attributes of a file before it is opened

Syntax : SET_FILE_ATR(file_id, atr_type <,atr_value>)

Input/Output Parameters :

[in] file_id: FILE

[in] atr_type : INTEGER expression

[in] atr_value : INTEGER expression

%ENVIRONMENT Group :PBCORE

Details:

- *file_id* is the file variable that will be used in the OPEN FILE, WRITE, READ, and/or CLOSE FILE statements.
- *atr_type* specifies the attribute type to set. The predefined constants as specified in Table 7-2 should be used.
- *atr_value* is optional depending on the attribute type being set.

XML Related

Purpose: Sets the attributes file to XML before it is opened

Syntax : SET_FILE_ATR(xml_file, ATR_XML)

Input/Output Parameters :

[in] xml_file: FILE

[in] ATR_XML: INTEGER expression

Details:

- *xml_file* is the file variable that will be used in the OPEN FILE, WRITE, READ, and/or CLOSE FILE statements.
- *ATR_XML* specifies the attribute type to set. The predefined constants as specified in Table 7-2 should be used.

Draft

- The file must then be opened as a RO file. You cannot do any other XMl operations until the file has been opened.

See Also: SET_PORT_ATR Built-In Function, Section Section 7.3.1 ., “Setting File Attributes" and Section Section 9.5 , “Formatting XML Input"

Example: Refer to Section Section 9.5 , “Formatting XML Input"

Draft

A.20.14 SET_FILE_POS Built-In Procedure

Purpose: Sets the file position for the next READ or WRITE operation to take place in the specified file to the value of the new specified file position

Syntax : SET_FILE_POS(file_id, new_file_pos, status)

Input/Output Parameters :

[in] file_id : FILE

[in] new_file_pos : INTEGER expression

[out] status : INTEGER variable

%ENVIRONMENT Group :FLBT

Details:

- The file associated with *file_id* must be opened and uncompressed on either the FROM or RAM disks. Otherwise, the program is aborted with an error.
- *new_file_pos* must be in the range of -1 to the number of bytes in the file.at_eof : specifies that the file position is to be set at the end of the file. at_sof : specifies that the file position is to be set at the start of the file.
  — Any other value causes the file to be set the specified number of bytes from the beginning of the file.
- *status* is set to 0 if the *new_file_pos* is between -1 and the number of bytes in the file, indicating the file position was successfully set. If not equal to 0, then an error occurred.

See Also: Chapter 7 *FILE INPUT/OUTPUT OPERATIONS*

Example: The following example opens the **filepos.dt** data file, sets the file position from a directory, reads the positions from the file, and stores the positions in the PATH, **my_path** .

SET_FILE_POS Built-In Procedure

```
OPEN FILE file_id ('RW','filepos.dt')
```

```
FOR i = 1 TO PATH_LEN(my_path) DO
  SET_FILE_POS(file_id, pos_dir[i], status)
  IF status = 0 THEN
    READ file_id (temp_pos)
    my_path[i].node_pos = temp_pos
  ENDIF
ENDFOR
```

A.20.15 SET_INT_REG Built-In Procedure

Purpose: Stores an integer value in the specified register

Syntax : SET_INT_REG(register_no, int_value, status)

Input/Output Parameters :

[in] register_no : INTEGER

[in] int_value : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the register into which *int_value* will be stored.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_INT_REG, GET_REAL_REG, SET_REAL_REG Built-in Procedures

Example: Refer to Section B.5 , "Using Register Built-ins" (REG_EX.KL), for a detailed program example.

A.20.16 SET_JPOS_REG Built-In Procedure

Purpose: Stores a JOINTPOS value in the specified register

Syntax : SET_JPOS_REG(register_no, jpos, status<, group_no>)

Input/Output Parameters :

[in] register_no : INTEGER

[in] jpos : JOINTPOS

[out] status :INTEGER

[in] group_no :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the position register in which to store the position, *jpos* .
- The position data is set in JOINTPOS representation with no conversion.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

See Also: GET_JPOS_REG, GET_POS_REG, SET_POS_REG, POS_REG_TYPE Built-in Procedures

Example: Refer to Section B.5 ,"Using Register Built-ins" (REG_EX.KL), for a detailed program example.

A.20.17 SET_JPOS_TPE Built-In Procedure

Purpose: Stores a JOINTPOS value in the specified position in the specified teach pendant program

Syntax : SET_JPOS_TPE(open_id, position_no, posn, status<, group_no>)

Input/Output Parameters :

[in] open_id : INTEGER

[in] position_no : INTEGER

[in] posn : JOINTPOS

[out] status : INTEGER

[in] group_no :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *open_id* specifies the opened teach pendant program. Before calling this built-in, a program must be opened using the OPEN_TPE Built-In, and have read/write access.
- *position_no* specifies the position in the program in which to store the value.
- The position data is set in JOINTPOS representation with no conversion.
- *status* explains the status of the attempted operation. If not equal to 0, then an error has occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

See Also: GET_JPOS_TPE, GET_POS_TPE, SET_POS_TPE, GET_POS_TYP Built-in Procedures

Example: Refer to Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY.TP.KL), for a detailed program example.

Draft

A.20.18 SET_LANG Built-In Procedure

Purpose: Changes the current language

Syntax : SET_LANG(lang_name, status)

Input/Output Parameters :

[in] lang_name :STRING

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *lang_name* specifies which language from which the dictionaries should be read/written. Use one of the following pre-defined constants:

  dp_default

  dp_english

  dp_japanese

  dp_french

  dp_german

dp_spanish

- The read-only system variable $LANGUAGE indicates which language is currently in use.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred setting the language.
- The error, 33003, "No dict found for language," will be returned if no dictionaries are loaded into the specified language. The KCL command "SHOW LANGS" can be used to view which languages are created in the system.

See Also: Chapter 10 *DICTIONARIES AND FORMS*

Example: Refer to Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL), for a detailed program example.

Draft

A.20.19 SET_PERCH Built-In Procedure

Purpose: Sets the perch position and tolerance for a group of axes

Syntax : SET_PERCH(jpos, tolerance, indx)

Input/Output Parameters :

[in] jpos : JOINTPOS

[in] tolerance : ARRAY[6] of REAL

[in] indx : INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- The values of *jpos* are converted to radians and stored in the system variable $REFPOS1[*indx*].$perch_pos.
- The *tolerance* array is converted to degrees and stored in the system variable $REFPOS1[*indx*].$perchtol. If the tolerance array is uninitialized, an error is generated.
- *indx* specifies the element number to be set in the $REFPOS1 array.
- The group of axes is implied from the specified position, *jpos* . If JOINTPOS is not in group 1, then the system variable $REFPOSn is used where n corresponds to the group number of *jpos* and *indx* must beset to 1.

See Also: The appropriate application-specific *FANUC Robotics Setup and Operations Manual* to setup Reference Positions

Example: In the following example, $REFPOS1[2].$perchpos and $REFPOS1[2].$perchtol are set according to **perch_pos** and **tolerance[i].**

SET_PERCH Built-In Procedure

```
VAR
  perch_pos: JOINTPOS IN GROUP[1]
BEGIN
  FOR i = 1 to 6 DO
    tolerance[i] = 0.01
  ENDFOR
  SET_PERCH (perch_pos, tolerance, 2)
END
```

Draft

A.20.20 SET_PORT_ASG Built-In Procedure

Purpose: Allows a KAREL program to assign one or more logical ports to specified physical port(s)

Syntax : SET_PORT_ASG(log_port_type, log_port_no, rack_no, slot_no, phy_port_type, phy_port_no, n_ports, status)

Input/Output Parameters :

[in] log_port_type : INTEGER

[in] log_port_no : INTEGER

[in] rack_no : INTEGER

[in] slot_no : INTEGER

[in] phy_port_type : INTEGER

[in] phy_port_no : INTEGER

[in] n_ports : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

- *log_port_type* specifies the code for the type of port to be assigned. Codes are defined in KLIOTYPS.KL.

- *log_port_no* specifies the number of the port to be assigned.
- *rack_no* is the rack containing the port module. For process I/O boards, memory-image, and dummy ports, this is zero; for Allen-Bradley and Genius ports, this is 16.
- *slot_no* is the slot containing the port module. For process I/O boards, this is the sequence in the SLC-2 chain. For memory-image and dummy ports, this is zero; for Allen-Bradley and Genius ports, this is 1.
- *phy_port_type* is the type of port to be assigned to. Often this will be the same as *log_port_type* . Exceptions are if *log_port_type* is a group type (*io_gpin* or *io_gpout*) or a port is assigned to memory-image or dummy ports.
- *phy_port_no* is the number of the port to be assigned to. If *log_port_type* is a group, this is the port number for the least-significant bit of the group.
- *n_ports* is the number of physical ports to be assigned to the logical port. If *log_port_type* is a group type, *n_ports* indicates the number of bits in the group. When setting digital I/O, *n_ports* is the number of points you are configuring. In most cases this will be 8, but may be 1 through 8.
- *status* is returned with zero if the parameters are valid. Otherwise, it is returned with an error code. The assignment is invalid if the specified port(s) do not exist or if the assignment of *log_port_type* to *phy_port_type* is not permitted.

For example, GINs cannot be assigned to DOUTs. Neither *log_port_type* nor *phy_port_type* can be a system port type (SOPIN, for example).

Note The assignment does not take effect until the next power-up.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.20.21 SET_PORT_ATR Built-In Function

Purpose: Sets the attributes of a port

Syntax : SET_PORT_ATR(port_id, atr_type, atr_value)

Function Return Type :INTEGER

Input/Output Parameters :

[in] port_id : INTEGER

[in] atr_type : INTEGER

[in] atr_value : INTEGER

%ENVIRONMENT Group :FLBT

Details:

- *port_id* is one of the predefined constants as follows:

  port_1

  port_2

  port_3

  port_4

- *atr_type* specifies the attribute type to set. One of the following predefined constants should be used:

  atr_readahd :Read ahead buffer

  atr_baud :Baud rate

  atr_parity :Parity

  atr_sbits :Stop bits

  atr_dbits :Data length

  atr_xonoff :XON/XOFF

  atr_eol :End of line

  atr_modem :Modem line

- *atr_value* specifies the value for the attribute type. See Table A–20 on the following page which contains acceptable pre-defined attribute types with corresponding values.

Table A–20. Attribute Values

ATR_TYPE	ATR_VALUE
atr_readahd	any integer, represents multiples of 128 bytes (for example: atr_value=1 means the buffer length is 128 bytes.
atr_baud	baud_9600 baud_4800 baud_2400 baud_1200
atr_parity	parity_none parity_even parity_odd

Draft

Table A–20. Attribute Values (Cont'd)

ATR_TYPE	ATR_VALUE
atr_sbits	sbits_1 sbits_15 sbits_2
atr_dbits	dbits_5 dbits_6 dbits_7 dbits_8
atr_xonoff	xf_not_used xf_used
atr_eol	an ASCII code value, Refer to Appendix D , "Character Codes"
atr_modem	md_not_used md_use_dsr md_nouse_dsr md_use_dtr md_nouse_dtr md_use_rts md_nouse_rts

- A returned integer is the status of this action to port.

See Also: SET_FILE_ATR Built-In Procedure, Section 7.3.1 , ‘‘Setting File and Port Attributes,’’ for more information

Example: Refer to the example for the GET_PORT_ATR Built_In Function.

A.20.22 SET_PORT_CMT Built-In Procedure

Purpose: Allows a KAREL program to set the comment displayed on the teach pendant, for a specified logical port

Syntax : SET_PORT_CMT(port_type, port_no, comment_str, status)

Input/Output Parameters :

[in] port_type : INTEGER

[in] port_no : INTEGER

[in] comment_str : STRING

[out] status : INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

- *port_type* specifies the code for the type of port whose mode is being set. Codes are defined in KLIOTYPS.KL.
- *port_no* specifies the port number whose mode is being set.
- *comment_str* is a string whose value is the comment for the specified port. This must not be over 16 characters long.
- *status* is returned with zero if the parameters are valid and the specified mode can be set for the specified port.

See Also: SET_PORT_VALUE, SET_PORT_MOD, GET_PORT_CMT, GET_PORT_VALUE, GET_PORT_MOD Built-in Procedures

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.20.23 SET_PORT_MOD Built-In Procedure

Purpose: Allows a KAREL program to set (or reset) special port modes for a specified logical port

Syntax : SET_PORT_MOD(port_type, port_no, mode_mask, status)

Input/Output Parameters :

[in] port_type : INTEGER

[in] port_no : INTEGER

[in] mode_mask : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

- *port_type* specifies the code for the type of port whose mode is being set. Codes are defined in KLIOTYPS.KL.
- *port_no* specifies the port number whose mode is being set.
- *mode_mask* is a mask specifying which modes are turned on. The following modes are defined:

1 :reverse mode - sense of the port is reversed; if the port is set to TRUE, the physical output is set toFALSE. If the port is set to FALSE, the physical output is set to TRUE. If a physical input is TRUE when the port is read, FALSE is returned. If a physical input is FALSE when the port is read, TRUE is returned.ports.

2 :complementary mode - the logical port is assigned to two physical ports whose values are complementary. In this case, port_no must be an odd number. If port n is set to TRUE, port n is set to TRUE, and port n + 1 is set to FALSE. If port n is set to FALSE, port n is set to FALSE and port n + 1 is set to TRUE. This is effective only for output

Note The mode setting does not take effect until the next power-up.

- *status* is returned with zero if the parameters are valid and the specified mode can be set for the specified port.

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

Draft

A.20.24 SET_PORT_SIM Built-In Procedure

Purpose: Sets port simulated

Syntax : SET_PORT_SIM(port_type, port_no, value, status)

Input/Output Parameters :

[in] port_type : INTEGER

[in] port_no : INTEGER

[in] value : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

- *port_type* specifies the code for the type of port to simulate. Codes are defined in KLIOTYPS.KL.
- *port_no* specifies the number of the port to simulate.
- *value* specifies the initial value to set.
- *status* is returned with zero if the port is simulated.

See Also: SET_PORT_ASG, GET_PORT_ASG, GET_PORT_SIM Built-in Procedures

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.20.25 SET_PORT_VAL Built-In Procedure

Purpose: Allows a KAREL program to set a specified output (or simulated input) for a specified logical port

Syntax : SET_PORT_VAL(port_type, port_no, value, status)

Input/Output Parameters :

[in] port_type : INTEGER

[in] port_no : INTEGER

[in] value : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

- *port_type* specifies the code for the type of port whose mode is being set. Codes are defined in KLIOTYPS.KL.
- *port_no* specifies the port number whose mode is being set.
- *value* indicates the value to be assigned to a specified port. If the *port_type* is BOOLEAN (i.e. DOUT), this should be 0 = OFF, or 1 = ON. This field can be used to set input ports if the port is simulated.
- *status* is returned with zero if the parameters are valid and the specified mode can be set for the specified port.

See Also: SET_PORT_VALUE, SET_PORT_MOD, GET_PORT_CMT, GET_PORT_VALUE, GET_PORT_MOD Built-in Procedures

Example: The following example sets the value for a specified port.

SET_PORT_VAL Built-In Procedure

```
PROGRAM setvalprog
%ENVIRONMENT IOSETUP
%INCLUDE FR:\kliotyps
 ROUTINE set_value(port_type: INTEGER;
       port_no:  INTEGER;
```

```
     g_value:  BOOLEAN): INTEGER
 VAR
  value: INTEGER
  status: INTEGER
 BEGIN
  IF g_value THEN
   value = 1
  ELSE
   value = 0;
  ENDIF
  SET_PORT_VAL (port_type, port_no, value, status)
  RETURN (status)
 END set_value
BEGIN
END setvalprog
```

Draft

A.20.26 SET_POS_REG Built-In Procedure

Purpose: Stores an XYZWPR value in the specified position register

Syntax : SET_POS_REG(register_no, posn, status<, group_no>)

Input/Output Parameters :

[in] register_no : INTEGER

[in] posn : XYZWPR

[out] status : INTEGER

[in] group_no : INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the position register in which to store the value.
- The position data is set in XYZWPR representation with no conversion.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

Example: Refer to Section B.5 ,"Using Register Built-ins" (REG_EX.KL), for a detailed program example.

A.20.27 SET_POS_TPE Built-In Procedure

Draft

Purpose: Stores an XYZWPR value in the specified position in the specified teach pendant program

Syntax : SET_POS_TPE(open_id, position_no, posn, status<, group_no>)

Input/Output Parameters :

[in] open_id : INTEGER

[in] position_no : INTEGER

[in] posn : XYZWPR

[out] status : INTEGER

[in] group_no : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *open_id* specifies the opened teach pendant program. Before calling this built-in, a program must be opened using the OPEN_TPE Built-In, and have read/write access.
- *position_no* specifies the position in the program in which to store the value.
- A motion instruction must already exist that uses the *position_no* or the position will not be used by the teach pendant program.
- The position data is set in XYZWPR representation with no conversion.
- *status* explains the status of the attempted operation. If not equal to 0, then an error has occurred.
- If *group_no* is omitted, the default group for the program is assumed. Data for other groups is not changed.
- If *group_no* is specified, it must be in the range of 1 to the total number of groups defined on the controller.

Example: Refer to Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL), for a detailed program example.

A.20.28 SET_PREG_CMT Built-In-Procedure

Purpose: To set the comment information of a KAREL position register based on a given register number and a given comment.

Syntax: SET_PREG_CMT (register_no, comment_string, status)

Input/Output Parameters:

[in] register_no: INTEGER

[in] comment_string: STRING

[out] status: INTEGER

%ENVIRONMENT group: REGOPE

Draft

A.20.29 SET_REAL_REG Built-In Procedure

Purpose: Stores a REAL value in the specified register

Syntax : SET_REAL_REG(register_no, real_value, status)

Input/Output Parameters :

[in] register_no : INTEGER

[in] real_value : REAL

[out] status : INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the register into which *real_value* will be stored.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: SET_INT_REG, GET_REAL_REG, GET_INT_REG Built-in Procedures

A.20.30 SET_REG_CMT Built-In-Procedure

Purpose: To set the comment information of a KAREL register based on a given register number and a given comment.

Syntax: SET_REG_CMT (register_no, comment_string, status

Input/Output Parameters:

[in] register_no: INTEGER

[in] comment_string: STRING

[out] status: INTEGER

%ENVIRONMENT group REGOPE

Details:

- Register_no specifies which register to retrieve the comments from. The comment_string represents the data which is to be used to set the comment of the given register. If the comment_string exceeds more than 16 characters, the built-in will truncate the string.

A.20.31 SET_SREG_CMT Built-in Procedure

Purpose: Sets the comment for the specified string register.

Syntax : SET_SREG_CMT(register_no, comment, status)

Input/Output Parameters:

[in] register_no :INTEGER

[in] comment :STRING[254]

[out] status :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the string register to get.
- *comment* contains the comment to set to the specified string register.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_STR_REG, GET_SREG_CMT, SET_STR_REG Built-in Procedures.

Draft

A.20.32 SET_STR_REG Built-in Procedure

Purpose: Sets the value for the specified string register.

Syntax : SET_STR_REG(register_no, value, status)

Input/Output Parameters:

[in] register_no :INTEGER

[in] value :STRING[254]

[out] status :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- *register_no* specifies the string register to get.
- *value* contains the value to set to the specified string register.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_STR_REG, GET_SREG_CMT, SET_SREG_CMT Built-in Procedures.

A.20.33 SET_TIME Built-In Procedure

Purpose: Set the current time within the KAREL system

Syntax : SET_TIME(i)

Input/Output Parameters :

[in] i : INTEGER

%ENVIRONMENT Group :TIM

Details:

- *i* holds the INTEGER representation of time within the KAREL system. This value is represented in 32-bit INTEGER format as follows:

Draft

Table A–21. 32–Bit INTEGER Format of Time

31–25	24–21	20–16
year	month	day
15–11	10–5	4–0
hour	minute	second

- The contents of the individual fields are as follows:
  - DATE:

    Bits 15-9 — Year since 1980

    Bits 8-5 — Month (1-12)

    Bits 4-0 — Day of the month
  - TIME:

    Bits 31-25 — Number of hours (0-23)

    Bits 24-21 — Number of minutes (0-59)

    Bits 20-16 — Number of 2-second increments (0-29)
- This value can be determined using the GET_TIME and CNV_STR_TIME Built-In procedures.
- If *i* is 0, the time on the system will not be changed.
- INTEGER values can be compared to determine if one time is more recent than another.

See Also: CNV_STR_TIME, GET_TIME Built-In Procedures

Example: The following example converts the STRING variable *str_time* , input by the user in "DD-MMM-YYY HH:MM:SS" format, to the INTEGER representation of time **int_time** using the CNV_STR_TIME Built-In procedure. SET_TIME is then used to set the time within the KAREL system to the time specified by **int_time** .

SET_TIME Built-In Procedure

```
WRITE('Enter the new time : ')
READ(str_time)
CNV_STR_TIME(str_time,int_time)
SET_TIME(int_time)
```

A.20.34 SET_TPE_CMT Built-In Procedure

Purpose: Provides the ability for a KAREL program to set the comment associated with a specified position in a teach pendant program.

Syntax : SET_TPE_CMT(open_id, pos_no, comment, status)

Input/Output Parameters :

[in] open_id :INTEGER

[in] pos_no :INTEGER

[in] comment :STRING

[out] status :INTEGER

%ENVIRONMENT Group :TPE

Details:

- *open_id* specifies the open_id returned from a previous call to OPEN_TPE. An open mode of TPE_RWACC must be used in the OPEN_TPE call.
- *pos_id* specifies the number of the position in the teach pendant program to get a comment from. The specified position must have been recorded.
- *comment* is the comment to be associated with the specified position. A zero length string can be used to ensure that a position has no comment. If the string is over 16 characters, it is truncated and used and a warning error is returned.
- *status* indicates zero if the operation was successful, otherwise an error code will be displayed.

See Also: GET_TPE_CMT and OPEN_TPE for more Built-in Procedures.

A.20.35 SET_TRNS_TPE Built-In Procedure

Purpose: Stores a POSITION value within the specified position in the specified teach pendant program

Syntax : SET_TRNS_TPE(open_id, position_no, posn, status)

Input/Output Parameters :

[in] open_id : INTEGER

Draft

[in] position_no : INTEGER

[in] posn : POSITION

[out] status : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *open_id* specifies the opened teach pendant program. A program must be opened before calling this built-in.
- *position_no* specifies the position in the program in which to store the value specified by *posn* . Data for other groups is not changed.
- The position data is set in POSITION representation with no conversion.
- *posn* is the group number of *position_no* .
- *status* explains the status of the attempted operation. If not equal to 0, then an error has occurred.

A.20.36 SET_TSK_ATTR Built-In Procedure

Purpose: Set the value of the specified running task attribute

Syntax : SET_TSK_ATTR(task_name, attribute, value, status)

Input/Output Parameters :

[in] task_name : STRING

[in] attribute : INTEGER

[in] value : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *task_name* is the name of the specified running task. A blank *task_name* will indicate the calling task.
- *attribute* is the task attribute whose value is to be set. The following attributes are valid:

TSK_PRIORITY :Priority, see %PRIORITY for value information

TSK_TIMESLIC :Time slice duration, see %TIMESLICE for value information

TSK_NOBUSY :Busy lamp off, see %NOBUSYLAMP

TSK_NOABORT :Ignore abort request

Pg_np_error :no abort on error

Pg_np_cmd :no abort on command

TSK_NOPAUSE :Ignore pause request

pg_np_error :no pause on error

pg_np_end :no pause on command when TP is enabled

pg_np_tpenb :no pause

TSK_TRACE :Trace enable

TSK_TRACELEN :Maximum number of lines to store when tracing

TSK_TPMOTION :TP motion enable, see %TPMOTION for value information

TSK_PAUSESFT :Pause on shift, reverse of %NOPAUSESHFT

- *value* depends on the task attribute being set.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_TSK_INFO Built-in Procedure

Example: See examples in Chapter 15 *MULTI-TASKING*

A.20.37 SET_TSK_NAME Built-In Procedure

Purpose: Set the name of the specified task

Syntax : SET_TSK_NAME(old_name, new_name, status)

Input/Output Parameters :

[in] old_name : STRING

Draft

[in] new_name : STRING

[out] status : INTEGER

%ENVIRONMENT Group :MULTI

Details:

- *task_name* is the name of the task of interest. A blank *task_name* will indicate the calling task.
- *new_name* will become the new task name.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: GET_ATTR_PRG Built-in Procedure

Example: See examples in Chapter 15 *MULTI-TASKING*

Draft

A.20.38 SET_VAR Built-In Procedure

Purpose: Allows a KAREL program to set the value of a specified variable

Syntax : SET_VAR(entry, prog_name, var_name, value, status)

Input/Output Parameters :

[in,out] entry : INTEGER

[in] prog_name : STRING

[in] var_name : STRING

[in] value : Any valid KAREL data type except PATH

[out] status : INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- *entry* returns the entry number in the variable data table of *var_name* in the device directory where *var_name* is located. **This variable should not be modified.**
- *prog_name* specifies the name of the program that contains the specified variable. If *prog_name*

is ”, then it defaults to the current task name being executed. *prog_name* can also access a system variable on a robot in a ring.

- Use *prog_name* of ’*SYSTEM*’ to set a system variable.
- *var_name* must refer to a static variable.
- *var_name* can contain node numbers, field names, and/or subscripts.
- If both *var_name* and *value* are ARRAYs, the number of elements copied will equal the size of the smaller of the two arrays.
- If both *var_name* and *value* are STRINGs, the number of characters copied will equal the size of the smaller of the two strings.
- If both *var_name* and *value* are STRUCTUREs of the same type, *value* will be an exact copy of *var_name* .
- *var_name* will be set to *value* .
- If *value* is uninitialized, the value of *var_name* will be set to uninitialized and *status* will be set to 12311. *value* must be a static variable within the calling program.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Caution

Using SET_VAR to modify system variables could cause unexpected results.

- The designated names of all the robots can be found in the system variable $PH_MEMBERS[]. This also include information about the state of the robot. The ring index is the array index for this system variable. KAREL users can write general purpose programs by referring to the names and other information in this system variable rather than explicit names.

See Also: CREATE_VAR, RENAME_VAR Built-In Procedures, *Internet Options Manual* for information on accessing system variables on a robot in a ring.

Example 1: To access $TP_DEFPROG on the MHROB03 robot in a ring, see Accessing $TP_DEFPROG on MHROB03 .

Accessing $TP_DEFPROG on MHROB03

```
SET_VAR(entry, '\\MHROB03\*system*', '$TP_DEFPROG', strvar, status)
```

Example 2: In GET_VAR SET_VAR Example , an array [ipgetset]set_data[x,y] is set on all robots in the ring from all robots in the ring. In this array x is the source robot index and y is the destination robot index:

GET_VAR SET_VAR Example

```
FOR idx = 1 TO $PH_ROSIP.$NUM_MEMBERS DO
 IF idx = $PH_ROSIP.$MY_INDEX THEN
    -- This will work but it this robot so is inefficient
 ELSE
  SELECT $PH_MEMBERS[idx].$STATE OF
   CASE (0) : -- Offline
    sstate = ' Offline'
   CASE (1) : -- Online
    sstate = ' Online '
   CASE (2) : -- Synchronized
    sstate = ' Synch  '
     CNV_INT_STR(idx, 1, 10, sidx)
     prog_name = '\\' + $PH_MEMBERS[idx].$NAME + '\ipgetset'
     var_name = 'set_data['+ smy_index + ',' + sidx + ']'
     GET_VAR(entry, prog_name, var_name, set_data[$PH_ROSIP.$MY_INDEX,
          idx], status[idx])
     IF status[idx] = 0 THEN
      IF uninit(set_data[$PH_ROSIP.$MY_INDEX, idx]) THEN
       set_data[$PH_ROSIP.$MY_INDEX, idx] = 0
      ELSE
        set_data[$PH_ROSIP.$MY_INDEX, idx] = set_data[$PH_ROSIP.$MY_INDEX,
          idx] + 1
      ENDIF
       SET_VAR(entry, prog_name, var_name, set_data[$PH_ROSIP.$MY_INDEX, idx],
          status[idx])
     ENDIF
    ENDSELECT
  ENDIF
ENDFOR
```

Example 3: Refer to Section B.2 , "Copying Path Variables" (CPY_PTH.KL), for a detailed program example.

Example 4 GET_VAR and SET_VAR can also be used to set register values.

This will work for the local robot with the program names *posreg* and *numreg*. For the local robot this has similar functionality to the GET_POS_REG, GET_REG and SET_REG, SET_POS_REG built-ins. The built-ins only work for the local robot. You can access robots in the ring via GET_VAR and SET_VAR by using the robot name as part of the program name.

For the case of SET_VAR on numeric registers the register type will be set to the type of the KAREL variable. In the example below after executing this code numeric register 20 will be an integer and numeric register 21 will be a real.

If a position register is locked and you attempt to set it the error position register locked is returned. See Using GET_VAR and SET_VAR To Set Register Values .

Draft

Using GET_VAR and SET_VAR To Set Register Values

```
program GETREG
%nolockgroup
VAR
entry: integer
status: integer
int_data: integer
real_data: real
posext_data: xyzwprext


BEGIN
  GET_VAR(entry, '\\mhrob01\*numreg*', '$NUMREG[10]', int_data, status)
  IF status <> 0 THEN
    GET_VAR(entry, '\\mhrob01\*numreg*', '$NUMREG[10]', real_data, status)
  ENDIF
  GET_VAR(entry, '\\mhrob01\*posreg*', '$POSREG[1, 10]', posext_data, status)
  SET_VAR(entry, '\\mhrob01\*numreg*', '$NUMREG[20]', int_data, status)
  SET_VAR(entry, '\\mhrob01\*numreg*', '$NUMREG[21]', real_data, status)
  SET_VAR(entry, '\\mhrob01\*posreg*', '$POSREG[1, 20]', posext_data, status)
end GETREG
```

Draft

A.20.39 %SHADOWVARS Translator Directive

Purpose: This directive specifies that all variables by default are created in SHADOW.

Syntax : %SHADOWVARS

A.20.40 SHORT Data Type

Purpose: Defines a variable as a SHORT data type

Syntax : SHORT

Details:

- SHORT, is defined as 2 bytes with the range of (-32768 <= n >= 32766). A SHORT variable assigned to (32767) is considered uninitialized.
- SHORTs are allowed only within an array or within a structure.

- SHORTs can be assigned to BYTEs and INTEGERs, and BYTEs and INTEGERs can be assigned to SHORTs. An assigned value outside the SHORT range is detected during execution and causes the program to be aborted.

A.20.41 SIGNAL EVENT Action

Purpose: Signals an event that might satisfy a condition handler or release a waiting program

Syntax : SIGNAL EVENT[event_no]

where:

event_no : an INTEGER expression

Details:

- You can use the SIGNAL EVENT action to indicate a user-defined event has occurred.
- *event_no* occurs when signaled and is not remembered. Thus, if a WHEN clause has the event as its only condition, the associated actions will occur.
- If other conditions are specified that are not met at the time the event is signaled, the actions are not taken, even if the other conditions are met at another time.
- *event_no* must be in the range of -32768 to 32767. Otherwise, the program is aborted with an error.

See Also: EVENT Condition

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.20.42 SIGNAL EVENT Statement

Purpose: Signals an event that might satisfy a condition handler or release a waiting program

Syntax : SIGNAL EVENT [event_no]

where:

event_no : an INTEGER

Details:

- You can use the SIGNAL EVENT statement to indicate a user-defined event has occurred.
- *event_no* occurs when signaled and is not remembered. Thus, if a WHEN clause has the event as its only condition, the associated actions will occur.

- If other conditions are specified that are not met at the time the event is signaled, the actions are not taken, even if the other conditions are met at another time.
- *event_no* must be in the range of -32768 to 32767. Otherwise, the program is aborted with an error.

See Also: Appendix E , “Syntax Diagrams” for more syntax information, EVENT Condition

Example: Refer to the DISABLE CONDITION Statement example program.

A.20.43 SIGNAL SEMAPHORE Action

Purpose: Adds one to the value of the indicated semaphore

Syntax : SIGNAL SEMAPHORE[semaphore_no]

where:

semaphore_no : an INTEGER expression

Details:

- The semaphore indicated by *semaphore_no* is incremented by one.
- *semaphore_no* must be in the range of 1 to the number of semaphores defined on the controller.

See Also: Section 15.8 , "Task Communication" for more information and examples.

A.20.44 SIN Built-In Function

Purpose: Returns a REAL value that is the sine of the specified angle argument

Syntax : SIN(angle)

Function Return Type :REAL

Input/Output Parameters :

[in] angle : REAL

%ENVIRONMENT Group :SYSTEM

Details:

- *angle* specifies an angle in degrees.

Draft

- *angle* must be in the range of ± 18000 degrees. Otherwise, the program will be aborted with an error.

Example: Refer to Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL), for a detailed program example.

A.20.45 SQRT Built-In Function

Purpose: Returns a REAL value that is the positive square root of the specified REAL argument

Syntax : SQRT(x)

Function Return Type :REAL

Input/Output Parameters :

[in] x : REAL

%ENVIRONMENT Group :SYSTEM

Details:

- *x* must not be negative. Otherwise, the program will be aborted with an error.

Example: The following example calculates the square root of the expression (a*a+b*b) and indicates that this is the hypotenuse of a triangle.

SQRT Built-In Function

```
c = SQRT(a*a+b*b)
WRITE ('The hypotenuse of the triangle is ',c::6::2)
```

A.20.46 %STACKSIZE Translator Directive

Purpose: Specifies stack size in long words.

Syntax : %STACKSIZE = n

Details:

- *n* is the stack size.
- The default value of *n* is 300 (1200 bytes).

Draft

See Also: Section 5.1.6 , “Stack Usage,” for information on computing stack size

A.20.47 STD_PTH_NODE Data Type

Purpose: Defines a data type for use in PATHs.

Syntax : STD_PTH_NODE = STRUCTURE

node_pos : POSITION in GROUP[1]

group_data : GROUP_ASSOC in GROUP[1] (no longer used)

common_data : COMMON_ASSOC (no longer used)

ENDSTRUCTURE

Details:

- If the NODEDATA clause is omitted from the PATH declaration, then STD_PTH_NODE will be the default.
- Each node in the PATH will be of this type.

Draft

A.20.48 STOP Action

Purpose: Stops any motion in progress, leaving it in a resumable state

Syntax : STOP <GROUP[n{,n}]>

Details:

- Any motion in progress is decelerated to a stop. The unfinished motion as well as any pending motions are grouped together in a motion set and placed on a stack.
- More than one motion might be stacked by a single STOP action.
- If the KAREL program was waiting for the completion of the motion in progress, it will continue to wait.
- The stacked motion set can be removed from the stack and restarted with either a RESUME statement or action or by issuing RESUME from the operator interface (CRT/KB).
- If the group clause is not present, all groups for which the task has control (when the condition is defined) will be stopped.
- If the motion that is stopped, resumed, canceled, or held is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are stopped, resumed, canceled, or held.

- Motion cannot be stopped for a different task.

See Also: RESUME Statement

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.20.49 STOP Statement

Purpose: Stops any motion in progress, leaving it in a resumable state

Syntax : STOP <GROUP[n{,n}]>

Details:

- Any motion in progress is decelerated to a stop. The unfinished motion as well as any pending motions are grouped together in a motion set and placed on a stack.
- More than one motion might be stacked by a single STOP statement.
- If the KAREL program was waiting for the completion of the motion in progress, it will continue to wait.
- The stacked motion set can be removed from the stack and restarted with either a RESUME statement or action, or by issuing RESUME from the CRT/KB.
- If the group clause is not present, all groups for which the task has control will be stopped.
- If the motion that is stopped, resumed, canceled, or held is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are stopped, resumed, canceled, or held.
- Motion cannot be stopped for a different task.

See Also: RESUME Action, RESUME Statement, Appendix E , “Syntax Diagrams,” for more syntax information

Example: The following example stops motion if the digital input is ON.

STOP Statement

```
IF DIN[2] THEN
  STOP
ENDIF
```

A.20.50 STRING Data Type

Purpose: Defines a variable or routine parameter as STRING data type

Syntax : STRING[length]

where:

length : an INTEGER constant or literal

Details:

- *length* , the physical length of the string, indicates the maximum number of characters for which space is allocated for a STRING variable.
- *length* must be in the range 1 through 254 and must be specified in a STRING variable declaration.
- A *length* value is not used when declaring STRING routine parameters; a STRING of any length can be passed to a STRING parameter.
- Attempting to assign a STRING to a STRING variable that is longer than the physical length of the variable results in the STRING value being truncated on the right to the physical length of the STRING variable.
- Only STRING expressions can be assigned to STRING variables or passed as arguments to STRING parameters.
- STRING values cannot be returned by functions.
- Valid STRING operators are:
  - Relational operators (>, >=, =, <>, <, and <=)
  - Concatenation operator (+)
- STRING literals consist of a series of ASCII characters enclosed in single quotes (apostrophes). Examples are given in the following table.

Table A–22. Example STRING Literals

VALID	INVALID	REASON
‘123456’	123456	Without quotes 123456 is an INTEGER literal

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.3 , "Saving Data to the Default Device" (SAVE_VR.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Draft

Section B.9 , "Using the File and Device Built-ins" (FILE_EX.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL)

Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)

Section B.14 , "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.20.51 STR_LEN Built-In Function

Purpose: Returns the current length of the specified STRING argument

Syntax : STR_LEN(str)

Function Return Type :INTEGER

Input/Output Parameters :

[in] str : STRING

%ENVIRONMENT Group :SYSTEM

Details:

- The returned value is the length of the STRING currently stored in the *str* argument, not the maximum length of the STRING specified in its declaration.

Example: Refer to Section B.12 , "Displaying a List from a Dictionary File" (DCLST_EX.KL) for a detailed program example.

A.20.52 STRUCTURE Data Type

Purpose: Defines a data type as a user-defined structure

Syntax : new_type_name = STRUCTURE

field_name_1: type_name_1

field_name_2: type_name_2

...

ENDSTRUCTURE

Details:

- A user-defined structure is a data type consisting of a list of component fields, each of which can be a standard data type or another, previously defined, user data type.
- When a program containing variables of user-defined types is loaded, the definitions of these types is checked against a previously created definition. If this does not exist, it is created.
- The following data types are not permitted as part of a data structure:
  - — STRUCTURE definitions (types that are declared structures are permitted)
  - — PATH types
  - — FILE types
  - — Vision types
  - — Variable length arrays
  - — The data structure itself, or any type that includes it, either directly or indirectly
- A variable may not be defined as a structure, but as a data type previously defined as a structure

See Also: Section 2.4.2 , “User-Defined Data Structures”

Example: Refer to Section B.2 , "Copying Path Variables" (CPY_PTH.KL), for a detailed program example.

A.20.53 SUB_STR Built-In Function

Purpose: Returns a copy of part of a specified STRING argument

Syntax : SUB_STR(src, strt, len)

Function Return Type :STRING

Input/Output Parameters :

[in] src : STRING

[in] strt : INTEGER

[in] len : INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- A substring of *src* is returned by the function.
- The length of substring is the number of characters, specified by *len* , that starts at the indexed position specified by *strt* .
- *strt* must be positive. Otherwise, the program is aborted with an error. If *strt* is greater than the length of *src* , then an empty string is returned.
- *len* must be positive. Otherwise, the program is aborted with an error. If *len* is greater than the declared length of the return STRING, then the returned STRING is truncated to fit.

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)

Section B.9 , "Using the File and Device Built-ins" (FILE_EX.KL)

A.21 - T - KAREL LANGUAGE DESCRIPTION

A.21.1 TAN Built-In Function

Purpose: Returns a REAL value that is the tangent of the specified REAL argument

Syntax : TAN(angle)

Function Return Type :REAL

Input/Output Parameters:

[in] angle : REAL

%ENVIRONMENT Group :SYSTEM

Details:

- The value of *angle* must be in the range of ± 18000 degrees. Otherwise, the program will be aborted with an error.

Example: The following example uses the TAN Built-In function to specify the tangent of the variable *angle* . The tangent should be equal to the SIN(*angle*) divided by COS(*angle*).

TAN Built-In Function

```
WRITE ('enter an angle:')
READ (angle,CR)

ratio = SIN(angle)/COS(angle)
```

Draft

```
IF ratio = TAN(angle) THEN
  WRITE ('ratio is correct',CR)
ENDIF
```

Draft

A.21.2 %TIMESLICE Translator Directive

Purpose: Supports round-robin type time slicing for tasks with the same priority

Syntax : %TIMESLICE = n

Details:

- *n* specifies task execution time in msec for one slice. The default value is 256 msec.
- The timeslice value must be greater than 0.
- This value is the maximum duration for executing the task continuously if there are other tasks with the same priority that are ready to run.
- This function is effective only when more than one KAREL task with the same priority is executing at the same time.
- The timeslice duration can be set during task execution by the SET_TSK_ATTR Built-In routine.

A.21.3 %TPMOTION Translator Directive

Purpose: Specifies that task motion is enabled when the teach pendant is on

Syntax : %TPMOTION

Details:

- This attribute can be set during task execution by the SET_TSK_ATTR Built-In routine.

A.21.4 TRANSLATE Built-In Procedure

Purpose: Translates a KAREL source file (.KL file type) into p-code (.PC file type), which can be loaded into memory and executed.

Syntax : TRANSLATE(file_spec, listing_sw, status)

Input/Output Parameters:

[in] file_spec : STRING

[in] listing_sw : BOOLEAN

[out] status : INTEGER

%ENVIRONMENT Group :trans

Details:

- *file_spec* specifies the device, name and type of the file to translate. If no device is specified, the default device will be used. The type, if specified, is ignored and .KL is used instead.
- The p-code file will be created on the default device. The default device should be set to the ram disk.
- *listing_sw* specifies whether a .LS file should be created. The .LS file contains a listing of the source lines and any errors which may have occurred. The .LS file will be created on the default device.
- The KAREL program will wait while the TRANSLATE Built-In executes. If the KAREL program is paused, translation will continue until completed. If the KAREL program is aborted, translation will also be aborted and the .PC file will not be created.
- If the KAREL program must continue to execute during translation, use the KCL_NO_WAIT Built-In instead.
- *status* explains the status of the attempted operation. If the number 0 is returned, the translation was successful. If not, an error occurred. Some of the status codes are shown below:

  0 Translation was successful

  268 Translator option is not installed

  35084 File cannot be opened or created.KL file cannot be found or default device is not the RAM disk

  -1 Translation was not successful (Please see .LS file for details)

Example: The following example program will create, translate, load, and run another program called hello.

TRANSLATE Built-In Procedure

```
OPEN FILE util_file ('RW', 'hello.kl')
  WRITE util_file ('PROGRAM hello', CR)
  WRITE util_file ('%NOLOCKGROUP', CR)
  WRITE util_file ('BEGIN', CR)
  WRITE util_file (' WRITE (''hello world'', CR)', CR)
  WRITE util_file ('END hello', CR)
  CLOSE FILE util_file
  TRANSLATE('hello', TRUE, status)
  IF status = 0 THEN
```

Draft

```
    LOAD('hello.pc', 0, status)
  ENDIF
  IF status = 0 THEN
    CALL_PROG('hello', prog_index)
  ENDIF
```

Draft

A.21.5 TRUNC Built-In Function

Purpose: Converts the specified REAL argument to an INTEGER value by removing the fractional part of the REAL argument

Syntax : TRUNC(x)

Function Return Type :INTEGER

Input/Output Parameters:

[in] x : REAL

%ENVIRONMENT Group :SYStem

Details:

- The returned value is the value of *x* after any fractional part has been removed. For example, if x = 2.3, the .3 is removed and a value of 2 is returned.
- *x* must be in the range of -2147483648 to +2147483583. Otherwise, the program is aborted with an error.
- ROUND and TRUNC can both be used to convert a REAL expression to an INTEGER expression.

See Also: ROUND Built-In Function

Example: The following example uses the TRUNC Built-In to determine the actual INTEGER value of **miles** divided by **hours** to get **mph.**

TRUNC Built-In Function

```
WRITE ('enter miles driven, hours used: ')
  READ (miles,hours,CR)
  mph = TRUNC(miles/hours)
  WRITE ('miles per hour=',mph::5)
```

A.22 - U - KAREL LANGUAGE DESCRIPTION

A.22.1 UNHOLD Action

Purpose: Releases a HOLD of motion

Syntax : UNHOLD <GROUP [n{,n}]>

Details:

- Any motion that was in progress when the last HOLD was executed is resumed.
- If motions are not held, the action has no effect.
- Held motions are canceled if a RELEASE statement is executed.
- If the group clause is not present, all groups or which the task has control (when the condition is defined) will be resumed.
- Motion cannot be stopped for a different task.

Draft

Example: The following example shows a global condition handler that issues an UNHOLD action to resume robot motion is when TPIN[1] is pressed.

UNHOLD Action

```
CONDITION[1]:
  WHEN TPIN[1]+ DO
    UNHOLD
ENDCONDITION
```

A.22.2 UNHOLD Statement

Purpose: Releases a HOLD of motion

Syntax : UNHOLD <GROUP [n{,n}]>

Details:

- Any motion that was in progress when the last HOLD was executed is resumed.
- If motions are not held, the statement has no effect.
- Held motions are canceled if a RELEASE statement is executed.
- If the group clause is not present, all groups or which the task has control (when the condition is defined) will be resumed.

- Motion cannot be stopped for a different task.

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: The following example initiates a move to **PR[1]** and HOLDs the motion. If **DIN[1]** is ON, UNHOLD allows the program to resume motion.

UNHOLD Statement

```
move_to_pr1 –– Call TP program to move to PR[1]
HOLD
IF DIN[1] THEN
  UNHOLD
ENDIF
```

Draft

A.22.3 UNINIT Built-In Function

Purpose: Returns a BOOLEAN value indicating whether or not the specified argument is uninitialized

Syntax : UNINIT(variable)

Function Return Type :BOOLEAN

Input/Output Parameters :

[in] variable :any KAREL variable

%ENVIRONMENT Group :SYSTEM

Details:

- A value of TRUE is returned if *variable* is uninitialized. Otherwise, FALSE is returned.
- *variable* can be of any data type except an unsubscripted ARRAY, PATH, or structure.

Example: Refer to Section B.12 , "Displaying a List from a Dictionary File" (DCLST_EX.KL) for a detailed program example.

A.22.4 %UNINITVARS Translator Directive

Purpose: This directive specifies that all variables are by default uninitialized.

Syntax : %UNINITVARS

A.22.5 UNLOCK_GROUP Built-In Procedure

Purpose: Unlocks motion control for the specified group of axes

Syntax : UNLOCK_GROUP(group_mask, status)

Input/Output Parameters:

[in] group_mask :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :MULTI

Details:

- *group_mask* specifies the group of axes to unlock for the running task. The group numbers must be in the range of 1 to the total number of groups defined on the controller.

Table A–23. Group_mask Settings

GROUP	DECIMAL	BIT
Group 1	1	1
Group 2	2	2
Group 3	4	3

To specify multiple groups select the decimal values, shown in Table A–23 , which correspond to the desired groups. Then connect them together using the OR operator. For example to specify groups 1, 3, enter "1 OR 4".

- Unlocking a group indicates that the task is done with the motion group.
- When a task completes execution (or is aborted), all motion groups that are locked by the program will be unlocked automatically.
- If motion is executing or pending when the UNLOCK_GROUP Built-In is called, then status is set to 17039, "Executing motion exists."
- If motion is stopped when the UNLOCK_GROUP Built-In is called, then status is set to 17040, "Stopped motion exists."
- If a motion statement is encountered in a program that has the %NOLOCKGROUP Directive, the task will attempt to get motion control for all the required groups if it does not already have it. The task will pause if it cannot get motion control.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Draft

See Also: LOCK_GROUP Built-In Procedure, Chapter 15 *MULTI-TASKING* , for more information and examples.

Example: The following example unlocks groups 1, 2, and 3, and then locks group 3.

UNLOCK_GROUP Built-In Procedure

```
PROGRAM lock_grp_ex
%ENVIRONMENT MOTN
%ENVIRONMENT MULTI
  VAR
    status: INTEGER
  BEGIN
    REPEAT
      -- Unlock groups 1, 2, and 3
      UNLOCK_GROUP(1 OR 2 OR 4, status)
      IF status = 17040 THEN
        CNCL_STP_MTN       -- or RESUME
      ENDIF
      DELAY 500
    UNTIL status = 0
    -- Lock only group 3
    LOCK_GROUP(4, status)
END lock_grp_ex
```

Draft

A.22.6 UNPAUSE Action

Purpose: Resumes program execution long enough for a routine action to be executed

Syntax : UNPAUSE

Details:

- If a routine is called as an action, but program execution is paused, execution is resumed only for the duration of the routine and then is paused again.
- If more than one routine is called, all of the routines will be executed before execution is paused again.
- The resume and pause caused by UNPAUSE do not satisfy any RESUME and PAUSE conditions.

See Also: RESUME, PAUSE Actions

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.22.7 UNPOS Built-In Procedure

Purpose: Sets the specified REAL variables to the location (x,y,z) and orientation (w,p,r) components of the specified XYZWPR variable and sets the specified CONFIG variable to the configuration component of the XYZWPR

Syntax : UNPOS(posn, x, y, z, w, p, r, c)

Input/Output Parameters:

[in] posn :XYZWPR

[out] x, y, z :REAL

[out] w, p, r :REAL

[out] c :CONFIG

%ENVIRONMENT Group :SYSTEM

Details:

- *x, y, z, w, p* , and *r* arguments are set to the x, y, and z location coordinates and yaw, pitch, and roll orientation angles of *posn* .
- *c* returns the configuration of *posn* .

Example: The following example uses the UNPOS Built-In to add 100 to the **x** location argument.

UNPOS Built-In Procedure

```
UNPOS (CURPOS,x,y,z,w,p,r,config)
next_pos = POS (x+100,y,z,w,p,r,config)
SET_POS_REG(1, next_pos, status) –– Put next_pos in PR[1]
move_to_pr1 –– Call TP program to move to PR[1]
```

A.22.8 USING ... ENDUSING Statement

Purpose: Defines a range of executable statements in which fields of a variable of a STRUCTURE type can be accessed without repeating the name of the variable.

Syntax : USING struct_var{,struct_var} DO

{statement} ENDUSING

where:

struct_var : a variable of STRUCTURE type

statement : an executable KAREL statement

Details:

- In the executable statement, if the same name is both a field name and a variable name, the field name is used.
- If the same field name appears in more than one variable, the right-most variable in the USING statement is used.
- When the translator sees any field, it searches the structure type variables listed in the USING statement from right to left.

Example: Refer to Section B.2 , "Copying Path Variables" (CPY_PTH.KL), for a detailed program example.

Draft

A.23 - V - KAREL LANGUAGE DESCRIPTION

A.23.1 V_CAM_CALIB *i*RVision Built-In Procedure

Purpose: Finds the calibration grid for either a single plane or multiple plane calibration.

Syntax:V_CAM_CALIB (cal_name, func_code, status)

[in] cal_name : STRING

[in] func_code : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group: CVIS

Details:

- *cal_name* is the name of the calibration setup file created in setup mode.
- *func_code* is the plane number to find. One (1) for first plane or single plane calibration and two (2) for second plane in multiplane calibration. See Table A–24 .

Table A–24. Function Code Values

Calibration Type	Function Code
Grid Pattern Calibration	Specify the index of the calibration plane: 1 or 2.
3DL Calibration	Specify the index of the calibration plane: 1 or 2.
Visual Tracking Calibration	Not supported
Simple 2D Calibration	Not supported

- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- The calibration file must already exist and be set up. This built-in will recalibrate an already calibrated file.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Example:

```
------------------------------------------------------------------
PROGRAM cal
------------------------------------------------------------------
%NOLOCKGROUP
%ENVIRONMENT CVIS
%ALPHABETIZE
%COMMENT = 'IRVision CAL Built-in'
%NOPAUSE = ERROR + COMMAND + TPENABLE


VAR
  STATUS      : INTEGER
----------------------------------------

BEGIN
 -- Find calibration plane one
  V_CAM_CALIB('CAL1',1,STATUS)
    -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_CAM_CALIB failed - ERROR ', STATUS, CR)
    ABORT
  ENDIF

END cal
```

Draft

A.23.2 V_GET_OFFSET *i*RVision Built-In Procedure

Purpose: Gets a vision offset from a vision process and stores it in a specified vision register.

Syntax:V_GET_OFFSET(vp_name, register_no, status)

[in] vp_name: STRING

[in] register_no: INTEGER

[out] status: INTEGER

% ENVIRONMENT GROUP: CVIS

Details:

- *vp_name* is the name of the vision process created in setup mode
- *register_no* is the register number in which the offset and vision data is placed.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This command is used after a V_RUN_FIND built-in procedure. If image processing is not yet completed when V_GET_OFFSET is executed, it waits for the completion of the image processing. V_GET_OFFSET stores the vision offset for a workpiece in a vision register. When the vision process finds more than one workpiece, V_GET_OFFSET should be called repeatedly.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Example Program:

```
-----------------------------------------------------------------
PROGRAM vision
-----------------------------------------------------------------
%NOLOCKGROUP
%ENVIRONMENT cvis
%ALPHABETIZE
%COMMENT = 'IRVision Built-in'
%NOPAUSE = ERROR + COMMAND + TPENABLE


VAR
  STATUS      : INTEGER
  visprocess  : STRING[8]
  int_value   : INTEGER
  real_value  : REAL
```

```
--
----------------------------------------

BEGIN


  -- the name of the vision process passed from a TP program or MACRO
  -- TP example CALL VISION('VP1')
  GET_TPE_PRM(1,3,int_value,real_value,visprocess,STATUS)

  -- V_RUN_FIND, snap an image and run the vision process
  V_RUN_FIND(visprocess, 0, STATUS)

  -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_RUN_FIND FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

  -- V_GET_OFFSET, get the first offset from the run_find command
  -- put the offset into VR[1]
  -- call V_GET_OFFSET multiple times to get offsets from multiple parts
  V_GET_OFFSET(visprocess, 1, STATUS)

  -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_GET_OFFSET FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

  -- Get all the offset values from VR[1] so they can be put into a PR
VREG_OFFSET(1,1,status)
END vision
```

A.23.3 V_GET_PASSFL *i*RVision Built-In Procedure

Purpose: Gets the status of the error proofing vision process. It then stores the result in a specified numeric register.

Syntax:V_GET_PASSFL (vp_name, register_no, status)

[in] vp_name : STRING

[in] register_no : INTEGER

[out] status : INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *vp_name* is the name of the error proofing vision process created in setup mode
- *register_no* is the register number the command will set with the error proofing operations status.
  - Zero - error proofing operation failed
  - One - error proofing operation was successful
  - Two - error proofing operation is undetermined because the parent tool failed
- *status* explains the status of the attempted operation. If status is not equal to 0, then an error occurred.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Example:

```
------------------------------------------------------------------
PROGRAM eproof
------------------------------------------------------------------
%NOLOCKGROUP
%ENVIRONMENT CVIS
%ALPHABETIZE
%COMMENT = 'IRVision EP Built-in'
%NOPAUSE = ERROR + COMMAND + TPENABLE


VAR
  STATUS      : INTEGER
----------------------------------------

BEGIN
    -- Run error proofing vision process 'EP1'
  V_RUN_FIND('EP1',0,STATUS)
    -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_RUN_FIND FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

   -- Put error proofing result into R[1]
  V_GET_PASSFL('EP1',1,STATUS)
```

Draft

```
    -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_GET_PASSFL FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

END eproof
```

Draft

A.23.4 V_GET_QUEUE *i*RVision Built-in Procedure

Purpose: Gets a part from the specified queue and stores the information in a vision register.

Syntax: V_GET_QUEUE (queue_num, vr_num, timeout, status)

[in] queue_num: INTEGER

[in] vr_num: INTEGER

[in] timeout: INTEGER

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Queue_num* is the number of the queue from which to get the parts.
- *Vr_num* is the number of the vision register (VR) in which to put the offsets.
- *Timeout* is the time to wait until a timeout occurs.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This command sets, as a tracking trigger, the value of the existing encoder when the picked up target is found.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

A.23.5 V_INIT_QUEUE *i*RVision Built-in Procedure

Purpose: Initializes the specified queue.

Syntax: V_INIT_QUEUE (queue_num, status)

[in] queue_num: INTEGER

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Queue_num* is the number of the queue to be initialized.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- All workpiece information held in the queue when the command is executed is cleared.
- When multiple robots are used to pick up workpieces on one conveyor, this command must be executed individually for all robots.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Draft

A.23.6 V_RALC_QUEUE *i*RVision Built-in Procedure

Purpose: Places or reallocates a part received from the queue back onto the queue for a downstream robot.

Syntax: V_RALC_QUEUE (queue_num, vr_num, status)

[in] queue_num: INTEGER

[in] vr_num: INTEGER

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Queue_num* is the number of the queue in which to reallocate the parts.
- *Vr_num* is the number of the vision register (VR) of the offset to reallocate.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

A.23.7 V_RUN_FIND *i*RVision Built-In Procedure

Purpose: Starts an *i*RVision process. When a specified vision process has more than one camera view, location is performed for the specified camera views.

Syntax:

V_RUN_FIND(vp_name,camera_view,status)

[in] vp_name: STRING

[in] camera_view: INTEGER

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *vp_name* is the name of the vision process created in setup mode
- *camera_view* is the number of the camera view. Used for multi camera vision processes. A value of -1 will run all of the views.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Example Program:

```
-----------------------------------------------------------------
PROGRAM vision
-----------------------------------------------------------------
%NOLOCKGROUP
%ENVIRONMENT cvis
%ALPHABETIZE
%COMMENT = 'IRVision Built-in'
%NOPAUSE = ERROR + COMMAND + TPENABLE


VAR
  STATUS       : INTEGER
  visprocess   : STRING[8]
  int_value    : INTEGER
  real_value   : REAL



--
----------------------------------------

BEGIN


  -- the name of the vision process passed from a TP program or MACRO
  -- TP example CALL VISION('VP1')
```

Draft

```
  GET_TPE_PRM(1,3,int_value,real_value,visprocess,STATUS)

  -- V_RUN_FIND, snap an image and run the vision process
  V_RUN_FIND(visprocess, 0, STATUS)

  -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_RUN_FIND FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

  -- V_GET_OFFSET, get the first offset from the run_find command
  -- put the offset into VR[1]
  -- call V_GET_OFFSET multiple times to get offsets from multiple parts
  V_GET_OFFSET(visprocess, 1, STATUS)

  -- success if status is zero
  IF STATUS <> 0 THEN
    WRITE ('V_GET_OFFSET FAILED with ERROR CODE ', STATUS, CR)
    ABORT
  ENDIF

  -- Get all the offset values from VR[1] so they can be put into a PR
VREG_OFFSET(1,1,status)
END vision
```

Draft

A.23.8 V_SET_REF *i*RVision Built-In Procedure

Purpose: Sets the reference position in the specified vision process after V_RUN_FIND has been run.

Syntax:

V_SET_REF(vp_name, status)

[in] vp_name: STRING

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *vp_name* is the name of the vision process created in setup mode.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

- If a vision process remains open on the setup PC when SET_REFERENCE is executed for the vision process, the reference position cannot be written to the vision process, which results in an alarm. Close the setup window, then re-execute the command.
- When the vision process finds more than one workpiece, the position of the workpiece having the highest score is set as the reference position.
- It is recommended that only one workpiece be placed within the camera view so that an incorrect position is not set as the reference position.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Draft

A.23.9 V_START_VTRK *i*RVision Built-in Procedure

Purpose: Starts the specified visual tracking vision process.

Syntax: V_START_VTRK (vp_name, status)

[in] vp_name: STRING

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Vp_name* is the name of the visual tracking vision process.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- After this command is executed, *i*RVision monitors a specified condition (such as a conveyor move distance). *i*RVision executes the specified vision process each time a specified condition is satisfied. When multiple robots are used with one conveyor, this command is executed only in the robots on which *i*RVision resides.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

A.23.10 V_STOP_VTRK *i*RVision Built-in Procedure

Purpose: Stops the specified visual tracking vision process.

Syntax: V_STOP_VTRK (vp_name, status)

[in] vp_name: STRING

[out] status: INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Vp_name* is the name of the visual tracking vision process.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- When multiple robots are used with one conveyor, this command is executed only in the robots on which *i*RVision resides.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Draft

A.23.11 VAR_INFO Built-In Procedure

Purpose: Allows a KAREL program to determine data type and numerical information regarding internal or external program variables

Syntax: VAR_INFO(prog_name, var_name, uninit, type_nam, type_value, dims, slen, status)

Input/Output Parameters:

[in] prog_name: STRING

[in] var_name: STRING

[out] uninit_b :BOOLEAN

[out] type_nam :STRING

[out] dims:ARRAY[3] OF INTEGER

[out] type_value :INTEGER

[out] status :INTEGER

[out] slen :INTEGER

%ENVIRONMENT Group :BYNAM

Details:

- *prog_name* specifies the name of the program that contains the specified variable. If *prog_name* is blank, then it defaults to the current program being executed.
- *var_name* must refer to a static program variable.
- *var_name* can contain node numbers, field names, and/or subscripts.
- *uninit_b* will return a value of TRUE if the variable specified by *var_name* is uninitialized and FALSE if the variable specified by *var_name* is initialized.

- *type_nam* returns a STRING specifying the type name of *var_name*
- *type_value* returns an INTEGER corresponding to the data type of *var_name*. The following table lists valid data types and their representative INTEGER values.

Table A–25. Valid Data Types

Data Type	Value
POSITION	1
XYZWPR	2
XYZWPREXT	6
INTEGER	16
REAL	17
BOOLEAN	18
VECTOR	19
VIS_PROCESS	21
MODEL	22
SHORT	23
BYTE	24
JOINTPOS1	25
CONFIG	28
FILE	29
PATH	31
CAM_SETUP	32
JOINTPOS2	41
JOINTPOS3	57
JOINTPOS4	73
JOINTPOS5	89
JOINTPOS6	105
JOINTPOS7	121
JOINTPOS8	137
JOINTPOS9	153
JOINTPOS	153
STRING	209
user-defined type	210

Draft

- *dims* returns the dimensions of the array, if any. The size of the *dims* array should be 3.
  - dims[1] = 0 if not an array
  - dims[2] = 0 if not a two-dimensional array
  - dims[3] = 0 if not a three-dimensional array
- *slen* returns the declared length of the variable specified by *var_name* if it is a STRING variable.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

Example: The following example retrieves information regarding the variable **counter** , located in **util_prog** , from within the program **task.**

VAR_INFO Built-In Procedure

```
PROGRAM util_prog
  VAR
    counter, i : INTEGER
  BEGIN
    counter = 0
    FOR i = 1 TO 10 DO
      counter = counter + 1
    ENDFOR
  END util_prog
PROGRAM task
  VAR
    uninit_b           : BOOLEAN
    type_name          : STRING[12]
    type_code          : INTEGER
    slen, status       : INTEGER
    alen               : ARRAY[3] OF INTEGER
  BEGIN
    VAR_INFO('util_prog', 'counter', uninit_b, type_name, type_code,
             alen, slen, status)
    WRITE('counter : ', CR)
    WRITE('UNINIT : ', uninit_b, '    TYPE : ', type_name, cr)
  END task
```

A.23.12 VAR_LIST Built-In Procedure

Purpose: Locates variables in the specified KAREL program with the specified name and data type

Syntax : VAR_LIST(prog_name, var_name, var_type, n_skip, format, ary_nam, n_vars, status)

Input/Output Parameters:

[in] prog_name : STRING

[in] var_name : STRING

[in] var_type : INTEGER

[in] n_skip : INTEGER

[in] format : INTEGER

[out] ary_nam : ARRAY of STRING

[out] n_vars : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :BYNAM

Details:

- *prog_name* specifies the name of the program that contains the specified variables. *prog_name* can be specified using the wildcard (*) character, which specifies all loaded programs.
- *var_name* is the name of the variable to be found. *var_name* can be specified using the wildcard (*) character, which specifies that all variables for *prog_name* be found.
- *var_type* represents the data type of the variables to be found. The following is a list of valid data types:

Table A–26. Valid Data Types

Data Type	Value
All variable types	0
POSITION	1
XYZWPR	2
INTEGER	16
REAL	17
BOOLEAN	18
VECTOR	19
VIS_PROCESS	21
MODEL	22
SHORT	23
BYTE	24

Table A–26. Valid Data Types (Cont'd)

Data Type	Value
JOINTPOS1	25
CONFIG	28
FILE	29
PATH	31
CAM_SETUP	32
JOINTPOS2	41
JOINTPOS3	57
JOINTPOS4	73
JOINTPOS5	89
JOINTPOS6	105
JOINTPOS7	121
JOINTPOS8	137
JOINTPOS9	153
JOINTPOS	153
STRING	209
user-defined type	210

Draft

- *n_skip* is used when more variables exist than the declared length of *ary_nam* . Set *n_skip* to 0 the first time you use VAR_LIST. If *ary_nam* is completely filled with variable names, copy the array to another ARRAY of STRINGs and execute the VAR_LIST again with *n_skip* equal to *n_vars* . The call to VAR_LIST will skip the variables found in the first pass and locate only the remaining variables.
- *format* specifies the format of the program name and variable name. The following values are valid for *format* :

```
1=prog_name only, no blanks
2 =var_name only, no blanks
3 =[prog_name]var_name, no blanks
4 ='prog_name var_name   ',
Total length = 27 characters, prog_name
starts with character 1 and var_name starts with character 16.
```

- *ary_nam* is an ARRAY of STRINGs to store the variable names. If the declared length of the STRING in *ary_nam* is not long enough to store the formatted data, then status is returned with an error.

- *n_vars* is the number of variables stored in *ary_name*.
- *status* will return zero if successful.

See Also: FILE_LIST, PROG_LIST Built-In Procedures

Example: Refer to Section B.2 , "Copying Path Variables" (CPY_PTH.KL), for a detailed program example.

A.23.13 VECTOR Data Type

Purpose: Defines a variable, function return type, or routine parameter as VECTOR data type

Syntax : VECTOR

Details:

- A VECTOR consists of three REAL values representing a location or direction in three dimensional Cartesian coordinates.
- Only VECTOR expressions can be assigned to VECTOR variables, returned from VECTOR function routines, or passed as arguments to VECTOR parameters.
- Valid VECTOR operators are:
  - Addition (+) and subtraction (-) mathematical operators
  - Equal (=) and the not equal (<>) relational operators
  - Cross product (#) and the inner product (@) operators.
  - Multiplication (*) and division (÷) operators
  - Relative position (:) operator
- Component fields of VECTOR variables can be accessed or set as if they were defined as follows:

VECTOR Data Type

```
VECTOR = STRUCTURE
 X:  REAL
 Y:  REAL
 Z:  REAL
ENDSTRUCTURE
Note:  All fields are read-write
```

Example: The following example shows VECTOR as variable declarations, as parameters in a routine, and as a function routine return type.

Draft

VECTOR Data Type

```
VAR
  direction, offset : VECTOR
ROUTINE calc_offset(offset_vec:VECTOR):VECTOR FROM util_prog
```

A.23.14 VOL_SPACE Built-In Procedure

Purpose: Returns the total bytes, free bytes, and volume name for the specified device

Syntax : VOL_SPACE(device, total, free, volume)

Input/Output Parameters:

[in] device :STRING

[out] total :INTEGER

[out] free :INTEGER

[out] volume :STRING

%ENVIRONMENT Group :FLBT

Details:

- *devices* can be:

  RD: The RAM disk returns all three parameters, but the volume name is '' since it is not supported. The RAM disk must be mounted in order to query it.

  FR: The FROM disk returns all three parameters, but the volume name is '' since it is not supported. The FROM disk must be mounted in order to query it.

  FR: Size of the Flash ROM. This only sets the *total* parameter.

  DRAM: Size of the DRAM. This only sets the *total* parameter.

  CMOS: Size of the CMOS ROM. This only sets the *total* parameter.

  TPP : The area of system memory where teach pendant programs are stored.

  PERM: The area of permanent CMOS RAM memory where system variables and selected KAREL variables are stored.

Draft

TEMP: The area of temporary DRAM memory used for loaded KAREL programs, KAREL variables, program execution information, and system operations.

SYSTEM: The area of temporary DRAM memory where the system software and options are stored. This memory is saved to FROM and restored on power up.

Note All device names must end with a *colon* (:).

- *total* is the original size of the memory, in bytes.
- *free* is the amount of available memory, in bytes.
- *volume* is the name of the storage device used.

See Also: Section 1.4.1 , Status Memory in the "Status Displays and Indicators," chapter of the appropriate application-specific *FANUC Robotics Setup and Operations Manual* .

Example: The following example gets information about the different devices.

VOL_SPACE Built-In Procedure

```
PROGRAM space
%NOLOCKGROUP
%ENVIRONMENT FLBT
VAR
  total:  INTEGER
  free:   INTEGER
  volume: STRING [30]
BEGIN
  VOL_SPACE('rd:', total, free, volume)
  VOL_SPACE('fr:', total, free, volume)
  VOL_SPACE('frp:', total, free, volume)
  VOL_SPACE('fr:', total, free, volume)
  VOL_SPACE('dram:', total, free, volume)
  VOL_SPACE('cmos:', total, free, volume)
  VOL_SPACE('tpp:', total, free, volume)
  VOL_SPACE('perm:', total, free, volume)
  VOL_SPACE('temp:', total, free, volume)
END space
```

A.23.15 VREG_FND_POS *i*RVision Built-in Procedure

Purpose: Populates the specified position register with the found position data in the specified vision register.

Syntax: VREG_FND_POS (visreg_no, camera_view, posreg_no, status)

[in] visreg_no : INTEGER

[in] camera_view : INTEGER

[in] posreg_no : INTEGER

[out] status : INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Visreg_no* is the vision register (VR) number that contains the offset data.
- *Camera_view* is the specified camera view for a multi view vision process.
- *Posreg_no* is the position register (PR) number that is to be populated with the offset data.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

Draft

A.23.16 VREG_OFFSET *i*RVision Built-in Procedure

Purpose: Populates the specified position register with the offset data in the specified vision register.

Syntax: VREG_OFFSET(visreg_no, posreg_no, status)

[in] visreg_no : INTEGER

[in] posreg_no : INTEGER

[out] status : INTEGER

%ENVIRONMENT GROUP: CVIS

Details:

- *Visreg_no* is the vision register (VR) number that contains the offset data
- *Posreg_no* is the position register (PR) number that is to be populated with the offset data.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- This built-in requires the *i*RVision KAREL Interface Software Option (J870).

A.24 - W - KAREL LANGUAGE DESCRIPTION

A.24.1 WAIT FOR Statement

Purpose: Delays continuation of program execution until some condition(s) are met

Syntax : WAIT FOR cond_list

where:

cond_list: one or more conditions

Details:

- All of the conditions in a single WAIT FOR statement must be satisfied simultaneously for execution to continue.

See Also: Chapter 6 *CONDITION HANDLERS* , Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.24.2 WHEN Clause

Purpose: Used to specify a conditions/actions pair in a global condition handler

Syntax : WHEN cond_list DO action_list

where:

cond_list : one or more conditions

action_list : one or more conditions

Details:

- All of the conditions in the *cond_list* of a single WHEN clause must be satisfied simultaneously for the condition handler to be triggered.
- The *action_list* represents a list of actions to be taken when the corresponding conditions of a WHEN clause are satisfied simultaneously.

- Calls to function routines are not allowed in a CONDITION statement and, therefore, cannot be used in a WHEN clause.
- CONDITION statements can include multiple WHEN clauses.

See Also: Chapter 6 *CONDITION HANDLERS* , Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to the following sections for detailed program examples:

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.10 , "Using Dynamic Display Built-ins" (DYN_DISP.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

Draft

A.24.3 WHILE...ENDWHILE Statement

Purpose: Used when an action is to be executed as long as a BOOLEAN expression remains TRUE

Syntax : WHILE boolean_exp DO{ statement }ENDWHILE

where:

boolean_exp : a BOOLEAN expression

statement : a valid KAREL executable statement

Details:

- *boolean_exp* is evaluated before each iteration.
- As long as *boolean_exp* is TRUE, the statements in the loop are executed.
- If *boolean_exp* is FALSE, control is transferred to the statement following ENDWHILE, and the statement or statements in the body of the loop are not executed.

See Also: Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL) for a detailed program example.

A.24.4 WITH Clause

Purpose: Used in condition handlers to specify condition handler qualifiers

Syntax : WITH param_spec {, param_spec}

where:

param_spec is of the form : with_sys_var = value

with_sys_var : one of the system variables available for use in the WITH clause

value : an expression of the type corresponding to the type of the system variable

Details:

- The actual system variables specified are not changed.
- $PRIORITY and $SCAN_TIME are condition handler qualifiers that can be used in a WITH clause only when the WITH clause is part of a condition handler statement.

Draft

A.24.5 WRITE Statement

Purpose: Writes data to a serial device or file

Syntax : WRITE <file_var> (data_item { ,data_item })

where:

file_var : a FILE variable

data_item : an expression and its optional format specifiers or the reserved word CR

Details:

- If *file_var* is not specified in a WRITE statement the default TPDISPLAY is used. %CRTDEVICE directive will change the default to OUTPUT.
- If *file_var* is specified, it must be one of the output devices or a variable that has been equated to one of them.
- If *file_var* attribute was set with the UF option, data is transmitted to the specified file or device in binary form. Otherwise, data is transmitted as ASCII text.
- *data_item* can be any valid KAREL expression.
- If *data_item* is of type ARRAY, a subscript must be provided.
- If *data_item* is of type PATH, you can specify that the entire path be read, a specific node be read [n], or a range of nodes be read [n .. m].
- Optional format specifiers can be used to control the amount of data that is written for each *data_item* .

- The reserved word CR, which can be used as a data item, specifies that the next data item to be written to the file_var will start on the next line.
- Use the IO_STATUS Built-In to determine if the write operation was successful.

See Also: PATH Data Type, for more information on writing PATH variables, Chapter 7 *FILE INPUT/OUTPUT OPERATIONS* , for more information on format specifiers and file_vars. Appendix E , “Syntax Diagrams,” for more syntax information

Example: Refer to Appendix B , "KAREL Example Programs" for more detailed program examples.

A.24.6 WRITE_DICT Built-In Procedure

Purpose: Writes information from a dictionary

Syntax : WRITE_DICT(file_var, dict_name, element_no, status)

Input/Output Parameters:

[in] file_var :FILE

[in] dict_name :STRING

[in] element_no :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *file_var* must be opened to the window where the dictionary text is to appear.
- *dict_name* specifies the name of the dictionary from which to write.
- *element_no* specifies the element number to write. This number is designated with a “$” in the dictionary file.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred writing the element from the dictionary file.

See Also: READ_DICT, REMOVE_DICT Built-In Procedures, Chapter 10 *DICTIONARIES AND FORMS*

Example: Refer to Section B.12 , "Displaying a List From a Dictionary File" (DCLST_EX.KL), for a detailed program example.

Draft

A.24.7 WRITE_DICT_V Built-In Procedure

Purpose: Writes information from a dictionary with formatted variables

Syntax : WRITE_DICT_V(file_var, dict_name, element_no, value_array, status)

Input/Output Parameters:

[in] file_var :FILE

[in] dict_name :STRING

[in] element_no :INTEGER

[in] value_array :ARRAY OF STRING

[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

- *file_var* must be opened to the window where the dictionary text is to appear.
- *dict_name* specifies the name of the dictionary from which to write.
- *element_no* specifies the element number to write. This number is designated with a $ in the dictionary file.
- *value_array* is an array of variable names that corresponds to each formatted data item in the dictionary text. Each variable name may be specified as '[prog_name]var_name'.
  - *[prog_name]* specifies the name of the program that contains the specified variable. If not specified, then the current program being executed is used.
  - *var_name* must refer to a static, global program variable.
  - *var_name* may contain node numbers, field names, and/or subscripts.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred writing the element from the dictionary file.

See Also: READ_DICT_V Built-In Procedure, Chapter 10 *DICTIONARIES AND FORMS*

Example: In the following example, TPTASKEG.TX contains dictionary text information which will display a system variable. This information is the first element in the dictionary and element numbers start at 0. **util_prog** uses WRITE_DICT_V to display the text on the teach pendant.

WRITE_DICT_V Built-In Procedure

```
---------------------------------------------
TPTASKEG.TX
```

```
----------------------------------------------
$ "Maximum number of tasks = %d"
----------------------------------------------
UTILITY PROGRAM:
----------------------------------------------
PROGRAM util_prog
  %ENVIRONMENT uif
  VAR
    status: INTEGER
    value_array: ARRAY[1] OF STRING[30]
  BEGIN
    value_array[1] = '[*system*].$scr.$maxnumtask'
    ADD_DICT('TPTASKEG', 'TASK', dp_default, dp_open, status)
    WRITE_DICT_V(TPDISPLAY, 'TASK', 0, value_array, status)
END util_prog
```

A.25 - X - KAREL LANGUAGE DESCRIPTION

A.25.1 XML_ADDTAG Built-In Procedure

Purpose: Associates the tag “tag_name” with the “xml_file”.

Syntax : XML_ADDTAG(xml_file, tag_name, numchar, caseflag, tag_ident, status)

Input/Output Parameters:

[in] xml_file :FILE

[in] tag_name:STRING

[in] numchar:INTEGER

[in] caseflag:boolean

[in] tag_ident:INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- *xml_file* specifies an open KAREL file with AR_XML attribute set
- *tag_name* Name of the tag you want to know about

Draft

- *numchar* specifies the number of characters to use when looking for the tag
- *caseflag* If TRUE, specifies whether the tag_name is case sensitive
- *tag_ident* Application identifier that user associates with tag There are some system tag idents that the user cannot use. This is used to allow you to switch on his tags and the user return codes. When the scanning encounters the registered tag it will return with the tag_ident. The file MUST be open before you can register a tag. tag_ident should be a unique number within the application. This allows the application to do a select based on the return identifier. NOTE: The system reserves some identifiers for error and scan limit status returns. This allows the application to easily include these constants in the select statement.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred writing the element from the dictionary file. The return status will be bad if the user has not opened the file and set the XML attribute.

Example: Refer to Section 9.5 .

Draft

A.25.2 XML_GETDATA Built-In Procedure

Purpose: Returns the attribute names and values associated with the tag causing the return.

Syntax : xml_getdata(xml_file, numattr, attrnames, attrvalues, text, textdone)

Input/Output Parameters:

[in] xml_file :FILE

[out] numattr: INTEGER

[out] attrnames: ARRAY OF STRING

[out] attrvalues: ARRAY OF STRING

[out] textdata: STRING

[out] textdone: BOOLEAN

[out] status: INTEGER

Details:

- *xml_file* An open KAREL file with AR_XML attribute set
- *numattr* indicates the number of attributes
- *attrnames* indicates attribute names
- *attrvalues* indicates attribute values

- *textdata* indicates the text that follows the tag
- *textdone* If this is FALSE, more text is to be read
- *status* indicates the result of the operation

A.25.3 XML_REMTAG Built-In Procedure

Purpose: Removes the tag name from the list.

Syntax: xml_remtag(xml_file, tag_name, tag_ident, status)

Input/Output Parameters:

[in] xml_file :FILE

[in] tag_name: STRING

[OUT] status: INTEGER

Details:

- *xml_file* Open KAREL file with AR_XML attribute set
- *tag_name* Indicates the name of the tag to remove
- *status* Indicates the result of the operation

A.25.4 XML_SCAN Built-In Procedure

Purpose: Scan through a previously opened XML file

Syntax : XML_SCAN(xml_file, tag_name, func_code, status)

Input/Output Parameters:

[in] xml_file :FILE

[out] tag_name:STRING

[out] tag_ident:INTEGER

[out] func_code::INTEGER

[out] status :INTEGER

Draft

Details:

- *xml_file* Open KAREL file with AR_XML attribute set
- *tag_name* Name of the tag system found
- *tag_ident* Tag user associated in addtag call
- *func_code* Function code, start etc
- *status* Result of operation

- When a registered tag is found
  - Func_code = XML_START or XML_END or XML_STEND
  - Tag_ident = The value associated with the tag when it was registered
- When a text buffer is full
  - Func_code = XML_TXCONT
  - Tag_ident = The value associated with the tag when it was registered
- After 50 lines are scanned
  - Status = XML_SCANLIM, just recall the built-in when the is encountered
  - Tag_ident = XML_SCANLIM
  - This is not an error and indicates that there is more to come
- If it encounters a parsing error
  - Status = Some error
  - Tag_ident = XML_ERROR
- At the end of file, Status = SUCCESS
- Valid Parse Errors are:
  - XML_TAG_SIZE Too many characters in tag
  - XML_ATTR_SIZE Too many characters in attribute
  - XML_NOSLASH Invalid use of / character
  - XML_INVTAG Invalid character in tag
  - XML_UNMATCHATTR No value for attribute
  - XML_UNMATCHTAG End tag with no matching start
  - XML_INVATTR Invalid character in attribute
  - XML_NOFILE Cannot find file
  - XML_TAGNEST Tag nesting level too deep
  - XML_COMMENT Error in comment

- The system will provide a separate return for the start of the tag and the end of the tag if the tag does not contain both starting and ending information. The attribute data is NOT valid when the call is made for the END tag.
- The *tag_ident* will be the tag_ident that the user registered for the registered tag return. If the system returns for other reasons then the tag_ident may contain system tag data.

A.25.5 XML_SETVAR Built-In Procedure

Purpose: Sets the variable [prog_name]var_name according to the attributes that were associated with the tag causing the return.

Syntax : xml_setvar(xml_file, prog_name, var_name, status)

Input/Output Parameters:

[in] xml_file :FILE

[in] prog_name:STRING

[in] var_name:STRING

[out] status:INTEGER

Details:

- In this case the text for the attribute will be matched with the text field name of the KAREL variable. So a variable of this type:

```
xmlstrct_t = STRUCTURE
  first: integer
  second: real
  third: BOOLEAN
  fourth: string[20]
ENDSTRUCTURE
```

  Can be set via the XML:

  <xmlstrct_t first="123456" second="7.8910" third="1" fourth="A string"\>

- The XML tag name does not need to match the TYPE name. The association of the field names and attribute names is based on the [program]variable in the call to xml_setvar.

Draft

A.25.6 XYZWPR Data Type

Purpose: Defines a variable, function return type, or routine parameter as XYZWPR data type

Syntax : XYZWPR <IN GROUP [n]>

Details:

- An XYZWPR consists of three REAL components specifying a Cartesian location (x,y,z), three REAL components specifying an orientation (w,p,r), and a component specifying a CONFIG Data Type, 32 bytes total.
- The configuration string indicates the joint placements and multiple turns that describe the configuration of the robot when it is at a particular position.
- A position is always referenced with respect to a specific coordinate frame.
- Components of XYZWPR variables can be accessed or set as if they were defined as follows:

XYZWPR Data Type

```
XYZWPR = STRUCTURE
 X: REAL
 Y: REAL
 Z: REAL
 W: REAL
 P: REAL
 R: REAL
 CONFIG_DATA:  CONFIG
ENDSTRUCTURE
Note:  All fields are read-write access.
```

Example: Refer to the following sections for detailed program examples:

Section B.2 , "Copying Path Variables" (CPY_PTH.KL)

Section B.5 ,"Using Register Built-ins" (REG_EX.KL)

Section B.6 , "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)

Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

Draft

A.25.7 XYZWPREXT Data Type

Purpose: Defines a variable, function return type, or routine parameter as an XYZWPREXT

Syntax : XYZWPREXT <IN GROUP [n]>

Details:

- An XYZWPREXT consists of three REAL components specifying a Cartesian location (x,y,z), three REAL components specifying an orientation (w,p,r), and a component specifying a configuration string. It also includes three extended axes, 44 bytes total.
- The configuration string indicates the joint placements and multiple turns that describe the configuration of the robot when it is at a particular position.
- A position is always referenced with respect to a specific coordinate frame.
- Components of XYZWPREXT variables can be accessed or set as if they were defined as follows:

Draft

XYZWPREXT Data Type

```
XYZWPRext = STRUCTURE
 X: REAL
 Y: REAL
 Z: REAL
 W: REAL
 P: REAL
 R: REAL
 CONFIG_DATA:  CONFIG
 EXT1: REAL
 EXT2: REAL
 EXT3: REAL
ENDSTRUCTURE
--Note:  All fields are read-write access.
```

A.26 - Y - KAREL LANGUAGE DESCRIPTION

There are no KAREL descriptions beginning with "Y".

A.27 - Z - KAREL LANGUAGE DESCRIPTION

There are no KAREL descriptions beginning with "Z".

Appendix B

KAREL EXAMPLE PROGRAMS

Contents

Appendix B	KAREL EXAMPLE PROGRAMS	B–1
B.1	SETTING UP DIGITAL OUTPUT PORTS FOR PROCESS MONITORING	B–6
B.2	COPYING PATH VARIABLES	B–18
B.3	SAVING DATA TO THE DEFAULT DEVICE	B–28
B.4	STANDARD ROUTINES	B–31
B.5	USING REGISTER BUILT-INS	B–33
B.6	PATH VARIABLES AND CONDITION HANDLERS PROGRAM	B–38
B.7	LISTING FILES AND PROGRAMS AND MANIPULATING STRINGS	B–44
B.8	GENERATING AND MOVING ALONG A HEXAGON PATH	B–49
B.9	USING THE FILE AND DEVICE BUILT-INS	B–54
B.10	USING DYNAMIC DISPLAY BUILT-INS	B–58
B.11	MANIPULATING VALUES OF DYNAMICALLY DISPLAYED VARIABLES	B–68
B.12	DISPLAYING A LIST FROM A DICTIONARY FILE	B–70
B.12.1	Dictionary Files	B–80
B.13	USING THE DISCTRL_ALPHA BUILT-IN	B–81
B.13.1	Dictionary Files	B–85
B.14	APPLYING OFFSETS TO A COPIED TEACH PENDANT PROGRAM	B–85

Draft

This appendix contains some KAREL program examples. These programs are meant to show you how to use the KAREL built-ins and commands described in Appendix A , "KAREL Language Alphabetical Description."

This section includes examples of how to use the KAREL built-ins and commands in a program. Refer to Appendix A , for more detailed information on each of the KAREL built-ins and commands.

Table B–1 lists the programs in this section, their main function, the built-ins used in each program, and the section to refer to for the program listing.

Conventions

Each program in this appendix is divided into five sections.

Section 0 - Lists each element of the KAREL language that is used in the example program.

Section 1 - Contains the program and environment declarations.

Section 2 - Contains the constant, variable, and type declarations.

Section 3 - Contains the routine declarations.

Section 4 - Contains the main body of the program.

Table B–1. KAREL Example Programs

Program Name	Program Function	Built-ins Used	Section to Refer
CPY_PTH.KL	Copies path variables.	APPEND_NODE BY_NAME CALL_PROG CNV_INT_STR COPY_PATH CREATE_VAR CURPOS DELETE_NODE LOAD PATH_LEN PROG_LIST READ_KB SET_CURSOR SET_FILE_ATR SET_VAR SUB_STR VAR_LIST	Section B.2
SAVE_VRS.KL	Saves data to the default device.	DELETE_FILE SAVE	Section B.3

Draft

Table B–1. KAREL Example Programs (Cont'd)

Program Name	Program Function	Built-ins Used	Section to Refer
ROUT_EX.KL	Contains standard routines that are used throughout the program examples.	CHR FORCE_SPMENU	Section B.4
REG_EX.KL	Uses Register built-ins.	CALL_PROGLIN CHR CURPOS GET_JPOS_REG GET_POS_REG GET_REG POS_REG_TYP SET_INT_REG SET_JPOS_REG SET_POS_REG FORCE_SPMENU	Section B.5
PTH_MOVE.KL	Teaches and moves along a path. Also uses condition handlers.	CHR CNV_REL_JPOS PATH_LEN SET_CURSOR	Section B.6
LIST_EX.KL	Lists files and programs, and manipulate strings.	ABS ARRAY_LEN CNV_INT_STR FILE_LIST LOAD LOAD_STATUS PROG_LIST ROUND SUB_STR	Section B.7
GEN_HEX.KL	Generates a hexagon, and moves along the path.	CNV_REL_JPOS COS CURPOS SIN	Section B.8
FILE_EX.KL	Uses the File and Device built-ins.	CNV_TIME_STR COPY_FILE DISMOUNT_DEV FORMAT_DEV GET_TIME MOUNT_DEV SUB_STR	Section B.9

Draft

Table B–1. KAREL Example Programs (Cont'd)

Program Name	Program Function	Built-ins Used	Section to Refer
DYN_DISP.KL	Uses Dynamic Display built-ins.	ABORT_TASK CNC_DYN_DISB CNC_DYN_DISE CNC_DYN_DISP CNC_DYN_DISS CNC_DYN_DISI CNC_DYN_DISR INI_DYN_DISB INI_DYN_DISE INI_DYN_DISP INI_DYN_DISS INI_DYN_DISI INI_DYN_DISR LOAD LOAD_STATUS RUN_TASK	Section B.10
CHG_DATA.KL	Processes and changes values of dynamically displayed variables.		Section B.11
DCLST_EX.KL	Displays a list from a dictionary file.	ADD_DICT ACT_SCREEN ATT_WINDOW_S CHECK_DICT CLR_IO_STAT CNV_STR_INT DEF_SCREEN DISCTRL_LIST FORCE_SPENU IO_STATUS ORD READ_DICT REMOVE_DICT SET_FILE_ATR SET_WINDOW STR_LEN UNINIT WRITE_DICT	Section B.12.1
DCLISTEG.UTX	Dictionary file.	N/A	Section B.12

Draft

Table B–1. KAREL Example Programs (Cont'd)

Program Name	Program Function	Built-ins Used	Section to Refer
DCALP_EX.KL	Uses the DISCTRL_ALPHA Built-in.	ADD_DICT CHR DISCTRL_ALPH FORCE_SPEMU POST_ERR SET_CURSOR SET_LANG	Section B.13
DCALPHEG.UTX	Dictionary file.	N/A	Section B.13.1
CPY_TP.KL	Applies offsets to copied teach pendant pro grams.	AVL_POS_NUM CHR CLOSE_TPE CNV_JPOS_REL CNV_REL_JPOS COPY_TPE GET_JPOS_TYP GET_POS_TPE GET_POS_TYP OPEN TPE PROG_LIST SELECT_TPE SET_JPOS_TPE SET_POS_TPE	Section B.14

Draft

Table B–1. KAREL Example Programs (Cont'd)

Program Name	Program Function	Built-ins Used	Section to Refer
DOUT_EX.KL	Sets up digital output ports for process monitoring. The DOUTs are used to monitor the status of the external equipment and to show the current status of the process. The equipment status DOUTs are simulated, but in practice they are looked up to the actual external equipment as a feedback response. The robot is moved along a path until the external equipment needs servicing, which is triggered by the DOUT values.	CHRPATH_LEN CURPOS DELAY FORCE_SPMENU RESET SET_PORT_ASG SET_PORT_CMT SET_PORT_MOD SET_PORT_SIM	Section B.1

B.1 SETTING UP DIGITAL OUTPUT PORTS FOR PROCESS MONITORING

This program sets up digital output ports for process monitoring. The DOUT are to monitor the external equipment status and show the current status of the process. The equipment status DOUT's are simulated, but in practice are hooked up to the actual external equipments as a feedback response. The robot is moved along a path until external equipment needs to be serviced, which is triggered by the DOUT values.

Setting Up Digital Output Ports for Process Monitoring - Overview

```
---------------------------------------------------------------------
----    DOUT_EX.Kl
---------------------------------------------------------------------
---------------------------------------------------------------------
----    Section 0:  Detail about DOUT_EX.kl
---------------------------------------------------------------------
---- Elements of KAREL Language Covered:
```

```
----    Action:
----            CONTINUE                                    Sec 4-A
----            ENABLE CONDITION                            Sec 3-B; 4-C
----            NOMESSAGE                                   Sec 4-A
----            RESUME                                      Sec 4-C
----            ROUTINE CALL                                Sec 4-A,C
----            SIGNAL EVENT                                Sec 4-C
----            STOP                                        Sec 4-C
----            UNPAUSE                                     Sec 4-A
----    Clauses:
----            FROM                                        Sec 3-A
----            WHEN                                        Sec 4-A,C
----    Conditions:
----            ERROR[xxx]                                  Sec 4-A
----            EVENT                                       Sec 4-C
----            RELATIONAL condition                        Sec 4-A
----    Data types:
----            BOOLEAN                                     Sec 2
----            INTEGER                                     Sec 2
----            PATH                                        Sec 2
----            XYZWPREXT                                   Sec 2
----            STRING                                      Sec 2
----            XYZWPR                                      Sec 2
```

Draft

Setting Up Digital Output Ports for Monitoring Teach Pendant Program - Overview Continued

```
----    Directives:
----            ALPHABETIZE                                 Sec 1
----            COMMENT                                     Sec 1
----            CMOSVARS                                    Sec 1
----            INCLUDE                                     Sec 1
----    Built-in Functions & Procedures:
----            CHR                                         Sec 3-E; 4-D
----            CURPOS                                      Sec 3-E
----            DELAY                                       Sec 3-B,E
----            FORCE_SPMENU                                Sec 3-E; 4-D
----            PATH_LEN                                    Sec 4-B; 4-D
----            RESET                                       Sec 3-B
----            SET_PORT_ASG                                Sec 3-D
----            SET_PORT_CMT                                Sec 3-D
----            SET_PORT_MOD                                Sec 3-C
----            SET_PORT_SIM                                Sec 4-D
----            SET_POS_REG                                 Sec 3-E, 4-D
----            SET_EPOS_REG                                Sec 3-E
----    Statements:
----            ABORT                                       Sec 3-D; 4-B,D
```

```
----            ATTACH                                      Sec 4-B
----            CONNECT TIMER                               Sec 4-A
----            CONDITION...ENDCONDITON                     Sec 4-A,C
----            ENABLE CONDITION                            Sec 3-B,E; 4-A,C
----            FOR...ENDFOR                                Sec 3-D
----            IF...THEN...ENDIF                           Sec 3-B,C,D; 4-B,C,D
----            RELEASE                                     Sec 4-B
----            ROUTINE                                     Sec 3-A,B,C,D,E,F
----            WAIT FOR                                    Sec 3-E
----            WHILE...ENDWHILE                            Sec 4-B
----            WRITE                                       Sec 3-B,D,E; 4-B,D
----    Reserve Words:
----            BEGIN                                       Sec 3-B,C,D,E; 4
----            CONST                                       Sec 2
----            CR                                          Sec 3-B,D,E; 4-B,D
----            END                                         Sec 3-B,C,D,E, 4-D
----            NOT                                         Sec 3-B; 4-C
----            PROGRAM                                     Sec 1
----            VAR                                         Sec 2
----    Predefined FILE names:
----            TPFUNC                                      Sec 4-D
```

Setting Up Digital Output Ports for Process Monitoring - Declaration Section

```
----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
----------------------------------------------------------------------------
PROGRAM DOUT_EX                 -- Define the program name
%ALPHABETIZE                    -- CReate the variables in alphabetical order
%NOPAUSE  = TPENABLE            -- Do not pause the program if TP is ENABLED.
                                -- during execution.
%COMMENT  = 'PORT/CH DOUT_EX'
%CMOSVARS                       -- Make sure variables are stored in CMOS
%INCLUDE KLIOTYPS
----------------------------------------------------------------------------
----    Section 2:  Constant and Variable Declarations
----------------------------------------------------------------------------
CONST
-- Condition Handler Numbers
   CONT_CH    = 2                           -- Continue execution condition
   EQIP_FAIL  = 3                           -- Equipment Failure Condition
   RESTART    = 6                           -- Restart condition Handler
   SERV_DONE  = 4                           -- Servicing Done condition
   UNINIT_CH  = 10                          -- Monitor for uninit error
   WARMED_UP  = 5                           -- Event to notify eqip is ready
-- Process DOUT numbers ( 1 thru 6 are complementary DOUT )
```

```
--                      ( 3 and 4 are simulated DOUT )
   EQIP_READY = 1                           -- Equipment Ready
   EQIP_NOT_RD= 2                           -- Equipment Not Ready
   EQIP_ERROR = 3                           -- Equipment Failed during process
   EQIP_FIXED = 4                           -- Equipment Fixed after failure
   EQIP_ON    = 5                           -- Turn Eqip-1 ON DOUT
   EQIP_OFF   = 6                           -- Turn Eqip-1 OFF DOUT
   NODE_PULSE = 7                           -- Node Pulsing DOUT
   FINISH     = 8                           -- Path Finishing signal DOUT
-- Process Constants
   SUCCESS    = 0                           -- Successful Operation Status
   UNASIGNED  = 13007                       -- Unassigned Port Deletion Error
VAR
   cont_timer,
   last_node, node_ind,
   status               :INTEGER            -- Status from builtin calls
   prg_abrt             :BOOLEAN            -- Set when the program is aborted
   pth1                 :PATH               -- Process Path
   stop_pos             :XYZWPREXT          -- Process Stop Position
   perch_pos            :XYZWPR             -- Perch Position
   tmp_xyz              :XYZWPR             -- XYZWPR variable for temporary use
   indx                 :INTEGER            -- Used a FOR loop counter
   ports_ready          :BOOLEAN            -- Check if ports assigned
   cmt_str              :STRING[10]         -- Comment String
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```
---------------------------------------------------------------------------
----    Section 3:  Routine Declaration
---------------------------------------------------------------------------
---------------------------------------------------------------------------
----    Section 3-A:  TP_CLS  Declaration
----                  This routine is from ROUT_EX.kl and will
----                  clear the TP USER menu screen and force
----                  it to be visible.
---------------------------------------------------------------------------
ROUTINE tp_cls FROM rout_ex              -- ROUT_EX must also be loaded.
---------------------------------------------------------------------------
----    Section 3-B:  port_init  Declaration
----                  This routine assigns a value to ports_ready, which
----                  allows the ports to be initialized.  It resets the
----                  controller so that program execution may be continued
----                  automatically though the CONT_CH condition handler.
---------------------------------------------------------------------------
ROUTINE init_port
VAR
```

Draft

```
        reset_ok: BOOLEAN
BEGIN
  ports_ready = FALSE              -- Set false so ports will be initialized
  DELAY 500;
  RESET(reset_ok)                  -- Reset the controller
  IF (NOT reset_ok) THEN
     WRITE('Reset Failed', CR)
   ENDIF
  cont_timer = 0                   -- Set a timer to continue the process
  ENABLE CONDITION[CONT_CH]        -- Enabled the CONT_CH which continues
                                   -- program execution
END init_port
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```
-----------------------------------------------------------------------------
----    Section 3-C:  SET_MODE Declaration
----                  Sets up the mode of IO's.  Depending on the passed
----                  parameter the IO ports will be set to REVERSE
----                  and/or COMPLEMENTARY mode.  When the ports are set
----                  to REVERSE mode, the TRUE condition is represented by
----                  a FALSE signal.  When COMPLEMENTARY mode is selected
----                  for a port (odd number port), the  port n and n+1 are
----                  complementary signal of each other.
-----------------------------------------------------------------------------
ROUTINE set_mode(port_type:     INTEGER;
                 port_no:       INTEGER;
                 reverse:       BOOLEAN;
                 complmnt:      BOOLEAN)
VAR
    mode:            INTEGER
BEGIN  -- set_mode
 IF reverse THEN
    mode = 1                                 -- Set the reverse mode
 ELSE
    mode = 0
 ENDIF
 IF complmnt THEN
    mode = mode OR 2                         -- Set complementary mode
 ENDIF
 SET_PORT_MOD(port_type, port_no, mode, status)
END set_mode
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```
---------------------------------------------------------------------------
----    Section 3-D:  SETUP_PORTS Declaration
----          This section assumes that you do not have an AB or GENIUS I/O
----          or any other external I/O board.  Therefore, any previous port
----          assignments are no longer needed for this application, and
----          can be deleted.
---------------------------------------------------------------------------
ROUTINE setup_ports
VAR
    port_n  : INTEGER
BEGIN
-- Delete DIGITAL OUTPUT PORTS 1 thru 48
  FOR port_n = 0 to 5 DO
    -- Indexing of 0 to 5 may not be obvious, But look into the DIGITAL
    -- OUT Configuration screen in TP, you will see the 8 DIGITAL OUTPUT
    -- ports are grouped together in configuration.
    SET_PORT_ASG(IO_DOUT, port_n*8+1, 0, 0, 0, 0, 0, status)
    IF (status <> SUCCESS) AND (status <> UNASIGNED) THEN
      -- Verify that deletion by SET_PORT_ASG was successful
      WRITE ('SET_PORT_ASG built-in  for DOUT (deletion) failed',CR)
      WRITE ('Status = ',status,CR)
    ENDIF
  ENDFOR
    --  Assign the DIGITAL PORTS 1 THRU 48 as memory images.
  FOR port_n = 0 TO 5 DO
    SET_PORT_ASG(IO_DOUT, port_n*8+1, 0, 0, io_mem_boo, port_n*8+1, 8, status)
    IF (status <> 0 ) THEN     -- Verify that SET_PORT_ASG was successful
      WRITE ('SET_PORT_ASG built-in  for DOUT (assignment) failed',CR)
      WRITE ('Status = ',status,CR)
    ENDIF
  ENDFOR
  -- Suppose equipment-1 is turned ON by the DOUT[1] = TRUE signal and
  -- turned OFF by the DOUT[2] = TRUE signal. To avoid both signals being
  -- TRUE or FALSE at the same time, set DOUT[1] to be a complement.
  -- Once the DOUT[1] is set in complementary mode, the DOUT[1] and
  -- DOUT[2] will always show the opposite signal of each other.
  -- Thus avoiding the confusion of turning the equipment OFF and ON
  -- at the same time.
  -- Set port-1, port-3 and port-5 to COMPLEMENTARY mode.
  FOR port_n = 1 to 6 DO
    SET_MODE(io_dout, port_n, TRUE, TRUE)
    IF (status <> SUCCESS) THEN
      WRITE ('SET_PORT_MODE Failed on port ',1,CR)
      WRITE ('With Status = ',status,CR)
    ENDIF
  ENDFOR
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```
-- Set appropriate comments for the ports.
  SET_PORT_CMT(IO_DOUT, EQIP_READY, 'Equip-READY ',status)
  IF (status <> 0 ) THEN             -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, EQIP_NOT_RD, 'E - NOT READY',status)
  IF (status <> 0 ) THEN             -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, EQIP_ERROR, 'Equip- ERROR',status)
  IF (status <> 0 ) THEN            -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, EQIP_FIXED, 'Equip- FIXED',status)
  IF (status <> 0 ) THEN             -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, EQIP_ON, 'Equip- ON',status)
  IF (status <> 0 ) THEN            -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, EQIP_OFF, 'Equip- OFF',status)
  IF (status <> 0 ) THEN            -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, NODE_PULSE, 'Pulse @ node',status)
  IF (status <> 0 ) THEN            -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
  SET_PORT_CMT(IO_DOUT, FINISH, 'Finish PATH',status)
  IF (status <> 0 ) THEN            -- Verify SET_PORT_CMT was successful
    WRITE ('SET_PORT_CMT built-in failed',CR)
    WRITE ('Status = ',status,CR)
  ENDIF
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

Draft

```
  TP_CLS   -- clear the teach pendant USER screen

  WRITE ('PORT SETUP IS COMPLETE',CR)
  WRITE ('AT THIS POINT YOU NEED TO COLD START',CR)
  WRITE ('Configuration changes of PORTs will not',CR)
  WRITE ('take effect until after a COLD START.',CR,CR)
  WRITE ('Once the controller is ready after',CR)
  WRITE ('COLD START, re-load this program',CR)
  WRITE ('rerun.',CR)

  ports_ready = TRUE      -- Set the ports_ready variable so re-execution of
                          -- this routine, setup_ports, is not performed.
  -- Aborting program to allow for the cold start.
  ABORT
END setup_ports
----------------------------------------------------------------------------
----    Section 3-E:  SERVICE_RTN interrupt routine Declaration
----                  This routine waits until the equipment has been
----                  serviced and then moves the robot back to where
----                  it was before servicing.  It then sets the DOUT
----                  to notify that the equipment is ready.
----------------------------------------------------------------------------
ROUTINE service_rtn
BEGIN
  TP_CLS
-- store the current position, where the process is stopped due to failure
-- so after resuming the process can be started from this point.
  stop_pos = CURPOS(0,0)
-- move the robot to the perch position so the equipment
-- can be worked on safely.
  SET_POS_REG(1, perch_pos, status) —— Put perch_pos in PR[1]
  move_to_pr -- Call TP program to move to PR[1]
  WRITE (chr(139),' PLEASE READ ',chr(143),CR)     --Display in reverse video
  WRITE ('Equipment - 1 failed during',CR)
  WRITE ('processing. Motions have been stopped.',CR)
  WRITE ('Please Fix the equipment then',CR)
  WRITE ('SET DOUT[',EQIP_FIXED,'] = TRUE ',CR)
        --Display the following message in reverse video
  WRITE (chr(139), 'IMPORTANT: Once the DOUT is set, current',CR)
  WRITE ('STOPPED motion will be RESUMED',chr(143),CR)
  WAIT FOR DOUT[EQIP_FIXED]  -- wait until equipment has been fixed
```

Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```
  -- Move to the point where the process was stopped
  SET_EPOS_REG(1, stop_pos, status)
```

```
  move_to_pr -- Call TP program to move to PR[1]
  -- Enable the SERVICE-DONE condition handler to resume the process.
  ENABLE CONDITION[SERV_DONE]
  -- Wait a sufficient time to allow equipment to warm up and get ready for
  -- processing after the fix is completed.
  WRITE ('Continuing the process.....',CR)
  DELAY 2000

  --Signal that the equipment is now ready.
  DOUT[EQIP_READY] = TRUE

  -- Force the teach pendant back to the IO screen
  FORCE_SPMENU(tp_panel, SPI_TPDIGIO, 1)
END service_rtn
-----------------------------------------------------------------------------
----    Section 3-F:  Routines frst_nod, mid_nods and end_nod are TP
----    routines for doing moves with Time Before clauses
-----------------------------------------------------------------------------
ROUTINE frst_nod FROM frst_nod      -- frst_nod must also be loaded.
--   1:L PR[1] 100mm/sec CNT100 TB   0.00sec,DO[1:NODE_PULSE]=PULSE,1.0sec  ;
--   2:L PR[1] 100mm/sec CNT100 TB   0.00sec,DO[2:EQUIP_ON]=PULSE,2.0sec    ;
ROUTINE mid_nods FROM mid_nods      -- mid_nods must also be loaded.
--   1:L P[1] 100mm/sec CNT100 TB     0.00sec,DO[1:NODE_PULSE]=PULSE,1.0sec  ;
ROUTINE end_nod FROM end_nod        -- end_nod must also be loaded.
--   1:L P[1] 100mm/sec FINE TB       .20sec,DO[3:FINISH]=ON    ;
--   2:L P[1] 100mm/sec FINE TB       0.00sec,DO[1:NODE_PULSE]=PULSE,1.0sec  ;
```

Draft

Setting Up Digital Output Ports for Process Monitoring - Main

```
------------------------------------------------------------------------------
----    Section 4:  Main Program
------------------------------------------------------------------------------
BEGIN   -- DOUT_EX
------------------------------------------------------------------------------
----    Section 4-A: Global Condition Handler Declaration
------------------------------------------------------------------------------
CONDITION[UNINIT_CH]:
 WHEN ERROR[12311] DO                  -- Trap UNINITIALIZATION error
   NOMESSAGE                           -- Supress the error message
   UNPAUSE                             -- UNPAUSE
   init_port                           -- Allow ports to be initialized.
ENDCONDITION
ENABLE CONDITION[UNINIT_CH]
CONNECT TIMER to cont_timer
CONDITION [CONT_CH]:
   WHEN cont_timer > 1000 DO
```

```
    CONTINUE
ENDCONDITION
```

Setting Up Digital Output Ports for Process Monitoring - Main

```
------------------------------------------------------------------------------
----    Section 4-B: Verify PATH variable, pth1, has been taught.
------------------------------------------------------------------------------
tp_cls                         -- Routine Call; Clears the TP USER menu and
                               -- forces the TP USER menu to be visible.
-- Check the number of nodes in the path
IF PATH_LEN(pth1) = 0 THEN                  -- Path is empty (no nodes)
   WRITE ('You need to teach the path.',CR) -- Display instructions
   WRITE ('before executing this program.',CR)
   WRITE ('Teach the PATH variable pth1', CR, 'and restart the program',CR)
   WRITE ('PROGRAM ABORTED',CR)
   ABORT                           -- ABORT the task. do not continue
                                   -- There are no nodes to move to
ENDIF
-- Set Perch Position
-- This position is used in the service_rtn routine
IF UNINIT(perch_pos) THEN

   WRITE ('PERCH POSITION is not recorded.',cr)
   WRITE ('RELEASing Motion Control to TP.',cr)
   WRITE ('Please Move robot to desired Perch Pos',cr)
   -- Wait until the DEADMAN switch is HELD and
   -- TP is TURNED ON to move robot from TP.
   WHILE ((TPIN[248] = ON) AND (TPIN[247] = ON)) DO
     WRITE TPPROMPT(CHR(128),CHR(137),'Hold Down the DEAD-MAN switch')
     DELAY 500
   ENDWHILE
   -- Release motion control from the KAREL program to the
   -- TP control.  Robot can be moved to desired Perch
   -- position with out disturbing the flow of this KAREL task.
   RELEASE
   WHILE (TPIN[249] = OFF ) DO
     WRITE TPPROMPT(CHR(128),CHR(137),'Turn the TP ON')
     DELAY 1000
   ENDWHILE
   WRITE ('ROBOT is ready to move from TP',cr)
   WRITE ('After moving ROBOT to PERCH position ',cr)
   WRITE ('Turn OFF the TP then RELEASE DEADMAN ',cr)
   WHILE (TPIN[249] = ON ) DO
     WRITE TPPROMPT(CHR(128),CHR(137),'Turn OFF TP, after MOVE is done
     DELAY 10000
```

```
ENDWHILE
```

Setting Up Digital Output Ports for Process Monitoring - Main

```
-- KAREL program execution will not continue passed ATTACH
   -- statement until the TP is turned OFF.
   -- Wait until the TP is TURNED OFF after move from TP is completed.
   WHILE (TPIN[249] = ON ) DO
     DELAY 2000
   ENDWHILE
  -- At this point the robot is positioned to the desired
   -- Perch position. Get the motion
   -- control back from TP and record the perch position.
   ATTACH
   perch_pos = CURPOS(0,0,1)
ENDIF
-----------------------------------------------------------------------------
----Section 4-C: Set up Ports and Declare Process dependant condition
handler
-----------------------------------------------------------------------------
-- Port assignments need to be assigned only once and take effect
-- after the controller is COLD STARTED.
-- The ports_ready variable is used to determine if the ports have
-- already been assigned by this program.
-- Therefore only the first execution of this program will assign the ports
IF  NOT(ports_ready) THEN
   setup_ports
ENDIF
-- Define a condition handler to trap equipment failure.
-- If equipment fails during the process, then the DOUT[EQIP_ERROR] is
-- set to TRUE. Which will stop the motion and require the equipment to be
-- fixed before motion can be resumed.
CONDITION[EQIP_FAIL]:
  WHEN DOUT[EQIP_ERROR]  DO
    STOP
    DOUT[EQIP_FIXED] = FALSE
    DOUT[EQIP_READY] = FALSE
    ENABLE CONDITION[RESTART]
    service_rtn
ENDCONDITION
ENABLE CONDITION[EQIP_FAIL]
-- Define a condition handler to monitor the servicing process.
-- Once Servicing/Fixing of equipment is complete, wait for the equipment
-- to be in READY mode.  When the equipment is READY, signal an event
-- which will restart the process where it left off.  The SERV_DONE
-- condition handler is ENABLED from the SERVICE_RTN interrupt routine.
```

```
CONDITION[SERV_DONE]:
   WHEN DOUT[EQIP_READY] DO
    SIGNAL EVENT[WARMED_UP]
    DOUT[EQIP_ERROR] = FALSE
ENDCONDITION
```

Setting Up Digital Output Ports for Process Monitoring - Main

```
-- Define a condition handler to monitor when the warm up is complete, then
-- resume the stopped motion and continue the process.  Also re-enable
-- the EQIP_FAIL condition handler to continue monitoring for equipment
-- failure.
CONDITION[RESTART]:
  WHEN EVENT[WARMED_UP] DO
    RESUME
    ENABLE CONDITION[EQIP_FAIL]
ENDCONDITION
-----------------------------------------------------------------------------
----    Section 4-D: Do process manipulation
-----------------------------------------------------------------------------
-- Using the PATH_LEN built-in find out the last node of the path
last_node = PATH_LEN(pth1)
-- Setting EQIP_ERROR/EQIP_FIXED number ports to be simulated.
-- This setup does not require cold start, can change the port to be
-- simulated on the fly.
SET_PORT_SIM(io_dout, NODE_PULSE, 1, status)
IF (status <> SUCCESS) THEN
  WRITE ('SET_PORT_SIM Failed on port ',indx,CR)
  WRITE ('With Status = ',status,CR)
ENDIF
SET_PORT_SIM(io_dout, FINISH, 1, status)
IF (status <> SUCCESS) THEN
  WRITE ('SET_PORT_SIM Failed on port ',indx,CR)
  WRITE ('With Status = ',status,CR)
ENDIF
```

Setting Up Digital Output Ports for Process Monitoring - Main

```
WRITE (' NOW YOU WILL SEE THE DOUT[',NODE_PULSE,'] PULSE',CR)
WRITE (' as the robot moves through every node.',CR,CR)
WRITE (' To simulate EQUIPMENT failure, change ',CR)
WRITE (' DOUT[',EQIP_ERROR,'] = TRUE. ',CR)
WRITE (' Press ''ENTER'' to Continue',CR)
READ(CR)
-- Change the TP display to the DI/O Screen
```

Draft

```
FORCE_SPMENU(tp_panel, SPI_TPDIGIO, 1)
-- Moving along path when equipment is ready.
-- Need to turn on equipment-1 for 1/2 second when robot position
-- is at 1st node.  Pulse the DOUT[NODE_PULSE] for every node
-- Turn on the DOUT[FINISH] about 200 ms before the last node.
IF DOUT[EQIP_READY] THEN
  tmp_xyz = pth1[1] -- Convert path node to XYZWPR
  SET_POS_REG(1, tmp_xyz, status) -- Put position in PR[1]
  frst_nod -- Call TP program to do move
  FOR node_ind = 2 TO (last_node - 1) DO
    tmp_xyz = pth1[node_ind]
    SET_POS_REG(1, tmp_xyz, status)
    mid_nods
  ENDFOR
  tmp_xyz = pth1[last_node]
  SET_POS_REG(1, tmp_xyz, status)
  end_nod
  ELSE
  FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
  WRITE (' Equipment is not READY',CR)
  WRITE (' Set equipment to READY MODE',CR)
  WRITE (' before executing this program.',CR)
  WRITE (' SET DOUT[',EQIP_READY,'] = TRUE ',CR)
  ABORT
ENDIF
WRITE TPFUNC    (CHR(128),CHR(137))  -- Home Cursor and Clear to End-of-line
                                     -- This will remove the ABORT displayed
                                     -- above F1.
END DOUT_EX
```

B.2 COPYING PATH VARIABLES

This example shows the different ways of copying and appending PATH variables. The PATH Data Type can be copied from one to another only with hard coded path variable names. However, user defined paths can be copied from one to another. The path variable names can be determined during execution of the program.

Copy Path Variables Program - Overview

```
-----------------------------------------------------------------------------
----    Detail about CPY_PTH.Kl
-----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:    In Section:
----    Action:
----    Clauses:
```

```
----           FROM                         Sec 3-A
----           IN DRAM                      Sec 2
----           WHEN                         Sec 4-A
----      Conditions:
----      Data types:
----           ARRAY OF STRING              Sec 2
----           BOOLEAN                      Sec 2; 3-C
----           FILE                         Sec 2
----           INTEGER                      Sec 2; 3-B,C
----           PATH                         Sec 2
----           STRING                       Sec 2; 3-B
----           STRUCTURE...ENDSTRUCTURE     Sec 2
----           USER DEFINED PATH            Sec 2
----           XYZWPR                       Sec 2
----      Directives:
----           ALPHABETIZE                  Sec 1
----           COMMENT                      Sec 1
----           CMOSVARS                     Sec 1
----           CRTDEVICE                    Sec 1
----           INCLUDE                      Sec 2
```

Copy Path Variables Program - Overview and Declaration Section

```
----      Built-in Functions & Procedures:
----           APPEND_NODE                  Sec 4-D
----           BYNAME                       Sec 4-E
----           CALL_PROG                    Sec 4-B
----           COPY_PATH                    Sec 3-C; 4-D
----           CNV_INT_STR                  Sec 4-E
----           CREATE_VAR                   Sec 4-E
----           CURPOS                       Sec 4-B
----           DELETE_NODE                  Sec 4-C
----           LOAD                         Sec 4-B
----           PATH_LEN                     Sec 4-C,E
----           PROG_LIST                    Sec 4-B
----           READ_KB                      Sec 3-B
----           SET_CURSOR                   Sec 4-E
----           SET_FILE_ATR                 Sec 4-A
----           SET_POS_REG                  Sec 4-D
----           SET_VAR                      Sec 4-B
----           SUB_STR                      Sec 4-E
----           VAR_LIST                     Sec 4-E
----      Statements:
----           ABORT                        Sec 4-C,E
----           CLOSE FILE                   Sec 4-E
----           FOR .... ENDFOR              Sec 3-C; 4-C,D,E
```

Draft

```
----            IF...THEN...ENDIF            Sec 3-B,C; 4-B,C,D,E
----            OPEN FILE                    Sec 4-A
----            REPEAT...UNTIL               Sec 3-B; 4-E
----            ROUTINE                      Sec 3
----            WRITE                        Sec 3-B,C; 4-A,B,C,E
----            USING...ENDUSING             Sec 4-D
----    Reserve Word:
----            BEGIN                        Sec 3-B,C; 4
----            END                          Sec 3-B,C; 4-E
----            PROGRAM                      Sec 1
----            TYPE                         Sec 2
----            VAR                          Sec 2
----     Predefined File Names:
----           CRTFUNC                       Sec 3-B
----           CRTPROMPT                     Sec 3-B,C
-------------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-------------------------------------------------------------------------------
PROGRAM CPY_PTH
%ALPHABETIZE
%COMMENT = 'COPY PATH'         -- Display information by default to
CRT/KB
%CRTDEVICE
%CMOSVARS                     -- Use CMOS RAM to store all static variables,
                              -- except those specified with IN DRAM
```

Copy Path Variables Program - Declaration Section

```
-------------------------------------------------------------------------------
----    Section 2:  Constant, Variable and Type Declarations
-------------------------------------------------------------------------------
CONST
   SUCCESS       = 0 -- The value returned from all built-ins when successful
TYPE
    node_struc    = STRUCTURE      -- Create a user defined node structure
      posn_dat      :XYZWPR
    ENDSTRUCTURE
    user_path     = PATH nodedata = node_struc  --Create a user defined path
VAR
    pth1,
    pth2,
    pth3
    pth4            :PATH         -- These are system defined PATHs
    upth1,
    upth2,
    upth3,
```

```
    upth4           :user_path    -- These are user defined PATHs

    p1_len,
    p2_len,
    status, node_ind,
    total_node      :INTEGER
    F1_press,
    F2_press        :BOOLEAN
    src_num,
    des_num         :INTEGER
    dummy_str,
    src_var,
    des_var         :STRING[20]
    cur_name        :STRING[12]
    entry           :INTEGER
    var_type        :INTEGER
    mem_loc         :INTEGER
```

Draft

Copy Path Variables Program - Storing Variables in Memory

```
-- Store the following variables in DRAM, which is temporary memory
    indx      IN DRAM  :INTEGER
    prog_name IN DRAM  :STRING[10]
    prog_type IN DRAM  :INTEGER
    n_match   IN DRAM  :INTEGER
    n_skip    IN DRAM  :INTEGER
    format    IN DRAM  :INTEGER
    ary_nam   IN DRAM  :ARRAY[5] OF STRING[20]
    prog_indx IN DRAM  :INTEGER
    do_copy   IN DRAM  :BOOLEAN
    crt_kb    IN DRAM  :FILE
%INCLUDE KLEVKMSK     -- system supplied file: definition of KC_FUNC_KEY
%INCLUDE KLEVKEYS     -- system supplied file: definition of KY_F1 & KY_F2
```

Copy Path Variables Program - Monitor User Response

```
------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----    Section 3-A:  CRT_CLS Declaration
------------------------------------------------------------------------------
ROUTINE CRT_CLS FROM rout_ex -- include this routine from the file rout_ex.kl
------------------------------------------------------------------------------
----  Section 3-B:  YES_NO Declaration
----                LABEL the F1 key as YES and F2 key as NO, ask for user
```

```
----                  confirmation.  These two keys are monitored by global
----                  condition handler,so User response can be trapped.
------------------------------------------------------------------------------
ROUTINE YES_NO
VAR
 key_press : INTEGER
 str : STRING[1]
 n_chars: INTEGER
 l_status: INTEGER
 BEGIN  ---- YES_NO
  WRITE CRTFUNC   (CHR(128),CHR(137))  --- Clear  Window, Home Cursor
  -- Display YES above F1 & NO above F2 & clear rest of Function window
  WRITE CRTFUNC   ('   YES     NO ',chr(129))
  F1_press = FALSE
  F2_press = FALSE
  REPEAT -- until user presses either the F1 or F2 key
    -- Read just the function keys of the CRT/KB.
    -- The read will be satisfied only when a function key is pressed.
    READ_KB (crt_kb, str , 0, 0, kc_func_key, -1,
             '', n_chars, key_press, l_status)
    -- key_press must be converted from a "raw" CRT character to the teach
    -- pendant equivalent character.g
    key_press = $CRT_KEY_TBL[key_press+1]
    IF (key_press = ky_f1) THEN -- The user pressed F1
      F1_press = true
    ENDIF
    IF (key_press = ky_f2) THEN -- The user pressed F2
      F2_press = true
    ENDIF
  UNTIL ((f1_press = TRUE) OR (F2_press = TRUE))
  WRITE CRTFUNC    (CHR(128),CHR(137))  --- Clear Window, Home Cursor
  WRITE CRTPROMPT  (CHR(128),CHR(137))  --- Clear Window, Home Cursor
END YES_NO
```

Copy Path Variables Program - Copying Path Variables

```
------------------------------------------------------------------------------
---- Section 3-C: PTH_CPY Declaration
----        Copy one user defined path variable to another user defined path
----        variable. The first parameter is the source path.  The second
----        parameter is the destination path. The path parameters can only
----        be passed using BYNAME and the paths must be user defined
------------------------------------------------------------------------------
ROUTINE PTH_CPY(src_path: USER_PATH; des_path: USER_PATH)
VAR
    node_indx  :INTEGER
```

Draft

```
    do_it       :BOOLEAN
    l_stat      :INTEGER
BEGIN --- pth_cpy
CRT_CLS     -- Clear the CRT/KB USER Menu screen
do_it = true
    WRITE ('Perform copy?',CR)
    yes_no
    do_it = F1_press -- F1_press will be true only if the user selected
YES
    IF (do_it) THEN
     --  Copy the entire path of src_path to des_path
       COPY_PATH (src_path, 0,0, des_path, l_stat)
        IF (l_stat <> 0) THEN
          WRITE ('Error in COPY_PATH', l_stat, CR)
        ELSE
          WRITE ('Path Copy function Completed ',cr)
        ENDIF
    ELSE
      WRITE ('Path Copy function canceled by choice',cr,cr)
    ENDIF
END PTH_CPY
----------------------------------------------------------------------------
----   Section 3-D:  Routine move_to_pr is a TP
----   routine for doing moves
----------------------------------------------------------------------------
ROUTINE move_to_pr FROM move_to_pr      -- move_to_pr must also be loaded.
--   1:J PR[1] 100% FINE    ;
```

Draft

Copy Path Variables Program - Opens CRT/KB & Sends Data to Default Device

```
-----------------------------------------------------------------------------
----    Section 4:  Main Program
-----------------------------------------------------------------------------
BEGIN --- CPY_PATH
-----------------------------------------------------------------------------
----    Section 4-A: Open CRT KB for reading YES/NO inputs from user
-----------------------------------------------------------------------------
CRT_CLS --  will force the CRT USER menu to be visible & clear the screen
 SET_FILE_ATR(crt_kb, ATR_FIELD) -- Needed so the read is satisfied with one
                                 -- character.
 OPEN FILE crt_kb ('RO', 'KB:crkb') -- Open a file to the CRT/KB
                                    -- Used within the YES_NO routine.
-----------------------------------------------------------------------------
----    Section 4-B: Check if SAVE_VRS.PC is loaded. If loaded then execute
-----------------------------------------------------------------------------
---- First check if the "SAVE_VRS" program is loaded or not.
```

```
prog_name = 'SAVE_VRS' -- Only interested in SAVE_VRS program
prog_type = 6          -- Interested only in PC type files
n_skip = 0             -- First time do not skip any files
format = 1             -- Return just the filename
do_copy = TRUE
WRITE ('Checking Program List',cr)
PROG_LIST(prog_name, prog_type, n_skip, format, ary_nam, n_match, status)
  IF (status <>SUCCESS ) THEN
    IF (status = 7073 ) THEN  ---- Program does not exist error
        --- Program SAVE_VRS is not loaded on the controller.
      WRITE ( 'LOADING ',prog_name, CR)
      LOAD (prog_name+'.PC', 0, status)
      IF (status <> SUCCESS) THEN
        WRITE ('Error loading ', prog_name,cr)
        WRITE CRTPROMPT('Copy paths WITHOUT saving program variables?',CR)
        YES_NO
        do_copy = F1_press -- F1_press is true only if user selected
YES
                           -- Copy without saving variables.
      ENDIF
    ELSE
     -- The program listing failed.
      WRITE ('PROG_LIST built-in failed',cr,' with Status = ',status,cr)
      WRITE CRTPROMPT('Copy paths WITHOUT saving program variables?',CR)
      YES_NO
      do_copy = F1_press -- F1_press is true only if user selected
YES
    ENDIF               -- Copy without saving variables.
  ENDIF
```

Copy Path Variables Program - Checks Path Initialization

```
IF (status = SUCCESS)  THEN
  -- This is one way to set variables within another program without
  -- using the FROM clause in the variable section.
  -- It is very useful if you want to have run-time independent code,
  -- where the program or variable name you are setting is not
  -- known until run-time.
  cur_name = CURR_PROG
   SET_VAR (entry, prog_name, 'del_vr', TRUE, status)
   SET_VAR (entry, prog_name, 'prog_name',cur_name , status)
   SET_VAR (entry, prog_name, 'sav_type', 1, status)
   SET_VAR (entry, prog_name, 'dev', 'FLPY:', status)
   WRITE ('Saving program variables before copy', CR)
   CALL_PROG(prog_name,prog_indx) -- call SAVE_VRS
ENDIF
```

```
---------------------------------------------------------------------------
----    Section 4-C: Check for initialization of PATHs pth1 and pth2.
---------------------------------------------------------------------------
IF (NOT do_copy) THEN
  WRITE ('Program exiting, unable to save variables,',cr)
  WRITE ('before copying path''s content',cr)
  -- NOTICE:
  --  Two single quotes will display as one single quote
  --  so this write statement will appear as :
  --  "before copying path's content"
  ABORT
ENDIF
WRITE ('Checking Variable initialization',cr)
-- Check if the pth variables are initialized.
p1_len = PATH_LEN(pth1)  ; p2_len = PATH_LEN(pth2)
IF ( (p1_len = 0) OR (p2_len = 0) )THEN
  WRITE  ('PTH1 or PTH2 is empty path',cr)
  WRITE  ('Please make sure both paths are taught then restart',cr)
  ABORT  -- Cannot copy uninitialized variables.
ENDIF
-- Check if the pth3 variable is initialized.
IF  (PATH_LEN(pth3) <> 0) THEN
  WRITE ('Deleting nodes from pth3',cr) -- Delete the old path of pth3
  FOR indx = PATH_LEN(pth3) DOWNTO 1 DO
    -- its easy to delete nodes from the end instead of deleting node from
    -- the front end. Since after every deletion the nodes are renumbered.
    DELETE_NODE(pth3, indx, status) -- Delete last node of pth3
    IF status <> SUCCESS THEN
       WRITE ('While Deleting ',indx, ' node',cr)
       WRITE ('DELETE_NODE unsuccessful: Status = ',status,cr)
    ENDIF
  ENDFOR
ENDIF
```

Draft

Copy Path Variables Program - Path Initialization

```
-----------------------------------------------------------------------------
----    Section 4-D: Add pth1 and pth2 together to create pth3.
----                 Move along pth1 and pth2.
----                 Move backwards through pth3.
-----------------------------------------------------------------------------
total_node = p1_len + p2_len     -- Total number of nodes needed for pth3
-- Copy the node data from pth1 to pth3
WRITE ('copying pth1 to pth3',cr)
COPY_PATH (pth1, 0,0, pth3, status)
     IF (status <> 0) THEN
```

```
     WRITE ('ERROR in COPY_PATH', status, CR)
   ENDIF
-- Create the required number of nodes for pth3.
-- We know that pth3 now has PATH_LEN(pth3) nodes.
WRITE ('Appending nodes to pth3',cr)
FOR indx = p1_len+1 TO total_node DO -- Append the correct number of nodes.
   APPEND_NODE(pth3, status)
   IF (status <> 0) THEN
     WRITE ('While Appending ',indx, ' node',cr)
     WRITE ('APPEND_NODE unsuccessful: Status = ',status,cr)
   ENDIF
ENDFOR
-- Append the node data of pth2 to pth3.
WRITE ('Appending pth2 to pth3',cr)
FOR indx = p1_len+1 TO total_node DO
  USING pth2[indx - p1_len] DO
    pth3[indx].node_pos    = node_pos
  ENDUSING
ENDFOR
-- Move along the path pth1 and pth2
WRITE ('Moving Along Path pth1',cr)
FOR node_ind = 1 TO p1_len DO
  tmp_xyz = pth1[node_ind]
  SET_POS_REG(1, tmp_xyz, status)
  move_to_pr -- Call TP program to move to PR[1]
ENDFOR
WRITE ('Moving Along Path pth2',cr)
FOR node_ind = 1 TO p2_len DO
  tmp_xyz = pth2[node_ind]
  SET_POS_REG(1, tmp_xyz, status)
  move_to_pr -- Call TP program to move to PR[1]
ENDFOR
--Copy pth3 in reverse order to pth4
COPY_PATH (pth3, PATH_LEN(pth3), 1, pth4, status)
IF (status <> 0) THEN
  WRITE ('ERROR in COPY_PATH', status, CR)
ENDIF--- Move along pth4 which is a reverse order of pth3.
WRITE ('Moving Along Path pth4',cr)
FOR node_ind = 1 TO PATH_LEN(pth4)  DO
  tmp_xyz = pth4[node_ind]
  SET_POS_REG(1, tmp_xyz, status)
  move_to_pr -- Call TP program to move to PR[1]
ENDFOR
```

Copy Path Variables Program - Copy User Defined Paths

```
-----------------------------------------------------------------------------
----    Section 4-E: Copy User Defined Paths.
----                 Copy one user defined path to another user defined path,
----                 where the user specifies which paths to be copied.
-----------------------------------------------------------------------------
CRT_CLS
SET_CURSOR(OUTPUT,2,10, status)     -- Position cursor nicely on CRT
IF (status <> 0 ) THEN
   WRITE ('SET_CURSOR built-in failed with status = ',status,cr)
ENDIF
-- write message in reverse video and then set back to normal video
WRITE (chr(139),' COPY PATH FUNCTION',chr(143),CR,cr)
WRITE ('Currently you have the following ',cr)
WRITE ('User Defined Paths',cr,cr)
n_skip = 0
var_type = 31                     -- Get listing of only PATH type variables
REPEAT
VAR_LIST ('CPY_PTH', '*',var_type, n_skip, 2, ary_nam, n_match, status)
  FOR indx = 1 TO n_match DO
   IF (SUB_STR (ary_nam[indx], 1, 4) = 'UPTH') THEN -- Verify it's one of
                                                    -- the user defined paths
    WRITE (ary_nam[indx], CR)
   ENDIF
  ENDFOR
  n_skip = n_skip + n_match
UNTIL (n_match < ARRAY_LEN(ary_nam))
Write ('Enter the source path number:')
  READ(src_num);
Write ('Enter the destination path number:')
  READ(des_num);
CNV_INT_STR(src_num,2,0,dummy_str)       -- Convert source number to string
src_var = 'UPTH'+ SUB_STR(dummy_str,2,1) -- SUB_STR will remove the leading
                                        -- blank from dummy_str before
                                        -- concatenating to create the
var_type = 0                            -- source variable name
VAR_LIST ('CPY_PTH', src_var, var_type, 0, 2, ary_nam, n_match, status)
IF (status <> SUCCESS) THEN
  WRITE ('Var_list unsuccessful for src_var: status ', status, cr)
ENDIF
```

Copy Path Variables Program - Copy User Defined Paths Continued

```
-- If the variable does not exist create it.
IF (n_match = 0) THEN
  CREATE_VAR ('', src_var, '', 'USER_PATH', 1, 0, 0, 0, status, mem_loc)
  IF (status <> SUCCESS) THEN
```

```
    WRITE ('Error creating ', src_var, ':', status, cr)
  ENDIF
ENDIF
--Create the destination variable name
CNV_INT_STR(des_num,2,0,dummy_str)      -- Convert des_num to a string
des_var = 'UPTH'+ SUB_STR(dummy_str,2,1)-- The SUB_STR will remove the leading
                                        -- blank from dummy_str before
                                        -- concatenating to create the
                                        -- source variable name
-- Verify that the des_var variable exists.
VAR_LIST ('CPY_PTH', des_var, var_type, 0, 2, ary_nam, n_match, status)
IF (status <> SUCCESS) THEN
  WRITE ('Var_list unsuccessful for des_vr: status', status, cr)
ENDIF
-- If the variable does not exist create it.
IF (n_match = 0) THEN
  CREATE_VAR ('', des_var, '', 'USER_PATH', 1, 0, 0, 0, status, mem_loc)
  IF (status <> SUCCESS) THEN
    WRITE ('Error creating ', des_var, ':', status, cr)
  ENDIF
ENDIF
-- Copy the specified source path to the specified destination path
pth_cpy(BYNAME('', src_var, indx), BYNAME('', des_var, indx) )
-- Close file before quitting
CLOSE FILE crt_kb
WRITE ('CPY_PTH example completed',cr)
END CPY_PTH
```

B.3 SAVING DATA TO THE DEFAULT DEVICE

This program will save variables or teach pendant programs to the default device. If the user specified to overwrite the file then the file will be deleted before performing the save.

Note This program is called by the CPY_PTH.KL program. Refer to Section B.2 , for information on CPY_PTH.KL.

Saving Data Program - Overview

```
-----------------------------------------------------------------------------
----    SAVE_VRS.KL
-----------------------------------------------------------------------------
----    Section 0:  Detail about SAVE_VRS.KL
-----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:    In Section:
----    Actions:
```

```
----    Clauses:
----    Conditions:
----    Data types:
----            BOOLEAN                     Sec 2
----            INTEGER                     Sec 2
----            STRING                      Sec 2
----    Directives:
----            COMMENT                     Sec 1
----            ENVIRONMENT                 Sec 1
----            NOLOCKGROUP                 Sec 1
----    Built-in Functions & Procedures:
----            DELETE_FILE                 Sec 4-B
----            SAVE                        Sec 4-B
----    Statements:
----            IF, THEN, ENDIF             Sec 4-B
----           SELECT, CASE, ENDSELECT      Sec 4-A
----            WRITE                       Sec 4-B
----    Reserve Words:
----            BEGIN                       Sec 4
----            CONST                       Sec 2
----            CR                          Sec 4-B
----            END                         Sec 4-B
----            PROGRAM                     Sec 1
----            VAR                         Sec 2
```

Draft

Saving Data Program - Declarations Section

```
---------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
---------------------------------------------------------------------------
PROGRAM SAVE_VRS
%NOLOCKGROUP
%COMMENT = 'Save .vr, .tp, .sv'
%ENVIRONMENT MEMO
%ENVIRONMENT FDEV
---------------------------------------------------------------------------
----    Section 2:  Constant, Variable and Type Declarations
---------------------------------------------------------------------------
CONST
  DO_VR  = 1       -- Save variable file(s)
  DO_TP  = 2       -- Save TP program(s)
  DO_SYS = 3       -- Save system variables
  SUCCESS = 0      -- The value expected from all built-in calls.
VAR
  sav_type  : INTEGER     -- Specifies the type of save to perform
  prog_name : STRING[12]  -- The program name to save
```

```
  status    : INTEGER    -- The status returned from the built-in calls
  file_spec : STRING[30]  -- The created file specification for SAVE
  dev       : STRING[5]   -- The device to save to specify whether to
  del_vr    : BOOLEAN   -- delete file_spec before performing the SAVE.
--------------------------------------------------------------------------
----    Section 3:  Routine Declaration
--------------------------------------------------------------------------
```

Saving Data Program - Create File Spec

```
--------------------------------------------------------------------------
----    Section 4:  Main Program
--------------------------------------------------------------------------
BEGIN -- SAVE_VRS
--------------------------------------------------------------------------
----    Section 4-A: Create the file_spec, which contains the device, file
----                 name and type to be saved.
---------------------------------------------------------------------------
  SELECT (sav_type) OF
    CASE (DO_VR):
     -- If prog_name is '*' then all PC variables will be saved with the
     -- correct program name, irregardless of the file name part of
     -- file_spec.
     file_spec = dev+prog_name+'.VR'   -- Create the variable file name
    CASE (DO_TP):
     -- If prog_name is '*' then all TP programs will be saved with the
     -- correct TP program name, irregardless of the prog_name part of
     -- file_spec.
     file_spec = dev+prog_name+'.TP'   -- Create the TP program name
    CASE (DO_SYS):
     prog_name = '*SYSTEM*'
     file_spec = dev+'ALLSYS.SV'       -- All system variables will be
                                       -- saved into this one file.
  ENDSELECT
```

Saving Data Program - Delete/Overwrite

```
---------------------------------------------------------------------------
----    Section 4-B: Decide whether to delete the file before saving
----                and then perform the SAVE.
---------------------------------------------------------------------------
   -- If the user specified to delete the file before saving, then
   -- delete the file and verify that the delete was successful.
   -- It is possible that the delete will return a status of:
   --  10003 : "file does not exist", for the FLPY: device
```

```
--    OR
--  85014 : "file not found", for all RD: and FR: devices
-- We will disregard these errors since we do not care if the
-- file did not previously exist.
IF (del_vr = TRUE)  THEN
  DELETE_FILE (file_spec, FALSE, status) -- Delete the file.
  IF (status <> SUCCESS) AND (status <> 10003) AND
     (status <> 85014) THEN
    WRITE ('Error ', status,' in attempt to delete ',cr, file_spec,cr)
  ENDIF
ENDIF
-- If prog_name is specified as an '*' for either .tp or .vr files then
-- the SAVE builtin will save the appropriate files/programs with the
-- correct names.
SAVE (prog_name, file_spec, status) -- Save the variable/program
IF (status <> SUCCESS) THEN        -- Verify SAVE was successful
  WRITE ('error saving ', file_spec, 'variables', status, cr)
ENDIF
END  SAVE_VRS
```

B.4 STANDARD ROUTINES

This program is made up of several routines which are used through out the examples. The following is a list of the routines within this file:

- CRT_CLS Clears the CRT/KB USER Menu screen
- TP_CLS Clears the teach pendant USER Menu screen

Standard Routines - Overview

```
-------------------------------------------------------------------------
----    ROUT_EX.KL
-------------------------------------------------------------------------
----    Section 0:  Detail about ROUT_EX.kl
-------------------------------------------------------------------------
---- Elements of KAREL Language Covered:    In Section:
----    Actions:
----    Clauses:
----    Conditions:
----    Data types:
----    Built-in Functions & Procedures:
----            CHR                          Sec 3-A,B
----            FORCE_SPMENU                 Sec 3-A,B
----    Statements:
----            ROUTINE                      Sec 3-A,B
```

```
----            WRITE                         Sec 3-A,B
----
----    Reserve Words:
----            BEGIN                          Sec 3-A,B; 4
----            CR                             Sec 3-B
----            END                            Sec 3-A,B; 4
----            PROGRAM                        Sec 1
----    Predefined File Names:
----            CRTERROR                      Sec 3-A
----            CRTFUNC                       Sec 3-A
----            CRTPROMPT                     Sec 3-A
----            CRTSTATUS                     Sec 3-A
----            OUTPUT                        Sec 3-A
----            TPERROR                       Sec 3-B
----            TPFUNC                        Sec 3-B
----            TPSTATUS                      Sec 3-B
----            TPPROMPT                      Sec 3-B
```

Standard Routines - Declaration Section

```
-----------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-----------------------------------------------------------------------
PROGRAM ROUT_EX
%NOLOCKGROUP   ---- Don’t lock any motion groups
%COMMENT = ’MISC_ROUTINES’
-----------------------------------------------------------------------
----    Section 2:  Constant and Variable Declarations
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----    Section 3:  Routine Declarations
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----    Section 3-A:  CRT_CLS Declaration
----                  Clear the predefined windows:
----                  CRTPROMPT, CRTSTATUS, CRTFUNC, CRTERROR, OUTPUT
----                  Force Display of the CRT/KB USER SCREEN.
-----------------------------------------------------------------------
ROUTINE CRT_CLS
BEGIN  ---- CRT_CLS
--See Chapter 7.9.2 for more information on the PREDEFINED window names
  WRITE CRTERROR   (CHR(128),CHR(137))  -- Clear Window, Home Cursor
  WRITE CRTSTATUS  (CHR(128),CHR(137))  -- Clear Window, Home Cursor
  WRITE CRTPROMPT  (CHR(128),CHR(137))  -- Clear Window, Home Cursor
```

```
  WRITE CRTFUNC    (CHR(128),CHR(137))  -- Clear Window, Home Cursor
  WRITE OUTPUT     (CHR(128),CHR(137))  -- Clear Window, Home Cursor
  FORCE_SPMENU(CRT_PANEL,SPI_TPUSER,1) -- Force the CRT USER Menu
                                       -- to be visible last.  This will
                                       -- avoid the screen from flashing
                                       -- since the screen will be clean
                                       -- when you see it.
END CRT_CLS
```

Standard Routines - Clears Screen and Displays Menu

```
------------------------------------------------------------------------
----    Section 3-B:   TP_CLS Declaration
                       Clear the predefined windows:
----                   TPERROR, TPSTATUS, TPPROMPT, TPFUNC TPDISPLAY
----                   Force Display of the TP USER Menu SCREEN.
------------------------------------------------------------------------
ROUTINE TP_CLS
BEGIN
  WRITE (CHR(128),CHR(137)) -- By default this will clear TPDISPLAY
  WRITE TPERROR (CR,'                                        ',CR)
  WRITE TPSTATUS(CR,'                                        ',CR)
  WRITE TPPROMPT(CR,'                                        ',CR)
  WRITE TPFUNC  (CR,'                                        ',CR)
  FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the USER menu screen
     -- to be visible last.
     -- This will avoid the screen from
     -- flashing since the screen will
     -- be clean when you see it.
END TP_CLS
------------------------------------------------------------------------
----    Section 4:  Main Program
------------------------------------------------------------------------
BEGIN -- ROUT_EX
END  ROUT_EX
```

B.5 USING REGISTER BUILT-INS

This program demonstrates the use of the REGISTER builtins. REG_EX.KL retrieves the current position and stores it in PR[1]. Then it executes the program PROG_VAL.TP. PROG_VAL will modify the value within the Position Register PR[1].

After PROG_VAL is completed, REG_EX.KL retrieves the PR[1] position. The position is then manipulated and restored in PR[2], and an INTEGER number is stored in R[1]. A different teach

pendant program, PROG_1.TP, is executed which loops through some positions and stores a value to R[2]. The number of loops depends on the value of the R[1] (which was initially set by the KAREL program.)

After PROG_1.TP has completed, the KAREL program gets the value from R[2] and verifies it was the expected value.

The PROG_VAL.TP teach pendant program should look similar to the following.

```
PROG_VAL                                  JOINT 10%
1:   !POSITION REG VALUE ;
2:J P[1:ABOVE JOINT] 100% FINE ;
3:   PR[1,2]=600 ;
4:L PR[1] 100.0 Inch/mm FINE ;
5:J P[1:ABOVE JOINT]100% FINE ;
```

The PROG_VAL.TP teach pendant program does the following:

- Moves to position 1 in joint mode.
- Changes the ’y’ location of the position in Position Register 1, PR[1] (which was set by the KAREL program).
- Moves to the new PR[1] position.
- Finally moves back to position 1.

The PROG_1.TP teach pendant program should look similar to the following.

```
PROG_1                                    JOINT 10%
1:   LBL[1:START] ;
2:   IF R[1]=0, JMP LBL[2] ;
3:J P[1] 100% FINE ;
4:J P[2] 100% FINE ;
5:   R[1]=R[1] 1 ;
6:   JMP LBL[1] ;
7:   LBL[2:DONE] ;
8:   R[2]=1 ;
```

The PROG_1.TP teach pendant program does the following:

- Checks the value of the R[1].
- If the value of R[1] is not 0, then moves to J P[1] and J P[2] and decrements the value of R[1]. PROG_1.TP continues in this loop until the Register R[1] is zero.

Draft

- After the looping is complete, PROG_1.TP stores value 1 in R[2], which will be checked by the KAREL program.

Using Register Built-ins Program - Overview

```
---------------------------------------------------------------------------
----    REG_EX.Kl
---------------------------------------------------------------------------
---- Elements of KAREL Language Covered:         In Section:
----    Actions:
----    Clauses:
----    Conditions:
----    Data types:
----                BOOLEAN                        Sec 2
----                JOINTPOS                       Sec 2
----                REAL                           Sec 2
----                XYZWPR                         Sec 2
----    Directives:
----                ALPHABETIZE                    Sec 1
----                COMMENT                        Sec 1
----                NOLOCKGROUP                    Sec 1
----    Built-in Functions & Procedures:
----                CALL_PROGLIN                    Sec 4-A, 4-C
----                CHR                             Sec 4
----                CURPOS                          Sec 4-A
----                FORCE_SPMENU                    Sec 4
----                GET_POS_REG                     Sec 4-B
----                GET_JPOS_REG                    Sec 4-B
----                GET_REG                         Sec 4-C
----                POS_REG_TYP                     Sec 4-B
----                SET_JPOS_REG                    Sec 4-B
----                SET_INT_REG                     Sec 4-B
----                SET_POS_REG                     Sec 4-A
----    Statements:
----                WRITE                           Sec 4, 4-A,B,C
----                IF..THEN..ELSE..ENDIF           Sec 4-A,B,C
----                SELECT...CASE...ENDSELECT       Sec 4-B
----    Reserve Words:
----                BEGIN                           Sec 4
----                CONST                           Sec 2
----                CR                              Sec 4-A,B,C
----                END                             Sec 4-C
----                PROGRAM                         Sec 4
----                VAR                             Sec 2
```

Using Register Built-ins Program - Declaration Section

```
----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
----------------------------------------------------------------------------
PROGRAM reg_ex
%nolockgroup
%comment = 'Reg-Ops'
%alphabetize
----------------------------------------------------------------------------
----    Section 2:  Variable Declaration
----------------------------------------------------------------------------
CONST
    cc_success     = 0        -- Success status
    cc_xyzwpr      = 2     -- Position Register has an XYZWPR
    cc_jntpos      = 9      -- Position Register has a JOINTPOS
VAR
    xyz                 :XYZWPR
    jpos                :JOINTPOS
    r_val               :REAL
    prg_indx,
    i_val,
    pos_type,
    num_axes,
    status              :INTEGER
    r_flg               :BOOLEAN
---------------------------------------------------------------------------
----    Section 3: Routine Declaration
---------------------------------------------------------------------------
---------------------------------------------------------------------------
----    Section 4: Main program
---------------------------------------------------------------------------
BEGIN -- REG_EX
 write(chr(137),chr(128));            -- Clear the TP USER menu screen
 FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the TP USER menu to be
                                     -- visible
```

Using Register Built-ins - Storing and Manipulating Positions

```
-------------------------------------------------------------------------------
---- Section 4-A: Store current position in PR[1] and  execute PROG_VAL.TP
-------------------------------------------------------------------------------
 WRITE('Getting Current Position',cr)
 xyz = CURPOS(0,0)                    -- Get the current position
 WRITE('Storing Current position to PR[1]',cr)
 SET_POS_REG(1,xyz, status)           -- Store the position in PR[1]
```

```
 IF (status = cc_success) THEN        -- verify SET_POS_REG is successful
   WRITE('Executing "PROG_VAL.TP"',cr)
   CALL_PROGLIN('PROG_VAL',2,prg_indx, FALSE)
                                    --Execute 'PROG_VAL.TP' starting
                                   -- at line 2.  Do not pause on
                                   -- entry of PROG_VAL.
----------------------------------------------------------------------------
---- Section 4-B: Get new position from PR[1]. Manipulate and store in PR[2]
----------------------------------------------------------------------------
  WRITE('Getting Position back from PR[1]',cr)
    -- Decide what type of position is stored in Position Register 1, PR[1]
   POS_REG_TYPE(1, 1, pos_type, num_axes, status)
   IF (status = cc_success) THEN
   -- Get the position back from PR[1], using the correct builtin.
   -- This position was modified in PROG_VAL.TP
     SELECT pos_type OF
       CASE (cc_xyzwpr):
         xyz= GET_POS_REG(1, status)
       CASE (cc_jntpos):
         jpos = GET_JPOS_REG(1, status)
         xyz = jpos
       ELSE:
         write ('The position register set to invalid type', pos_type,CR)
         status = -1                  -- set status so do not continue.
     ENDSELECT
     IF (status = cc_success) THEN     -- Verify GET_POS_REG/GET_JPOS_REG is
                                   -- successful
       xyz.x = xyz.x+10                -- Manipulate the position.
       xyz.z = xyz.z-10
       jpos  = xyz                     -- Convert to a JOINTPOS
       WRITE('Setting New Position to PR[2]',cr)
       SET_JPOS_REG(2,jpos,status)     -- Set the JOINTPOS into PR[2]
       IF (status = cc_success) THEN   -- Verify SET_JPOS_REG is successful
          WRITE('Setting Integer Value to R[1]',cr)
          SET_INT_REG(1, 10, status)   -- Set the value 10 into R[1]
```

Using Register Built-ins - Executing Program and Checking Register

```
----------------------------------------------------------------------
----    Section 4-C: Execute PROG_1.TP and check the R[2]
----------------------------------------------------------------------
        IF (status=cc_success) THEN --Verify SET_INT_REG is successful
            WRITE('Executing "PROG_1.TP"',cr)
            CALL_PROGLIN('PROG_1',1, prg_indx, FALSE)
   --Execute PROG_1.TPstarting on first line.
   --Do not pause on entry of PROG_1.
```

```
            WRITE('Getting Value from R[2]',cr)
            GET_REG(2,r_flg, i_val, r_val, status) --Get R[2] value
            IF (status = cc_success) THEN      --Verify GET_REG success
              IF (r_flg) THEN                   --REAL value in register
              WRITE('Got REAL value from R[2]',cr)
                IF (r_val <> 1.0) THEN                  --Verify value set
                 WRITE ('PROG_1 failed to set R[2]',cr)-- by PROG_1_TP
                  WRITE ('PROG_1 failed to set R[2]',cr)
                ENDIF
              ELSE                  --Register contained an INTEGER
                WRITE('Got INTEGER value from R[2]',cr)
                IF (i_val <> 1) THEN     --Verify value set by
  WRITE ('PROG_1 failed to set R[2]',cr) --PROG_1.TP
                ENDIF
             ENDIF
           ELSE                      --GET_REG was NOT successful
             WRITE('GET_REG Failed',cr,' Status = ',status,cr)
           ENDIF
         ELSE                        --SET_INT_REG was NOT successful
           WRITE('SET_INT_REG Failed, Status = ',status,cr)
         ENDIF
       ELSE                          --SET_JPOS_REG was NOT successful
         WRITE('SET_JPOS_REG Failed, Status = ',status,cr)
       ENDIF
     ELSE                             -- GET_POS_REG was NOT Successful
       WRITE('GET_POS_REG Failed, Status = ',status,cr)
     ENDIF
   ELSE
     WRITE ('POS_REG_TYPE Failed, Status =', status, cr)
   ENDIF
 ELSE                                 -- SET_POS_REG was NOT successful
   WRITE('SET_POS_REG Failed, Status = ',status,cr)
 ENDIF
 IF (status = cc_success) THEN; WRITE ('Program Completed
Successfully',cr)
 ELSE ;                      WRITE ('Program Aborted due to error',cr)
 ENDIF
END reg_ex
```

B.6 PATH VARIABLES AND CONDITION HANDLERS PROGRAM

This program checks to determine if PATH variables are taught or not. If the paths are taught, the robot moves to a joint position and then loops along a path 5 times.

This example also sets up two global condition handlers.

- The first condition handler detects if the user has pushed a teach pendant key, and if so aborts the program.
- The second condition handler sets a variable when the program is aborted.

Path Variables and Condition Handlers Program - Overview

```
---------------------------------------------------------------------
----    PTH_MOVE.Kl
---------------------------------------------------------------------
---------------------------------------------------------------------
----    Section 0:  Detail about PTH_MOVE.kl
---------------------------------------------------------------------
---- Elements of KAREL Language Covered:       In Section:
----    Actions:
----            ABORT                                   Sec 4-A
----    Clauses:
----            WHEN                                    Sec 4-A
----            FROM                                    Sec 3-A
----    Conditions:
----            ABORT                                   Sec 4-A
----    Data types:
----            ARRAY OF REAL                           Sec 2
----            BOOLEAN                                 Sec 2
----            INTEGER                                 Sec 2
----            JOINTPOS6                               Sec 2
----            PATH                                    Sec 2
----            XYZWPR                                  Sec 2
----    Directives:
----            ALPHABETIZE                             Sec 1
----            COMMENT                                 Sec 1
----            ENVIRONMENT                             Sec 1
----    Built-in Functions & Procedures:
----            PATH_LEN                                Sec 4-C
----            CHR                                     Sec 3-B; 4-B,D
----            CNV_REL_JPOS                            Sec 4-D
----            SET_CURSOR                              Sec 4-B
----            SET_POS_REG                             4-D
```

Path Variables and Condition Handlers Program - Overview Continued

```
----    Statements:
----            Abort                                   Sec 4-C
----            CONDITION...ENDCONDITION                Sec 4-A
----            FOR...ENDFOR                            Sec 4-D
```

Draft

```
----            ROUTINE                               Sec 3-A, B, C
----            WAIT FOR                              Sec 3-B
----            WRITE                                 Sec 3-B; 4-B,C,D
----    Reserved Words:
----            BEGIN                                 Sec 3-A,B, 4
----            CONST                                 Sec 2
----            END                                   Sec 3-A,B: 4-D
----            VAR                                   Sec 2
----            PROGRAM                               Sec 1
----    Predefined File Names:
----            TPFUNC                                Sec 3-B; 4-D
----            TPDISPLAY                             Sec 4-B
```

Path Variables and Condition Handlers Program - Declaration Section

```
-------------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-------------------------------------------------------------------------------
PROGRAM PTH_MOVE         -- Define the program name
%ALPHABETIZE             -- Create the variables in alphabetical order
%COMMENT     = 'PATH MOVES'
%ENVIRONMENT PATHOP      -- Necessary for PATH_LEN
%ENVIRONMENT UIF         -- Necessary for SET_CURSOR
-------------------------------------------------------------------------------
----    Section 2:  Constant and Variable Declarations
-------------------------------------------------------------------------------
CONST
    CH_ABORT   = 1                          -- Number associated with the
                                            -- abort Condition handler
    CH_F1      = 2                          -- Number associated with the
                                            -- F1 key Condition handler
VAR
    status              :INTEGER            -- Status from built-in calls
    node_ind            :INTEGER            -- Index used when moving along path
    loop_pth            :INTEGER            -- Used in a FOR loop counter
    prg_abrt            :BOOLEAN            -- Set when program is aborted
    pth1                :PATH
    strt_jnt            :JOINTPOS6          -- Starting position of a move
    via_pos             :XYZWPR             -- Via point for a circular move
    des_pos             :XYZWPR             -- Destination point
    tmp_xyz             :XYZWPR             -- Temporary XYZWPR
    real_ary            :ARRAY[6] OF REAL   -- This is used for creating
                                            -- a joint position with 6 axes
    index               :INTEGER            -- FOR loop counter
```

Path Variables and Condition Handlers Program - Declare Routines

```
--------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
--------------------------------------------------------------------------------
----    Section 3-A:  TP_CLS Declaration
----     ROUTINE TP_CLS FROM ROUT_EX        -- ROUT_EX must also be loaded.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----    Section 3-B:  YES_NO Declaration
---                  Display choices on the function line of the TP.
----                  Asks for user response.
---                  F1 key is monitored by the Global condition handler
----                 [CH_F1] and the F2 is monitored here.
----                 If F1 is pressed the program will abort.
---                  But, if the F2 is pressed the program will continue.
--------------------------------------------------------------------------------
ROUTINE YES_NO
BEGIN
  WRITE TPFUNC (CHR(137))                  -- Home Cursor in Function window
  WRITE TPFUNC ('  ABORT  CONT')           -- Display Function key options
  WAIT FOR TPIN[131]                       -- Wait for user to respond to
                                           -- continue.  If the user presses
                                           -- F1 (abort) condition handler
                                           -- CH_ABORT will abort program.
  WRITE TPFUNC (CHR(137))                  -- Home Cursor in Function window
  WRITE TPFUNC ('  ABORT',chr(129))        -- Redisplay just Abort option and
                                           -- clear rest of Function window
END YES_NO
------------------------------------------------------------------------------
----   Section 3-C:  Routines move_to_pr and move_circ are TP
----   routines for doing moves
------------------------------------------------------------------------------
ROUTINE move_to_pr FROM move_to_pr      -- move_to_pr must also be loaded.
--   1:J PR[1] 100% FINE    ;
ROUTINE move_circ FROM move_circ        -- move_circ must also be loaded.
--   1:C PR[1]
--     :  PR[2] 100mm/sec FINE    ;
```

Path Variables and Condition Handlers Program - Declare Condition Handlers

```
-------------------------------------------------------------------------------
----    Section 4:  Main Program
-------------------------------------------------------------------------------
BEGIN   -- PTH_MOVE
-------------------------------------------------------------------------------
```

```
----    Section 4-A: Global Condition Handler Declaration
------------------------------------------------------------------------------
CONDITION[CH_ABORT]:
  WHEN ABORT DO       -- When the program is aborting set prg_abrt flag.
                      -- This will be triggered if this program aborts itself
                      --  or if an external mechanism aborts this program.
    prg_abrt = TRUE   -- You may then have another task which detects
                      -- prg_abrt being set, and does shutdown operations
                      -- (ie: set DOUT/GOUT's, send signals to a PLC)
ENDCONDITION
CONDITION[CH_F1]:
  WHEN TPIN[129] DO   -- Monitor TP 'F1' Key. If 'F1' key is pressed,
    ABORT             -- abort the program.
ENDCONDITION
prg_abrt = false                -- Initialize variable which is set only if
                                -- the program is aborted and CH_ABORT is
                                -- enabled.
ENABLE CONDITION[CH_ABORT]      -- Start scanning abort condition as defined.
ENABLE CONDITION[CH_F1]         -- Start scanning F1 key condition as defined.
------------------------------------------------------------------------------
----    Section 4-B: Display banner message and wait for users response
------------------------------------------------------------------------------
TP_CLS                                -- Routine Call; Clears the TP USER
                                      -- menu, and forces the TP USER menu
                                      -- to be visible.
SET_CURSOR(TPDISPLAY,2,13, status)   -- Set cursor position in TP USER menu
IF (status <> 0 ) THEN               -- Verify that SET_CURSOR was successful
   WRITE ('SET_CURSOR built-in failed with status = ',status,cr)
   YES_NO                            -- Ask whether to quit, due to error.
ENDIF
--- Write heading in REVERSE video, then turn reverse video off
WRITE (chr(139),' PLEASE READ ',chr(143),CR)
WRITE (cr,' *** F1 Key is labelled as ABORT key *** ')
WRITE (cr,' Any time the F1 key is pressed the program')
WRITE (cr,' will abort. However, the F2 key is active ')
WRITE (cr,' only when the function key is labeled.',cr,cr)
YES_NO  -- Wait for user response
```

Path Variables and Condition Handlers Program - Teach and Move Along Path

```
-----------------------------------------------------------------------------
----    Section 4-C: Verify PATH variable, pth1, has been taught
-----------------------------------------------------------------------------
-- Check the number of nodes in the path
IF PATH_LEN(pth1) = 0 THEN                    -- Path is empty (has no nodes)
   WRITE ('You need to teach the path.',cr)  -- Display instructions to user
```

```
      WRITE ('before executing this program.',cr)
      WRITE ('Teach the PATH variable pth1', CR, ' and restart the program',cr)
      ABORT                                      -- Simply ABORT the task
                                                 -- do not continue since there
   ENDIF                                         -- are no nodes to move to
   -----------------------------------------------------------------------------
   ----    Section 4-D: Creating a joint position and moving along paths
   -----------------------------------------------------------------------------
   FOR indx = 1 to 6 DO                       -- Set all joint angles to zero
       real_ary[indx] = 0.0
   ENDFOR
   real_ary[5] = 90.0                         -- Make sure that the position
                                              -- is not at a singularity point.
   CNV_REL_JPOS(real_ary, strt_jnt, status)  -- Convert real_ary values into
                                              -- a joint position, strt_jnt
   IF (status <> 0 ) THEN                     -- Converting joint position
                                              -- was NOT successful
      WRITE ('CNV_REL_JPOS built-in failed with status = ',status,cr)
      YES_NO                                  -- Ask user if want to continue.
   ELSE                                       -- Converting joint position was
                                              -- successful.
      -- The start position, strt_jnt, has been created and is located at
      -- axes 1-4 = 0.0, axes 5 = 90.0, axes 6 = 0.0.
      via_pos = strt_jnt                      -- Copy the strt_jnt to via_pos
      via_pos.x = via_pos.x +200              -- Add offset to the x location
      via_pos.y = via_pos.y +200              -- Add offset to the y location
      -- The via position, via_pos, has been created to be the same position
      -- as strt_jnt except it has been offset in the x and y locations by
      -- 200 mm.
      des_pos = strt_jnt                      -- Copy the strt_jnt to des_pos
      des_pos.x = des_pos.x + 400             -- Add offset to the x location
     -- The destination position, des_pos, has been created to be the same
     -- position as strt_jnt except it has been offset in the x location by
     -- 400 mm.
```

Draft

Path Variables and Condition Handlers Program - Move Along Path

```
      tmp_xyz =  strt_jnt             -- Convert start position to XYZWPR
      SET_POS_REG(1, tmp_xyz, status) -- Put start position in PR[1]
      move_to_pr                      -- Call TP program to move to PR[1]
      tmp_xyz =  des_pos              -- Convert destination position to XYZWPR
      SET_POS_REG(1, tmp_xyz, status) -- Put destination position in PR[1]
      tmp_xyz =  via_pos              -- Convert via position to XYZWPR
      SET_POS_REG(2, tmp_xyz, status) -- Put via position in PR[2]
      WRITE (cr,'Circular move to Destination Position',cr)
      move_circ                         -- Call TP program to
```

```
                                                   -- Move robot to destination
                                                   -- position using circular motion
                                                   -- via the via_pos
ENDIF
--- Execute the same path for 5 times.
FOR loop_pth = 1 TO 5 DO
   WRITE ('Moving along pth1 ',loop_pth::2, ' times',cr)
      -- Display the loop iteration
      -- NOTICE: that "loop_pth::2" will cause 2 blanks to be
      -- displayed after "pth1 '" and before loop_pth.
   FOR node_ind = 1 TO PATH_LEN(pth1) DO
     tmp_xyz =  pth1[node_ind]
     SET_POS_REG(1, tmp_xyz, status)
     move_to_pr
   ENDFOR
ENDFOR
WRITE TPFUNC    (CHR(128),CHR(137))  -- Home Cursor and Clear to
                                     -- End-of-line.  This will remove
                                     -- the ABORT displayed above F1.
WRITE ('pth_move Successfully Completed',cr)
END PTH_MOVE
```

Draft

B.7 LISTING FILES AND PROGRAMS AND MANIPULATING STRINGS

This program displays the list of files on the FLPY: device, and lists the programs loaded on the controller. It also shows basic STRING manipulating capabilities, using semi-colons(;) as a statement separator, and nesting IF statements.

Listing Files and Programs and Manipulating Strings - Overview

```
------------------------------------------------------------------------------
----    LIST_EX.Kl
------------------------------------------------------------------------------
----    Section 0:  Detail about LIST_EX.kl
------------------------------------------------------------------------------
---- Elements of KAREL Language Covered:    In Section:
----    Actions:
----    Clauses:
----            FROM                        Sec 3-B
----
----    Conditions:
----    Data types:
----            ARRAY OF STRING             Sec 2
----            BOOLEAN                     Sec 2, 3
```

```
----            INTEGER                         Sec 2, 3
----            STRING                          Sec 2
----    Directives:
----            %COMMENT                        Sec 1
----            %NOLOCKGROUP                    Sec 1
----    Built-in Functions & Procedures:
----            ABS                             Sec 4-A
----            ARRAY_LEN                       Sec 4-C&D
----            CNV_INT_STR                     Sec 4-A
----            FILE_LIST                       Sec 4-C
----            LOAD                            Sec 4-B
----            LOAD_STATUS                     Sec 4-B
----            PROG_LIST                       Sec 4-D
----            ROUND                           Sec 4-A
----            SUB_STR                         Sec 4-A
```

Listing Files and Programs and Manipulating Strings - Overview Continued

```
----    Statements:
----           FOR .... ENDFOR                  Sec 3-B
----           IF...THEN...ENDIF                Sec 4-A,B,C,D
----           ROUTINE                          Sec 3-A,B,C
----           REPEAT...UNTIL                   Sec 4-C,D
----           RETURN                           Sec 3-A
----           WRITE                            Sec 3-B; 4-A,B
----    Reserve Words:
----            BEGIN                           Sec 3-A,B; 4
----            CONST                           Sec 2
----            CR                              Sec 3-B; 4-A,B
----            END                             Sec 3-A,B, 4-B
----            PROGRAM                         Sec 1
----            VAR                             Sec 2
----
----    Operators:
----            MOD                             Sec 3-A
----            /                               Sec 3-A
----            *                               Sec 3-A
----    Devices Used:
----            FLPY:                           Sec 4-C
----    Basic Concepts:
----         Semi-colon(;) as statement separator
----         Nested IF..THEN..ELSE..IF..THEN..ELSE..ENDIF..ENDIF structure
----         Concatenation of STRINGS using '+'
----
```

Draft

Listing Files and Programs and Manipulating Strings - Declarations Section

```
-------------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-------------------------------------------------------------------------------
PROGRAM LIST_EX
%NOLOCKGROUP   ---- Don't lock any motion groups
%COMMENT = 'FILE_LIST'
-------------------------------------------------------------------------------
----    Section 2:  Constant and Variable Declarations
-------------------------------------------------------------------------------
CONST
      INCREMENT   = 13849
      MODULUS     = 65536
      MULTIPLIER  = 25173
VAR
      pr_cases       :STRING[6]  -- psuedo random number converted to string
      prg_nm         :STRING[50] -- Concatenated program name
      loaded         :BOOLEAN    -- Used to see if program is loaded
      initi          :BOOLEAN    -- Used to see if variables initialized
      indx1     :INTEGER   -- FOR loop index
      cases,                     -- Random number returned
      max_number,                -- Maximum random number
      seed      :INTEGER   -- Seed for generating a random number
      file_spec      :STRING[20] -- File specification for FILE_LIST
      n_files        :INTEGER    -- Number of files returned from FILE_LIST
      n_skip         :INTEGER    -- Number to skip for FILE_LIST & PROG_LIST
      format         :INTEGER    -- Format of returned names
                                 -- For FILE_LIST & PROG_LIST
      ary_nam        :ARRAY[9] OF STRING[20] -- Returned names
                                             -- from FILE_LIST & PROG_LIST
      prog_name      :STRING[10] -- Program names to list from PROG_LIST
      prog_type      :INTEGER    -- Program types to list form PROG_LIST
      n_progs        :INTEGER    -- Number of programs returned from PROG_LIST
      status    :INTEGER   -- Status of built-in procedure call
```

Listing Files and Programs and Manipulating Strings - Declare Routines

```
------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----    Section 3-A:  RANDOM Declaration
---                  Creates a pseudo-random number and returns the number.
------------------------------------------------------------------------------
ROUTINE random(seed : INTEGER) : REAL
```

Draft

```
BEGIN
  seed = (seed * MULTIPLIER + INCREMENT) MOD MODULUS
  RETURN(seed/65535.0)
END random
-----------------------------------------------------------------------------
----    Section 3-B:  DISPL_LIST  Declaration
----                  Display maxnum elements of ary_nam.
-----------------------------------------------------------------------------
ROUTINE displ_list(maxnum :INTEGER)
BEGIN
  FOR indx1 = 1 TO maxnum DO ;  WRITE (ary_nam[indx1],cr);   ENDFOR
  -- Notice the use of the semi-colon, which allows multiple statements
  -- on a line.
END displ_list
-----------------------------------------------------------------------------
----    Section 3-C:  TP_CLS  Declaration
----                 This routine is from ROUT_EX.KL and will
----                 clear the TP USER menu screen and force it to be visible.
-----------------------------------------------------------------------------
ROUTINE tp_cls FROM rout_ex
```

Draft

Listing Files and Programs and Manipulating Strings - Main Program

```
-----------------------------------------------------------------------------
----    Section 4:  Main Program
-----------------------------------------------------------------------------
BEGIN -- LIST_EX
tp_cls                -- Use routine from the rout_ex.kl file
-----------------------------------------------------------------------------
----  Section 4-A: Generate a pseudo random number, convert INTEGER to STRING
-----------------------------------------------------------------------------
max_number = 255 ;                          -- So the random number is 0..255
seed = 259 ;
  WRITE ('Manupulating String',cr)
  cases = ROUND(ABS((random(seed)*max_number)))-- Call random then take the
                                               -- absolute value of the number
                                               -- returned and round off the
                                               -- number.
  CNV_INT_STR(cases, 1, 0, pr_cases)           -- Convert cases to its
                                               -- ascii representation
  pr_cases =  SUB_STR(pr_cases, 2,3)           -- get at most 3 characters,
                                               -- starting at the second
                                               -- character, since first
                                               -- character is a blank.
```

Listing Files and Programs and Manipulating Strings - Create and Load Program

```
-------------------------------------------------------------------------------
----    Section 4-B: Build a program name from the number and try to load it
-------------------------------------------------------------------------------
  --- Build a random program name to show the manipulation of
  --- STRINGS and INTEGERs.
  prg_nm = 'MYPROG' + pr_cases + '.PC'     -- Concatenate the STRINGs together
                                           -- which create a program name
  --- Verify that the program is not already loaded
  WRITE ('Checking load status of ',prg_nm,cr)
  LOAD_STATUS(prg_nm, loaded, initi)
  IF (NOT loaded) THEN                      -- The program is not loaded
     WRITE ('Loading ',prg_nm,cr)
     LOAD(prg_nm, 0 , status)               -- Load in the program
     IF (status =  0 ) THEN                 -- Verify load is successful
        WRITE ('Loading ','MYPROG' + pr_cases + '.VR',cr)
        LOAD('MYPROG' + pr_cases + '.VR', 0, status) -- Load the .vr file
        IF (status <> 0 ) THEN              -- Loading variables failed
           WRITE ('Loading of ', 'MYPROG' + pr_cases + '.VR', ' failed',cr)
           WRITE ('Status = ',status);
        ENDIF
     ELSE                                   -- Load of program failed
        IF (status = 10003) THEN            -- File does not exist
           WRITE (prg_nm, ' file does not exist',cr)
        ELSE
           WRITE ('Loading of ',prg_nm, ' failed',cr,'Status = ',status);
        ENDIF
     ENDIF

 ELSE                                       -- The program is already loaded
     IF (NOT initi) THEN                    -- Variables not initialized
        WRITE ('Loading ','MYPROG' + pr_cases + '.VR',cr)
        LOAD('MYPROG' + pr_cases + '.VR', 0, status) -- Load in variables
        IF (status <> 0 ) THEN              -- Load of variables failed
           WRITE ('Loading of ', 'MYPROG' + pr_cases + '.VR', ' failed',cr)
           WRITE ('Status = ',status);
        ENDIF
     ENDIF
 ENDIF
```

Listing Files and Programs and Manipulating Strings - List Programs

```
-------------------------------------------------------------------------------
---- Section 4-C: Check the file listing of the drive FLPY: and display them
-------------------------------------------------------------------------------
```

Draft

```
  --- Display a directory listing of files on the Flpy:
  file_spec = 'FLPY:*.*'       -- All files in FLPY: drive
  n_skip = 0                   -- First time do not skip any files
  format = 3                   -- Return list in filename.filetype format
  WRITE ('Doing File list',cr)
  REPEAT                       -- UNTIL all files have been listed
    FILE_LIST(file_spec, n_skip, format, ary_nam, n_files, status)
    IF (status <>0 ) THEN      -- Error occurred
       WRITE ('FILE_LIST builtin failed with Status = ',status,cr)
    ELSE
       displ_list(n_files)     -- Write the names to the TP USER menu
       n_skip = n_skip + n_files  -- Skip the files we already got.
    ENDIF
  UNTIL (ARRAY_LEN(ary_nam) <> n_files) -- When n_files does not equal
                                        -- declared size of ary_name then
                                        --  all files have been listed.
------------------------------------------------------------------------------
----    Section 4-D: Show the programs loaded in controller
------------------------------------------------------------------------------
  --- Display the list of programs loaded on the controller
  prog_name = '*'             -- All program names should be listed
  prog_type = 6               -- Only PC type files should be listed
  n_skip = 0                  -- First time do not skip any file
  format = 2                  -- Return list in  filename.filetype format
  WRITE ('Doing Program list',cr)
  REPEAT                     -- UNTIL all programs have been listed
    PROG_LIST(prog_name, prog_type, n_skip, format, ary_nam, n_progs, status)
    -- The program names are stored in ary_nam
    -- n_progs is the number of program names stored in ary_nam
    IF (status <>0 ) THEN
       WRITE ('PROG_LIST builtin failed with Status = ',status,cr)
    ELSE
       displ_list(n_progs)              -- Display the current list
       n_skip = n_skip + n_progs        -- Skip the programs already listed
    ENDIF
  UNTIL (ARRAY_LEN(ary_nam) <> n_progs)  -- When n_files does not equal the
                                        -- declared size of ary_name then all
                                        --  programs have been listed.
END LIST_EX
```

B.8 GENERATING AND MOVING ALONG A HEXAGON PATH

This program generates a hexagonal path and moves along each side of the hexagon.

Draft

Generate and Move Along Hexagon Path - Overview

```
---------------------------------------------------------------------------
----    GEN_HEX.KL
---------------------------------------------------------------------------
----    Section 0:  Detail about GEN_HEX.KL
---------------------------------------------------------------------------
---- Elements of KAREL Language Covered: In Section:
----    Action:
----    Clauses:
----    Conditions:
----    Data types:
----            ARRAY OF REAL                Sec 3-C
----            ARRAY OF XYZWPR              Sec 2
----            INTEGER                      Sec 2; 3-B,C
----            JOINTPOS6                    Sec 2
----            REAL                         Sec 3-B
----    Directives:
               %COMMENT                     Sec 1
----    Built-in Functions & Procedures:
----            CHECK_EPOS                   Sec 4-B
----            CNV_REL_JPOS                 Sec 3-C
----            COS                          Sec 3-B
----            SIN                          Sec 3-B
----            SET_POS_REG                  4-B
----            SET_JPOS_REG                 Sec 3-C
----    Statements:
----            CONNECT TIMER                Sec 4-A
----            FOR ... ENDFOR               Sec 3-B,C; 4-B
----            ROUTINE                      Sec 3-A,B,C
----            WRITE                        Sec 4-A,B
----    Reserve Word:
----            BEGIN                        Sec 3-B,C; 4
----            CONST                        Sec 2
----            CR                           Sec 4-A
----            END                          Sec 3-B,C; 4-B
----            PROGRAM                      Sec 1
----            VAR                          Sec 2
```

Generate and Move Along Hexagon Path - Declaration Section

```
---------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
---------------------------------------------------------------------------
PROGRAM gen_hex
%COMMENT = 'HEXAGON'
```

Draft

```
--------------------------------------------------------------------------------
----    Section 2:  Constant and Variable Declaration
--------------------------------------------------------------------------------
CONST
  L_HEX_SIDE = 300            -- Length of one side of the hexagon
  NUM_AXES   = 6              -- Number of robot axes
VAR
  p_cntr        : JOINTPOS6   -- Center of the hexagon
  p_xyzwpr      : ARRAY[NUM_AXES] OF XYZWPR
                 -- Six vertices of the hexagon
  tmp_xyz       :XYZWPR
  clock,
  t_start,
  t_end,
  t_total       : INTEGER
  status,
  p_indx        : INTEGER
```

Generate and Move Along Hexagon Path - Declare Routines

```
--------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
--------------------------------------------------------------------------------
------------------------------------------------------------------------------
----   Section 3-A:  Routines move_to_pr and movl_to_pr are TP
----   routines for doing moves
------------------------------------------------------------------------------
ROUTINE move_to_pr FROM move_to_pr      -- move_to_pr must also be loaded.
--   1:J PR[1] 100% FINE    ;
ROUTINE movl_to_pr FROM movl_to_pr      -- movl_to_pr must also be loaded.
--   1:L PR[1] 1000mm/sec FINE    ;
--------------------------------------------------------------------------------
----    Section 3-B:  R_HEX_CENTER Declaration
----                  Calculates the hexagon points based on distance
----                  between point 1 and 4 of the hexagon.
--------------------------------------------------------------------------------
ROUTINE r_calc_hex
VAR
  p1_to_pcntr   : REAL -- Distance from the center of the hex to point 1
  vertice       : INTEGER   -- the index used specify each vertice of hexagon
BEGIN
  p1_to_pcntr = (L_HEX_SIDE / 2) + (L_HEX_SIDE * COS(60))
  p_xyzwpr[1] = p_cntr      -- p_cntr was calculated in r_hex_center
  p_xyzwpr[1].y = p_xyzwpr[1].y - p1_to_pcntr --set the first vertice of hex
  FOR vertice = 2 TO NUM_AXES DO    -- start at 2 since 1 is already set
    p_xyzwpr[vertice] = p_xyzwpr[1]   -- Intialize all vertices
```

Draft

```
  ENDFOR
  -- Calculating individual components for each vertice of the hexagon
  p_xyzwpr[2].x = p_xyzwpr[1].x + (L_HEX_SIDE * SIN(60))
  p_xyzwpr[2].y = p_xyzwpr[1].y + (L_HEX_SIDE * COS(60))
  p_xyzwpr[3].x = p_xyzwpr[1].x + (L_HEX_SIDE * SIN(60))
  p_xyzwpr[3].y = p_xyzwpr[1].y + (L_HEX_SIDE + (L_HEX_SIDE * COS(60)))

  p_xyzwpr[4].y = p_xyzwpr[1].y + (L_HEX_SIDE + (2 * (L_HEX_SIDE * COS(60)))
  p_xyzwpr[5].x = p_xyzwpr[1].x - (L_HEX_SIDE * SIN(60))
  p_xyzwpr[5].y = p_xyzwpr[3].y
  p_xyzwpr[6].x = p_xyzwpr[1].x - (L_HEX_SIDE * SIN(60))
  p_xyzwpr[6].y = p_xyzwpr[2].y
END r_calc_hex
```

Generate and Move Along Hexagon Path - Declare Routines

```
-----------------------------------------------------------------------------
----    Section 3-C:  R_HEX_CENTER Declaration
----                  Positions the face plate perpendicular
----                  to the xy world coordinate plane.
-----------------------------------------------------------------------------
ROUTINE r_hex_center
VAR
  status, indx   : INTEGER
  p_cntr_arry    : ARRAY[NUM_AXES] OF REAL
BEGIN
-- Initalize the center position array to zero
  FOR indx = 1 TO NUM_AXES DO
     p_cntr_arry[indx] = 0
  ENDFOR
-- Set JOINT 3 and 5 to -45 and 45 degrees
  p_cntr_arry[3] = -45
  p_cntr_arry[5] =  45  -- Convert the REAL array to a joint position,
                        -- p_cntr
  CNV_REL_JPOS(p_cntr_arry, p_cntr, status)
  SET_JPOS_REG(1, p_cntr, status) -- Put p_cntr in PR[1]
  move_to_pr    -- Call TP program to move to PR[1]

END r_hex_center
```

Generate and Move Along Hexagon Path - Main Program

```
-----------------------------------------------------------------------------
----    Section 4:  Main Program
-----------------------------------------------------------------------------
```

Draft

```
BEGIN  --- GEN_HEX
----------------------------------------------------------------------------
----    Section 4-A: Connect timer, set uframe, call routines
----------------------------------------------------------------------------
  clock = 0                    -- Initialize clock value to zero
  CONNECT TIMER TO clock       -- Connect the timer
  WRITE ('Moving to the center of the HEXAGON',CR) -- update user of process
  r_hex_center                 -- position the face plate of robot.
  WRITE ('Calculating the sides of HEXAGON',CR)  -- update user
  r_calc_hex                   -- Calculate the hexagon points
----------------------------------------------------------------------------
----    Section 4-B: Move on sides of hexagon
----------------------------------------------------------------------------
  WRITE ('Moving along the sides of the Hexagon',CR) -- Update user
  t_start = clock             -- Record the time before motion begins

  FOR p_indx = 1 TO 6 DO
   -- Verify that the position is reachable
   CHECK_EPOS ((p_xyzwpr[p_indx]), $UFRAME, $UTOOL, status)
   IF (status <> 0) THEN
    WRITE ('Unable to move to p_xyzwpr[', p_indx,']',CR);
   ELSE
    SET_POS_REG(1, p_xyzwpr[p_index], status)
    movl_to_pr             -- Call TP program to move to PR[1]
                           -- Move to each vertex of hexagon
   ENDIF
  ENDFOR
  SET_POS_REG(1, p_xyzwpr[1], status)
  movl_to_pr             -- Call TP program to move to PR[1]
                         -- Move back to first vertex of hexagon

  tmp_xyz = p_cntr
  SET_POS_REG(1, tmp_xyz, status)
  movl_to_pr               -- Move TCP in a straight
                           -- line to the center position
  t_end = clock            -- Record ending time
  WRITE('Total motion time = ',t_end-t_start,CR) --Display the total time for
                                                 -- motion.
                                                 -- NOTE that the total was
                                                 -- computed in the WRITE
                                                 -- statement.
END GEN_HEX
```

Draft

B.9 USING THE FILE AND DEVICE BUILT-INS

This program demonstrates how to use the File and Device built-ins. This program FORMATS and MOUNTS the RAM disk. Then copies files from the FLPY: device to RD:. If the RAM disk gets full the RAM disk size is increased and reformatted. This program continues until either all the files are copied successfully, or the built-in operations fail.

File and Device Built-ins Program - Overview

```
-------------------------------------------------------------------------------
----    FILE_EX.Kl
-------------------------------------------------------------------------------
----    Section 0:  Detail about FILE_EX.kl
-------------------------------------------------------------------------------
---- Elements of KAREL Language Covered:            In Section:
----    Actions:
----    Clauses:
----            FROM                                Sec 3
----    Conditions:
----    Data types:
----             BOOLEAN                              Sec 2
----            INTEGER                               Sec 2
----             STRING                               Sec 2
----    Directives:
----             COMMENT                              Sec 1
----            NOLOCKGROUP                           Sec 1
----    Built-in Functions & Procedures:
----             CNV_TIME_STR                         Sec 4-A
----             COPY_FILE                            Sec 4-B
----             DISMOUNT_DEV                         Sec 4-B
----             FORMAT_DEV                           Sec 4-B
----             GET_TIME                             Sec 4-A
----             MOUNT_DEV                            Sec 4-B
----             PURGE_DEV                            Sec 4-B
----             SUB_STR                              Sec 4-A
----    Statements:
----             IF...THEN...ELSE...ENDIF             Sec 4-B
----             REPEAT...UNTIL                       Sec 4-A
----             ROUTINE                              Sec 3
----             SELECT...ENDSELECT                   Sec 4-B
----             WRITE                                Sec 4-A,B
```

File and Device Built-ins Program - Overview Continued

```
----    Reserve Words:
----            BEGIN                               Sec 4
```

```
----            CONST                               Sec 2
----            CR                                  Sec 4-A,B
----            END                                 Sec 4-B
----            PROGRAM                             Sec 4
----            VAR                                 Sec 2
----    Devices Used:
----            FLPY                                Sec 4-B
----            MF3                                 Sec 4-B
----            RD                                  Sec 4-B
----            FR                                  Sec 4-B
```

File and Device Built-ins Program - Declaration Section, Declare Routines

```
-------------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-------------------------------------------------------------------------------
PROGRAM FILE_EX
%nolockgroup
%comment = 'COPY FILES'
-------------------------------------------------------------------------------
----    Section 2:  Variable Declaration
-------------------------------------------------------------------------------
CONST
    SUCCESS     = 0           -- Success status from builtins
    FINISHED    = TRUE         -- Finished Copy
    TRY_AGAIN   = FALSE        -- Try to copy again
    RD_FULL     = 85020        -- RAM disk full
    NOT_MOUNT   = 85005        -- Device not mounted
    FR_FULL     = 85001        -- FROM disk is full
    MNT_RD      = 85004        -- RAM disk must be mounted
  --Refer to
```

FANUC Robotics Controller KAREL

```
Setup and Operations Manual for an Error Code listing
VAR
    time_int    : INTEGER
    time_str    : STRING[30]
    status      : INTEGER
    cpy_stat    : BOOLEAN
    to_dev      : STRING[5]
-------------------------------------------------------------------------------
----    Section 3: Routine Declaration
-------------------------------------------------------------------------------
ROUTINE tp_cls FROM ROUT_EX
-------------------------------------------------------------------------------
----    Section 4: Main program
```

```
-------------------------------------------------------------------------------
BEGIN  -- FILE_EX
  tp_cls     -- from rout_ex.kl
-------------------------------------------------------------------------------
----    Section 4-A: Get Time and FORMAT ramdisk with date as volume name
-------------------------------------------------------------------------------
 GET_TIME(time_int)                                -- Get the system time
 CNV_TIME_STR(time_int, time_str)                  -- Convert the INTEGER time
                                                   -- to readable format
 WRITE ('Today is ', SUB_STR(time_str, 2,8),CR)    -- Display the date part
 WRITE ('Time is ', SUB_STR(time_str, 11,5),CR)    -- Display the time part
```

File and Device Built-ins Program - Mount and Copy to RAM Disk

```
------------------------------------------------------------------------------
----    Section 4-B: Mount RAMDISK and start copying from FLPY to
MF3:
------------------------------------------------------------------------------
to_dev = 'MF3:'
REPEAT                          -- Until all files have been copied
  cpy_stat = FINISHED
  WRITE('COPYing......',cr)
  -- Copy the files from FLPY: to to_dev and overwrite the file if it
  -- already exists.
  COPY_FILE('FLPY:*.kl', to_dev, TRUE, FALSE, status)
  SELECT (status) OF
    CASE (RD_FULL):            -- RAM disk is full
                               -- Dismount and re-size the RAM-DISK
      WRITE ('DISMOUNTing RD: ....',cr)
      DISMOUNT_DEV('RD:', status)
                               -- Verify DISMOUNT was successful or that
                               -- the device was not already mounted
      IF (status = SUCCESS) OR (status = NOT_MOUNT) THEN
           -- Increase the size of RD:
        WRITE('Increasing RD: size...',cr)
        $FILE_MAXSEC = ROUND($FILE_MAXSEC * 1.2)
                              -- Increase the RAM disk size
                              -- Format the RAM-DISK
        WRITE('FORMATTING RD:......',cr)
        FORMAT_DEV('RD:','' ,FALSE, status) -- Format the RAM disk
        IF (status <> SUCCESS) THEN
          WRITE ('FORMAT of RD: failed, status:', status,CR)
          WRITE ('Copy incomplete',cr)
        ELSE
          cpy_stat = TRY_AGAIN
        ENDIF
```

Draft

```
        WRITE('MOUNTing RD:......',cr)
        MOUNT_DEV ('RD:', status)
        IF (status <> SUCCESS) THEN
          WRITE ('MOUNTing of RD: failed, status:', status,CR)
          WRITE ('Copy incomplete',cr)
        ELSE
          cpy_stat = TRY_AGAIN
        ENDIF
      ELSE
        WRITE ('DISMOUNT of RD: failed, status:', status,cr)
        WRITE ('Copy incomplete',cr)
      ENDIF
```

File and Device Built-ins Program - Mount and Copy to RAM Disk Continued

```
CASE (FR_FULL):      -- FROM disk is full
      WRITE ('FROM disk is full',CR, 'PURGING FROM.....', CR)
      PURGE_DEV ('FR:', status)    -- Purge the FROM
      IF (status <> SUCCESS) THEN
        WRITE ('PURGE of FROM failed, status:', status, CR)
        WRITE ('Copy incomplete', CR)
      ELSE
        cpy_stat = TRY_AGAIN
      ENDIF
CASE (NOT_MOUNT, MNT_RD):    -- Device is not mounted
      WRITE ('MOUNTing ',to_dev,'.....',CR)
      MOUNT_DEV(to_dev, status)
      IF (status <> SUCCESS) THEN
        WRITE ('MOUNTing of ',to_dev,': failed, status:', status, CR)
        WRITE ('Copy incomplete', CR)
      ELSE
        cpy_stat = TRY_AGAIN
      ENDIF
CASE (SUCCESS):
      WRITE ('Copy completed successfully!',CR)
ELSE:
      WRITE ('Copy failed, status:', status,CR)
ENDSELECT
UNTIL (cpy_stat = FINISHED)
END file_ex
```

Draft

B.10 USING DYNAMIC DISPLAY BUILT-INS

This program demonstrates how to use the dynamic display built-ins. This program initiates the dynamic display of various data types. It then executes another task, CHG_DATA, which changes the values of these variables.

Before exiting this program the dynamic displays are cancelled and the other task is aborted. If DYN_DISP is aborted, it will set a variable which CHG_DATA detects. This ensures that CHG_DATA cannot continue executing once DYN_DISP is aborted.

Draft

Using Dynamic Display Built-ins - Overview

```
-----------------------------------------------------------------------------
----    DYN_DISP.Kl
-----------------------------------------------------------------------------
----    Section 0:  Detail about DYN_DISP.KL
-----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:         In Section:
----    Actions:
----    Clauses:
----                FROM                            Sec 3-C
----                IN CMOS                         Sec 2
----                WHEN                            Sec 4
----    Conditions:
----                ABORT                           Sec 4
----    Data types:
----                BOOLEAN                         Sec 2
----                INTEGER                         Sec 2
----                REAL                            Sec 2
----                STRING                          Sec 2
----    Directives:
----                ALPHABETIZE                     Sec 1
----                COMMENT                         Sec 1
----                NOLOCKGROUP                     Sec 1
```

Using Dynamic Display Built-ins - Overview Continued

```
----    Built-in Functions & Procedures:
----                ABORT_TASK                      Sec 4-C
----                CNC_DYN_DISI                    Sec 4-C
----                CNC_DYN_DISR                    Sec 4-C
----                CNC_DYN_DISB                    Sec 4-C
----                CNC_DYN_DISE                    Sec 4-C
----                CNC_DYN_DISP                    Sec 4-C
----                CNC_DYN_DISS                    Sec 4-C
----                INI_DYN_DISI                    Sec 3-A, 4-A
```

```
----                INI_DYN_DISR                    Sec 3-B, 4-A
----                INI_DYN_DISB                    Sec 3-C, 4-A
----                INI_DYN_DISE                    Sec 3-D, 4-A
----                INI_DYN_DISP                    Sec 3-E, 4-A
----                INI_DYN_DISS                    Sec 3-F, 4-A
----                LOAD_STATUS                     Sec 4-B
----                LOAD                            Sec 4-B
----                RUN_TASK                        Sec 4-B
----
----    Statements:
----                CONDITION...ENDCONDITION        Sec 4
----                IF...THEN...ENDIF               Sec 4-A,B,C
----                READ                            Sec 4-C
----                ROUTINE                         Sec 3-A,B,C
----                WRITE                           Sec 4-A,B,C
----
----    Reserve Words:
----                BEGIN                           Sec 3-A,B; 4
----                CR                              Sec 4-A
----                CONST                           Sec 2
----                END                             Sec 3-A,B; 4-C
----                PROGRAM                         Sec 4
----                VAR                             Sec 2
----    Predefined File Variables:
----               TPPROMPT                         Sec 4-B,C
----    Predefined Windows:
----               T_FU                             Sec 3-A,B
```

Using Dynamic Display Built-ins - Declaration Section

```
-----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-----------------------------------------------------------------------------
PROGRAM DYN_DISP
%nolockgroup
%comment = 'Dynamic Disp'
%alphabetize
%INCLUDE KLIOTYPS
-----------------------------------------------------------------------------
----    Section 2:  Variable Declaration
-----------------------------------------------------------------------------
CONST
  cc_success     = 0      -- Success status
  cc_clear_win   = 128    -- Clear window
  cc_clear_eol   = 129    -- Clear to end of line
  cc_clear_eow   = 130    -- Clear to end of window
```

```
  CH_ABORT       = 1      -- Condition Handler to detect when program aborts
VAR
  Int_wind           :STRING[10]
  Rel_wind           :STRING[10]
  Field_Width        :INTEGER
  Attr_Mask          :INTEGER
  Char_Size          :INTEGER
  Row                :INTEGER
  Col                :INTEGER
  Interval           :INTEGER
  Buffer_Size        :INTEGER
  Format             :STRING[7]
  bool_names         :ARRAY[2] OF STRING[10]
  enum_names         :ARRAY[4] OF STRING[10]
  pval_names         :ARRAY[2] OF STRING[10]
  bool1 IN CMOS      :BOOLEAN
  enum1 IN CMOS      :INTEGER
  port_type          :INTEGER
  port_no            :INTEGER
  Str1  IN CMOS      :STRING[10]
  Int1  IN CMOS      :INTEGER    -- Using IN CMOS will create the variables
  Real1 IN CMOS      :REAL       -- in CMOS RAM, which is permanent memory.
  status             :INTEGER
  loaded,
  initialized        :BOOLEAN
  dynd_abrt          :BOOLEAN  -- Set to true when program aborts.
```

Using Dynamic Display Built-ins - Declare Routines

```
-----------------------------------------------------------------------------
----    Section 3: Routine Declaration
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----    Section 3-A: SET_INT Declaration
----                  Set all the input parameters for the INI_DYN_DISI call.
-----------------------------------------------------------------------------
ROUTINE Set_Int
Begin
  -- Valid predefined windows are described in
  -- Chapter 7.9.1, "USER MENU on the Teach Pendant
  --    Error  Line     --> 'ERR'   1  line
  --    Status Line     --> 'T_ST'  3  lines
  --    Display Window  --> 'T_FU'  10 lines
  --    Prompt Line     --> 'T_PR'  1  line
  --    Function Key    --> 'T_FK'  1  line
  Int_Wind          = 'T_FU'            -- Use the predefined display window
```

```
  Field_Width       = 0                    -- Use the minimum width necessary
  Attr_Mask         = 1 OR 4               -- BOLD and UNDERLINED
  Char_Size         = 0                    -- Normal
  Row               = 1                    -- Specify the location within 'T_FU'
  Col               = 16                   --    to dynamically displayed
  Interval          = 250                   -- 250ms between updates
  Buffer_Size       = 10                   -- Minimum value required.
  Format            = '%-8d'               -- 8 character minimum field width
--- With this specification the INTEGER will be displayed as follows:
---               --------
---              |xxxxxxxx|
---               --------
---      Where the integer value  will be left justified.
---      The x's will be the integer value unless the integer value is
---      less then 8 characters, then the right side will be blanks up to
---      a total 8 characters.  If the integer value is greater than the 8
---      characters the width is dynamically increased to display the whole
---      integer value.  The INTEGER value will also be bold and underlined.
End Set_Int
```

Using Dynamic Display Built-ins - Declare Routines Continued

```
-----------------------------------------------------------------------------
----    Section 3-B: SET_REAL Declaration
----                 Set all the input parameters for the INI_DYN_DISR call.
-----------------------------------------------------------------------------
ROUTINE Set_Real
Begin
  Rel_Wind          = 'T_FU'               -- Use the predefined display window
  Field_Width       = 10                   -- Maximum width of display.
  Attr_Mask         = 2 OR 8               -- blinking and reverse video
  Char_Size         = 0                    -- Normal
  Row               = 2                    -- Specify the location within 'TFU'
  Col               = 16                   --    to dynamically display
  Interval          = 200                  -- 200ms between update
  Buffer_Size       = 10                   -- Minimum value required.
  Format            = '%2.2f'
--- With the format and field_width specification the REAL will be
--- displayed as follows:
---               ----------
---              |xxxx.xx   |
---               ----------
---      Where the real value  will be left justified.
---      There will always be two digits after the decimal point.
---      A maximum width of 10 will be used.
---      If the real value is less then 10 characters the right side will be
```

Draft

```
---      padded with blanks up to 10 character width.
---      If the real value exceeds 10 characters, the display width will not
---      expand but will display a ">" as the last character, indicating the
---      entire value is not displayed.
---      The value will also be blinking and in reverse video.
End Set_Real
```

Using Dynamic Display Built-ins - Declare Routines Continued

```
-----------------------------------------------------------------------------
----    Section 3-C: SET_BOOL Declaration
----                  Set all the input parameters for the INI_DYN_DISB call
-----------------------------------------------------------------------------
ROUTINE Set_Bool
Begin
  -- Valid predefined windows are described in
  -- Chapter 7.9.1, "USER MENU on the Teach Pendant
  --    Error  Line     --> 'ERR'   1  line
  --    Status Line     --> 'T_ST'  3  lines
  --    Display Window  --> 'T_FU'  10 lines
  --    Prompt Line     --> 'T_PR'  1  line
  --    Function Key    --> 'T_FK'  1  line
  Int_Wind            = 'T_FU'          -- Use the predefined display window
  Field_Width        = 10               -- Display 10 chars
  Attr_Mask          = 2                -- Blinking
  Char_Size          = 0                -- Normal
  Row                = 3                -- Specify the location within 'T_FU'
  Col                = 16               --    to dynamically displayed
  Interval           = 250              -- 250ms between updates
  Buffer_Size        = 10               -- Minimum value required
  bool_names[1]      = 'YES'            -- string display in bool_var is FALSE
  bool_names[2]      = 'NO'             -- string display in bool_var is TRUE
--- With this specification the BOOLEAN will be displayed as follows:
---               --------
---              |xxxxxxxx|
---               --------
---      Where the boolean value will be left justified.
---      The x's will be one of the strings 'YES' or 'NO', depending on
---      the value of bool1.  The string will be blinking.
End Set_Bool
```

Using Dynamic Display Built-ins - Declare Routines Continued

```
-----------------------------------------------------------------------------
----    Section 3-D: SET_ENUM Declaration
```

```
----                  Set all the input parameters for the INI_DYN_DISE call.
-----------------------------------------------------------------------------
ROUTINE Set_Enum
Begin
  -- Valid predefined windows are described in
  -- Chapter 7.9.1, "USER MENU on the Teach Pendant
  --    Error  Line     --> 'ERR'   1  line
  --    Status Line     --> 'T_ST'  3  lines
  --    Display Window  --> 'T_FU'  10 lines
  --    Prompt Line     --> 'T_PR'  1  line
  --    Function Key    --> 'T_FK'  1  line
  Int_Wind         = 'T_FU'         -- Use thE predefined display window
  Attr_Mask        = 8               -- REVERSED
  Field_Width      = 10              -- Display to characters
  Char_Size        = 0               -- Normal
  Row              = 4               -- Specify the location within 'T_FU'
  Col              = 16              --    to dynamically displayed
  Interval         = 250             -- 250ms between updates
  Buffer_Size      = 10              -- Minimum value required
  enum_names[1]    = 'Enum-0'        -- value displayed if enum_var = 0
  enum_names[2]    = 'Enum-1'        -- value displayed if enum_var = 1
  enum_names[3]    = 'Enum-2'        -- value displayed if enum_var = 2
  enum_names[4]    = 'Enum-3'        -- value displayed if enum_var = 3
--- With this specification enum_var will be displayed as follows:
---               --------
---              |xxxxxxxx|
---               --------
---      Where one of the strings enum_names will be displayed,
---      depending on the integer value enum1.  If enum1 is outside
---      the range 0-3, a string of 10 '?'s will be displayed.
---      The string will be displayed in reversed video.
End Set_Enum
```

Using Dynamic Display Built-ins - Declare Routines Continued

```
-----------------------------------------------------------------------------
----    Section 3-E: SET_PORT Declaration
----                  Set all the input parameters for the INI_DYN_DISP call.
-----------------------------------------------------------------------------
ROUTINE Set_Port
Begin
  -- Valid predefined windows are described in
  -- Chapter 7.9.1, "USER MENU on the Teach Pendant
  --    Error  Line     --> 'ERR'   1  line
  --    Status Line     --> 'T_ST'  3  lines
  --    Display Window  --> 'T_FU'  10 lines
```

```
  --    Prompt Line     --> 'T_PR'  1  line
  --    Function Key    --> 'T_FK'  1  line
  Int_Wind            = 'T_FU'          -- Use the predefined display window
  Field_Width         = 10              -- Display to characters
  Attr_Mask           = 1               -- BOLD
  Char_Size           = 0               -- Normal
  Row                 = 5               -- Specify the location within 'T_FU'
  Col                 = 16              --    to dynamically displayed
  Interval            = 250             -- 250ms between updates
  Buffer_Size         = 10              -- Minimum value required.
  pval_names[1]       = 'RELEASED'      -- text displayed if key is not pressed
  pval_names[2]       = 'PRESSED'       -- text displayed if key is pressed
  port_type           = io_tpin         -- port type = TP key
  port_no             = 175             -- user-key 3
--- With this specification PRESSED or RELEASED will be displayed as follows:
---               --------
---              |xxxxxxxx|
---               --------
---      Where the string will be left justified.
---      The x's will be either 'RELEASED' or 'PRESSED'.
---      The string will also be normal video.
---      (Bold is not supported on the teach pendant.)
End Set_Port
```

Using Dynamic Display Built-ins - Declare Routines Continued

```
-----------------------------------------------------------------------------
----    Section 3-F: SET_STR Declaration
----                  Set all the input parameters for the INI_DYN_DISS call.
-----------------------------------------------------------------------------
ROUTINE Set_Str
Begin
  -- Valid predefined windows are described in
  -- Chapter 7.9.1, "USER MENU on the Teach Pendant
  --    Error  Line     --> 'ERR'   1  line
  --    Status Line     --> 'T_ST'  3  lines
  --    Display Window  --> 'T_FU'  10 lines
  --    Prompt Line     --> 'T_PR'  1  line
  --    Function Key    --> 'T_FK'  1  line
  Int_Wind              = 'T_FU'           -- Use th predefined display window
  Field_Width         = 10                 -- Use the minimum width neccessary
  Attr_Mask           = 1 OR 4             -- BOLD and UNDERLINED
  Char_Size           = 0                  -- Normal
  Row                 = 6                  -- Specify the location within 'T_FU'
  Col                 = 16                 --    to dynamically displayed
  Interval            = 250                -- 250ms between updates
```

```
  Buffer_Size       = 10                -- Minimum value required.
  Format            = '%10s'            -- 10 character minimum field width
--- With this specification the STRING will be displayed as follows:
---               --------
---              |xxxxxxxx|
---               --------
---      Where the string value will be left justified.
---      The x's will be the string value.
---      The STRING will also be underlined.
End Set_Str
-----------------------------------------------------------------------------
----    Section 3-G: TP_CLS Declaration
----               Clear the TP USER menus screen and force it to be visible.
-----------------------------------------------------------------------------
ROUTINE tp_cls FROM rout_ex
```

Using Dynamic Display Built-ins - Initiate Dynamic Displays

```
-----------------------------------------------------------------------------
----    Section 4: Main program
-----------------------------------------------------------------------------
BEGIN --- DYN_DISP
dynd_abrt = FALSE
CONDITION[CH_ABORT]:
  WHEN ABORT DO      -- When the program is aborting set dynd_abrt flag.
                     -- This will be triggered if this program aborts itself
                     --  or if an external mechanism aborts this program.
    dynd_abrt = TRUE -- CHG_DATA will detect this and complete execution.
ENDCONDITION
ENABLE CONDITION [CH_ABORT]
-----------------------------------------------------------------------------
----    Section 4-A: Setup variables, initiate dynamic display
-----------------------------------------------------------------------------
  TP_CLS                               -- Clear the TP USER screen
                                       -- Force display of the TP USER screen
  STATUS = cc_success                  -- Initialize the status variable
  -- Initialize the dynamically displayed variables
  Int1       = 1
  Real1      = 1.0
  Bool1      = FALSE
  Enum1      = 0
  Strl       = ''
  -- Display messages to the TP USER screen
  WRITE ('Current INT1 =',CR)
  WRITE ('Current REAL1=',CR)
  WRITE ('Current BOOL1=',CR)
```

```
  WRITE ('Current ENUM1=',CR)
  WRITE ('Current PORT =',CR)
  WRITE ('Current STR1 =',CR)
  Set_Int                -- Set parameter values for INTEGER DYNAMIC DISPLAY
  INI_DYN_DISI(Int1,Int_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, Buffer_Size, Format ,Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISI failed, Status=',status,CR)
  ENDIF
 Set_Bool                -- Set parameter values for BOOLEAN DYNAMIC
DISPLAY
  INI_DYN_DISB(Bool1,Int_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, bool_names,Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISB failed, Status=',status,CR)
  ENDIF
```

Using Dynamic Display Built-ins - Initiate Dynamic Displays

```
Set_Enum                   -- Set parameter values for Enumerated Integer
                           --    DYNAMIC DISPLAY
  INI_DYN_DISE(Enum1,Int_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, enum_names,Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISE failed, Status=',status,CR)
  ENDIF
  Set_Port               -- Set parameter values for Port DYNAMIC DISPLAY
  INI_DYN_DISP(port_type, port_no ,Int_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, pval_names, Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISP failed, Status=',status,CR)
  ENDIF
  Set_Real                  -- Set parameter values for REAL DYNAMIC DISPLAY
  INI_DYN_DISR(Real1,Rel_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, Buffer_Size, Format ,Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISR failed, Status=',status,CR)
  ENDIF
 Set_Str                 -- Set parameter values for STRING DYNAMIC
DISPLAY
  INI_DYN_DISS(Str1,Int_Wind,Field_Width,Attr_Mask,Char_Size,
               Row,Col, Interval, Buffer_Size, Format ,Status)
  IF Status <> cc_success THEN -- Check the status
     WRITE(' INI_DYN_DISS failed, Status=',status,CR)
  ENDIF
```

Draft

Using Dynamic Display Built-ins - Execute Subordinate Task

```
-----------------------------------------------------------------------------
----    Section 4-B: Check on subordinate program and execute it.
-----------------------------------------------------------------------------
  -- Check the status of the other program which will change the value
  -- of the variables.
  LOAD_STATUS('chg_data', loaded, initialized)
  IF (loaded = FALSE ) THEN
    WRITE TPPROMPT(CHR(cc_clear_win))            -- Clear the prompt line
    WRITE TPPROMPT('CHG_DATA is not loaded. Loading now...')
    LOAD('chg_data.pc',0,status)
    IF (status =  cc_success) THEN               -- Check the status
      RUN_TASK('CHG_DATA',1,false,false,1,status)
      IF (Status <> cc_success) THEN               -- Check the status
        WRITE ('Changing the value of the variables',CR)
        WRITE ('by another program failed',CR)
        WRITE ('BUT you can try changing the values',CR)
        WRITE ('from KCL',CR)
      ENDIF
    ELSE
      WRITE ('LOAD Failed, status = ',status,CR)
    ENDIF
  ELSE
    RUN_TASK('CHG_DATA',1,false,false,1,status)
    IF (Status <> cc_success) THEN                 -- Check the status
      WRITE ('Changing the value of the variables',CR)
      WRITE ('by another program failed',CR)
      WRITE ('BUT you can try changing the values',CR)
      WRITE ('from KCL',CR)
    ENDIF
  ENDIF
```

Using Dynamic Display Built-ins - User Response Cancels Dynamic Displays

```
-----------------------------------------------------------------------------
----    Section 4-C: Wait for user response, and cancel dynamic displays
-----------------------------------------------------------------------------
  WRITE TPPROMPT(CHR(cc_clear_win))        -- Clear the prompt line
  WRITE TPPROMPT('Enter a number to cancel DYNAMIC display: ')
  READ (CR)                                -- Read only one character
                                           -- See Chapter 7.7.1,
                                           -- "Formatting INTEGER Data Items"
  ABORT_TASK('CHG_DATA',TRUE, TRUE,status)   -- Abort CHG_DATA
  IF (status <> cc_success) THEN             -- Check the status
     WRITE(' ABORT_TASK failed, Status=',status,CR)
```

Draft

```
  ENDIF
  CNC_DYN_DISI(Int1, Int_Wind,Status)       -- Cancel display of Intl
  IF Status <> 0 THEN                       -- Check the status
     WRITE(' CND_DYN_DISI failed, Status=',status,CR)
  ENDIF
  CNC_DYN_DISR(Real1,Rel_Wind,Status)       -- Cancel display of Real1
  IF Status <> 0 THEN                       -- Check the status
     WRITE(' CND_DYN_DISR failed, Status=',status,CR)
  ENDIF
  CNC_DYN_DISB(Bool1, Int_Wind,Status)      -- Cancel display of Bool1
  IF Status <> 0 THEN                       -- Check the status
     WRITE(' CND_DYN_DISB failed, Status=',status,CR)
  ENDIF
  CNC_DYN_DISE(Enum1, Int_Wind,Status)     -- Cancel display of Enum1
  IF Status <> 0 THEN                      -- Check the status
     WRITE(' CND_DYN_DISE failed, Status=',status,CR)
  ENDIF
  CNC_DYN_DISP(port_type, Port_no, Int_Wind,Status) -- Cancel display of Port
  IF Status <> 0 THEN                       -- Check the status
     WRITE(' CND_DYN_DISP failed, Status=',status,CR)
  ENDIF
  CNC_DYN_DISS(Str1, Int_Wind,Status)        -- Cancel display of String
  IF Status <> 0 THEN                        -- Check the status
     WRITE(' CND_DYN_DISS failed, Status=',status,CR)
  ENDIF
END DYN_DISP
```

B.11 MANIPULATING VALUES OF DYNAMICALLY DISPLAYED VARIABLES

The CHG_DATA.KL program is called by DYN_DISP, where the actual variables are displayed dynamically. This program does some processing and changes the value of those variables.

Manipulate Dynamically Displayed Variables - Overview

```
----------------------------------------------------------------------------
----    CHG_DATA.KL
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----    Section 0:  Detail about CHG_DATA.KL
----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:          In Section:
----    Actions:
----    Clauses:
----                FROM                             Sec 2
```

```
----    Conditions:
----    Data types:
----                INTEGER                             Sec 2
----                REAL                                Sec 2
----
----    Directives:
----    Built-in Functions & Procedures:
----    Statements:
----                DELAY                               Sec 4
----                FOR....ENDFOR                       Sec 4
----                REPEAT...UNTIL                      Sec 4
----
----    Reserve Words:
----                BEGIN                               Sec 4
----                END                                 Sec 4
----                PROGRAM                             Sec 1
----                VAR                                 Sec 2
----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
----------------------------------------------------------------------------
PROGRAM CHG_DATA
%nolockgroup
%comment = 'Dynamic Disp2'
```

Draft

Manipulate Dynamically Displayed Variables - Declaration Section

```
-----------------------------------------------------------------------------
----    Section 2:  Variable Declaration
-----------------------------------------------------------------------------
VAR
 -- IF the following variables did NOT have IN CMOS, the following errors
 -- would be posted when loading this program:
 --   VARS-012 Create var -INT1 failed  VARS-038 Cannot change CMOS/DRAM type
 --   VARS-012 Create var -REAL1 failed VARS-038 Cannot change CMOS/DRAM type
 -- This indicates that there is a discrepancy between DYN_DISP and CHG_DATA.
 -- One program has specified to create the variables in DRAM the
 -- other specified CMOS.
  Int1 IN CMOS FROM dyn_disp   :INTEGER    -- dynamically displayed variable
  Real1 IN CMOS FROM dyn_disp  :REAL       -- dynamically displayed variable
  Bool1 IN CMOS FROM dyn_disp  :BOOLEAN    -- dynamically displayed variable
  Enum1 IN CMOS FROM dyn_disp  :INTEGER    -- dynamically displayed variable
  Str1 IN CMOS FROM dyn_disp   :STRING[10] -- dynamically displayed variable
  indx                          :INTEGER
  dynd_abrt FROM dyn_disp       :BOOLEAN   -- Set in dyn_disp when dyn_disp
                                           -- is aborting
-----------------------------------------------------------------------------
```

```
----    Section 3: Routine Declaration
-----------------------------------------------------------------------------
```

Manipulate Dynamically Displayed Variables - Main Program

```
-----------------------------------------------------------------------------
----    Section 4: Main program
-----------------------------------------------------------------------------
BEGIN  -- CHG_DATA

 -- This demonstrates that the variables are changed from this task, CHG_DATA.
 -- The dynamic display initiated in task DYN_DISP, will continue
 -- to correctly display the updated values of these variables.
 -- Do real application processing.
 -- Simulated here in a FOR loop.
REPEAT
  FOR indx  = -9999 to 9999 DO
    int1  = (indx DIV 2) * 7
    real1 = (indx DIV 3)* 3.13
    bool1 = ((indx AND 4) = 0)
    enum1 = (ABS(indx) DIV 5) MOD 5
    Str1 = SUB_STR('123456789A', 1, (ABS(indx) DIV 6) MOD 7 + 1)
    delay 200   -- Delay for 1/5 of a second as if processing is going on.
  ENDFOR
UNTIL (DYND_ABRT)   -- This task is aborted from DYN_DISP.  However, if
                    -- DYN_DISP aborts abnormally (ie from a KCL> ABORT), it
                    -- will set DYND_ABRT, which will allow CHG_DATA to
                    -- complete execution.
END CHG_DATA
```

B.12 DISPLAYING A LIST FROM A DICTIONARY FILE

This program controls the display of a list which is read in from the dictionary file DCLISTEG.UTX. For more information on DCLISTEG.UTX refer to Section B.12.1 . DCLST_EX.KL controls the placement of the cursor along with the action taken for each command.

Note The use of DISCTRL_FORM is the preferred method of displaying information. DISCTRL_FORM will automatically take care of all the key inputs and is much easier to use. For more information, refer to Chapter 10 *DICTIONARIES AND FORMS* .

Display List from Dictionary File - Overview

```
----------------------------------------------------------------------------
----   DCLST_EX.KL
----------------------------------------------------------------------------
```

```
----    Section 0:  Detail about DCLST_EX.KL
----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:      In Section:
----    Actions:
----    Clauses:
----            FROM                                 Sec 3-E
----    Conditions:
----    Data types:
----            ARRAY OF STRING                       Sec 2
----            BOOLEAN                               Sec 2
----            DISP_DAT_T                            Sec 2
----            FILE                                  Sec 2
----            INTEGER                               Sec 2
----            STRING                                Sec 2
----    Directives:
----            ALPHABETIZE                           Sec 1
----            COMMENT                               Sec 1
----            INCLUDE                               Sec 1
----            NOLOCKGROUP                           Sec 1
----    Built-in Functions & Procedures:
----            ADD_DICT                              Sec 3-B
----            ACT_SCREEN                            Sec 4-I
----            ATT_WINDOW_S                          Sec 4-C
----            CHECK_DICT                            Sec 3-B
----            CLR_IO_STAT                           Sec 3-A,C
----            CNV_STR_INT                           Sec 4-E
----            DEF_SCREEN                            Sec 4-B
----            DET_WINDOW                            Sec 4-I
----            DISCTRL_LIST                          Sec 4-G,H
----            FORCE_SPMENU                          Sec 4-A,B,C
----            IO_STATUS                             Sec 3-A,
----            ORD                                   Sec 4-H
----            READ_DICT                             Sec 4-E,F
```

Display List from Dictionary File - Overview Continued

```
----            REMOVE_DICT                           Sec 4-I
----            SET_FILE_ATR                          Sec 4-G
----            STR_LEN                               Sec 4-F
----            UNINIT                                Sec 3-C
----            WRITE_DICT                            Sec 4-D,H,I
----    Statements:
----            ABORT                                 Sec 3-B
----            CLOSE FILE                            Sec 4-I
----            FOR...ENDFOR                          Sec 4-F
----            IF...THEN...ENDIF                     Sec 3-A,B,C,D; 4-F,H,I
```

Draft

```
----            OPEN FILE                           Sec 3-A; 4-H
----            READ                                Sec 4-A,B,H
----            REPEAT...UNTIL                      Sec 4-H
----            ROUTINE                             Sec 3-A,B,C,D,E
----            SELECT...ENDSELECT                  Sec 4-H
----            WRITE                               Sec 3-A,B,C,D;4-A,I
----    Reserve Words:
----            BEGIN                               Sec 3-A,B,C,D;4
----            CR                                  Sec 4-A,B,C
----            END                                 Sec 3-A,B,C,D; 4-I
----            PROGRAM                             Sec 1
----            VAR                                 Sec 2
----    Predefined File Names:
----            TPDISPLAY                           Sec 4-D,G,H,I
----            TPFUNC                              Sec 4-D,H
----            TPPROMPT                            Sec 4-D,H,I
----            TPSTATUS                            Sec 4-D,I
----    Devices Used:
----            RD2U                                Sec 3-B
----    Predefined Windows:
----            ERR                                 Sec 4-C
----            T_ST                                Sec 4-C
----            T_FU                                Sec 4-C
----            T_PR                                Sec 4-C
----            T_FR                                Sec 4-C
```

Display List from Dictionary File - Declaration Section

```
-----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-----------------------------------------------------------------------------
PROGRAM DCLST_EX
%COMMENT='DISCTRL_LIST '
%ALPHABETIZE
%NOLOCKGROUP
%INCLUDE DCLIST -- the include file from the dictionary DCLISTEG.UTX
-----------------------------------------------------------------------------
----    Section 2:   Variable  Declarations
-----------------------------------------------------------------------------
VAR
  exit_Cmnd       : INTEGER
  act_pending     : INTEGER             -- decide if any action is pending
  display_data    : DISP_DAT_T          -- information needed for DICTRL_LIST
  done            : BOOLEAN             -- decides when to complete execution
  Kb_file         : FILE                -- file opened to the TPKB
  i               : INTEGER             -- just a counter
```

```
  key             : INTEGER               -- which key was pressed
  last_line       : INTEGER               -- number of last line of information
  list_data       : ARRAY[20] OF STRING[40] -- exact string information
  num_options     : INTEGER               -- number of items in list
  old_screen      : STRING[4]             -- previously attached screen
  status          : INTEGER               -- status returned from built-in call
  str             : STRING[1]             -- string read in from teach pendant
  Err_file        : FILE                  -- err log file
  Opened          : BOOLEAN               -- err log file open or not
```

Display List from Dictionary File - Declare Routines

```
-----------------------------------------------------------------------------
----    Section 3:  Routine Declaration
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----    Section 3-A: Op_Err_File Declaration
----                 Open the error log file.
-----------------------------------------------------------------------------
Routine Op_Err_File
Begin
  Opened = false
  Write TPPROMPT(CR,'Creating Auto Error File .............')
  OPEN FILE Err_File ('RW','RD2U:\D_LIST.ERR')  -- open for output
  IF (IO_STATUS(Err_File) <> 0 ) THEN
     CLR_IO_STAT(Err_File)
     Write TPPROMPT('*** USE USER WINDOW FOR ERROR OUTPUT ***',CR)
  ELSE
     Opened = TRUE
  ENDIF
End Op_Err_File
-----------------------------------------------------------------------------
----    Section 3-B: Chk_Add_Dct  Declaration
----                 Check whether a dictionary is loaded.
----                 If not loaded then load in the dictionary.
-----------------------------------------------------------------------------
Routine Chk_Add_Dct
Begin -- Chk_Add_Dct
  -- Make sure 'DLST' dictionary is added.
    CHECK_DICT('DLST',TPTSSP_TITLE,STATUS)
    IF STATUS <> 0 THEN
       Write TPPROMPT(CR,'Loading Required Dictionary............')
       ADD_DICT('RD2U:\DCLISTEG','DLST',DP_DEFAULT,DP_DRAM,STATUS)
       IF status <> 0 THEN
           WRITE TPPROMPT('ADD_DICT failed, STATUS=',STATUS,CR)
           ABORT  -- Without the dictionary this program can not continue.
```

Draft

```
         ENDIF
      ELSE
         WRITE TPPROMPT ('Dictionary already loaded in system.    ')
      ENDIF
End Chk_Add_Dct
```

Display List from Dictionary File - Declare Error Routines

```
------------------------------------------------------------------------------
----    Section 3-C:  Log_Errors Declaration
----                   Log detected errors to a file to be reviewed later.
------------------------------------------------------------------------------
ROUTINE Log_Errors(Out: FILE; Err_Str:STRING;Err_No:INTEGER)
BEGIN
  IF NOT Opened THEN  -- If error log file not opened then write errors to
                      -- screen
    WRITE (Err_Str,Err_No,CR)
  ELSE
    IF NOT UNINIT(Out)  THEN
      CLR_IO_STAT(Out)
      WRITE Out(Err_Str,Err_No,CR,CR)
    ELSE
      WRITE (Err_Str,Err_No, CR)
    ENDIF
  ENDIF
END Log_Errors
------------------------------------------------------------------------------
----    Section 3-D:  Chk_Stat Declaration
----                   Check the global variable, status.
----                   If not zero then log input parameter, err_str,
----                   to error file.
------------------------------------------------------------------------------
ROUTINE Chk_Stat ( err_str: STRING)
BEGIN -- Chk_Stat
  IF( status <> 0) then
    Log_Errors(Err_File, err_str,Status)
  ENDIF
END Chk_Stat
------------------------------------------------------------------------------
----    Section 3-E:  TP_CLS Declaration
------------------------------------------------------------------------------
ROUTINE TP_CLS FROM ROUT_EX
```

Display List from Dictionary File - Setup and Define Screen

Draft

```
-----------------------------------------------------------------------------
----    Section 4:  Main Program
-----------------------------------------------------------------------------
BEGIN -- DCLST_EX
-----------------------------------------------------------------------------
----    Section 4-A:  Perform Setup operations
-----------------------------------------------------------------------------
  TP_CLS   -- Call routine to clear and force the TP USER menu to be visible
  Write ('   ***** Starting DISCTRL_LIST Example *****', CR, CR)
  Chk_Add_Dct     -- Call routine to check and add dictionary
  Op_Err_File     -- Call routine open error log file
-----------------------------------------------------------------------------
----    Section 4-B:  Define and Active a screen
-----------------------------------------------------------------------------
  DEF_SCREEN('LIST', 'TP', status)   -- Create/Define a screen called LIST
   Chk_Stat ('DEF_SCREEN LIST')      -- Verify DEF_SCREEN was successful
  ACT_SCREEN('LIST', old_screen, status) -- activate the LIST screen that
                                         -- that was just defined.
   Chk_Stat ('ACT_SCREEN LIST')          -- Verify ACT_SCREEN was successful
-----------------------------------------------------------------------------
----    Section 4-C:  Attach windows to the screen
-----------------------------------------------------------------------------
  -- Attach the required windows to the LIST screen.
  -- SEE:
  --  Chapter 7.9.1 "USER Menu on the Teach Pendant,
  --  for more details on predefined window names.
  ATT_WINDOW_S('ERR', 'LIST', 1, 1, status)  -- attach the error window
    Chk_Stat('Attaching ERR')
  ATT_WINDOW_S('T_ST', 'LIST', 2, 1, status) -- attach the status window
    Chk_Stat('T_ST not attached')
  ATT_WINDOW_S('T_FU', 'LIST', 5, 1, status) -- attach the full window
    Chk_Stat('T_FU not attached')
  ATT_WINDOW_S('T_PR', 'LIST', 15, 1, status)-- attach the prompt window
    Chk_Stat('T_PR not attached')
  ATT_WINDOW_S('T_FK', 'LIST', 16, 1, status)-- attach the function window
    Chk_Stat('T_FK not attached')
```

Display List from Dictionary File - Write Elements to the Screen

```
-----------------------------------------------------------------------------
----    Section 4-D:  Write dictionary elements to windows
-----------------------------------------------------------------------------
    -- Write dictionary element,TPTSSP_TITLE, from DLST dictionary.
    -- Which will clear the status window, and display intro message in
    -- reverse video.
    WRITE_DICT(TPSTATUS, 'DLST', TPTSSP_TITLE, status)
```

```
       Chk_Stat( 'TPTSSP_TITLE not written')
    -- Write dictionary element,TPTSSP_FK1, from DLST dictionary.
    -- Which will display "[TYPE]" to the function line window.
    WRITE_DICT(TPFUNC, 'DLST', TPTSSP_FK1, status)
       Chk_Stat( 'TPTSSP_FK1 not written')
    -- Write dictionary element, TPTSSP_CLRSC, from DLST dictionary.
    -- Which will clear the teach pendant display window.
    WRITE_DICT(TPDISPLAY, 'DLST', TPTSSP_CLRSC, status)
       Chk_Stat( 'TPTSSP_CLRSC not written')
    -- Write dictionary element, TPTSSP_INSTR, from DLST dictionary.
    -- Which will display instructions to the prompt line window.
    WRITE_DICT(TPPROMPT, 'DLST', TPTSSP_INSTR, status)
       Chk_Stat( 'TPTSSP_INSTR not written')
-----------------------------------------------------------------------------
----    Section 4-E:   Determine the number of menu options
-----------------------------------------------------------------------------
    -- Read the dictionary element, TPTSSP_NUM, from DLST dictionary,
    -- Into the first element of list_data.
    -- list_data[1] will be an ASCII representation of the number of
    --  menu options.  last_line will be returned with the number of
    --  lines/elements used in list_data.
    READ_DICT('DLST', TPTSSP_NUM, list_data, 1, last_line, status)
       Chk_Stat( 'TPTSSP_NUM not read')
    -- convert the string into the INTEGER, num_options
    CNV_STR_INT(list_data[1], num_options)
```

Display List from Dictionary File - Initialize Display Data

```
-----------------------------------------------------------------------------
----    Section 4-F:   Initialize the data structure, display_data
----                   Which is used to display the list of menu options.
-----------------------------------------------------------------------------
    -- Initialize the display data structure
    -- In this example we only deal with window 1.
    display_data.win_start[1] = 1  -- Starting row for window 1.
    display_data.win_end[1]  = 10  -- Ending row for window 1.
    display_data.curr_win = 0      -- The current window to display, where
                                   --   zero (0) specifies first window.
    display_data.cursor_row = 1    -- Current row the cursor is on.
    display_data.lins_per_pg = 10  -- The number of lines scrolled when the
                                   -- user pushes SHIFT Up or Down. Usually
                                   -- it is the same as the window size.
    display_data.curs_st_col[1] = 0 -- starting column for field 1
    display_data.curs_en_col[1] = 0 -- ending column for field 1, will be
                                    -- updated a little later
    display_data.curr_field = 0     -- Current field, where
```

Draft

```
                                    --   zero (0) specifies the first field
   display_data.last_field = 0     -- Last field in the list (only using one
                                    --   field in this example).
   display_data.curr_it_num = 1    -- Current item number the cursor is on.
   display_data.sob_it_num = 1     -- Starting item number.
   display_data.eob_it_num = num_options  -- Ending item number, which is
                                          --  the number of options read in.
   display_data.last_it_num  = num_options-- Last item number, also the
                                          --   number of options read in
 -- Make sure the window end is not beyond total number of elements in list.
   IF display_data.win_end[1] > display_data.last_it_num THEN
     display_data.win_end[1] = display_data.last_it_num --reset to last item
   ENDIF
   -- Read dictionary element, TPTSSP_MENU, from dictionary DLST.
   -- list_data will be populated with the menu list information
   -- list_data[1] will contain the first line of information from
   -- the TPTSSP_MENU and list_data[last_line] will contain the last
   -- line of information read from the dictionary.
   READ_DICT('DLST', TPTSSP_MENU, list_data, 1, last_line, status)
    Chk_Stat('Reading menu list failed')
-- Determine longest list element & reset cursor end column for first field.
   FOR i = 1 TO last_line DO
     IF (STR_LEN(list_data[i]) > display_data.curs_en_col[1]) THEN
       display_data.curs_en_col[1] = STR_LEN(list_data[i])
     ENDIF
   ENDFOR
```

Display List from Dictionary File - Control Cursor Movement

```
-----------------------------------------------------------------------------
----    Section 4-G:   Display the list.
-----------------------------------------------------------------------------
   -- Initial Display the menu list.
   DISCTRL_LIST(TPDISPLAY, display_data, list_data, DC_DISP, status)
     Chk_Stat('Error displaying list')
   -- Open a file to the TPDISPLAY window with PASSALL and FIELD attributes
   -- and NOECHO
   SET_FILE_ATR(kb_file, ATR_PASSALL)  -- Get row teach pendant input
   SET_FILE_ATR(kb_file, ATR_FIELD)    -- so that a single key will
                                       -- satisfy the reads.
   SET_FILE_ATR(kb_file, ATR_NOECHO)   -- don't echo the keys back to
                                       -- the screen
   OPEN FILE Kb_file ('RW', 'KB:TPKB') -- open a file to the Teach pendant
                                       -- keyboard (keys)
     status = IO_STATUS(Kb_file)
     Chk_Stat('Error opening TPKB')
```

```
    act_pending = 0
    done = FALSE
------------------------------------------------------------------------------
----    Section 4-H:  Control cursor movement within the list
------------------------------------------------------------------------------
    REPEAT      -- Wait for a key input
      READ Kb_file (str::1)
      key = ORD(str,1)
      key = key AND 255           -- Convert the key to correct value.
      SELECT key OF               -- Decide how to handle key inputs
       CASE (KY_UP_ARW) :         -- up arrow key pressed
        IF act_pending <> 0 THEN  -- If a menu item was selected...
                                  -- Clear confirmation prompt
          WRITE_DICT(TPPROMPT, 'DLST', TPTSSP_CLRSC, status)
             -- Clear confirmation function keys
          WRITE_DICT(TPFUNC, 'DLST', TPTSSP_CLRSC, status)
        ENDIF
        DISCTRL_LIST(TPDISPLAY, display_data, list_data, DC_UP, status)
          Chk_Stat ('Error displaying list')
       CASE (KY_DN_ARW) : -- down arrow key pressed
        IF act_pending <> 0 THEN -- If a menu item was selected...
          -- Clear confirmation prompt
          WRITE_DICT(TPPROMPT, 'DLST', TPTSSP_CLRSC, status)
          -- Clear confirmation function keys
          WRITE_DICT(TPFUNC, 'DLST', TPTSSP_CLRSC, status)
        ENDIF
        DISCTRL_LIST(TPDISPLAY, display_data, list_data,
DC_DN, status)
           Chk_Stat ('Error displaying list')
```

Display List from Dictionary File - Control Cursor Movement Continued

```
CASE (KY_ENTER) :
        -- Perform later
CASE (KY_F4) : -- "YES" function key pressed
       IF act_pending <> 0 THEN -- If a menu item was selected...
          -- Clear confirmation prompt
          WRITE_DICT(TPPROMPT, 'DLST', TPTSSP_CLRSC, status)
          -- Clear confirmation function keys
          WRITE_DICT(TPFUNC, 'DLST', TPTSSP_CLRSC, status)
          IF act_pending = num_options THEN
            -- Exit the routine
            done = TRUE
          ENDIF
          -- Clear action pending
          act_pending = 0
```

```
        ENDIF
CASE (KY_F5) : -- "NO" function key pressed
              -- Clear confirmation prompt
      WRITE_DICT(TPPROMPT, 'DLST', TPTSSP_INSTR, status)
              -- Clear confirmation function keys
      WRITE_DICT(TPFUNC, 'DLST', TPTSSP_CLRSC, status)
              -- Clear action pending
       act_pending = 0
ELSE :        -- User entered an actual item number.  Calculate which
              -- row the cursor should be on and redisplay the list.
       IF ((key > 48) AND (key <= (48 + num_options))) THEN
         -- Translate number to a row
         key = key - 48
         display_data.cursor_row = key
        DISCTRL_LIST(TPDISPLAY,display_data,list_data,DC_DISP,status)
          Chk_Stat ('Error displaying list')
         key = KY_ENTER
       ENDIF
ENDSELECT
IF key = KY_ENTER THEN -- User has specified an action
  -- Write confirmation prompt for selected item
  WRITE_DICT (TPPROMPT, 'DLST',
     (TPTSSP_CNF1 - 1 + display_data.cursor_row), status)
  -- Display confirmation function keys
  WRITE_DICT(TPFUNC, 'DLST', TPTSSP_FK2, status)
  -- Set action pending to selected item
  act_pending = display_data.cursor_row -- this is the item selected
ENDIF
UNTIL done   -- repeat until the user selects the exit option
```

Display List from Dictionary File - Cleanup and Exit Program

```
-----------------------------------------------------------------------------
----    Section 4-I:  Cleanup before exiting program
-----------------------------------------------------------------------------
  -- Clear the TP USER menu windows
  WRITE_DICT(TPDISPLAY, 'DLST', TPTSSP_CLRSC, status)
  WRITE_DICT(TPSTATUS, 'DLST', TPTSSP_CLRSC, status)
  -- Close the file connected to the TP keyboard.
  CLOSE FILE Kb_file
  -- Close the error log file if it is open.
  IF opened THEN
     close file Err_File
  ENDIF
  Write TPPROMPT (cr,'Example Finished ')
  REMOVE_DICT ( 'LIST', dp_default, status) -- remove the dictionary
```

```
      write ('remove dict', status,cr)
      Chk_Stat ('Removing dictionary')
    ACT_SCREEN(old_screen, old_screen, status) -- activate the previous screen
      Chk_stat ('Activating old screen')
    DET_WINDOW('ERR', 'LIST', status)  -- Detach all the windows that were
      Chk_stat ('Detaching ERR window')
    DET_WINDOW('T_ST', 'LIST', status) -- previously attached.
      Chk_stat ('Detaching T_ST window')
    DET_WINDOW('T_FU', 'LIST', status)
      Chk_stat ('Detaching T_FU window')
    DET_WINDOW('T_PR', 'LIST', status)
      Chk_stat ('Detaching T_PR window')
    DET_WINDOW('T_FK', 'LIST', status)
      Chk_stat ('Detaching T_FK window')
  END DCLST_EX
```

Draft

B.12.1 Dictionary Files

This ASCII dictionary file is used to create a teach pendant screen which is used with DCLST_EX.KL. For more information on DCLST_EX.KL refer to Section B.7 .

Dictionary File

```
.kl dclist
*
$-,TPTSSP_TITLE &home &reverse "Karel DISCTRL_LIST Example
"
&standard
$-,TPTSSP_CLRSC &home &clear_2_eow
$-,TPTSSP_FK1 &home" [TYPE]                                  "
$-,TPTSSP_FK2 &home"                          YES       NO    "
$-,TPTSSP_INSTR &home "Press 'ENTER' or number key to select." &clear_2_eol
* Add menu options here, "Exit" must be last option
* TPTSSP_NUM specifies the number of menu options
$-,TPTSSP_NUM "14"
$-,TPTSSP_MENU
" 1  Test Cycle 1" &new_line
" 2  Test Cycle 2" &new_line
" 3  Test Cycle 3" &new_line
" 4  Test Cycle 4" &new_line
" 5  Test Cycle 5" &new_line
" 6  Test Cycle 6" &new_line
" 7  Test Cycle 7" &new_line
" 8  Test Cycle 8" &new_line
```

```
" 9  Test Cycle 9" &new_line
" 10  Test Cycle 10" &new_line
" 11  Test Cycle 11" &new_line
" 12  Test Cycle 12" &new_line
" 13  Test Cycle 13" &new_line
" 14  EXIT"
* Confirmations must be in order
$-,TPTSSP_CNF1 &home"Perform test cycle 1? [NO]" &clear_2_eol
$-,TPTSSP_CNF2 &home"Perform test cycle 2? [NO]" &clear_2_eol
$-,TPTSSP_CNF3 &home"Perform test cycle 3? [NO]" &clear_2_eol
$-,TPTSSP_CNF4 &home"Perform test cycle 4? [NO]" &clear_2_eol
$-,TPTSSP_CNF5 &home"Perform test cycle 5? [NO]" &clear_2_eol
$-,TPTSSP_CNF6 &home"Perform test cycle 6? [NO]" &clear_2_eol
$-,TPTSSP_CNF7 &home"Perform test cycle 7? [NO]" &clear_2_eol
$-,TPTSSP_CNF8 &home"Perform test cycle 8? [NO]" &clear_2_eol
$-,TPTSSP_CNF9 &home"Perform test cycle 9? [NO]" &clear_2_eol
$-,TPTSSP_CNF10 &home"Perform test cycle 10? [NO]" &clear_2_eol
$-,TPTSSP_CNF11 &home"Perform test cycle 11? [NO]" &clear_2_eol
$-,TPTSSP_CNF12 &home"Perform test cycle 12? [NO]" &clear_2_eol
$-,TPTSSP_CNF13 &home"Perform test cycle 13? [NO]" &clear_2_eol
$-,TPTSSP_CNF14 &home"Exit? [NO]" &clear_2_eol
```

Draft

B.13 USING THE DISCTRL_ALPHA BUILT-IN

This program shows three different ways to use the DISCTRL_ALPH built-in. The DISCTRL_ALPH built-in displays and controls alphanumeric string entry in a specified window. Refer to Appendix A , "KAREL Language Alphabetic Description" for more information.

Method 1 allows a program name to be entered using the default value for the dictionary name. See Section 4-A in Using the DISCTRL_ALPHA Built-in - Enter Data from Teach Pendant .

Method 2 allows a comment to be entered using the default value for the dictionary name. See Section 4-B in Using the DISCTRL_ALPHA Built-in - Enter Data from Teach Pendant .

Method 3 uses a user specified dictionary name and element to enter a program name. See Section 4-C in Using the DISCTRL_ALPHA Built-in - Enter Data from CRT/KB .

This program also posts all errors to the controller.

Using the DISCTRL_ALPHA Built-in - Overview

```
-----------------------------------------------------------------------------
----    DCALP_EX.KL
-----------------------------------------------------------------------------
---- Elements of KAREL Language Covered:          In Section:
----    Actions:
```

```
----    Clauses:
----    Conditions:
----    Data types:
----            INTEGER                                    Sec 2
----            STRING                                     Sec 2
----    Directives:
----            COMMENT                                   Sec 1
----            INCLUDE                                   Sec 1
----            NOLOCKGROUP                               Sec 1
----    Built-in Functions & Procedures:
----            ADD_DICT                                   Sec 4-C
----            CHR                                        Sec 4-A,B,C
----            DISCTRL_ALPH                               Sec 4-A,B,C
----            FORCE_SPMENU                               Sec 4-A,B,C
----            POST_ERR                                   Sec 4-A,B,C
----            SET_CURSOR                                 Sec 4-A,B,C
----            SET_LANG                                   Sec 4-C
----    Statements:
----            READ                                       Sec 4-A,B
----            WRITE                                      Sec 4-A,B,C
----            IF...THEN...ENDIF                          Sec 4-A,B,C
----    Reserve Words:
----            BEGIN                                      Sec 4
----            CONST                                      Sec 2
----            CR                                         Sec 4-A,B,C
----            END                                        Sec 4-C
----            PROGRAM                                    Sec 1
----            VAR                                        Sec 2
----    Predefined File Names:
----            OUTPUT                                     Sec 4-C
----            TPDISPLAY                                  Sec 4-A,B
----     Predefined Windows:
----            T_FU                                       Sec 4-A,B
----            C_FU                                       Sec 4-C
```

Using the DISCTRL_ALPHA Built-in - Declaration Section

```
-----------------------------------------------------------------------------
----    Section 1:  Program and Environment Declaration
-----------------------------------------------------------------------------
PROGRAM DCALP_EX
%COMMENT   = 'Display Alpha'
%NOLOCKGROUP
%INCLUDE KLEVKEYS    -- Necessary for the KY_ENTER
%INCLUDE DCALPH      -- Necessary for the ALPH_PROG Element, see section 4-C
-----------------------------------------------------------------------------
```

```
----    Section 2:  Constant and Variable  Declarations
------------------------------------------------------------------------------
CONST
  cc_home      = 137
  cc_clear_win = 128
  cc_warn      = 0     -- Value passed to POST_ERR to display warning only.
  cc_pause     = 1     -- value passed to POST_ERR to pause program.
VAR
  status       : INTEGER
  device_stat  : INTEGER
  term_char    : INTEGER
  window_name  : STRING[4]
  prog_name    : STRING[12]
  comment      : STRING[40]
------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
------------------------------------------------------------------------------
```

Using the DISCTRL_ALPHA Built-in - Enter Data from Teach Pendant

```
------------------------------------------------------------------------------
----    Section 4:  Main Program
------------------------------------------------------------------------------
BEGIN   -- DCALP_EX
------------------------------------------------------------------------------
---- Section 4-A: Enter a program name from the teach pendant USER menu
------------------------------------------------------------------------------
  WRITE (CHR(cc_home), CHR(cc_clear_win))    -- Clear TP USER menu
  FORCE_SPMENU(tp_panel, SPI_TPUSER, 1)      -- Force TP USER menu to be
                                            -- visible
  SET_CURSOR(TPDISPLAY, 12, 1, status)      -- reposition cursor
  WRITE ('prog_name: ')
  prog_name = ''                           -- initialize program name
  DISCTRL_ALPH('t_fu', 12, 12, prog_name, 'PROG', 0, term_char, status)
  IF status <> 0 THEN
    POST_ERR(status, '', 0, cc_warn)
  ENDIF
  IF term_char = ky_enter THEN             -- User pressed the ENTER key
    WRITE (CR, 'prog_name was changed:', prog_name, CR)
  ELSE
    WRITE (CR, 'prog_name was not changed')
  ENDIF
  WRITE (CR, 'Press enter to continue')
  READ (CR)
------------------------------------------------------------------------------
---- Section 4-B: Enter a comment from the teach pendant
```

Draft

```
------------------------------------------------------------------------------
  WRITE (CHR(cc_home) + CHR(cc_clear_win))-- Clear TP USER menu
  SET_CURSOR(TPDISPLAY, 12, 1, status)   -- reposition cursor
  comment = ' '                         -- Initialize the comment
  WRITE ('comment: ')                   -- Display message
  DISCTRL_ALPH('t_fu', 12, 10, comment, 'COMM', 0, term_char, status)
  IF status <> 0 THEN                   -- Verify DISCTRL_ALPH was successful
    POST_ERR(status, '', 0, cc_warn)    -- Post the status as a warning
  ENDIF
  IF term_char = ky_enter THEN
    WRITE (CR, 'comment was changed:', comment, CR) -- Display new comment
  ELSE
    WRITE (CR, 'comment was not changed', CR)
  ENDIF
  WRITE (CR, 'Press enter to continue')
  READ (CR)
```

Using the DISCTRL_ALPHA Built-in - Enter Data from CRT/KB

```
------------------------------------------------------------------------------
---- Section 4-C: This section will perform program name entry from the
----             CRT/KB.  The dictionary name and element values are
----             explicitly stated here, instead of using the available
----              default values.
------------------------------------------------------------------------------
  -- Set the dictionary language to English
  -- This is useful if you want to use this same code for multiple
  -- languages.  Then any time you load in a dictionary you check
  -- to see what the current language, $language, is and load the
  -- correct dictionary.
  -- For instance you could have a DCALPHJP.TX file which
  -- would be the Japanese dictionary. If the current language, $language,
  -- was set to Japanese you would load this dictionary.
  SET_LANG ( dp_english, status)
  IF (status <> 0) THEN
    POST_ERR (status, '', 0,cc_warn)   -- Post the status as a warning
  ENDIF
  -- Load the dcalpheg.tx file, using ALPH as the dictionary name,
  --  to the English language, using DRAM as the memory storage device.
  ADD_DICT ('DCALPHEG', 'ALPH', dp_english,  dp_dram, status)
  IF (status <> 0 ) THEN
    POST_ERR (status, '', 0, cc_pause)     -- Post the status and pause the
                                           -- program, since the dictionary
  ENDIF                                    -- must be loaded to continue.
 device_stat = crt_panel                     -- Give control to the CRT/KB
  WRITE OUTPUT (CHR(cc_home) + CHR(cc_clear_win))--Clear CRT/KB USER menu
```

```
  FORCE_SPMENU (device_stat, SPI_TPUSER, 1) -- Force the CRT/KB USER menu
                                            --  to be visible
  SET_CURSOR(OUTPUT, 12, 1, status)         -- Reposition the cursor
  WRITE OUTPUT ('prog_name: ')
  prog_name = ' '                           -- Initialize program name
  DISCTRL_ALPH('c_fu',12,12,prog_name,'ALPH',alph_prog,term_char,status)
--DISCTRL_ALPHA uses the ALPH dictionary and ALPH_PROG dictionary element
  IF status <> 0 THEN                       -- Verify DISCTRL_ALPH was
                                            -- successful.
    POST_ERR(status, '', 0, cc_warn)        -- Post returned status to the
  ENDIF                                     -- error ('err') window.
  IF term_char = ky_enter THEN
    WRITE (CR, 'prog_name was changed:', prog_name, CR)
  ELSE
    WRITE (CR, 'prog_name was NOT changed.', CR)
  ENDIF
  device_stat = tp_panel                     -- Make sure to reset
 END DCALP_EX
```

B.13.1 Dictionary Files

This ASCII dictionary file is used to write text to the specified screen. DCALPH_EG.UTX is also used with DCAL_EX.KL. For more information on DCAL_EX.KL, refer to Section B.13 .

DCALPHEG.UTX Dictionary File

```
.KL DCALPH
$, alpha_prog
" PRG       MAIN     SUB       TEST     >"&new_line
" PROC      JOB      MACRO              >"&new_line
" TEST1     TEST2    TEST3     TEST4    >"
```

Note The greater than symbol (>) in DCALPHEG.UTX Dictionary File is a reminder to use the NEXT key to scroll through the multiple lines. Also notice that the &new_line appears only on the first two lines. This ensures that the lines will scroll correctly.

B.14 APPLYING OFFSETS TO A COPIED TEACH PENDANT PROGRAM

This program copies a teach pendant program and then applies an offset to the positions within the newly created program. This is useful when you create the teach pendant program offline and then

realized that all the teach pendant positions are off by some determined amount. However, you should be aware that the utility PROGRAM ADJUST is far more adequate for this job.

Applying Offsets to Copied Teach Pendant Program - Overview

```
-------------------------------------------------------------------------------
----    CPY_TP.KL
-------------------------------------------------------------------------------
---- Elements of KAREL Language Covered:         In Section:
----    Actions:
----
----    Clauses:
----            FROM                               Sec 3-F
----
----    Conditions:
----
----    Data Types:
----            ARRAY OF REAL                       Sec 2
----            ARRAY OF STRING                     Sec 2
----            BOOLEAN                             Sec 2
----            INTEGER                             Sec 2; 3-B,C,D,E
----            JOINTPOS                            Sec 2
----            REAL                                Sec 2
----            STRING                              Sec 2
----            XYZWPR                              Sec 2
----    Directives:
----            ENVIRONMENT                        Sec 1
----    Built-in Functions & Procedures:
----            AVL_POS_NUM                         Sec 3-E
----            CHR                                 Sec 3-B, 4
----            CLOSE_TPE                           Sec 3-E
----            CNV_REL_JPOS                        Sec 3-E
----            CNV_JPOS_REL                        Sec 3-E
----            COPY_TPE                            Sec 3-E
----            GET_JPOS_TPE                        Sec 3-E
----            GET_POS_TYP                         Sec 3-E
----            GET_POS_TPE                         Sec 3-E
----            OPEN_TPE                            Sec 3-E
----            PROG_LIST                           Sec 3-B
----            SELECT_TPE                          Sec 3-E
----            SET_JPOS_TPE                        Sec 3-E
----            SET_POS_TPE                         Sec 3-E
```

Applying Offsets to Copied Teach Pendant Program - Overview Continued

```
----    Statements:
```

Draft

```
----            FOR...ENDFOR                        Sec 3-B,D,E
----            IF ...THEN...ELSE...ENDIF           Sec 3-A,B,C,E
----            READ                                Sec 3-B,C,D
----            REPEAT...UNTIL                      Sec 3-B,C,D
----            RETURN                              Sec 3-E
----            ROUTINE                             Sec 3-A,B,C,D,E,F
----            SELECT...ENDSELECT                  Sec 3-E
----            WRITE                               Sec 3-B,C,D,E,4
----
----    Reserve Words:
----            BEGIN                               Sec 3-A,B,C,D,E;
4
----            CONST                               Sec 2
----            CR                                  Sec 3-B,C,D,E
----            END                                 Sec 3-A,B,C,D,E;
4
----            PROGRAM                             Sec 1
----            VAR                                 Sec 2
```

Draft

Applying Offsets to Copied Teach Pendant Program - Declaration Section

```
------------------------------------------------------------------------------
----   Section 1:  Program and Environment Declaration
------------------------------------------------------------------------------
PROGRAM CPY_TP
%ENVIRONMENT TPE             -- necessary for all xxx_TPE built-ins
%ENVIRONMENT BYNAM           -- necessary for PROG_LIST built-in
------------------------------------------------------------------------------
----    Section 2:  Constant and Variable Declaration
------------------------------------------------------------------------------
CONST
  ER_WARN = 0                    -- warning constant for use in POST_ERR
  SUCCESS = 0                    -- success constant
  JNT_POS = 9                    -- constant for GET_POS_TYP
  XYZ_POS = 2                    -- constant for GET_POS_TYP
  MAX_AXS = 9                    -- Maximum number of axes JOINTPOS has
VAR
 from_prog: STRING[13]           -- TP program name to be copied FROM
 to_prog  : STRING[13]           -- TP program name to be copied TO
 over_sw  : BOOLEAN              -- Decides whether to overwrite an existing
                                 --  program when performing COPY_TPE
 status   : INTEGER              -- Holds error status from the builtin calls
 off_xyz  : XYZWPR               -- Offset amount for the XYZWPR positions
 jp_off   : ARRAY [9] of REAL    -- Offset amount for the JOINT positions
 new_xyz  : XYZWPR               -- XYZWPR which has offset applied
 org_xyz  : XYZWPR               -- Original XYZWPR from to_prog
```

```
 new_jpos : JOINTPOS             -- JOINTPOS which as the offset applied
 org_jpos : JOINTPOS             -- Original JOINTPOS from to_prog
 open_id  : INTEGER              -- Identifier for the opened to_prog
 jp_org   : ARRAY [9] of REAL    -- REAL representation of org_jpos
 jp_new   : ARRAY [9] of REAL    -- REAL representation of jp_new
```

Applying Offsets to Copied Teach Pendant Program - Declare Routines

```
-------------------------------------------------------------------------------
----    Section 3:  Routine Declaration
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
----    Section 3-A: CHK_STAT Declaration
----                 Tests whether the status was successful or not.
----                 If the status was not successful the status is posted
-------------------------------------------------------------------------------
ROUTINE chk_stat (rec_stat: integer)
begin
 IF (rec_stat <> SUCCESS) THEN   -- if rec_stat is not SUCCESS
     -- then post the error
  POST_ERR (rec_stat, '', 0, ER_WARN)   -- Post the error to the system.
 ENDIF
END chk_stat
-------------------------------------------------------------------------------
----    Section 3-B: GetFromPrg Declaration
----                 Generate a list of loaded TPE programs.
----                 Lets the user select one of these programs
----                 to be the program to be copied, ie FROM_prog
-------------------------------------------------------------------------------
ROUTINE GetFromPrg
VAR
 tp_type   : INTEGER         -- Types of program to list
 n_skip    : INTEGER         -- Index into the list of programs
 format    : INTEGER         -- What type of format to store programs in
 n_progs   : INTEGER         -- Number of programs returned in prg_name
 prg_name  : ARRAY [8] of STRING[20] --Program names returned from PROG_LIST
 status    : INTEGER         -- Status of PROG_LIST call
 f_index   : INTEGER         -- Fast index for generating the program listing.
 arr_size  : INTEGER         -- Array size of prg_name
 prg_select: INTEGER         -- Users selection for which program to copy
 indx      : INTEGER         -- FOR loop counter which displays prg_name
```

Applying Offsets to Copied Teach Pendant Program - Generate Program List for User

```
BEGIN
```

Draft

```
  f_index = 0    -- Initialize the f_index
  n_skip  = 0    -- Initialize the n_skip
  tp_type = 2    -- find any TPE program
  format  = 1    -- return just the program name in prg_name
  n_progs = 0    -- Initialize the n_progs
  arr_size = 8   -- Set equal to the declared array size of prg_name
  prg_select = 0 -- Initialize the program selector
  REPEAT
     WRITE (chr(128),chr(137))  -- Clear the TP USER screen
     -- Get a listing of all TP program which begin with "TEST"
    PROG_LIST('TEST*',tp_type,n_skip,format,prg_name,n_progs,status,f_index)
     chk_stat (status)    --Check status from PROG_LIST
     FOR indx = 1 to n_progs DO
      WRITE (indx,':',prg_name[indx], CR) -- Write the list of programs out
     ENDFOR
     IF (n_skip > 0) OR ( n_prog >0) THEN
       WRITE ('select program to be copied:',CR)
       WRITE ('PRESS -1 to get next page of programs:')
       REPEAT
         READ (prg_select)  -- get program selection
       UNTIL ((prg_select >= -1) AND (prg_select <= n_progs)
                AND (prg_select <> 0))
     ELSE
       WRITE ('no TP programs to COPY', CR)
       WRITE ('Aborting program, since need',CR)
       WRITE ('at least one TP program to copy.',CR)
       ABORT
     ENDIF
     -- Check if listing is complete and user has not made a selection.
     IF ((prg_select = -1) AND (n_progs < arr_size)) THEN
       f_index = 0               --reset f_index to re-generate list.
       n_progs = arr_size       --set so the REPEAT/UNTIL will continue
     ENDIF
     -- Check if user user has made a selection
     IF (prg_select <> -1) then
       from_prog = prg_name[prg_select]-- Set from_prog to name selected
       n_progs = 0                -- Set n_prog to stop looping.
     ENDIF
  UNTIL (n_progs < arr_size)
END GetFromPrg
```

Applying Offsets to Copied Teach Pendant Program - Overwrite or Delete Program

```
-------------------------------------------------------------------------------
----    Section 3-C: GetOvrSw Declaration
----                 Ask user whether to overwrite the copied
```

```
----                    program, TO_prog, if it exists.
------------------------------------------------------------------------------
ROUTINE GetOvrSw
VAR
  yesno  : INTEGER
BEGIN
  WRITE (CR, 'If Program already exists do you want',CR)
  WRITE ('to overwrite the file Yes:1, No:0 ? ')
  REPEAT
    READ (yesno)
  UNTIL ((yesno = 0) OR( yesno = 1))
  IF yesno = 1 then   --Set over_sw so program is overwriten if it exists
     over_sw = TRUE
  ELSE                --Set over_sw so program is NOT overwriten if it exists
     over_sw = FALSE
  ENDIF
END GetOvrSw
```

Applying Offsets to Copied Teach Pendant Program - Input Offset Positions

```
------------------------------------------------------------------------------
----    Section 3-D: GetOffset Declaration
----                 Have the user input the offset for both
----                 XYZWPR and JOINTPOS positions.
------------------------------------------------------------------------------
ROUTINE GetOffset
VAR
 yesno  : INTEGER
 index   : INTEGER
BEGIN
     --Get the XYZWPR offset, off_xyz
 REPEAT
   WRITE ( 'Enter offset for XYZWPR positions',CR)
   WRITE (' X = ')
   READ  (off_xyz.x)
   WRITE (' Y = ')
   READ  (off_xyz.y)
   WRITE (' Z = ')
   READ  (off_xyz.z)
   WRITE (' W = ')
   READ  (off_xyz.w)
   WRITE (' P = ')
   READ  (off_xyz.p)
   WRITE (' R = ')
   READ  (off_xyz.r)
   --Display the offset values the user input
```

```
    WRITE (' Offset XYZWPR position is',CR, off_xyz,CR)
    WRITE ('Is this offset correct? Yes:1, No:0 ? ')
    READ  (yesno)
  UNTIL (yesno = 1)    -- enter offset amounts until the user
     -- is satisfied.
  --Get the JOINTPOS offset, jp_off
  REPEAT
    WRITE ( 'Enter offset for JOINT positions',CR)
    FOR indx = 1 TO 6 DO     -- loop for number of robot axes
       WRITE (' J',indx,' = ')
       READ  (jp_off[indx])
    ENDFOR   WRITE ('JOINT position offset is', CR)
    FOR indx = 1 TO 6 DO
       write ( jp_off[indx],CR)     -- Display the values the user input
    ENDFOR
    WRITE ('Is this offset correct? Yes:1, No:0 ? ')
    READ  (yesno)
  UNTIL (yesno = 1)    -- Enter offset amounts until the user
END GetOffset    -- is satisfied
```

Applying Offsets to Copied Teach Pendant Program - Apply Offsets to Positions

```
-------------------------------------------------------------------------------
----    Section 3-E: ModifyPrg Declaration
----                 Apply the offsets to each position within the TP program
-------------------------------------------------------------------------------
ROUTINE ModifyPrg
VAR
  pos_typ : INTEGER  --The type of position returned from GET_POS_TYP
  num_axs : INTEGER  -- The number of axes if position is a JOINTPOS type
  indx_pos: INTEGER  -- FOR loop counter, that increments through TP position
  group_no: INTEGER  -- The group number of the current position setting.
  num_pos : INTEGER  -- The next available position number within TP program
  indx_axs: INTEGER  -- FOR loop counter, increments through REAL array
BEGIN
  SELECT_TPE ('', status)  -- Make sure the to_prog is currently not selected
  to_prog = 'MDFY_TP'      -- Set the to_prog to desired name.
  ------ Copy the from_prog to to_prog -----
  COPY_TPE (from_prog, to_prog, over_sw, status)
  chk_stat(status)           -- check status of COPY_TPE
--- If the user specified not to overwrite the TPE program and
---  the status returned is 7015, "program already exist",
---  then quit the program.  This will mean not altering the already
---  existing to_prog.
  IF ((over_sw = FALSE) AND (status = 7015)) THEN
      WRITE ('ABORTING:: PROGAM ALREADY EXISTS!',CR)
```

```
        RETURN
    ENDIF
--- Open the to_prog with the Read/Write access
 OPEN_TPE (to_prog, TPE_RWACC, TPE_RDREJ, open_id, status)
 chk_stat(status)               -- check status of OPEN_TPE
 group_no = 1
--- apply offset to each position within to_prog
--- The current number of position that the TPE program has is num_pos -1
 FOR indx_pos = 1 to num_pos-1 DO
   --Get the DATA TYPE of each position within the to_prog
   --If it is a JOINTPOS also get the number of axes.
   GET_POS_TYP (open_id, indx_pos, group_no, pos_typ, num_axs, status)
   chk_stat (status)
   WRITE('get_pos_typ status', status,cr)
```

Applying Offsets to Copied Teach Pendant Program - Apply Offsets to Positions Cont.

```
--Decide if the position, indx_pos, is a JOINTPOS or a XYZWPR
   SELECT pos_typ OF
     CASE (JNT_POS):        -- The position is a JOINTPOS
        FOR indx_axs = 1 TO MAX_AXS DO    -- initialize with default values
          jp_org[indx_axs] = 0.0          -- This avoids problems with the
          jp_new[indx_axs] = 0.0          -- CNV_REL_JPOS
        ENDFOR
        -- get the JOINTPOS P[indx_pos] from to_prog  -----
        org_jpos = GET_JPOS_TPE (open_id, indx_pos, status)
        chk_stat( status)
        -- Convert the JOINTPOS to a REAL array, in order to perform offset
        CNV_JPOS_REL (org_jpos,  jp_org, status)
        chk_stat (status)
        -- Apply the offset to the REAL array
        FOR indx_axs = 1 to num_axs DO
          jp_new[indx_axs] = jp_org[indx_axs] + jp_off[indx_axs]
        ENDFOR
        -- Converted back to a JOINTPOS.
        -- The input array, jp_new, must not have any uninitialized values
        -- or the error 12311 - "Data uninitialized" will be posted.
        -- This is why we previously set all the values to zero.
        CNV_REL_JPOS (jp_new, new_jpos, status)
        chk_stat(status)
        -- Set the new offset position, new_jpos, into the indx_pos
        SET_JPOS_TPE (open_id, indx_pos, new_jpos, status)
        chk_stat(status)
        write ('indx_pos', indx_pos, 'new_jpos',cr, new_jpos,cr)
      CASE (XYZ_POS): -- The position is a XYZWPR
         -- Get the XYZWPR position P[indx_pos] from to_prog
```

Draft

```
        org_xyz = GET_POS_TPE (open_id , indx_pos, status)
        chk_stat( status)               -- Check status from GET_POS_TPE
```

Applying Offsets to Copied Teach Pendant Program - Clears TP User Menu

```
-- Apply offset to the XYZWPR
        new_xyz.x = org_xyz.x + off_xyz.x
        new_xyz.y = org_xyz.y + off_xyz.y
        new_xyz.z = org_xyz.z + off_xyz.z
        new_xyz.w = org_xyz.w + off_xyz.w
        new_xyz.p = org_xyz.p + off_xyz.p
        new_xyz.r = org_xyz.r + off_xyz.r
         --Set the new offset position, new_xyz, into the indx_pos
        SET_POS_TPE (open_id, indx_pos, new_xyz, status)
        chk_stat (status)             -- Check status from SET_POS_TPE
   ENDSELECT
 ENDFOR
 ---Close TP program before quitting program
 CLOSE_TPE (open_id, status)
 chk_stat (status)                    --Check status from CLOSE_TPE
END ModifyPrg
-----------------------------------------------------------------------------
----    Section 3-F:  TP_CLS Declaration
----                  Clears the TP USER Menu screen and forces it to
----                  become visible.  The actual code resides in
ROUT_EX.KL
-----------------------------------------------------------------------------
ROUTINE TP_CLS FROM rout_ex
-----------------------------------------------------------------------------
----    Section 4:  Main Program
-----------------------------------------------------------------------------
BEGIN  -- CPY_TP
 tp_cls                              -- Clear the TP USER Menu screen
 GetFromPrg                          -- Get the TPE program to copy FROM
 GetOvrSw                            -- Get the TPE program name to copy TO
 GetOffset                           -- Get the offset for modifying
                                     -- the teach pendant program
 ModifyPrg                          -- Modify the copied program by the offset
END CPY_TP
```

Draft

Draft

Appendix C

KCL COMMAND ALPHABETICAL DESCRIPTION

Draft

Contents

Appendix C KCL COMMAND ALPHABETICAL DESCRIPTION C–1
C.1 ABORT command C–6
C.2 APPEND FILE command C–6
C.3 APPEND NODE command C–6
C.4 CHDIR command C–7
C.5 CLEAR ALL command C–7
C.6 CLEAR BREAK CONDITION command C–8
C.7 CLEAR BREAK PROGRAM command C–8
C.8 CLEAR DICT command C–9
C.9 CLEAR PROGRAM command C–9
C.10 CLEAR VARS command C–9
C.11 COMPRESS DICT command C–10
C.12 COMPRESS FORM command C–10
C.13 CONTINUE command C–11
C.14 COPY FILE command C–11
C.15 CREATE VARIABLE command C–12
C.16 DELETE FILE command C–13
C.17 DELETE NODE command C–13
C.18 DELETE VARIABLE command C–14
C.19 DIRECTORY command C–14
C.20 DISABLE BREAK PROGRAM command C–15

C.21 DISABLE CONDITION command C–15
C.22 DISMOUNT command C–15
C.23 EDIT command C–16
C.24 ENABLE BREAK PROGRAM C–16
C.25 ENABLE CONDITION command C–16
C.26 FORMAT command C–17
C.27 HELP command C–17
C.28 HOLD command C–18
C.29 INSERT NODE command C–18
C.30 LOAD ALL command C–18
C.31 LOAD DICT command C–19
C.32 LOAD FORM command C–19
C.33 LOAD MASTER command C–20
C.34 LOAD PROGRAM command C–20
C.35 LOAD SERVO command C–21
C.36 LOAD SYSTEM command C–21
C.37 LOAD TP command C–22
C.38 LOAD VARS command C–23
C.39 LOGOUT command C–24
C.40 MKDIR command C–24
C.41 MOUNT command C–24
C.42 MOVE FILE command C–25
C.43 PAUSE command C–25
C.44 PURGE command C–26
C.45 PRINT command C–26
C.46 RECORD command C–27
C.47 RENAME FILE command C–27
C.48 RENAME VARIABLE command C–28
C.49 RENAME VARS command C–28
C.50 RESET command C–29
C.51 RMDIR command C–29
C.52 RUN command C–30
C.53 RUNCF command C–30
C.54 SAVE MASTER command C–31
C.55 SAVE SERVO command C–31
C.56 SAVE SYSTEM command C–31
C.57 SAVE TP command C–32
C.58 SAVE VARS command C–33
C.59 SET BREAK CONDITION command C–33
C.60 SET BREAK PROGRAM command C–34
C.61 SET CLOCK command C–35

Draft

C.62 SET DEFAULT command C–35
C.63 SET GROUP command C–35
C.64 SET LANGUAGE command C–36
C.65 SET LOCAL VARIABLE command C–36
C.66 SET PORT command C–37
C.67 SET TASK command C–37
C.68 SET TRACE command C–38
C.69 SET VARIABLE command C–38
C.70 SET VERIFY command C–39
C.71 SHOW BREAK command C–39
C.72 SHOW BUILTINS command C–40
C.73 SHOW CONDITION command C–40
C.74 SHOW CLOCK command C–40
C.75 SHOW CURPOS command C–41
C.76 SHOW DEFAULT command C–41
C.77 SHOW DEVICE command C–41
C.78 SHOW DICTS command C–41
C.79 SHOW GROUP command C–41
C.80 SHOW HISTORY command C–42
C.81 SHOW LANG command C–42
C.82 SHOW LANGS command C–42
C.83 SHOW LOCAL VARIABLE command C–42
C.84 SHOW LOCAL VARS command C–44
C.85 SHOW MEMORY command C–44
C.86 SHOW PROGRAM command C–44
C.87 SHOW PROGRAMS command C–45
C.88 SHOW SYSTEM command C–45
C.89 SHOW TASK command C–45
C.90 SHOW TASKS command C–46
C.91 SHOW TRACE command C–46
C.92 SHOW TYPES command C–46
C.93 SHOW VARIABLE command C–47
C.94 SHOW VARS command C–47
C.95 SHOW data_type command C–48
C.96 SIMULATE command C–48
C.97 SKIP command C–49
C.98 STEP OFF command C–50
C.99 STEP ON command C–50
C.100 TRANSLATE command C–50
C.101 TYPE command C–51
C.102 UNSIMULATE command C–51

Draft

C.103 WAIT command C–52

Draft

This section describes each KCL command in alphabetical order. Each description includes the purpose of the command, its syntax, and details of how to use it. Examples of each command are also provided.

The following notation is used to describe KCL command syntax:

- < > indicates optional arguments to a command
- | indicates a choice which must be made
- { } indicates an item can be repeated
- file_spec: <device_name:><\\host_name\><path_name\>file_name.file_type |

  <device_name:><\\host_name\>‘host_specific_name’
- path_name: <file_name\><dir\dir\. . .>
- file_name: maximum of 36 characters, no file type

device_name: is a two to five-character optional field, followed by a colon. The first character is a letter, the remaining characters must be alphanumeric. If this field is left blank, the default device from the system variable $DEVICE will be used.

host_name: is a one to eight character optional field. The host_name selects the network node that receives this command. It must be preceded by two backslashes and separated from the remaining fields by a backslash.

path_name : file_name\<path_name> - is a recursively defined optional field, each field consisting of a maximum of 36 characters. It is used to select the file subdirectory. The root or source directory is handled as a special case. It is designated by a file_name of zero length. For example, access to the subdirectory SYS linked off of the root would have path name ‘\SYS’. A fully qualified file_spec using this path_name would look like this, ‘C1:\HOST\SYS\FILE.KL’.

file_name: from one to 36 characters

file_type: from zero to three characters

KCL commands can be abbreviated allowing you to type in fewer letters as long as the abbreviated version remains unique among all keywords. For example, “ABORT” can be “AB” but “CONTINUE” must be “CONT” to distinguish it from “CONDITION.”

path_names, file_names, and file_types that contain special characters or begin with numbers can be specified as a host_specific_name inside single quotes. FR:\’00\’ is valid, FR:\’00\test.kl’ is valid, FR:\’00’\test.kl is invalid because the host_specific_name must be last.

KCL commands that have <prog_name> as part of the command syntax will use the default program if none is specified. KCL commands that have <file_name> as part of the command syntax will use the default program as the file name if none is specified.

Draft

C.1 ABORT command

Syntax: ABORT < (**prog_name**) | ALL) > <FORCE>

where:

prog_name : the name of any KAREL or TP program which is a task

ALL : aborts all running or paused tasks

FORCE : aborts the task even if the NOABORT attribute is set. FORCE only works with ABORT prog_name; FORCE does not work with ABORT ALL

Purpose: Aborts the specified running or paused task. If **prog_ name** is not specified, the default program is used.

Execution of the current program statement is completed before the task aborts except for the current motion, DELAY, WAIT, or READ statements, which are canceled.

Examples: KCL> ABORT test_prog FORCE

KCL> ABORT ALL

Draft

C.2 APPEND FILE command

Syntax: APPEND FILE **input_file_spec** TO **output_ file_spec**

where:

input_file_spec : a valid file specification

output_file_spec : a valid file specification

Purpose: Appends the contents of the specified input file to the end of the specified output file. The **input_file_spec** and the **output_file_spec** must include both the file name and the file type.

Examples: KCL> APPEND FILE flpy:test.kl TO productn.kl

KCL> APPEND FILE test.kl TO productn.kl

C.3 APPEND NODE command

Syntax: APPEND NODE <[**prog_name**]> **var_name**

where:

prog_name : the name of any KAREL or TP program

var_name : the name of any variable of type PATH

Purpose: Appends one node to the end of the specified PATH variable previously loaded in RAM. The appended node value is uninitialized and the index number is one more than the last node index. Execute the KCL> SAVE VARS command to make the change permanent.

Examples: KCL> APPEND NODE [test_prog]weld_pth

KCL> APPEND NODE weld_pth

C.4 CHDIR command

Syntax: CHDIR < **device_name** >\< **path_name** >\ or CD < **device_name** >\< **path_nam e** >

where:

device_name : a specified device

path_name : a subdirectory previously created on the memory card device using the mkdir command. When the chdir command is used to change to a subdirectory, the entire path will be displayed on the teach pendant screen as mc:\new_dir\new_file.

The double dot (..) can be used to represent the directory one level above the current directory.

Purpose: Changes the default device. If a **device_name** is not specified, displays the default device.

Examples: KCL> CHDIR rd:\

KCL> CD

KCL> CD mc:\a

KCL> CD ..

C.5 CLEAR ALL command

Syntax: CLEAR ALL <YES>

where:

YES : confirmation is not prompted

Purpose: Clears all KAREL and teach pendant programs and variable data from memory. All cleared programs and variables (if they were saved with the KCL> SAVE VARS command) can be reloaded into memory using the KCL> LOAD command.

Draft

Examples: KCL> CLEAR ALL

Are you sure? YES

KCL> CLEAR ALL Y

C.6 CLEAR BREAK CONDITION command

Syntax: CLEAR BREAK CONDITION < **prog_name** > (**brk_pnt_no** | ALL)

where:

prog_name : the name of any KAREL program in memory

brk_pnt_no : a particular condition break point

ALL : clears all condition break points

Purpose: Clears specified condition break point(s) from the specified or default program.

A condition break point only affects the program in which it is set.

Examples: KCL> CLEAR BREAK CONDITION test_prog 3

KCL> CLEAR BREAK COND ALL

C.7 CLEAR BREAK PROGRAM command

Syntax: CLEAR BREAK PROGRAM < **prog_name** > (**brk_pnt_no** | ALL)

where:

prog_name : the name of any KAREL program in memory

brk_pnt_no : a particular program break point

ALL : clears all break points

Purpose: Clears specified break point(s) from the specified or default program.

A break point only affects the program in which it is set.

Examples: KCL> CLEAR BREAK PROGRAM test_prog 3

KCL> CLEAR BREAK PROG ALL

Draft

C.8 CLEAR DICT command

Syntax: CLEAR DICT **dict_name** <(**lang_name** | ALL)>

where:

dict_name : the name of any dictionary to be cleared

lang_name : the name of the language. The available choices are ENGLISH, JAPANESE, FRENCH, GERMAN, SPANISH or DEFAULT.

ALL : clears the dictionary from all languages

Purpose: Clears a dictionary from the specified language or from all languages. If no language is specified, it is cleared from the DEFAULT language only.

Examples: KCL> CLEAR DICT tpsy ENGLISH

KCL> CLEAR DICT tpsy

Draft

C.9 CLEAR PROGRAM command

Syntax: CLEAR PROGRAM < **prog_name** > <YES>

where:

prog_name : the name of any KAREL or teach pendant program in memory

YES : confirmation is not prompted

Purpose: Clears the program data from memory for the specified or default program.

Examples: KCL> CLEAR PROGRAM test_prog

Are you sure? YES

KCL> CLEAR PROG test_prog Y

C.10 CLEAR VARS command

Syntax: CLEAR VARS < **prog_name** > <YES>

where:

prog_name : the name of any KAREL or teach pendant program with variables

YES : confirmation is not prompted

Purpose: Clears the variable and type data associated with the specified or default program from memory.

Variables and types that are referenced by a loaded program are not cleared.

Examples: KCL> CLEAR VARS test_prog

Are you sure? YES

KCL> CLEAR VARS test_prog Y

Draft

C.11 COMPRESS DICT command

Syntax: COMPRESS DICT **file_name**

where:

file_name : the file name of the user dictionary you want to compress.

Purpose: Compresses a dictionary file from the default storage device, using the specified dictionary name. The file type of the user dictionary must be “.UTX”. The compressed dictionary file will have the same file name as the user dictionary, and be of type “.TX”.

See Also: Chapter 10 *DICTIONARIES AND FORMS*

Example: KCL> COMPRESS DICT tphcmneg

C.12 COMPRESS FORM command

Syntax: COMPRESS FORM < **file_name** >

where:

file_name : the file name of the form you want to compress.

Purpose: Compresses a form file from the default storage device using the specified form name. The file type of the form must be “.FTX”. A compressed dictionary file and variable file will be created. The compressed dictionary file will have the same file name as the form file and be of type ".TX". The variable file will have a four character file name, that is extracted from the form file name, and be of type “.VR”. If no form file name is specified, the name “FORM” is used.

See Also: Chapter 10 *DICTIONARIES AND FORMS*

Examples: KCL> COMPRESS FORM

KCL> COMPRESS FORM mnpalteg

C.13 CONTINUE command

Syntax: CONTINUE <(**prog_name**) | ALL)>

where:

prog_name : the name of any KAREL or teach pendant program which is a task

ALL : continues all paused tasks

Purpose: Continues program execution of the specified task that has been paused by a hold, pause, or test run operation. If the program is aborted, the program execution is started at the first executable line.

When a task is paused, the CYCLE START button on the operator panel has the same effect as the KCL> CONTINUE command.

CONTINUE is a motion command; therefore, the device from which it is issued must have motion control.

See Also: Refer to the $RMT_MASTER description in the *FANUC Robotics Software Reference Manual* for more information about assigning motion control to a remote device.

Examples: KCL> CONTINUE test_prog

KCL> CONT ALL

Draft

C.14 COPY FILE command

Syntax: COPY <FILE> **from_file_spec** TO **to_file_spec** <OVERWRITE>

where:

from_file_spec : a valid file specification

to_file_spec : a valid file specification

OVERWRITE : specifies copy over (overwrite) an existing file

Purpose: Copies the contents of one file to another with overwrite option. Allows file transfers between different devices and between the controller and a host system.

The wildcard character (*) can be used to replace **from_file_spec** 's entire file name, the first part of the file name, the last part of the file name, or both the first and last parts of the file name. The file

type can also use the wildcard in the same manner. The wildcard character in the **to_file_spec** can only replace the entire file name or the entire file type.

Examples: KCL> COPY flpy:\test.kl TO rdu:newtest.kl

KCL> COPY mc:\test_dir\test.kl TO mc:\test_dir\newtest.kl

KCL> COPY FILE flpy:*.kl TO rd:*.kl

KCL> COPY *.* TO fr:

KCL> COPY FILE *.kl TO rd:*.bak OVERWRITE

KCL> COPY FILE flpy:*main*.kl TO rd:* OV

KCL> COPY mdb:*.tp TO mc:

Draft

C.15 CREATE VARIABLE command

Syntax: CREATE VARIABLE <[**prog_name**]> **var_name <IN (CMOS | DRAM)> : data_type**

where:

prog_name : the name of any KAREL or TP program

var_name:data_type : a valid variable name and data type

Purpose: Allows you to declare a variable that will be associated with the specified or default program. You must specify a valid identifier for the **var_name** and a valid **data_type** .

Only one variable can be declared with the CREATE VAR command. You must enter the KCL> SAVE VARS command to save the declared variable with the program variable data. Use the KCL> SET VARIABLE command to assign a value to a variable.

The following data types are valid (user types are also supported): ARRAY OF BYTE, JOINTPOS, ARRAY OF SHORT, JOINTPOS1 to JOINTPOS9, BOOLEAN, POSITION, REAL, CONFIG, VECTOR, FILE, XYZWPR, XYZWPREXT, INTEGER

You can create multi-dimensional arrays of the above type. A maximum of 3 dimensions may be specified. Paths may only be created from a user defined type.

By default, the variable will be created in temporary memory (in DRAM), and must be recreated every power up. The value will always be reset to uninitialized.

If IN CMOS is specified the variable will be created in permanent memory. The variable's value will be recovered every time the controller is turned on.

See Also: SET VARIABLE command

Examples: KCL> CREATE VAR [test_prog]count IN CMOS: INTEGER

KCL> CREATE VAR vec:ARRAY[3,2,4] OF VECTOR

C.16 DELETE FILE command

Syntax: DELETE FILE **file_spec** <YES>

where:

file_spec : a valid file specification

YES : confirmation is not prompted

Purpose: Deletes the specified file from the specified storage device. The wildcard character (*) can be used to replace the entire file name, the first part of the file name, the last part of the file name, or both the first and last parts of the file name. The file type can also use the wildcard in the same manner.

Examples: KCL> DELETE FILE testprog.pc

Are you sure? YES

KCL> DELETE FILE rd:\testprog.pc YES

KCL> DELETE FILE rd:*.* Y

C.17 DELETE NODE command

Syntax: DELETE NODE <[**prog_name**]> **var_name [node_index**]

where:

prog_name : the name of any KAREL or TP program

var_name : the name of any variable of type PATH

[**node_index**] : a node in the path

Purpose: Deletes the specified node from the specified PATH variable. The PATH variable must be loaded in memory. Enter the KCL> SAVE VARS command to make the change permanent.

Examples: KCL> DELETE NODE [test_prog]weld_pth[4]

KCL> DELETE NODE weld_pth[3]

Draft

C.18 DELETE VARIABLE command

Syntax: DELETE VARIABLE <[**prog_name**]> **var_name**

where:

prog_name : the name of any KAREL or TP program with variables

var_name : the name of any program variable

Purpose: Deletes the specified variable from memory. A variable that is linked with loaded p-code cannot be deleted. Enter the KCL> SAVE VARS command to make the change permanent.

Examples: KCL> DELETE VARIABLE [test_prog]weld_pth

KCL> DELETE VAR weld_pth

Draft

C.19 DIRECTORY command

Syntax: DIRECTORY < **file_spec** >

where:

file_spec : a valid file specification

Purpose: Displays a list of the files that are on a storage device. If **file_spec** is not specified, directory information is displayed for all of the files stored on a specified device. The directory information displayed includes the following:

The volume name of the device (if specified when the device was initialized)

The name of the subdirectory, if available

The names and types of files currently stored on the device and the sizes of the files in bytes

The number of files, the number of bytes left, and the number of bytes total, if available

The wildcard character (*) can be used to replace the entire file name, the first part of the file name, the last part of the file name, or both the first and last parts of the file name. The file type can also use the wildcard in the same manner.

Examples: KCL> DIRECTORY rd:

KCL> DIR *.kl

KCL> DIR *SPOT*.kl

KCL> CD MC: \test_dir

Use the CD command to change to the KCL> DIR subdirectory before you use the DIR commandor KCL> DIR \test_dir*.* display the subdirectory contents without using the CD command.

C.20 DISABLE BREAK PROGRAM command

Syntax: DISABLE BREAK PROGRAM < **prog_name** > **brk_pnt_no**

where:

prog_name : the name of any KAREL or TP program in memory

brk_pnt_no : a particular program break point

Purpose: Disables the specified break point in the specified or default program.

Examples: KCL> DISABLE BREAK PROGRAM test_prog 3

KCL> DISABLE BREAK PROG 3

C.21 DISABLE CONDITION command

Syntax: DISABLE CONDITION < **prog_name** > **condition_no**

where:

prog_name : the name of any KAREL program in memory

condition_no : a particular condition

Purpose: Disables the specified condition in the specified or default program.

Examples: KCL> DISABLE CONDITION test_prog

KCL> DISABLE COND 3

C.22 DISMOUNT command

Syntax: DISMOUNT **device_name:**

where:

device_name : device to be dismounted

Draft

Purpose: Dismounts a mounted storage device and indicates to the controller that a storage device is no longer available for reading or writing data.

Example: KCL> DISMOUNT rd:

C.23 EDIT command

Syntax: EDIT < **file_spec** >

where:

file_spec : a valid file specification

Purpose: Provides an ASCII text editor which can be used for editing dictionary files, command files and KAREL source files.

If **file_spec** is not specified, the default program name is used as the file name and the default file type is .KL (KAREL source code).

If a previous editing session exists, then **file_spec** is ignored and the editing session is resumed.

Examples: KCL> EDIT startup.cf

KCL> ED

C.24 ENABLE BREAK PROGRAM

Syntax: ENABLE BREAK PROGRAM < **prog_name** > **brk_pnt_no**

where:

prog_name : the name of any KAREL or TP program in memory

brk_pnt_no : a particular program break point

Purpose: Enables the specified break point in the specified or default program.

Examples: KCL> ENABLE BREAK PROGRAM test_prog 3

KCL> ENABLE BREAK PROG 3

C.25 ENABLE CONDITION command

Syntax: ENABLE CONDITION < **prog_name** > **condition_no**

where:

prog_name : the name of any KAREL program in memory

condition_no : a particular condition

Purpose: Enables the specified condition in the specified or default program.

Examples: KCL> ENABLE CONDITION test_prog

KCL> ENABLE COND 3

C.26 FORMAT command

Syntax: FORMAT **device_name:** < **volume_name** > <YES>

where:

device_name : the specified device to be initialized

volume_name : label for the device

YES : confirmation is not prompted

Purpose: Formats a specified device. A device must be formatted before storing files on it.

Examples: KCL> FORMAT rd:

Are you sure? YES

KCL> FORMAT rd: Y

C.27 HELP command

Syntax: HELP < **command_name** >

where:

command_name : a KCL command

Purpose: Displays on-line help for KCL commands. If you specify a **command_name** argument, the required syntax and a brief description of the specified command is displayed.

Examples: KCL> HELP LOAD PROG

KCL> HELP

Draft

C.28 HOLD command

Syntax: HOLD <(**prog_name** | ALL)>

where:

prog_name : the name of any KAREL or TP program

ALL : holds all executing programs

Purpose: Pauses the specified or default program that is being executed and holds motion at the current position (after a normal deceleration).

Use the KCL> CONTINUE command or the CYCLE START button on the operator panel to resume program execution.

Examples: KCL> HOLD test_prog

KCL> HO ALL

Draft

C.29 INSERT NODE command

Syntax: INSERT NODE <[**prog_name**]> **var_name [node_index]**

where:

prog_name : the name of any KAREL or TP program

var_name : the name of any variable of type PATH

[node_index] : a node in the path

Purpose: Inserts a node in front of the specified node in the PATH variable. The PATH variable must be loaded in memory.

The inserted node index number is the **node_index** you specify and the inserted node value is uninitialized. The index numbers for subsequent nodes are incremented by one. You must enter the KCL> SAVE VARS command to make the change permanent.

Examples: KCL> INSERT NODE [test_prog]weld_pth[2]

KCL> INSERT NODE weld_pth[3]

C.30 LOAD ALL command

Syntax: LOAD ALL < **file_name** > <CONVERT>

where:

file_name : a valid file name

CONVERT : converts variables to system definition

Purpose: Loads a p-code and variable file from the default storage device and default directory into memory using the specified or default file name. The file types for the p-code and variable files are assumed to be ".PC" and ".VR" respectively.

If **file_name** is not specified, the default program is used. If the default has not been set, then the message, "Default program name not set," will be displayed.

Examples: KCL> LOAD ALL test_prog

KCL> LOAD ALL

C.31 LOAD DICT command

Syntax: LOAD DICT **file_name dict_name** < **lang_name** >

where:

file_name : the name of the file to be loaded

dict_name : the name of any dictionary to be loaded. The name will be truncated to 4 characters.

lang_name : a particular language. The available choices are ENGLISH, JAPANESE, FRENCH, GERMAN, SPANISH or DEFAULT.

Purpose: Loads a dictionary file from the default storage device and default directory into memory using the specified file name. The file type is assumed to be ".TX."

See Also: Chapter 10 *DICTIONARIES AND FORMS*

Examples: KCL> LOAD DICT tpaleg tpal FRENCH

KCL> LOAD DICT tpaleg tpal

C.32 LOAD FORM command

Syntax: LOAD FORM < **form_name** >

where:

form_name : the name of the form to be loaded

Purpose: Loads the specified form, from the default storage device, into memory. A form consists of a compressed dictionary file and a variable file. If no name is specified, 'FORM.TX' and 'FORM.VR' are loaded.

If the specified **form_name** is greater than four characters, the first two characters are not used for the dictionary name or the variable file name.

See Also: For more information on creating and using forms, refer to Chapter 10 *DICTIONARIES AND FORMS*

Example: KCL> LOAD FORM

Loading FORM.TX with dictionary name FORM

Loading FORM.VR

KCL> LOAD FORM tpexameg

Loading TPEXAMEG.TX with dictionary name EXAM

Loading EXAM.VR

Draft

C.33 LOAD MASTER command

Syntax: LOAD MASTER < **file_name** > <CONVERT>

where:

file_name : a valid file name

CONVERT : converts variables to system definition

Purpose: Loads a mastering data file from the default storage device and default directory into memory using the specified or default file name. The file type is assumed to be ".SV."

If **file_name** is not specified, the default file name, "SYSMAST.SV," is used.

Example: KCL> LOAD MASTER

C.34 LOAD PROGRAM command

Syntax: LOAD PROGRAM < **file_name** >

where:

file_name : a valid file name

Purpose: Loads a p-code file from the default storage device and default directory into memory using the specified or default file name. The file type is assumed to be “.PC.”

If **file_spec** is not specified, the default program is used. If the default has not been set, then the message, “Default program name not set,” will be displayed.

The following note applies to R-30iB controllers:

Note The KAREL option must be installed on the robot controller in order to load KAREL programs.

Examples: KCL> LOAD PROGRAM test_prog

KCL> LOAD PROG

C.35 LOAD SERVO command

Syntax: LOAD SERVO < **file_name** > <CONVERT>

where:

file_name : a valid file name

CONVERT : converts variables to system definition

Purpose: Loads a servo parameter file from the default storage device and default directory into memory using the specified or default file name. The file type is assumed to be “.SV.”

If **file_name** is not specified, the default file name, “SYSSERVO.SV,” is used.

Example: KCL> LOAD SERVO

C.36 LOAD SYSTEM command

Syntax: LOAD SYSTEM < **file_name** > <CONVERT>

where:

file_name : a valid file name

CONVERT : converts variables to system definition

Draft

Purpose: Loads the specified system variable file into memory, assigning values to all of the saved system variables. The default storage device and default directory are used with the specified or default file name. The file type is assumed to be ‘‘.SV.’’

If **file_name** is not specified, the default file name, ‘‘SYSVARS.SV,’’ is used.

Examples: KCL> LOAD SYSTEM awdef

KCL> LOAD SYSTEM CONVERT

The following rules are applicable for system variables:

- If an array system variable that is not referenced by a program already exists when a .SV file is loaded, the size in the .SV file is used and the contents are loaded. No errors are posted.
- If an array system variable that is referenced by a program already exists when a .SV file with a LARGER size is loaded, the size in the .SV file is ignored, and NONE of the array values are loaded. The following errors are posted; " *var_name* memory not updated", "Array len creation mismatch".
- If an array system variable that is referenced by a program already exists when a .SV file with a SMALLER size is loaded, the size in the .SV file is ignored but ALL the array values are loaded. No errors are posted.
- If a .SV file with a different type definition is loaded, the .SV file will stop loading and detect the error. The following errors are posted; "Create type - *var_name* failed", "Duplicate creation mismatch".
- If a .SV. file with a different type definition is loaded, but the CONVERT option is specified, it tries to load as much as it can. For example, the controller has a SCR_T type which has the field $NEW but not the field $OLD. When an old .SV file is loaded that has $OLD but not $NEW, the load procedure creates the SCR_T type based on what is in the .SV file and posts a "Create type - var_name failed", "Duplicate creation mismatch" error. It then creates the type SCR_! which has the field $OLD but not the field $NEW. It then does a field by field copy of all of the old valid fields into the new type. Therefore, since there is not $NEW information in the old type that field is not updated and the $OLD information is discarded. Any fields whose types don’t match are discarded from the loaded type. So if a field was changed from integer to real, the integer field in the loaded data would be discarded. Any fields that are arrays will follow the same rules as array system variables.

C.37 LOAD TP command

Syntax: LOAD TP < **file_name** > <OVERWRITE>

where:

file_name : a valid file name

OVERWRITE : If specified, may overwrite a previously loaded TP program with the same name

Draft

Purpose: Loads a TP program from the default storage device and default directory into memory using the specified or default file name. The file type is assumed to be ‘‘.TP.’’

If **file_name** is not specified, the default program is used. If the default has not been set, then the message, ‘‘Default program name not set,’’ will be displayed.

Examples: KCL> LOAD TP testprog

KCL> LOAD TP

C.38 LOAD VARS command

Syntax: LOAD VARS < **file_name** > <CONVERT>

where:

file_name : a valid file name

CONVERT : converts variables to system definition

Purpose: Loads the specified or default variable data file from the default storage device and directory into memory. The file type is assumed to be ‘‘.VR.’’

If **file_name** is not specified, the default program is used. If the default has not been set then the message, ‘‘Default program name not set,’’ will be displayed.

Examples: KCL> LOAD VARS test_prog

KCL> LOAD VARS

The following rules are applicable for array variables:

- If an array variable that is not referenced by a program already exists when a .VR file is loaded, the size in the .VR file is used and the contents are loaded. No errors are posted.
- If an array variable already exists when a program is loaded, the size in the .PC file is ignored and the program is loaded anyway. The following errors are posted: " *var_name* PC array length ignored", and "Array len creation mismatch".
- If an array variable that is referenced by a program already exists when a .VR file with a LARGER size is loaded, the size in the .VR file is ignored and NONE of the array values are loaded. The following errors are posted; " *var_name* memory not updated," "Array len creation mismatch."
- If an array variable that is referenced by a program already exists when a .VR file with a SMALLER size is loaded, the size in the .VR file is ignored but ALL the array values are loaded. The following errors are posted; " *var_name* array length updated," "Array len creation mismatch."

The following rules are applicable for user-defined types in KAREL programs:

Draft

- Once a type is created it can never be changed, regardless of whether a program references it or not. If all the variables referencing the type are deleted, the type will also be deleted. A new version can then be loaded.
- If a type already exists when a program with a different type definition is loaded, the .PC file will not be loaded. The following errors are posted; "Create type - *var_name* failed," "Duplicate creation mismatch."
- If a type already exists when a .VR file with a different type definition is loaded, the .VR file will stop loading when it detects the error. The following errors are posted; "Create type - *var_name* failed," "Duplicate creation mismatch".

Draft

C.39 LOGOUT command

Syntax: LOGOUT

Purpose: Logs the current user from the KCL device out of the system. The password level reverts to the OPERATOR level. If passwords are not enabled, an error message will be displayed by KCL such as, "No user currently logged in".

Example: KCL>LOGOUT

(The alarm message: "Logout (SAM) SETUP from KCL")

KCL Username>

C.40 MKDIR command

Syntax: MKDIR **<device_name>\path_name**

where:

device_name : a valid storage device

path_name : a subdirectory previously created on the memory card device using the mkdir command.

Purpose: MKDIR creates a subdirectory on the memory card (MC:) device. FANUC Robotics recommends you nest subdirectories only to 8 levels.

Example: KCL> MKDIR mc:\test_dir

KCL> MKDIR mc:\prog_dir\tpnx_dir

C.41 MOUNT command

Syntax: MOUNT **device_name**

where:

device_name : a valid storage device

Purpose: MOUNT indicates to the controller that a storage device is available for reading or writing data.

A device must be formatted with the KCL> FORMAT command before it can be mounted successfully.

Example: KCL> MOUNT rd:

C.42 MOVE FILE command

Syntax: MOVE <FILE> **file_spec**

where:

file_spec : a valid file specification.

Purpose: Moves the specified file from one memory file device to another. The file should exist on the FROM or RAM disks. If file_spec is a file on the FROM disk, the file is moved to the RAM disk, and vice versa.

The wildcard character (*) can be used to replace the entire file name, the first part of the file name, the last part of the file name, or both the first and last parts of the file name. The file type can also use the wildcard in the same manner. If file_spec specifies multiple files, then they are all moved to the other disk.

Examples: KCL> MOVE FILE fr:*.kl

KCL> MOVE rd:*.*

C.43 PAUSE command

Syntax: PAUSE <(**prog_name** | ALL)> <FORCE>

where:

prog_name : the name of any KAREL or TP program which is a task

ALL : pauses all running tasks

FORCE : pauses the task even if the NOPAUSE attribute is set

Purpose: Pauses the specified running task. If **prog_name** is not specified, the default program is used.

Draft

Execution of the current motion segment and the current program statement is completed before the task is paused.

Condition handlers remain active. If the condition handler action is NOPAUSE and the condition is satisfied, task execution resumes.

If the statement is a WAIT FOR and the wait condition is satisfied while the task is paused, the statement following the WAIT FOR is executed immediately when the task is resumed.

If the statement is a DELAY, timing will continue while the task is paused. If the delay time is finished while the task is paused, the statement following the DELAY is immediately executed when the task is resumed. If the statement is a READ, it will accept input even though the task is paused.

The KCL> CONTINUE command resumes execution of a paused task. When a task is paused, the CYCLE START button on the operator panel has the same effect as the KCL> CONTINUE command.

Examples: KCL> PAUSE test_prog FORCE

KCL> PAUSE ALL

Draft

C.44 PURGE command

Syntax: PURGE **device_name**

where:

device_name : the name of the memory file device to be purged

Purpose: Purges the specified memory file device by freeing any used blocks that are no longer needed. The device should be set to "FR:" for FROM disk, "RD:" for RAM disk, or "MF:" for both disks.

The purge operation is only necessary when the device does not have enough memory to perform an operation. The purge operation will erase file blocks that were previously used, but are no longer needed. These are called garbage blocks. The FROM disk can contain many garbage blocks if files are deleted or overwritten. The RAM disk will not normally contain garbage blocks, but they may occur when power is removed during a file copy.

The device must be mounted and no files can be open on the device or an error will be displayed.

Examples: KCL> PURGE fr:

KCL> PURGE mf:

C.45 PRINT command

Syntax: PRINT **file_spec**

where:

file_spec : a valid file specification

Purpose: Allows you to print the contents of an ASCII file to the default device.

Example: KCL> PRINT testprog.kl

C.46 RECORD command

Syntax: RECORD <[**prog_name**]> **var_name**

where:

prog_name : the name of any KAREL or TP program

var_name : the name of any POSITION, XYZWPR, or JOINTPOS variable

Purpose: Records the position of the TCP and/or auxiliary or extended axes. The robot must be calibrated before the RECORD command is issued. The variable can be a system variable or a program variable that exists in memory. The position is recorded relative to the user frame of reference.

You must enter the KCL> SAVE command to permanently assign the recorded position. The Record function key, F3, under the teach pendant TEACH menu also allows you to record positions.

Example: KCL> RECORD [paint_prog]start_pos

KCL> RECORD $GROUP[1].$uframe

C.47 RENAME FILE command

Syntax: RENAME FILE **old_file_spec** TO **new_file_ spec**

where:

old_file_spec : a valid file specification

new_file_spec : a valid file specification

Purpose: Changes the **old_file_spec** to the **new_file_spec** . The file will no longer exist under the **old_file_spec** . The **old_file_spec** and the **new_file_spec** must include both the file name and the file type. The same file type must be used in both file_specs but they cannot be the same file.

Use the KCL> COPY FILE command to change the device name of a file.

Examples: KCL> RENAME FILE test.kl TO productn.kl

Draft

KCL> RENAME FILE mycmd.cf TO yourcmd.cf

C.48 RENAME VARIABLE command

Syntax: RENAME VARIABLE <[**prog_name**]> **old_var_ name new_var_name**

where:

prog_name : the name of any KAREL or TP program

old_var_name : the name of any program variable

new_var_name : a valid program variable name

Purpose: Changes the **old_var_name** to the **new_var_name** in the program specified with the **old_var_name** . The variable will no longer exist under the **old_var_name** . The variable must exist in memory under the **old_var_name** in the specified program.

The **new_var_name** cannot already exist in memory. The variable still belongs to the same program. You cannot specify a **prog_name** with the **new_var_name.**

You must enter the KCL> SAVE VARS command to make the change permanent.

Examples: KCL> RENAME VARIABLE [test_prog]count part_count

KCL> RENAME VAR count part_count

C.49 RENAME VARS command

Syntax: RENAME VARS **old_prog_name new_prog_name**

where:

old_prog_name : the name of any KAREL or TP program

new_prog_name : the name of any KAREL or TP program

Purpose: Changes the name of the variable data associated with the **old_prog_name** to the **new_prog_name** . The variable data will no longer exist under the **old_prog_name** .

Before you use the RENAME VARS command, the variable data must exist in memory under the **old_prog_name** . Variable data cannot already exist in memory under the **new_prog_name** .

The command does not rename the program. To rename a KAREL program, use the KCL> RENAME FILE to rename the .KL file, edit the program name in the .KL file, translate the program, and load the new C file. To rename a TP program, use the SELECT menu.

Draft

You must enter the KCL> SAVE VARS command to make the change permanent.

Example: KCL> RENAME VARS test_1 test_2

Use this sequence of KCL commands to copy the variable data of one program (**prog_1**) into a variable file that is then used by another program (**prog_2**):

LOAD VARS **prog_1**

RENAME VARS **prog_1 prog_2**

SAVE VARS **prog_2**

LOAD ALL **prog_2**

The effect of this sequence of commands cannot be accomplished with the KCL> COPY FILE command.

The name of the program to which the variable data belongs is stored in the variable file. The KCL> COPY FILE command does not change that stored program name, so the data cannot be used with another program.

Draft

C.50 RESET command

Syntax: RESET

Purpose: Enables servo power after an error condition has shut off servo power, provided the cause of the error has been cleared. The command also clears the message line on the CRT/KB display. The error message remains displayed if the error condition still exists.

The RESET command has no effect on a program that is being executed. It has the same effect as the FAULT RESET button on the operator panel and the RESET function key on the teach pendant RESET screen.

Example: KCL> RESET

C.51 RMDIR command

Syntax: RMDIR **<device_name>\path_name**

where:

device_name : a valid storage device

path_name : a subdirectory previously created on the memory card device using the mkdir command.

Purpose: RMDIR deletes a subdirectory on the memory card (MC:) device. The directory must be empty before it can be deleted.

Example: KCL> RMDIR mc:\test_dir

KCL> RMDIR mc:\test_dir\prog_dir

C.52 RUN command

Syntax: RUN < **prog_name** >

where:

prog_name :the name of any KAREL or TP program

Purpose: Executes the specified program. The program must be loaded in memory. If no program is specified the default program is run. If uninitialized variables are encountered, program execution is paused.

Execution begins at the first executable line. RUN is a motion command; therefore, the device from which it is issued must have motion control. If a RUN command is issued in a command file, it is executed as a NOWAIT command. Therefore, the statement following the RUN command will be executed immediately after the RUN command is issued without waiting for the program, specified by the RUN command, to end.

See Also: Refer to the $RMT_MASTER description in the *FANUC Robotics Software Reference Manual* for more information about assigning motion control to a remote device.

Example: KCL> RUN test_prog

C.53 RUNCF command

Syntax: RUNCF **input_file_spec** < **output_file_spec** >

where:

input_file_spec : a valid file specification

output_file_spec : a valid file specification

Purpose: Executes the KCL command procedure that is stored in the specified input file and displays the output to the specified output file. The input file type is assumed to be .CF. The output file type is assumed to be .LS if no file type is supplied.

If **output_file_spec** is not specified, the output will be displayed to the KCL output window.

Draft

The RUNCF command can be nested within command files up to four levels. Use **%INCLUDE input_file_spec** to include another .CF file into the command procedure. RUNCF command itself is not allowed inside a command procedure.

If the command file contains motion commands, the device from which the RUNCF command is issued must have motion control.

See Also: Refer to Section 13.4 , “Command Procedures,” for more information

Examples: KCL> RUNCF startup output

KCL> RUNCF startup

C.54 SAVE MASTER command

Syntax: SAVE MASTER < **file_name** >

where:

file_name : a valid file name

Purpose: Saves the mastering data file from the default storage device and default directory into memory using the specified or default file name. The file type will be “.SV.”

If **file_name** is not specified, the default file name, “SYSMAST.SV,” is used.

Example: KCL> SAVE MASTER

C.55 SAVE SERVO command

Syntax: SAVE SERVO < **file_name** >

where:

file_name :a valid file name

Purpose: Saves the servo parameters into the default storage device using the specified or default file name. The file type will be “.SV.”

If **file_name** is not specified, the default file name, “SYSSERVO.SV,” is used.

Example: KCL> SAVE SERVO

C.56 SAVE SYSTEM command

Syntax: SAVE SYSTEM < **file_name** >

Draft

where:

file_name :a valid file name

Purpose: Saves the system variable values into the default storage device and default directory using the specified system variable file (.SV). If you do not specify a **file_spec** the default name, ‘‘SYSVARS.SV,’’is used. For example:

SAVE SYSTEM **file_1**

In this case, the system variable data is saved in a variable file called **file_1.SV** .

SAVE SYSTEM

In this case, the system variable data is saved in a system variable file ‘‘SYSVARS.SV.’’

Examples: KCL> SAVE SYSTEM file_1

KCL> SAVE SYSTEM

Draft

C.57 SAVE TP command

Syntax: SAVE TP < **file_name** > <= **prog_name** >

where:

file_name : a valid file name

prog_name : the name of any TP program

Purpose: Saves the specified TP program to the specified file (.TP). If you do not specify a **file_name** or a **prog_name** , the default program name is used. If only a **file_name** is specified, that name will also be used for **prog_name** . For example:

SAVE TP **file_1**

In this case, the TP program **file_1** is saved in a file called **file_1.TP** .

SAVE TP = **prog_1**

In this case, the TP program **prog_1** is saved in a file whose name is the default program name.

If you specify a program name, it must be preceded by an equal sign (=).

Examples: KCL> SAVE TP file_1 = prog_1

KCL> SAVE TP file_1

KCL> SAVE TP = prog_1

KCL> SAVE TP

C.58 SAVE VARS command

Syntax: SAVE VARS < **file_name** > <= **prog_name** >

where:

file_name : a valid file name

prog_name : the name of any KAREL or TP program

Purpose: Saves variable data from the specified program, including the currently assigned values, to the specified variable file (.VR). If you do not specify a **file_name** or a **prog_name** , the default program name is used. If only a **file_name** is specified, that name will also be used for **prog_name** . For example:

SAVE VARS **file_1**

In this case, the variable data for the program **file_1** is saved in a variable file called **file_1.VR.**

SAVE VARS = **prog_** 1

In this case, the variable data for **prog_** 1 is saved in a variable file whose name is the default program name.

If you specify a program name, it must be preceded by an equal sign (=).

Any variable data that is not saved is lost when an initial start of the controller is performed.

Examples: KCL> SAVE VARS file_1 = prog_1

KCL> SAVE VARS file_1

KCL> SAVE VARS = prog_1

KCL> SAVE VARS

C.59 SET BREAK CONDITION command

Syntax: SET BREAK CONDITION < **prog_name** > **condition_no**

where:

prog_name : the name of any running or paused KAREL program

Draft

condition_no : a particular condition

Purpose: Allows you to set a break point on the specified condition in the specified program or default program. The specified condition must already exist so the program must be running or paused. When the break point is triggered, a message will be posted to the error log and the break point will be cleared.

Examples: KCL> SET BREAK CONDITION test_prog 1

KCL> SET BREAK COND 2

C.60 SET BREAK PROGRAM command

Syntax: SET BREAK PROGRAM < **prog_name** > **brk_pnt_no line_no** <(PAUSE | DISPLAY | TRACE ON | TRACE OFF)>

where:

prog_name : the name of any KAREL or TP program in memory

brk_pnt_no : a particular program break point

line_no : a line number

PAUSE : task is paused when break point is executed

DISPLAY : message is displayed on the teach pendant USER menu when break point is executed

TRACE ON : trace is enabled when break point is executed

TRACE OFF : trace is disabled when break point is executed

Purpose: Allows you to set a break point at a specified line in the specified or default program. The specified line must be an executable line of source code. Break points will be executed before the specified line in the program. By default the task will pause when the break point is executed. DISPLAY, TRACE ON, and TRACE OFF will not pause task execution.

Break points are local only to the program in which the break points were set. For example, break point #1 can exist among one or more loaded programs with each at a unique line number. If you specify an existing break point number, the existing break point is cleared and a new one is set in the specified program at the specified line.

Break points in a program are cleared if the program is cleared from memory. You also use the KCL> CLEAR BREAK PROGRAM command to clear break points from memory.

Use the KCL> CONTINUE command or the operator panel CYCLE START button to resume execution of a paused program.

Draft

Examples: KCL> SET BREAK PROGRAM test_prog 1 22 DISPLAY

KCL> SET BREAK PROG 3 30

C.61 SET CLOCK command

Syntax: SET CLOCK **'dd-mmm-yy hh:mm'**

where:

The date is specified using two numeric characters for the day, a three letter abbreviation for the month, and two numeric characters for the year; for example, 01-JAN- 00.

The time is specified using two numeric characters for the hour and two numeric characters for the minutes; for example, 12:45.

Purpose: Sets the date and time of the internal controller clock.

The date and time are included in directory and translator listings.

See Also: SHOW CLOCK command

Example: KCL> SET CLOCK '02-JAN-xx 21:45'

Draft

C.62 SET DEFAULT command

Syntax: SET DEFAULT **prog_name**

where:

prog_name : the name of any KAREL or TP program

Purpose: Sets the default program name to be used as an argument default for program and file names. The default program name can also be set at the teach pendant.

See Also: Section 13.1.1 , "Default Program"

Examples: KCL> SET DEFAULT test_prog

KCL> SET DEF test_prog

C.63 SET GROUP command

Syntax: SET GROUP **group_no**

where:

group_no : a valid group number

Purpose: Sets the default group number to use in other commands.

Example: KCL> SET GROUP 1

C.64 SET LANGUAGE command

Syntax: SET LANGUAGE **lang_name**

where:

lang_name : a particular language. The available choices are ENGLISH, JAPANESE, FRENCH, GERMAN, SPANISH or DEFAULT.

Purpose: Sets the $LANGUAGE system variable which determines the language to use.

Example: KCL> SET LANG ENGLISH

C.65 SET LOCAL VARIABLE command

Syntax: SET LOCAL VARIABLE **var_name** <IN **rout_name** > <FROM **prog_name** > < **task_name** > = **value** <{, **value** }>

where:

var_name :a local variable or parameter name

rout_name :the name of any KAREL routine

prog_name :the name of any KAREL program

task_name : the name of any KAREL task

value : new value for variable

Purpose: Assigns the specified value to the specified local variable or routine parameter. You can assign constant values or variable values, but the value must be of the data type that has been declared for the variable.

Please use the HELP SET VAR command for more information on assigning data types.

If the IN clause is omitted, the routine at the top of the stack is assumed. If the FROM clause is omitted, the default program is assumed. If the **task_name** is omitted, the stack of the KCL default task is searched.

Note The file RD: **prog_name.rs** is required to obtain local variable information.

Draft

Example: See SHOW LOCAL VARIABLE command.

See Also: SHOW LOCAL VARIABLE and TRANSLATE command.

C.66 SET PORT command

Syntax: SET PORT **port_name [index] = value**

where:

port_name[index] : a valid I/O port **value** : a new value for the port

Purpose: Assigns the specified value to a specified input or output port. SET PORT can be used with either physical or simulated output ports, but only with simulated input ports.

The valid ports are:

DIN, DOUT, RDO, OPOUT, TPOUT, WDI, WDO (BOOLEAN)-AIN, AOUT, GIN, GOUT (INTEGER)

See Also: SIMULATE, UNSIMULATE command, Chapter 14 *INPUT/OUTPUT SYSTEM* , application-specific FANUC Robotics Setup and Operations Manual.

Example: KCL> SET PORT DOUT [1] = ON

KCL> SET PORT GOUT [2] = 255

KCL> SET PORT AIN [1] = 1000

C.67 SET TASK command

Syntax: SET TASK <[**prog_name**]> **attr_name** = **value**

where:

prog_name : the name of any KAREL or TP program which is a task

attr_name : PRIORITY or TRACELEN

value : new integer value for attribute

Purpose: Sets the specified task attribute. PRIORITY sets the task priority. The lower the number, the higher the priority. 1 to 89 is lower than motion, but higher than the user interface. 90 to 99 is lower than the user interface. The default is 50. TRACELEN sets the trace buffer length. The default is 10 lines.

Draft

C.68 SET TRACE command

Syntax: SET TRACE (OFF | ON) <[**prog_name**]>

where:

prog_name : the name of any KAREL or TP program loaded in memory

Purpose: Turns the trace function ON or OFF (default is OFF). The program statement currently being executed and its line number are stored in a buffer when TRACE is ON. TRACE should only be set to ON during debugging operations because it slows program execution. To see the trace data, SHOW TRACE command must be used.

See Also: SHOW TRACE command

C.69 SET VARIABLE command

Syntax: SET VARIABLE <[**prog_name**]> **var_name** = **value** <{, **value** }>

where:

prog_name : the name of any KAREL or TP program

var_name : a valid program variable

value : new value for variable or a program or system variable

Purpose: Assigns the specified value to the specified variable. You can assign constant values or variable values, but the value must be of the data type that has been declared for the variable.

You can assign values to system variables with KCL write access, to program variables, or to standard and user-defined variables and fields. You can assign only one ARRAY element. Use brackets ([]) after the variable name to specify an element.

Certain data types like positions and vectors might have more than one value specified.

KCL> SET VAR position_var = 0,0,0,0,0,0

The SET VARIABLE command displays the previous value of the specified variable followed by the value which you have just assigned, providing you with an opportunity to check the assignment. The DATA key on the teach pendant also allows you to assign values to variables.

When you use SET VARIABLE to define a position you can use one of the following formats:

KCL> SET VARIABLE var_name.X = value

KCL> SET VARIABLE var_name.Y = value

Draft

KCL> SET VARIABLE var_name.Z = value

KCL> SET VARIABLE var_name.W = value

KCL> SET VARIABLE var_name = value

where X,Y,Z,W,P, and R specify the location and orientation, c_str is a string value representing configuration in terms of joint placement and turn numbers. Refer to Section 8.1, “Positional Data.” For example, to set X=200.0, W=60.0 and the turn numbers for axes 4 and 6 to 1 and 0 you would type the following lines:

KCL> SET VARIABLE var_name.X = 200

KCL> SET VARIABLE var_name.W = 60

KCL> SET VARIABLE var_name.C = ’1,0’

You must enter the KCL>SAVE VARS command to make the changes permanent.

See Also: Section 2.3 , “Data Types”

Examples: KCL> SET VARIABLE [prog1] scale = $MCR.$GENOVERRIDE

KCL> SET VAR weld_pgm.angle = 45.0

KCL> SET VAR v[2,1,3].r = -0.897

KCL> SET VAR part_array[2] = part_array[1]

KCL> SET VAR weld_pos.x = 50.0

KCL> SET VAR pth_b[3].nodepos = pth_a[3].nodepos

C.70 SET VERIFY command

Syntax: SET VERIFY (ON | OFF)

Purpose: This turns the display of KCL commands ON or OFF during execution of a KCL command procedure (default is ON, meaning each command is displayed as it is executed). Only the RUNCF command is displayed when VERIFY is OFF.

C.71 SHOW BREAK command

Syntax: SHOW BREAK < **prog_name** >

where:

Draft

prog_name : the name of any KAREL or TP program in memory

Purpose: Displays a list of program break points for the specified or default program. The following information is displayed for each break point:

- Break point number
- Line number of the break point in the program

Examples: KCL> SHOW BREAK test_prog

KCL> SH BREAK

C.72 SHOW BUILTINS command

Syntax: SHOW BUILTINS

Purpose: Displays all the softpart built-ins that are loaded on the controller.

Example: KCL> SHOW BUILTINS

C.73 SHOW CONDITION command

Syntax: SHOW CONDITION < **prog_name** > < **condition_no** >

where:

prog_name : the name of any running or paused KAREL program

condition_no : a particular condition

Purpose: Displays the specified condition handler or a list of condition handlers for the specified or default program. Also displays enabled/disabled status and whether a break point is set. Condition handlers only exist when a program is running or paused.

Examples: KCL> SHOW CONDITION test_prog

KCL> SH COND

C.74 SHOW CLOCK command

Syntax: SHOW CLOCK

Purpose: Displays the current date and time of the controller clock.

See Also: SET CLOCK command

Draft

Example: KCL> SHOW CLOCK

C.75 SHOW CURPOS command

Syntax: SHOW CURPOS

Purpose: Displays the position of the TCP relative to the current user frame of reference with an x, y, and z location in millimeters; w, p, and r orientation in degrees; and the current configuration string. Be sure the robot is calibrated.

Example: KCL> SHOW CURPOS

C.76 SHOW DEFAULT command

Syntax: SHOW DEFAULT

Purpose: Shows the current default program name.

Example: KCL> SHOW DEFAULT

C.77 SHOW DEVICE command

Syntax: SHOW DEVICE **device_name:**

where:

device_name : device to be shown

Purpose: Shows the status of the device.

Example: KCL> SHOW DEVICE rd:

C.78 SHOW DICTS command

Syntax: SHOW DICTS

Purpose: Shows the dictionaries loaded in the system for the language specified in the system variable $LANGUAGE.

Example: KCL> SHOW DICTS

C.79 SHOW GROUP command

Syntax: SHOW GROUP

Draft

Purpose: Shows the default group number.

Example: KCL> SHOW GROUP

C.80 SHOW HISTORY command

Syntax: SHOW HISTORY

Purpose: Shows the nesting information of the routine calls. To display the source lines of KAREL programs, the .KL programs must exist on the RAM disk.

Example: KCL> SHOW HIST

C.81 SHOW LANG command

Syntax: SHOW LANG

Purpose: Shows the language specified in the system variable $LANGUAGE.

Example: KCL> SHOW LANG

C.82 SHOW LANGS command

Syntax: SHOW LANGS

Purpose: Shows all language currently available in the system.

Example: KCL> SHOW LANGS

C.83 SHOW LOCAL VARIABLE command

Syntax: SHOW LOCAL VARIABLE **var_name** <(HEXADECIMAL | BINARY)> <IN **rout_name** > <FROM **prog_name** > < **task_name** >

where:

var_name : a local variable or parameter name

rout_name : the name of any KAREL routine

prog_name : the name of any KAREL program

task_name : the name of any KAREL task

Draft

Purpose: Displays the name, type, and value of the specified local variable or routine parameter. Use brackets ([ ]) after the variable name to specify a specific ARRAY element. If you do not specify a specific element the entire variable is displayed.

If the IN clause is omitted, the routine at the top of the stack is assumed. If the FROM clause is omitted, the default program is assumed. If the **task_name** is omitted, the stack of the KCL default task is searched.

Note The file RD: **prog_name.rs** is required to obtain local variable information.

Example: Generate a .rs file from the KAREL translator.

KCL> TRANS testprog RS

Copy the .rs file to the RD device.

This is done automatically when you load the program from the KCL.

KCL> SET DEF testprog

KCL> LOAD PROG

Copied testprog.rs to RD:testprog.rs

To show local variables, the program must be running, paused, or aborted in the routine specified.

KCL> RUN

KCL> SHOW LOCAL VARS

KCL> SHOW LOCAL VARS IN testprog VALUES

KCL> SHOW LOCAL VAR var_1 IN rout_1 FROM testprog testtask

KCL> SHOW LOCAL VAR param_1

To set local variables, the program must be paused.

KCL> pause

KCL> set local var int_var = 12345

KCL> set local var strparam = "This is a string parameter"

See Also: TRANSLATE command.

Draft

C.84 SHOW LOCAL VARS command

Syntax: SHOW LOCAL VARS <VALUES> <IN **rout_name** > <FROM **prog_name** > < **task_name** >

where:

VALUES: :specifies values should be displayed

rout_name :the name of any KAREL routine

prog_name :the name of any KAREL program

task_name :the name of any KAREL task

Purpose: Displays a list including the name, type, and if specified, the current value of each local variable and each routine parameter.

If the IN clause is omitted, the routine at the top of the stack is assumed. If the FROM clause is omitted, the default program is assumed. If the **task_name** is omitted, the stack of the KCL default task is searched

Note The file RD: **prog_name.rs** is required to obtain local variable information.

Example: See SHOW LOCAL VARIABLE command.

See Also: TRANSLATE command and SHOW LOCAL VARIABLE.

Draft

C.85 SHOW MEMORY command

Syntax: SHOW MEMORY

Purpose: Displays current memory status. The command displays the following status information for memory and lists each memory pool separately:

Total number of bytes in the pool

Available number of bytes in the pool

Example: KCL> SHOW MEMORY

C.86 SHOW PROGRAM command

Syntax: SHOW PROGRAM < **prog_name** >

where:

prog_name : the name of any KAREL or TP program in memory

Purpose: Displays the status information of the specified or default program being executed.

Example: KCL> SHOW PROGRAM test_prog

KCL> SH PROG

Draft

C.87 SHOW PROGRAMS command

Syntax: SHOW PROGRAMS

Purpose: Shows a list of programs and variable data that are currently loaded in memory.

Examples: KCL> SHOW PROGRAMS

KCL> SH PROGS

C.88 SHOW SYSTEM command

Syntax: SHOW SYSTEM < **data_type** > <VALUES>

where:

data_type : any valid KAREL data type

Purpose: Displays a list including the name, type, and if specified, the current value of each system variable. If you specify a **data_type** , only the system variables of that type are listed.

See Also: SHOW VARIABLE command

Examples: KCL> SHOW SYSTEM REAL VALUES

KCL> SH SYS

C.89 SHOW TASK command

Syntax: SHOW TASK < **prog_name** >

where:

prog_name : the name of any KAREL or TP program which is a task

Purpose: Displays the task control data for the specified task. If **prog_name** is not specified, the default program is used.

Examples: KCL> SHOW TASK test_prog

KCL> SH TASK

C.90 SHOW TASKS command

Syntax: SHOW TASKS

Purpose: Displays the status of all known tasks running KAREL programs or TP programs.

You may see extra tasks running that are not yours. If the teach pendant is displaying a menu that was written using KAREL, such as Program Adjustment or Setup Passwords, you will see the status for this task also.

Examples: KCL> SHOW TASKS

C.91 SHOW TRACE command

Syntax: SHOW TRACE < **prog_name** >

where:

prog_name : the name of any KAREL or TP program which is a task

Purpose: Shows all the program statements and line numbers that have been executed since TRACE has been turned on.

The number of lines that are shown depends on the trace buffer length, which can be set with the SET_TASK command or the SET_TSK_ATTR built-in routine. To display the source lines of KAREL programs, the .KL files must exist on the RAM disk.

See Also: SET TRACE command

Example: KCL> SHOW TRACE

C.92 SHOW TYPES command

Syntax: SHOW TYPES < **prog_name** > < **FIELDS** >

where:

prog_name : the name of any KAREL or TP program

FIELDS : specifies fields should be displayed

Draft

Purpose: Displays a list including the name, type, and if specified, the fields of each user-defined type in the specified or default program. The actual array dimensions and string sizes are not shown.

See Also: SHOW VARS command, SHOW VARIABLE command

Examples: KCL> SHOW TYPES test_prog FIELDS

KCL> SH TYPES

C.93 SHOW VARIABLE command

Syntax: SHOW VARIABLE <[**prog_name**]> **var_name** <(HEXADECIMAL | BINARY)>

where:

prog_name : the name of any KAREL or TP program

var_name : a valid program variable

Purpose: Displays the name, type, and value of the specified variable.

You can display the values of system variables that allow KCL read access or the values of program variables. Use brackets ([]) after the variable name to specify a specific ARRAY element. If you do not specify a specific element the entire variable is displayed.

See Also: SHOW VARS command, SHOW SYSTEM command

Examples: KCL> SHOW VARIABLE $UTOOL

KCL> SH VAR [test_prog]group_mask HEX

KCL> SH VAR [test_prog]group_mask BINARY

KCL> SH VAR weld_pth[3]

C.94 SHOW VARS command

Syntax: SHOW VARS < **prog_name** > <VALUES>

where:

prog_name : the name of any KAREL or TP program

VALUES : specifies values should be displayed

Purpose: Displays a list including the name, type and, if specified, the current value of each variable in the specified or default program.

Draft

See Also: SHOW VARIABLE command, SHOW SYSTEM command, SHOW TYPES command

Example: KCL> SHOW VARS test_prog VALUES

KCL> SH VARS

C.95 SHOW data_type command

Syntax: SHOW **data_type** < **prog_name** > <VALUES>

where:

data_type : any valid KAREL data type

prog_name : the name of any KAREL or TP program

VALUES : specifies values should be displayed

Purpose: Displays a list of variables in the specified or default program (**prog_name**) of the specified data type (**data_type**). The list includes the name, type, and if specified, the current value of each variable.

See Also: SHOW VARS command, SHOW VARIABLE command

Examples: KCL> SHOW REAL test_prog VALUES

KCL> SH INTEGER

C.96 SIMULATE command

Syntax: SIMULATE **port_name[index]** < **= value** >

where:

port_name[index] : a valid I/O port

value : a new value for the port

Purpose: Simulating I/O allows you to test a program that uses I/O. Simulating I/O does not actually send output signals or receive input signals.

Warning

Depending on how signals are used, simulating signals might alter program execution. Do not simulate signals that are set up for safety checks. If you do, you could injure personnel or damage equipment.

Draft

When simulating a port value, you can specify its initial simulated value or allow the initial value to be the same as the physical port value. If no value is specified, the current physical port value is used.

The valid ports are:

DIN, DOUT, WDI, WDO (BOOLEAN)AIN, AOUT, GIN, GOUT (INTEGER)

See Also: UNSIMULATE command

Examples: KCL> SIMULATE DIN[17]

KCL> SIM DIN[1] = ON

KCL> SIM AIN[1] = 100

Draft

C.97 SKIP command

Syntax: SKIP < **prog_name** >

where:

prog_name : the name of any KAREL or TP program which is a task

Purpose: Skips execution of the current statement in the specified task. If **prog_name** is not specified, the default program is used. It has no effect when a task is running or when the system is in a READY state.

Entire motion statements are skipped with this command. You cannot skip single motion segments. The KCL> CONTINUE command resumes execution of the paused task with the statement following the last skipped statement. END statements cannot be skipped.

If you skip the last RETURN statement in a function routine, there is no way to return the value of the function to the calling program. Therefore, when executing the END statement of the routine, the task will abort.

If you skip into a FOR loop, you have skipped the statement that initializes the loop counter. When the ENDFOR statement is executed the program will try to remove the loop counter from the stack. If the FOR loop was nested in another FOR loop, the loop counter for the previous FOR loop will be removed from the stack, causing potentially invalid results. If the FOR loop was not nested, a stack underflow error will occur, causing the task to abort.

READ, MOVE, DELAY, WAIT FOR, and PULSE statements can be paused after they have begun execution. In these cases, when the task is resumed, execution of the paused statement must be finished before subsequent statements are executed. Subsequent skipped statements will not be executed. In particular, READ and WAIT FOR statements often require user intervention, such as entering data, before statement execution is completed.

Step mode operation and step mode type have no effect on the KCL> SKIP command.

Examples: KCL> SKIP test_prog

KCL> SKIP

C.98 STEP OFF command

Syntax: STEP OFF

Purpose: Disables single stepping for the program in which it was enabled.

Example: KCL> STEP OFF

C.99 STEP ON command

Syntax: STEP ON < **prog_name** >

where:

prog_name : the name of any KAREL or TP program which is a task

Purpose: Enables single stepping for the specified or default program.

Examples: KCL> STEP ON test_prog

KCL> STEP ON

C.100 TRANSLATE command

Syntax: TRANSLATE < **file_spec** > <DISPLAY> <LIST> <RS>

where:

file_spec : a valid file specification

DISPLAY : display source during translation

LIST : create listing file

RS : create routine stack (.rs) file for local var access

Purpose: Translates KAREL source code (.KL type files) into p-code (.PC type files), which can be loaded into memory and executed.

Draft

Translation of a program can be canceled using the CANCEL COMMAND key, CTRL-C, or CTRL-Y on the CRT/KB.

Examples: KCL> TRANSLATE testprog DISPLAY LIST

KCL> TRAN

C.101 TYPE command

Syntax: TYPE **file_spec**

where:

file_spec : a valid file specification

Purpose: This command allows you to display the contents of the specified ASCII file on the CRT/KB. You can specify any type of ASCII file.

Examples: KCL> TYPE rd:testprog.kl

KCL> TYPE testprog.kl

C.102 UNSIMULATE command

Syntax: UNSIMULATE (**port_name[index]** | **ALL**)

where:

port_name[index] : a valid I/O port

ALL : all simulated I/O ports

Purpose: Discontinues simulation of the specified input or output port. When a port is unsimulated, the physical value replaces the simulated value.

Warning

Depending on how signals are used, unsimulating signals might alter program execution or activate peripheral equipment. Do not unsimulate a signal unless you are sure of the result. If you do, you could injure personnel or damage equipment.

If you specify ALL instead of a particular port, simulation on all the simulated ports is discontinued.

The valid ports are:

Draft

DIN, DOUT, WDI, WDOAIN, AOUT, GIN, GOUT

See Also: SIMULATE command

Examples: KCL> UNSIMULATE DIN[17]

KCL> UNSIM ALL

Draft

C.103 WAIT command

Syntax: WAIT < **prog_name** > (DONE | PAUSE)

where:

prog_name : the name of any KAREL or TP program which is a task

DONE : specifies that the command procedure wait until execution of the current task is completed or aborted

PAUSE : specifies that the command procedure wait until execution of the current task is paused, completed, or aborted.

Purpose: Defers execution of the commands that follow the KCL> WAIT command in a command procedure until a task pauses or completes execution.

The command procedure waits until the condition specified with the DONE or PAUSE argument is met.

See Also: Section 13.4 , “Command Procedures”

Example: The following is an example of an executable command procedure:

```
> SET DEF testprog
> LOAD ALL
> RUN -- execute program
> WAIT PAUSE
> SHOW CURPOS -- display position of TCP when program pauses
> CONTINUE
> WAIT DONE
> CLEAR ALL YES -- clear after execution
```

Appendix D

CHARACTER CODES

Contents

Appendix D CHARACTER CODES D–1
D.1 CHARACTER CODES D–2

Draft

D.1 CHARACTER CODES

This appendix lists the ASCII numeric decimal codes and their corresponding ASCII, Multinational, graphic, and European characters as implemented on the KAREL system. The ASCII character set is the default character set for the KAREL system. Use the CHR Built-In Function, in Appendix A, to access the Multinational and Graphics character sets.

Table D–1. ASCII Character Codes

Decimal Code	Character Value	Decimal Code	Character Value	Decimal Code	Character Value	Decimal Code	Character Value
000	(NUL)	032	SP	064	@	096	‘
001	(SOH)	033	!	065	A	097	a
002	(STX)	034	"	066	B	098	b
003	(ETX)	035	#	067	C	099	c
004	(EOT)	036	$	068	D	100	d
005	(ENQ)	037	%	069	E	101	e
006	(ACK)	038	&	070	F	102	f
007	(BEL)	039	’	071	G	103	g
008	(BS)	040	(	072	H	104	h
009	(HT)	041	)	073	I	105	i
010	(LF)	042	*	074	J	106	j
011	(VT)	043	+	075	K	107	k
012	(FF)	044	’	076	L	108	l
013	(CR)	045	-	077	M	109	m
014	(SO)	046	.	078	N	110	n
015	(SI)	047	/	079	O	111	o
016	(DLE)	048	0	080	P	112	p
017	(DC1)	049	1	081	Q	113	q
018	(DC2)	050	2	082	R	114	r
019	(DC3)	051	3	083	S	115	s
020	(DC4)	052	4	084	T	116	t
021	(NAK)	053	5	085	U	117	u
022	(SYN)	054	6	086	V	118	v
023	(ETB)	055	7	087	W	119	w

Draft

Table D–1. ASCII Character Codes (Cont'd)

<table>
<tr><th>Decimal Code</th><th>Character Value</th><th>Decimal Code</th><th>Character Value</th><th>Decimal Code</th><th>Character Value</th><th>Decimal Code</th><th>Character Value</th></tr>
<tr><td>024</td><td>(CAN)</td><td>056</td><td>8</td><td>088</td><td>X</td><td>120</td><td>x</td></tr>
<tr><td>025</td><td>(EM)</td><td>057</td><td>9</td><td>089</td><td>Y</td><td>121</td><td>y</td></tr>
<tr><td>026</td><td>(SUB)</td><td>058</td><td>:</td><td>090</td><td>Z</td><td>122</td><td>z</td></tr>
<tr><td>027</td><td>(ESC)</td><td>059</td><td>;</td><td>091</td><td>[</td><td>123</td><td>{</td></tr>
<tr><td>028</td><td>(FS)</td><td>060</td><td><</td><td>092</td><td>\</td><td>124</td><td>|</td></tr>
<tr><td>029</td><td>(GS)</td><td>061</td><td>=</td><td>093</td><td>]</td><td>125</td><td>}</td></tr>
<tr><td>030</td><td>(RS)</td><td>062</td><td>></td><td>094</td><td>^</td><td>126</td><td>~</td></tr>
<tr><td>031</td><td>(US)</td><td>063</td><td>?</td><td>095</td><td>—</td><td>127</td><td>(DEL)</td></tr>
</table>

Draft

Table D–2. Special ASCII Character Codes

Decimal Code	Character Value	Decimal Code	Character Value
128	Clear window	154	Turn Multinational mode on
129	Clear to end of line	155 48	Foreground color black
130	Clear to end of window	155 49	Foreground color red
131	Set cursor position	155 50	Foreground color green
132	Carriage return	155 51	Foreground color yellow
133	Line feed	155 52	Foreground color blue
134	Reverse line feed	155 53	Foreground color magenta
135	Carriage return & line feed	155 54	Foreground color cyan
136	Back Space	155 55	Foreground color white

Table D–2. Special ASCII Character Codes (Cont'd)

Decimal Code	Character Value	Decimal Code	Character Value
137	Home cursor in window	155 127	Foreground color default
138	Blink video attribute	156 48	Background color black
139	Reverse video attribute	156 49	Background color red
140	Bold video attribute	156 50	Background color green
141	Underline video attribute	156 51	Background color yellow
142	Wide video size	156 52	Background color blue
143	Normal video attribute	156 53	Background color magenta
146	Turn Graphics mode on	156 54	Background color cyan
147	Turn ASCII mode on	156 55	Background color white
148	High/wide video size	156 127	Background color default
153	Normal video size		

Table D–3. Multinational Character Codes

Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value
000		032		064	À	096	à
001		033	¡	065	Á	097	á
002		034	©	066	Â	098	â
003		035	£	067	Ã	099	ã
004	(IND)	036		068	Ä	100	ä
005	(NEL)	037	¥	069	Å	101	å
006	(SSA)	038		070	Æ	102	æ
007	(ESA)	039	§	071	Ç	103	ç

Draft

Table D–3. Multinational Character Codes (Cont'd)

Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value
008	(HTS)	040	¤	072	È	104	è
009	(HTJ)	041	©	073	É	105	é
010	(VTS)	042	ª	074	Ê	106	ê
011	(PLD)	043	«	075	Ë	107	ë
012	(PLU)	044		076	Ì	108	ì
013	(RI)	045		077	Í	109	í
014	(SS2)	046		078	Î	110	î
015	(SS3)	047		079	Ï	111	ï
016	(DCS)	048	○	080		112	
017	(PU1)	049	±	081	Ñ	113	ñ
018	(PU2)	050	^{2}	082	Ò	114	ò
019	(STS)	051	^{3}	083	Ó	115	ó
020	(CCH)	052		084	Ô	116	ô
021	(MW)	053	µ	085	Õ	117	õ
022	(SPA)	054	¶	086	Ö	118	ö
023	(EPA)	055	•	087	Œ	119	œ
024		056		088	Ø	120	ø
025		057	^{1}	089	Ù	121	ù
026		058		090	Ú	122	ú
027	(CSI)	059	»	091	Û	123	û
028	(ST)	060	¼	092	Ü	124	ü
029	(OSC)	061	½	093	Y	125	ÿ
030	(PM)	062		094		126	
031	(APC)	063	¿	095	ß	127	

Draft

Table D–4. Graphics Character Codes (not available in R-30iB)

Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value
000	(NUL)	032	SP	064	@	096	♦
001	(SOH)	033	!	065	A	097	▒
002	(STX)	034	"	066	B	098	HT
003	(ETX)	035	#	067	C	099	FF
004	(EOT)	036	$	068	D	100	CR
005	(ENQ)	037	%	069	E	101	LF
006	(ACK)	038	&	070	F	102	○
007	(BEL)	039	’	071	G	103	±
008	(BS)	040	(	072	H	104	NL
009	(HT)	041	)	073	I	105	VT
010	(LF)	042	*	074	J	106	┘
011	(VT)	043	+	075	K	107	┐
012	(FF)	044	’	076	L	108	┌
013	(CR)	045	-	077	M	109	└
014	(SO)	046	.	078	N	110	+
015	(SI)	047	/	079	O	111	-
016	(DLE)	048	0	080	P	112	-
017	(DC1)	049	1	081	Q	113	-
018	(DC2)	050	2	082	R	114	-
019	(DC3)	051	3	083	S	115	_
020	(DC4)	052	4	084	T	116	├

Draft

Table D–4. Graphics Character Codes (not available in R-30iB) (Cont'd)

Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value	Decimal Codes	Character Value
021	(NAK)	053	5	085	U	117	├
022	(SYN)	054	6	086	V	118	┴
023	(ETB)	055	7	087	W	119	┬
024	(CAN)	056	8	088	X	120	|
025	(EM)	057	9	089	Y	121	≤
026	(SUB)	058	:	090	Z	122	≥
027	(ESC)	059	;	091	[	123	Π
028	(FS)	060	<	092		124	≠
029	(GS)	061	=	093	]	125	£
030	(RS)	062	>	094	^	126	•
031	(US)	063	?	095	(blank)	127	(DEL)

Table D–5. Teach Pendant Input Codes

Code	Value	Code	Value
0	48	174	USER KEY 2
1	49	175	USER KEY 3
2	50	176	USER KEY 4
3	51	177	USER KEY 5
4	52	178	USER KEY 6
5	53	185	FWD
6	54	186	BWD
7	55	187	COORD

Draft

Table D–5. Teach Pendant Input Codes (Cont'd)

Code	Value	Code	Value
8	56	188	+X
9	57	189	+Y
128	PREV	190	+Z
129	F1	191	+X rotation
131	F2	192	+Y rotation
132	F3	193	+Z rotation
133	F4	194	-X
134	F5	195	-Y
135	NEXT	196	-Z
143	SELECT	197	-X rotation
144	MENUS	198	-Y rotation
145	EDIT	199	-Z rotation
146	DATA	210	USER KEY
147	FCTN	212	Up arrow
148	ITEM	213	Down arrow
149	+%	214	Right arrow
150	-%	215	Left arrow
151	HOLD	147	DEADMAN switch, left
152	STEP	248	DEADMAN switch, right
153	RESET	249	ON/OFF switch
173	USER KEY 1	250	EMERGENCY STOP

Table D–6. European Character Codes

Code	Value	Code	Value	Code	Value
192	A‘	213	O~	234	e^
193	A‘	214	O:	235	e:
194	A^	215	OE	236	i‘
195	A~	216	O/	237	i‘
196	A:	217	U‘	238	i^
197	Ao	218	U‘	239	i:

Table D–6. European Character Codes (Cont'd)

Code	Value	Code	Value	Code	Value
198	AE	219	U^	240	
199	CC	220	U:	241	n~
200	E‘	221	Y:	242	o‘
201	E‘	222		243	o‘
202	E^	223	Bb	244	o^
203	E:	224	a‘	245	o~
204	I‘	225	a‘	246	o:
205	I‘	226	a^	247	oe
206	I^	227	a~	248	
207	I:	228	a:	249	u‘
208		229	ao	250	u‘
209	N~	230	ae	251	u^
210	O‘	231		252	u:
211	O‘	232	e‘	253	y:
212	O^	233	e‘	254	

Draft

A^ = A with ^ on top

A‘ = A with ‘ on top

Ao = A with o on top

A~ = A with ~ on top

A: = A with .. on top

AE = A and E run together

OE = A and E run together

Bb = Beta

Table D–7. Graphics Characters

Decimal Value	ASCII Character	Graphic Character
97	a	solid box
102	f	diamond
103	g	plus/minus
106	j	lower-right box corner
107	k	upper-right box corner
108	l	upper-left box corner
109	m	lower-left box corner
110	n	intersection lines
111	o	pixel row 1 horizontal line
112	p	pixel row 2 horizontal line
113	q	pixel row 3 horizontal line
114	r	pixel row 4 horizontal line
115	s	pixel row 5 horizontal line
116	t	T from right
117	u	T from left
119	v	T from above
119	w	T from below
120	x	Vertical Line
121	y	Less than or equal
122	z	Greater than or equal
123	{	Pi
124	|	Not equal
125	}	British pound symbol

Draft

SYNTAX DIAGRAMS

Contents

Appendix E SYNTAX DIAGRAMS E–1

Draft

KAREL syntax diagrams use the following symbols:

- Rectangle

  A rectangle encloses elements that are defined in another syntax diagram or in accompanying text.

- Oval

  An oval encloses KAREL reserved words that are entered exactly as shown.

- Circle

  A circle encloses special characters that are entered exactly as shown.

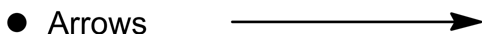
- Dot

  A dot indicates a mandatory line end (; or ENTER key) before the next syntax element.

- Caret

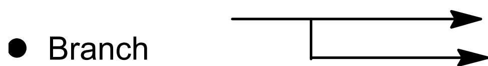
  A caret indicates an optional line end.

- Arrows



  Arrows indicate allowed paths and the correct sequence in a diagram.

- Branch

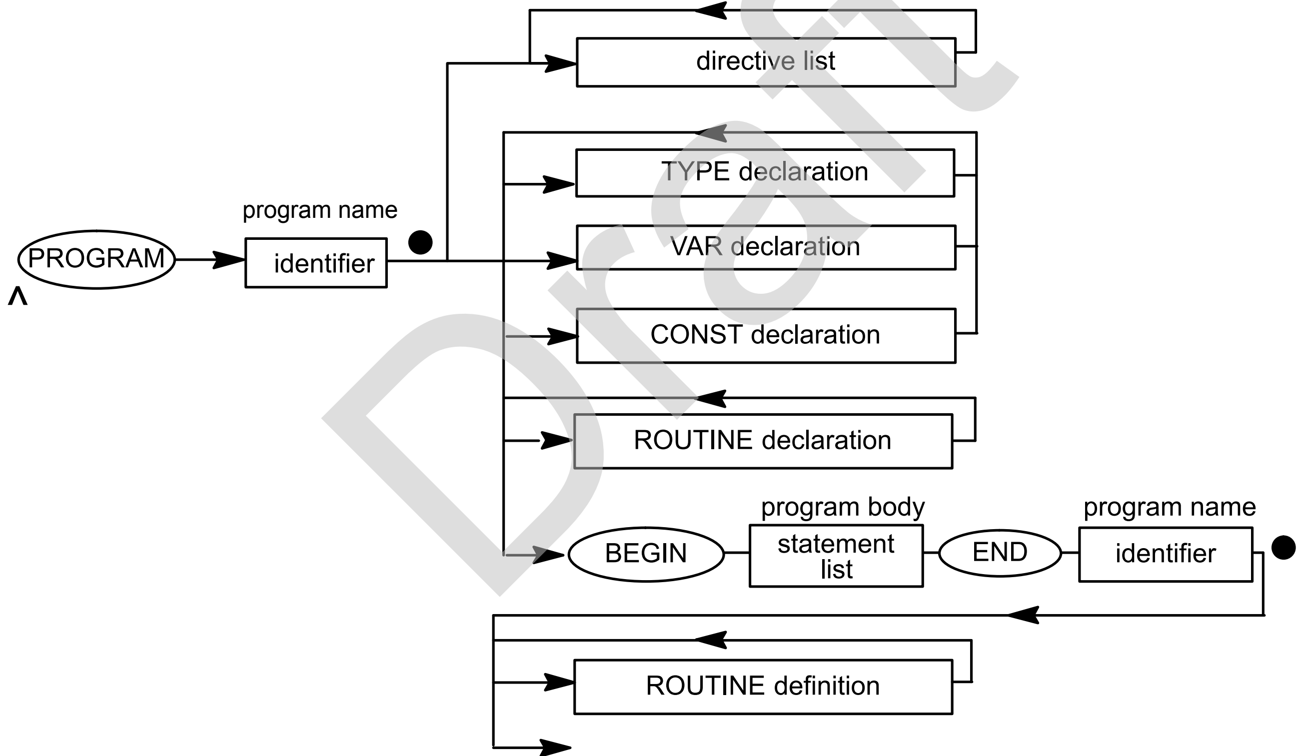


  Branches indicate optional paths or sequences.

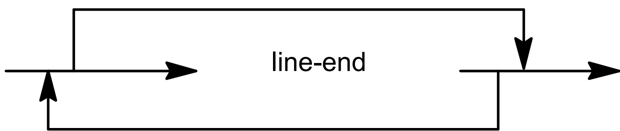
Draft

Figure E–1.

PROGRAM module definition



∧ -- 0 or more line-ends



● --newline

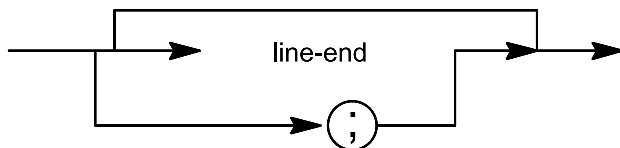
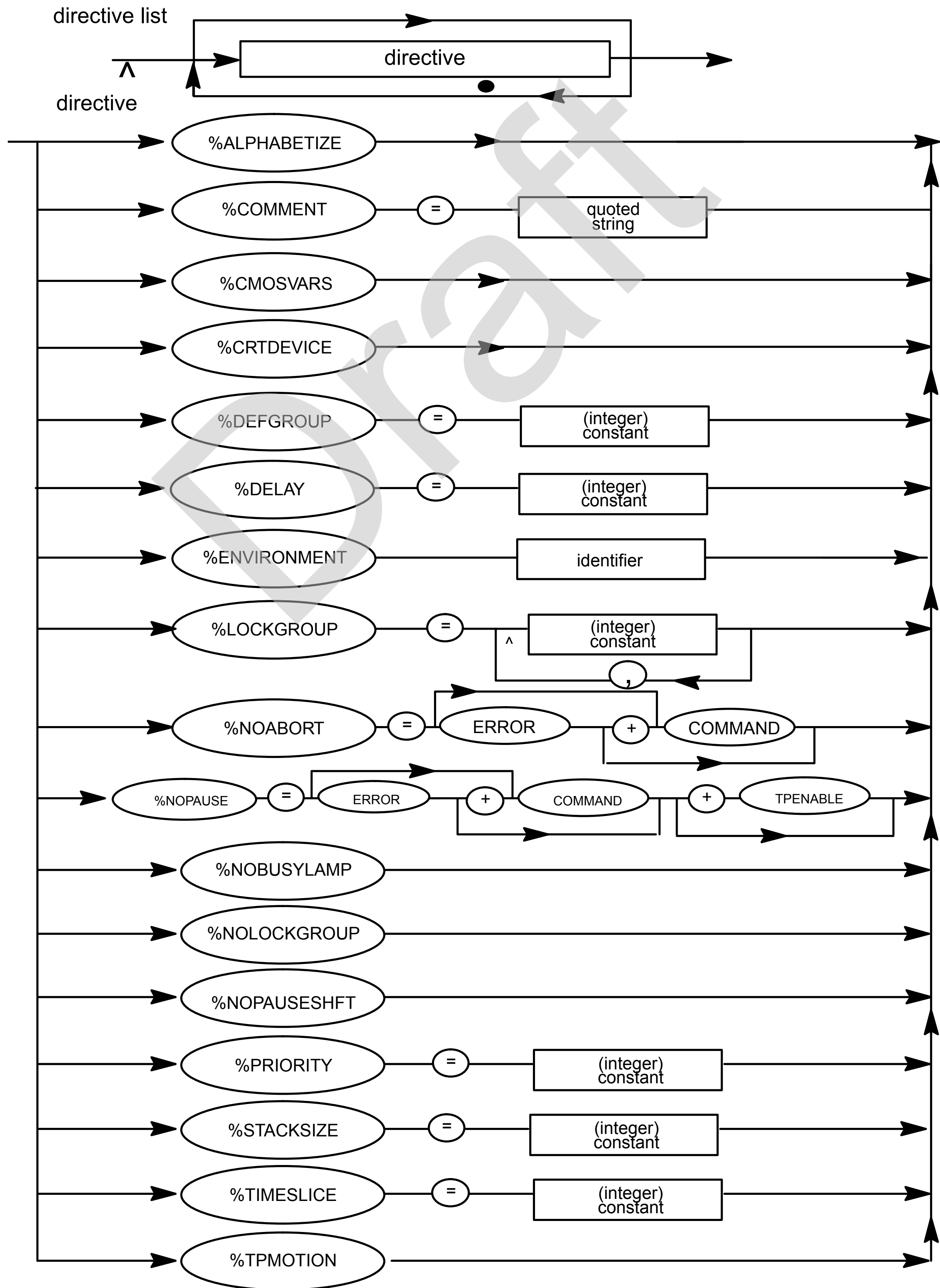
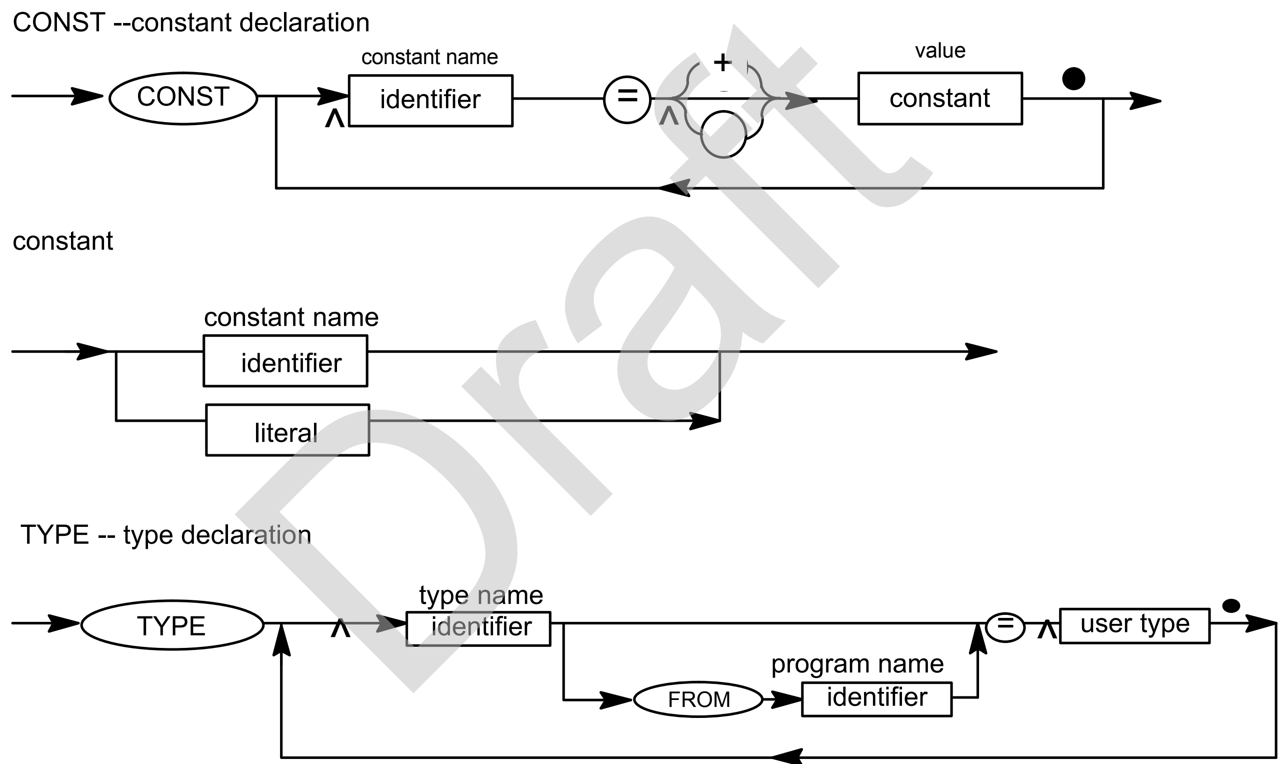


Figure E–2.



Draft

Figure E–3.



Draft

Figure E–4.

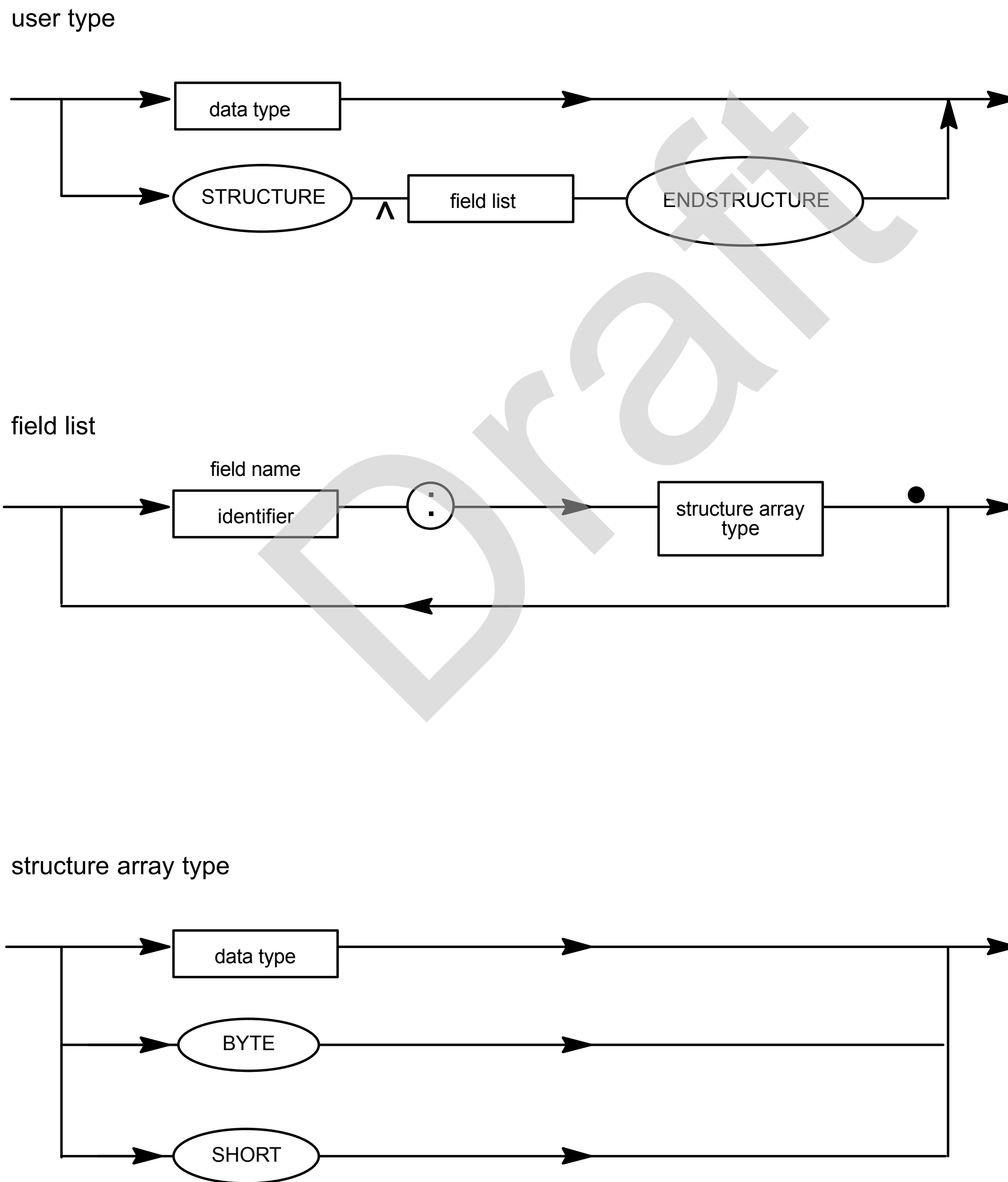


Figure E–5.

data type

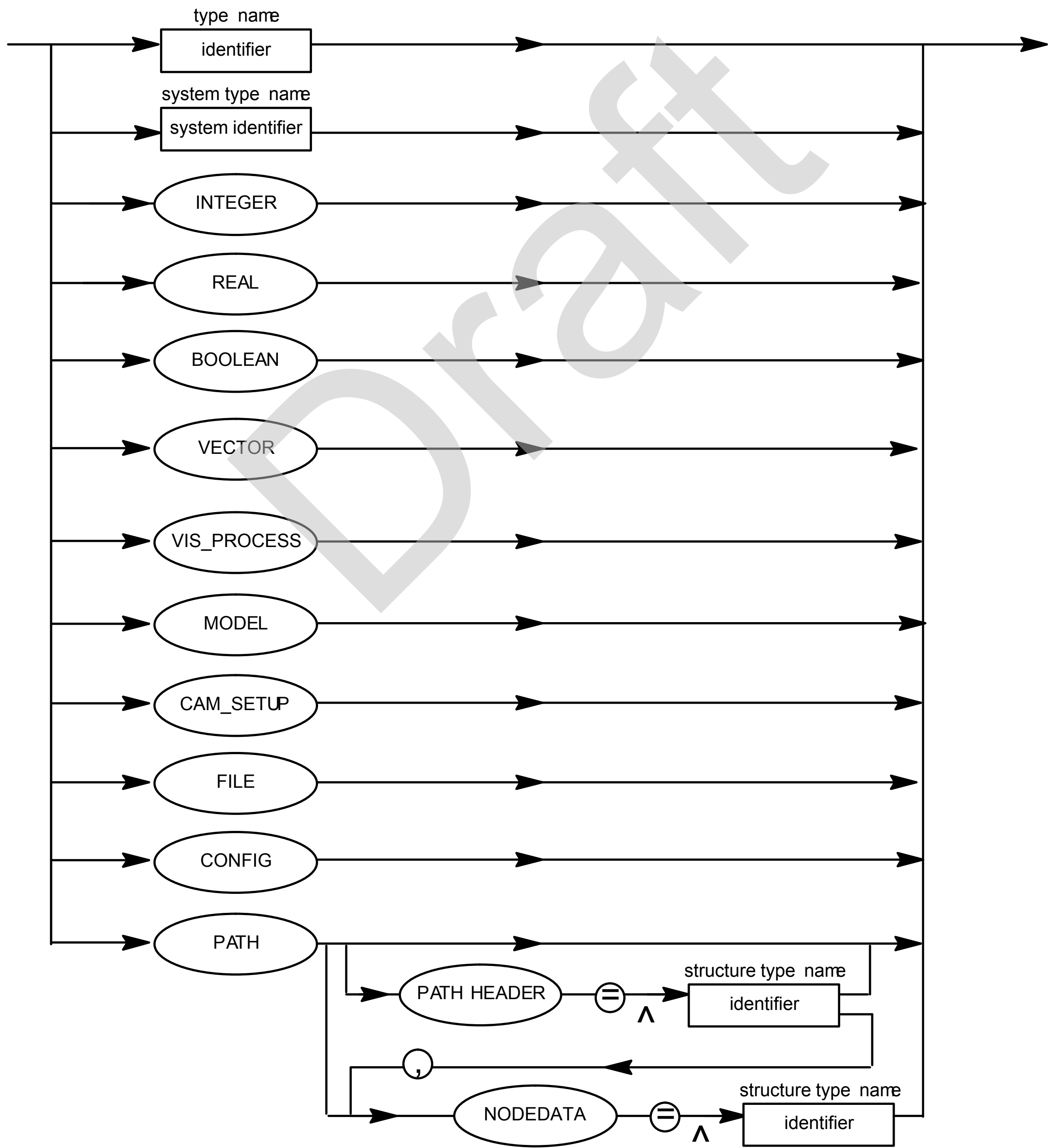
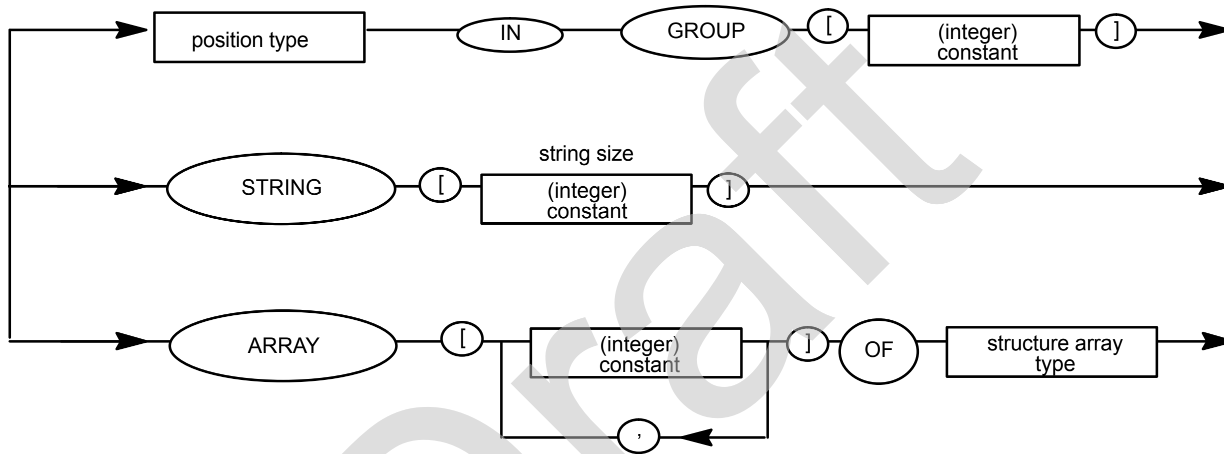


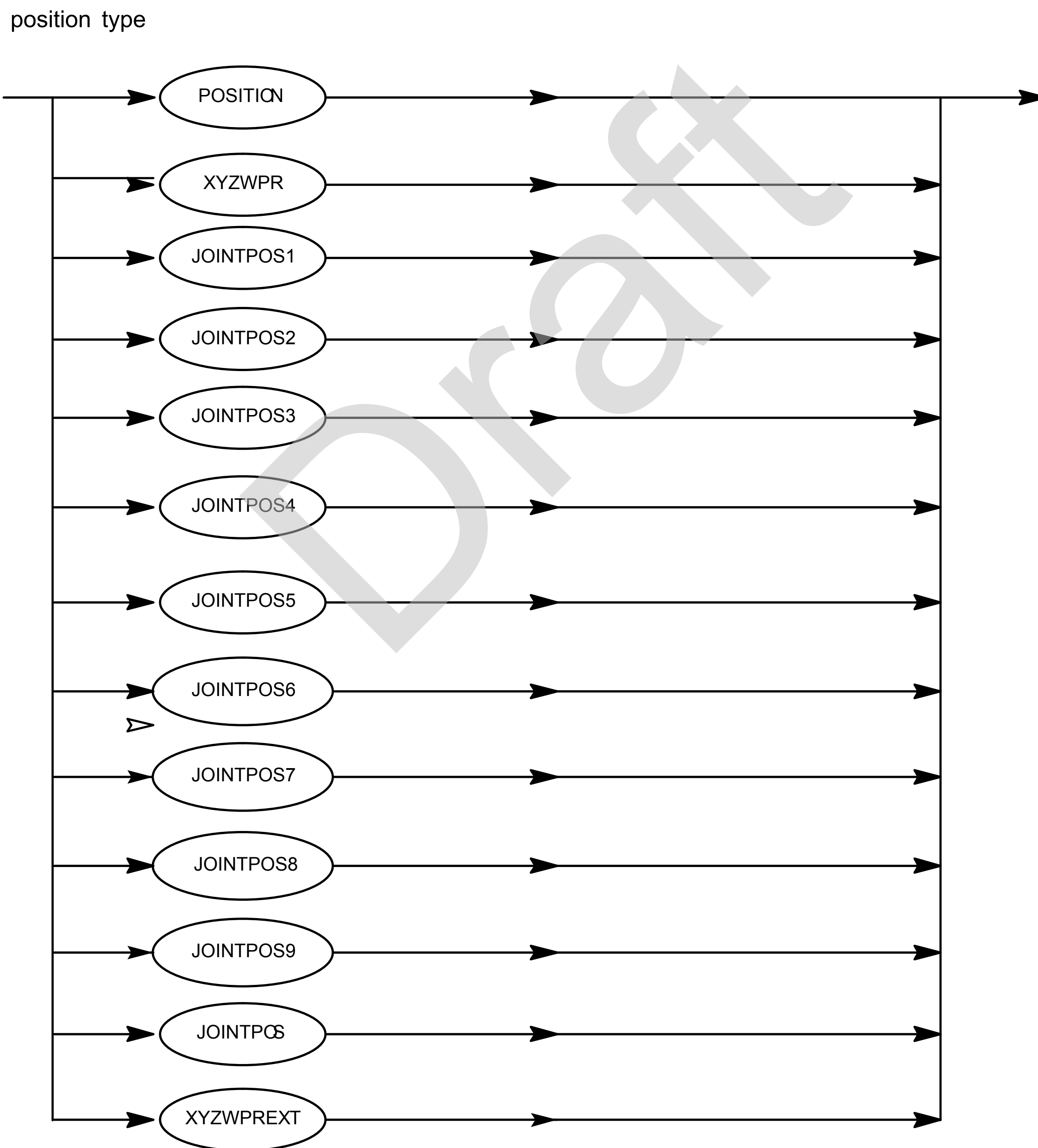
Figure E–6.

data type continued



Draft

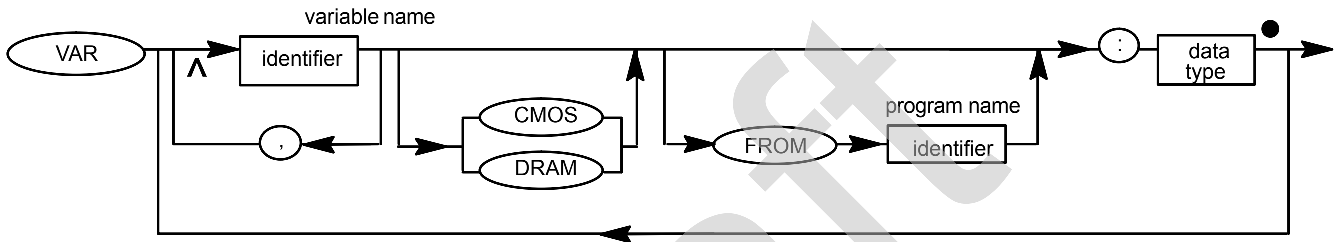
Figure E–7.



Draft

Figure E–8.

VAR -- variable declaration



Draft

ROUTINE -- routine declaration

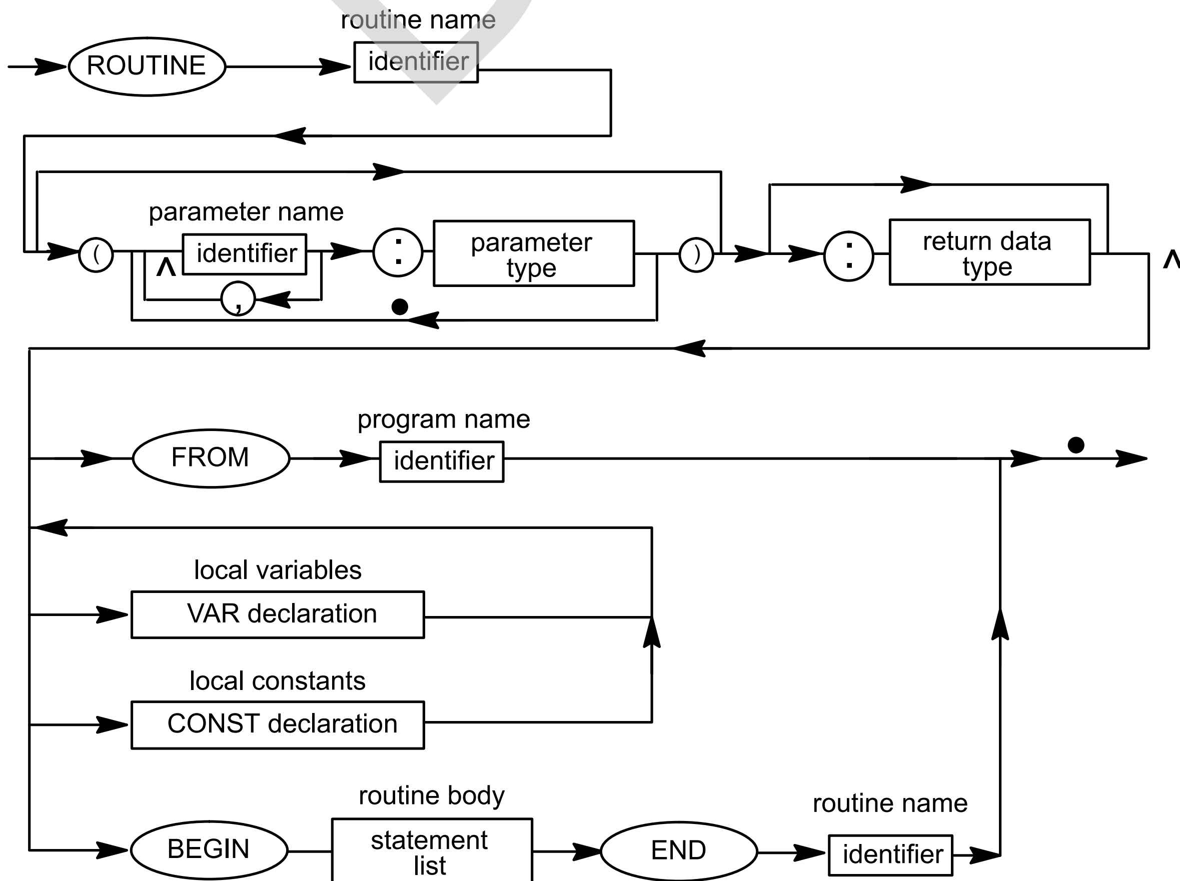
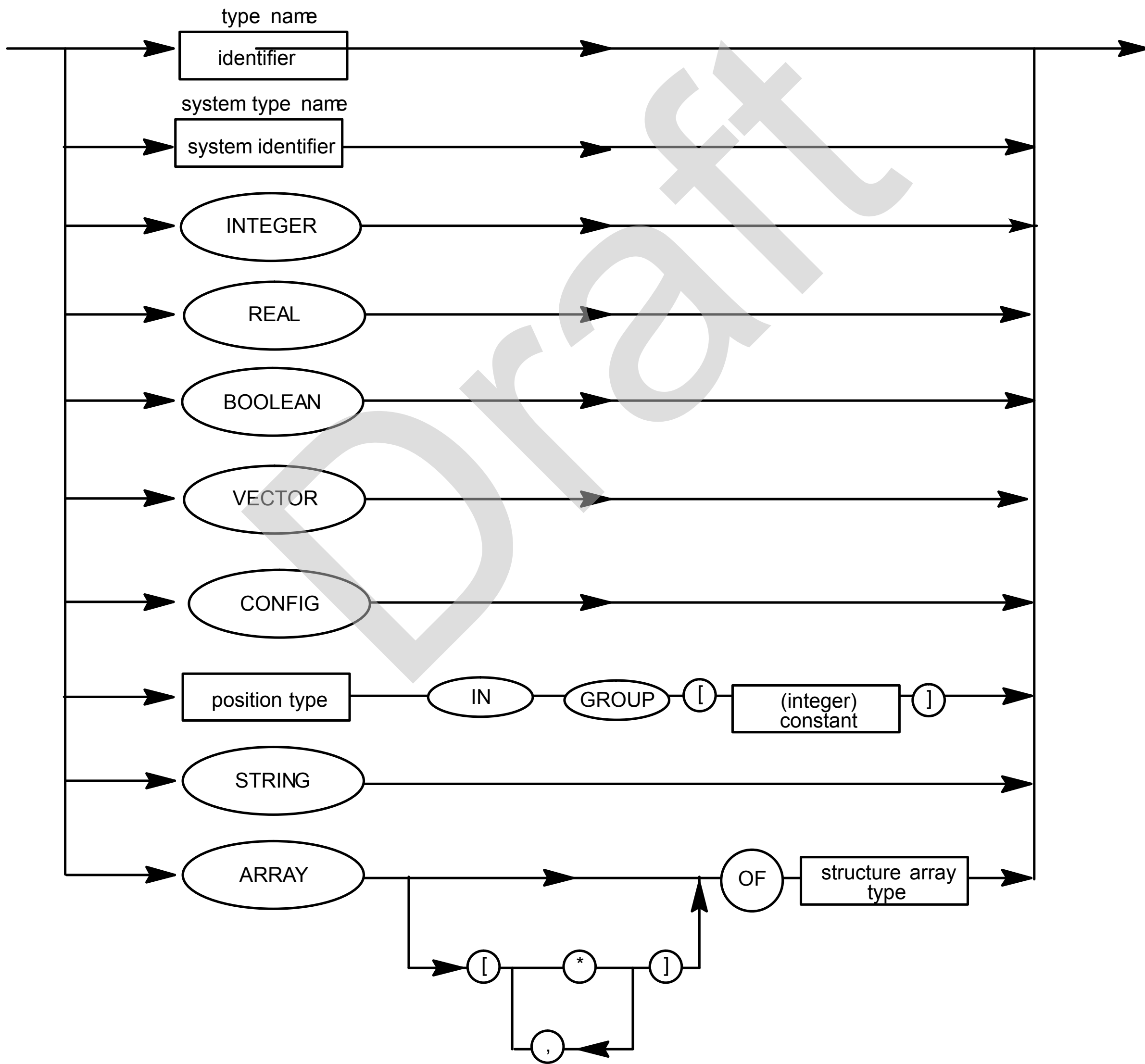


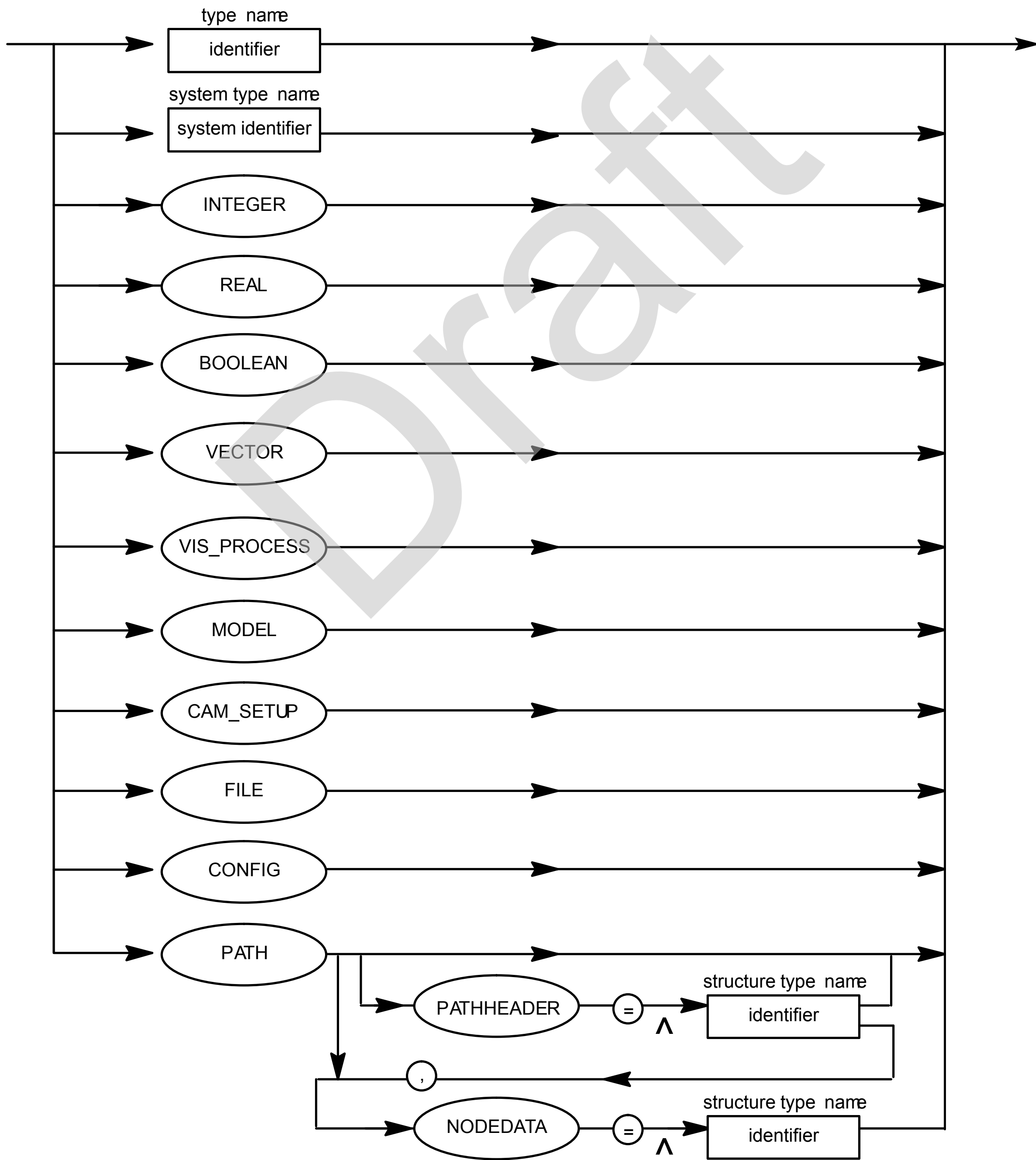
Figure E–9.

return data type



Draft

Figure E–10.



Draft

Figure E–11.

parameter type continued

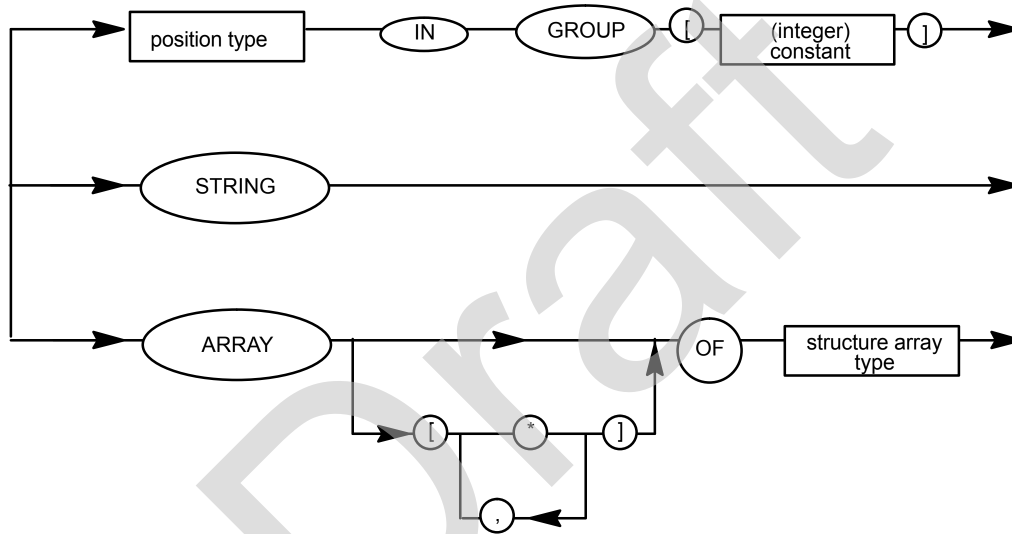
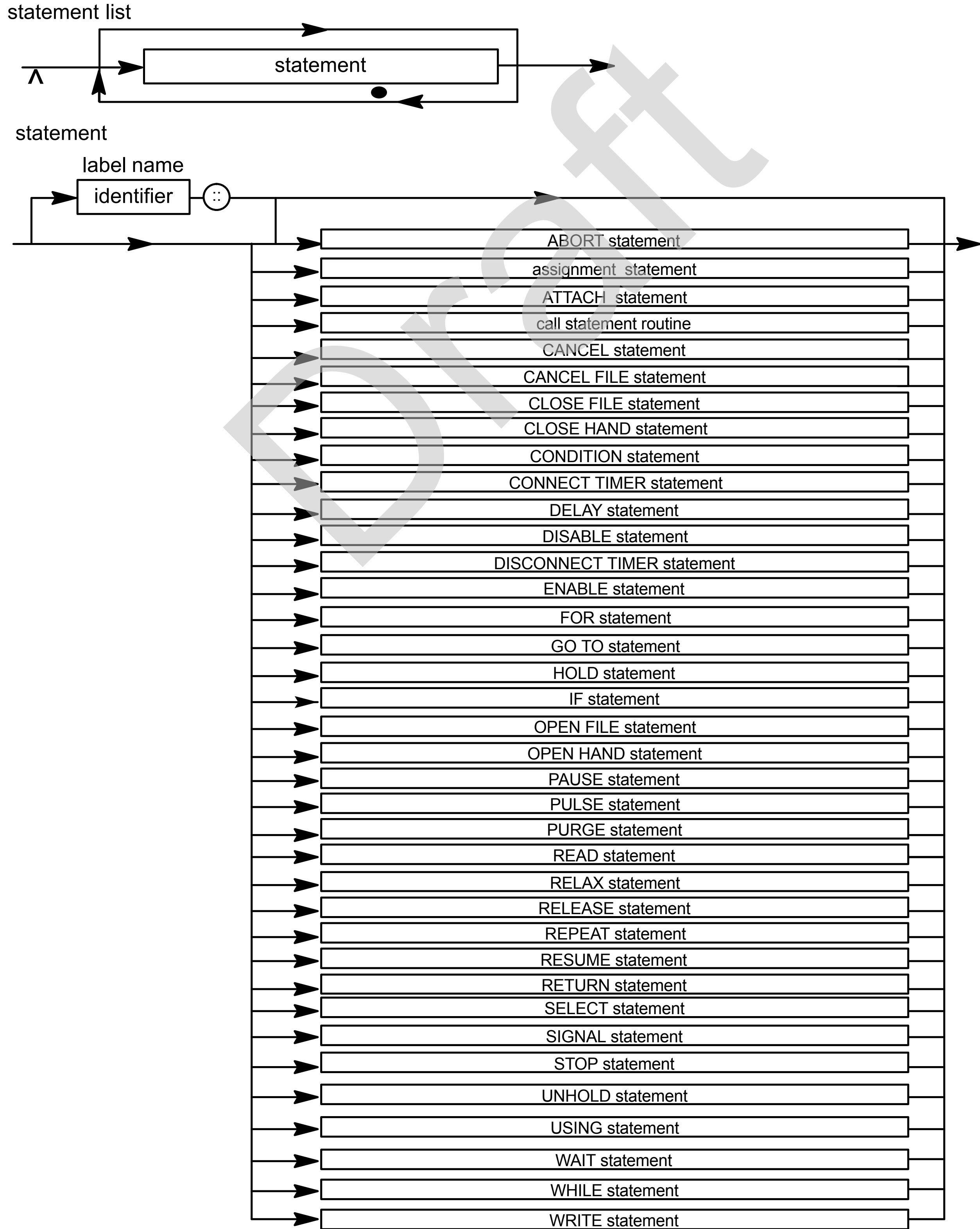
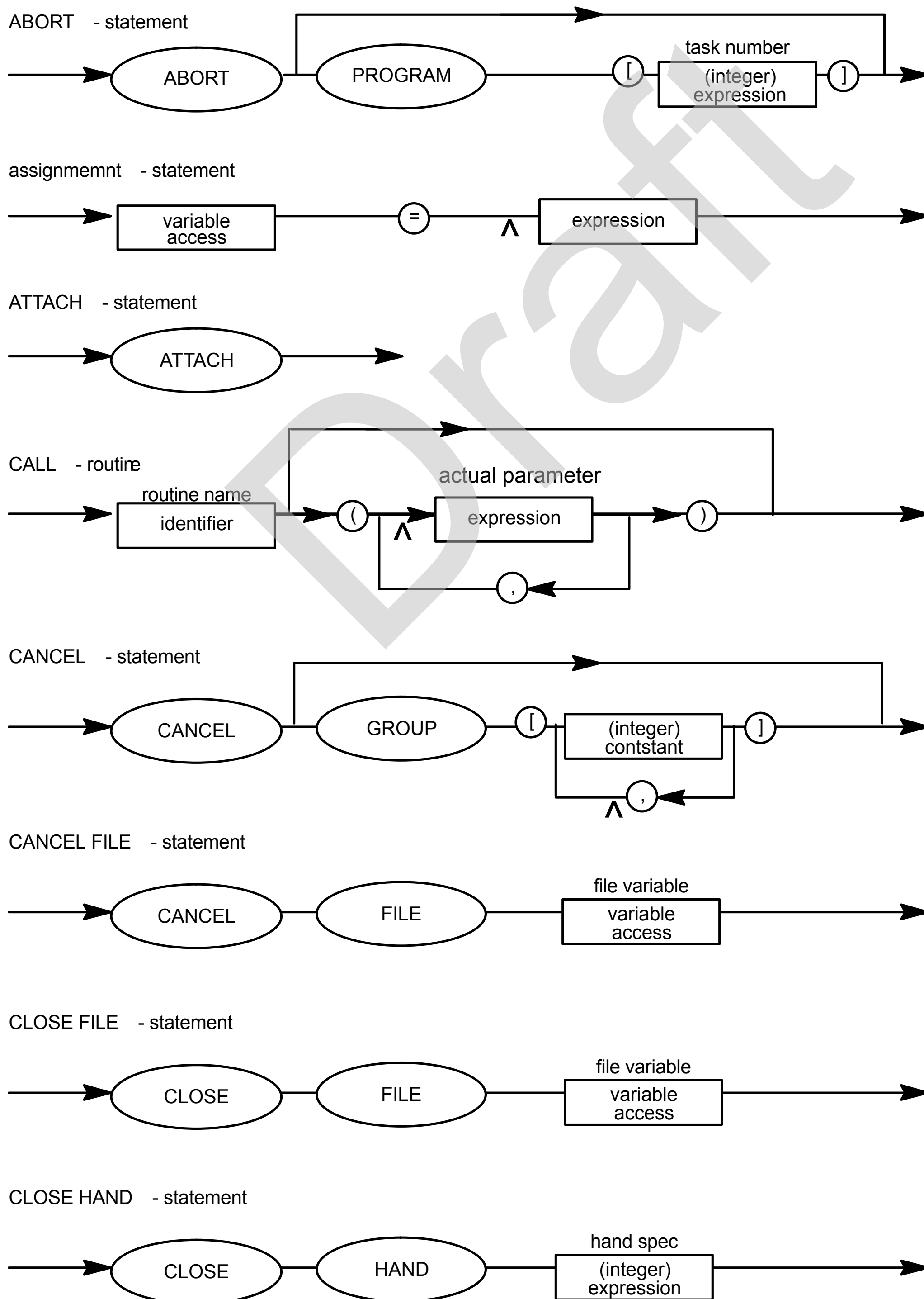


Figure E–12.



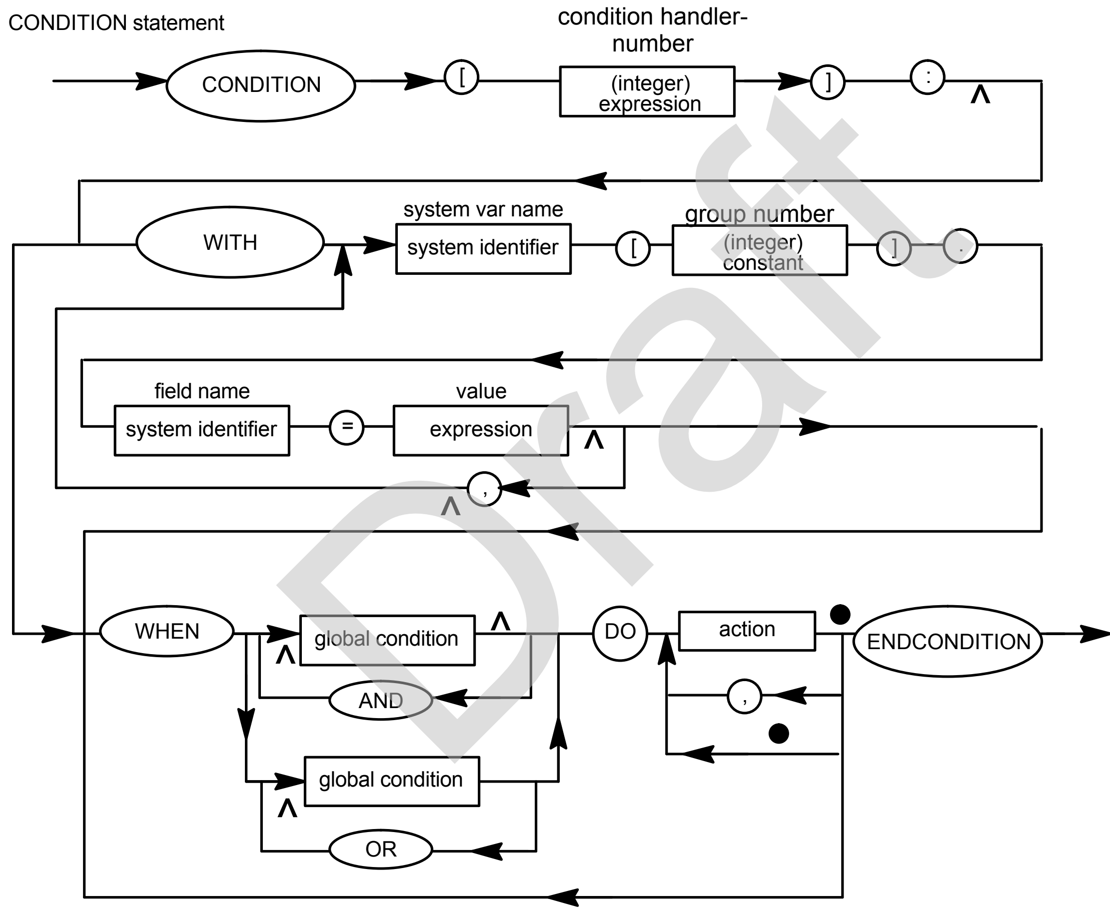
Draft

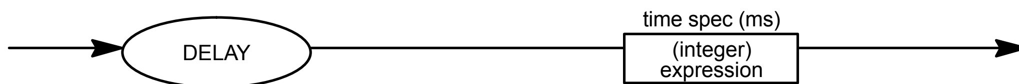
Figure E–13.



Draft

Figure E–14.





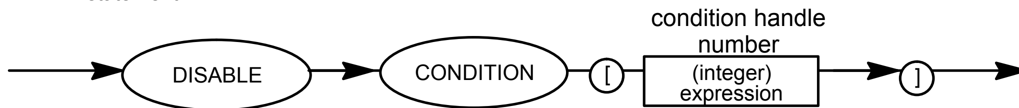
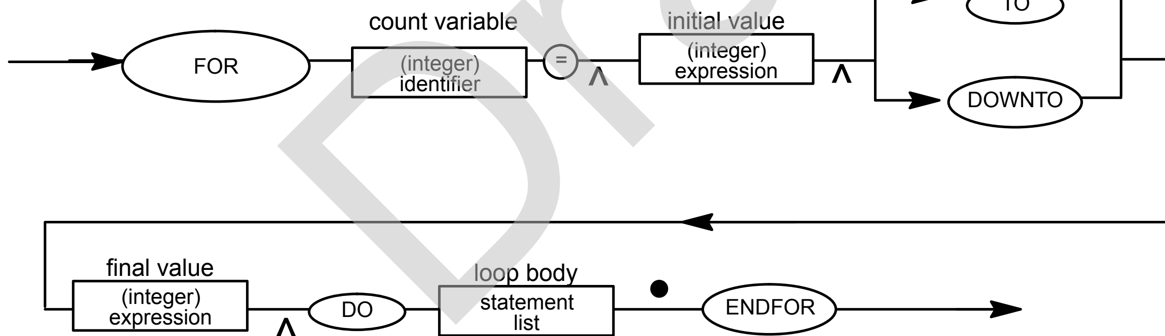


Figure E–15.

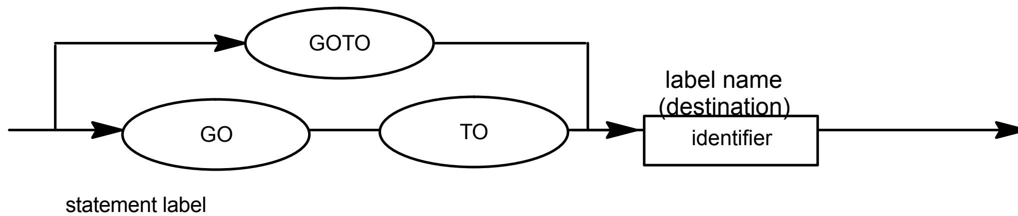
DISCONNECT TIMER - statement

ENABLE - statement

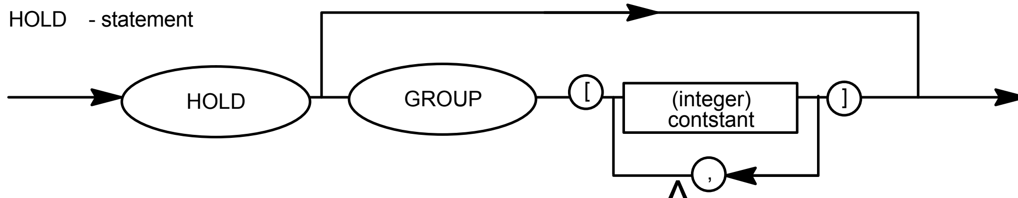
FOR - statement



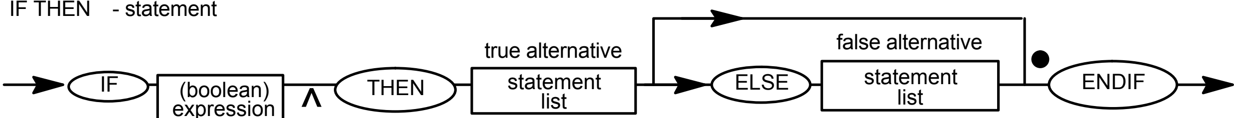
GO TO - statement



HOLD - statement



IF THEN - statement



Draft

Figure E–16.

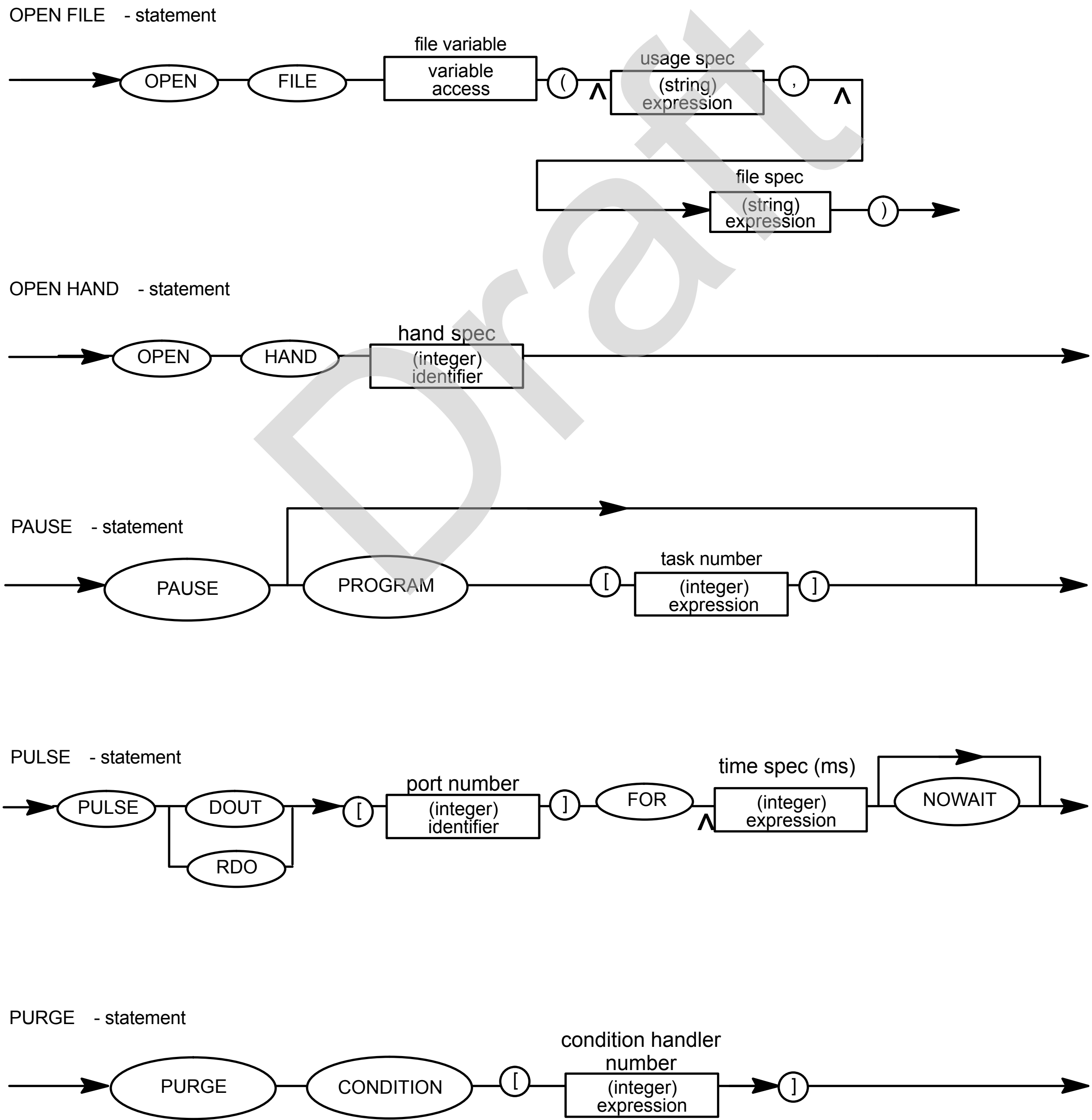
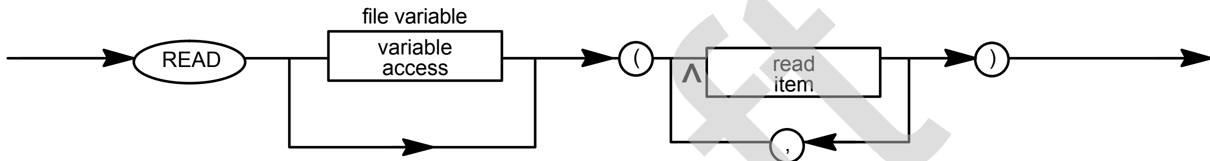
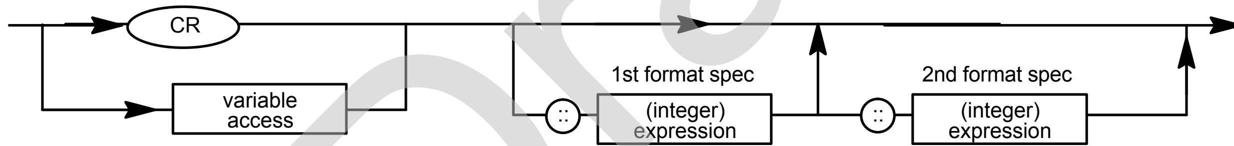


Figure E–17.

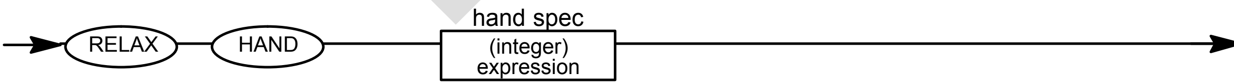
READ - statement



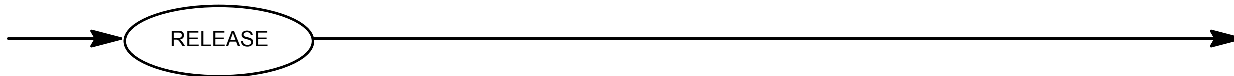
read item - statement



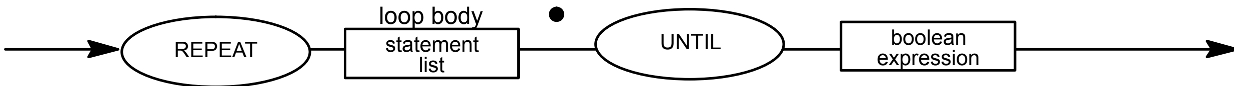
RELAX - statement



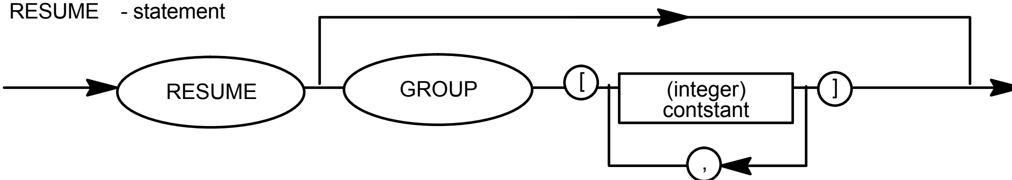
RELEASE - statement



REPEAT - statement



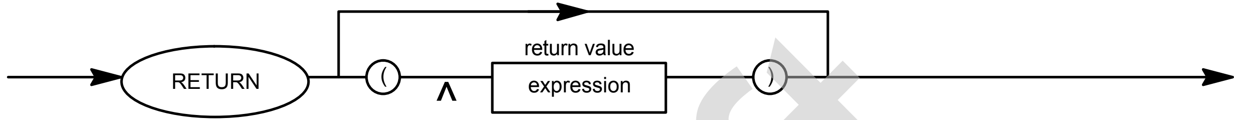
RESUME - statement



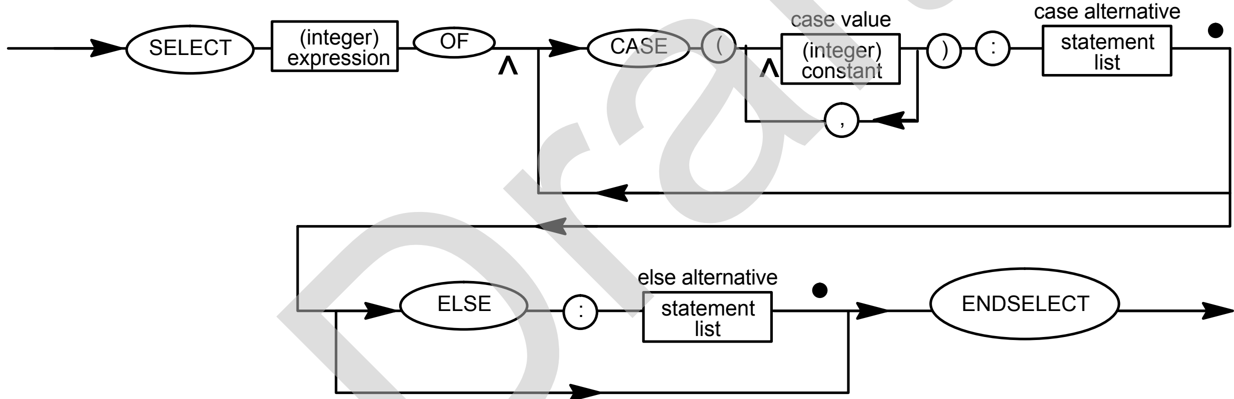
Draft

Figure E–18.

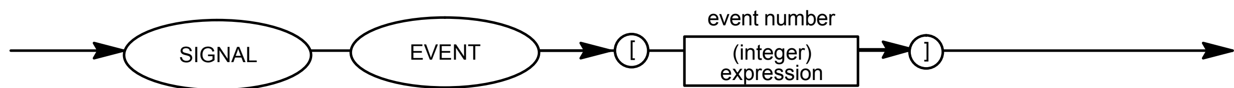
RETURN - statement



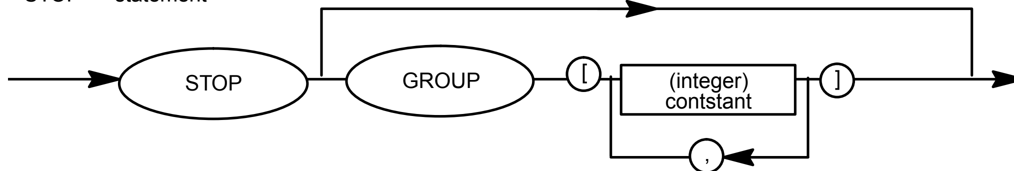
SELECT - statement



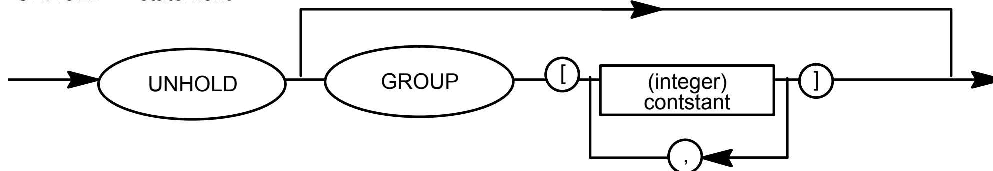
SIGNAL - statement



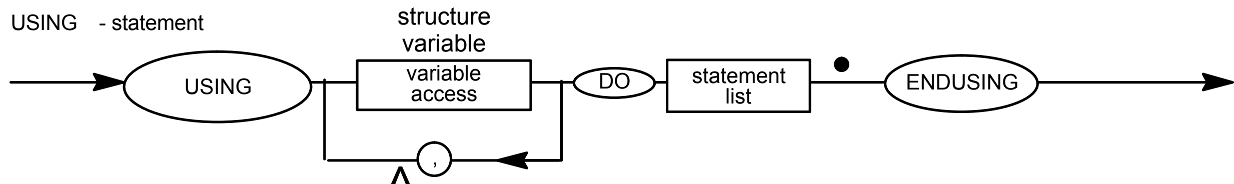
STOP - statement



UNHOLD - statement



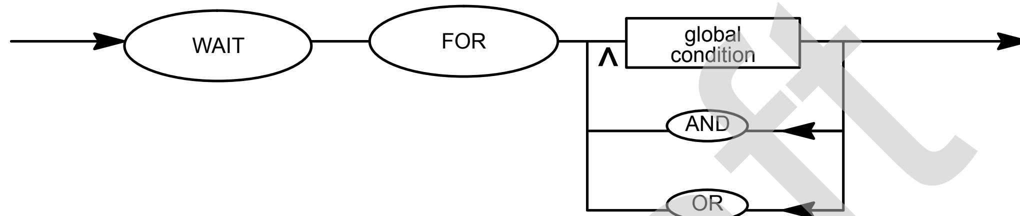
USING - statement



Draft

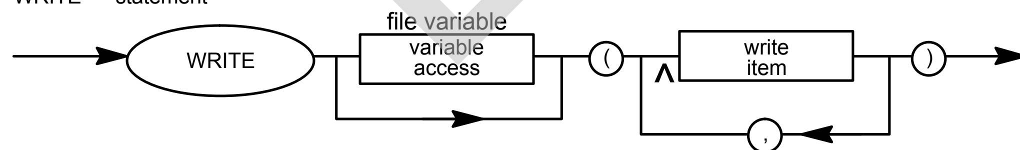
Figure E–19.

WAIT - statement

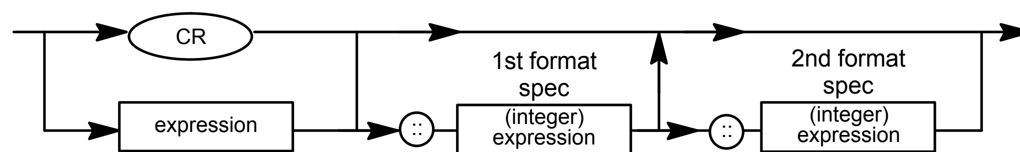


WHILE - statement

WRITE - statement

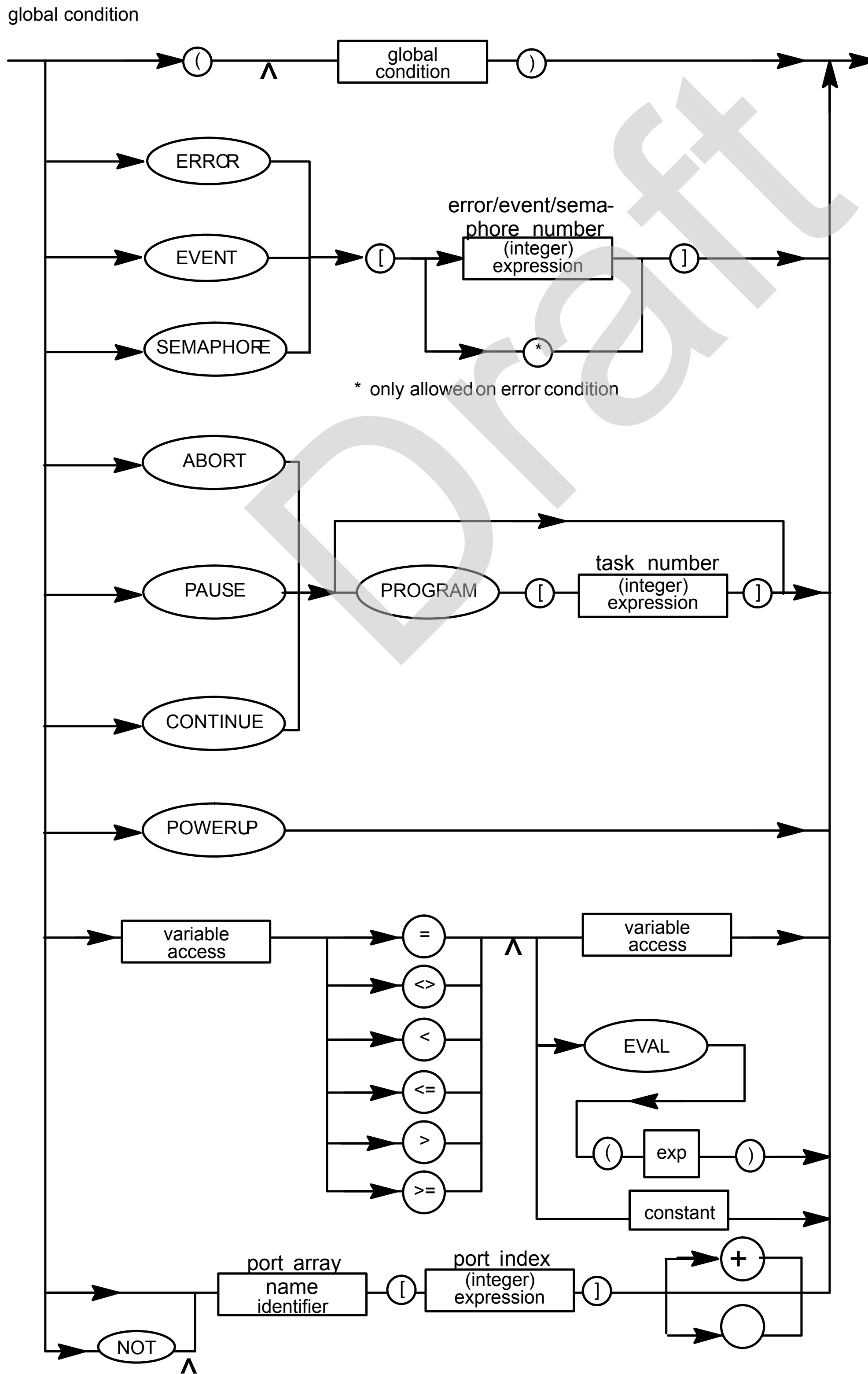


write item



Draft

Figure E–20.



Draft

Figure E–21.

condition handler action

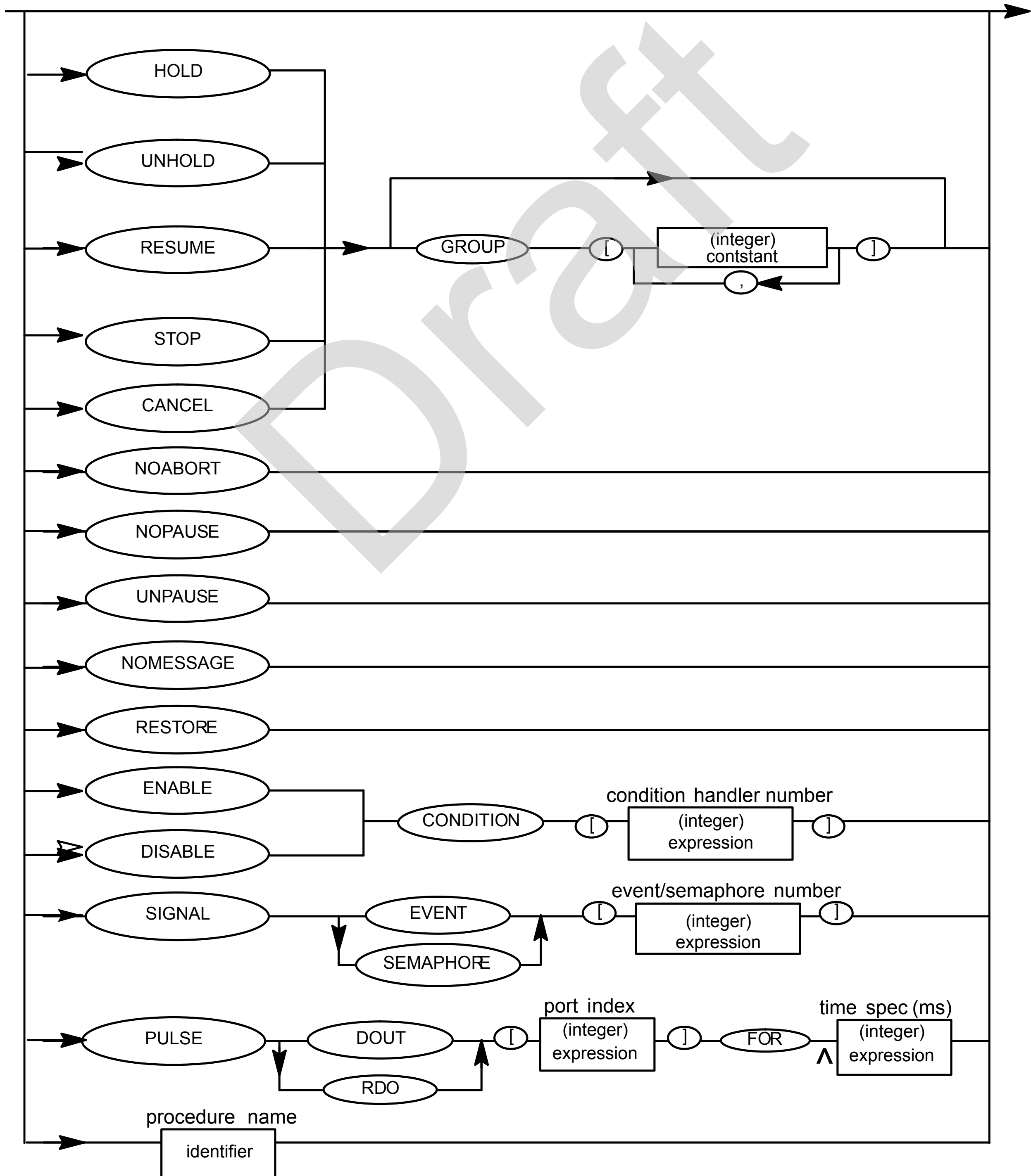
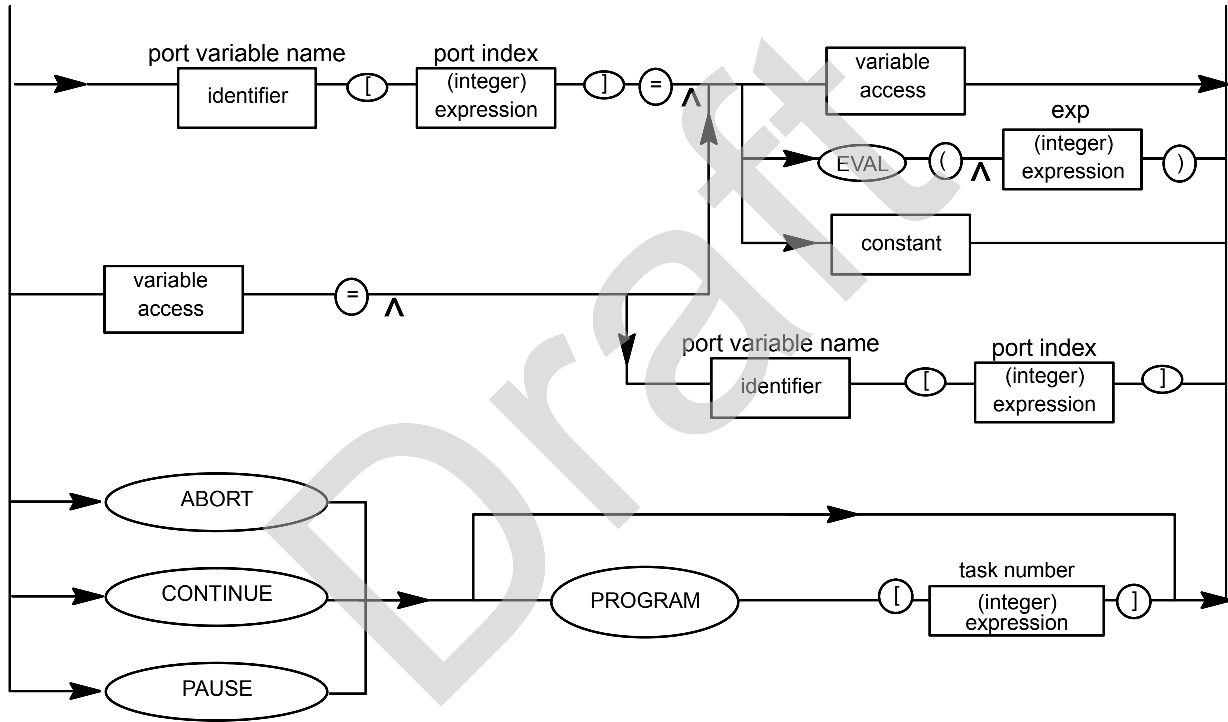


Figure E–22.

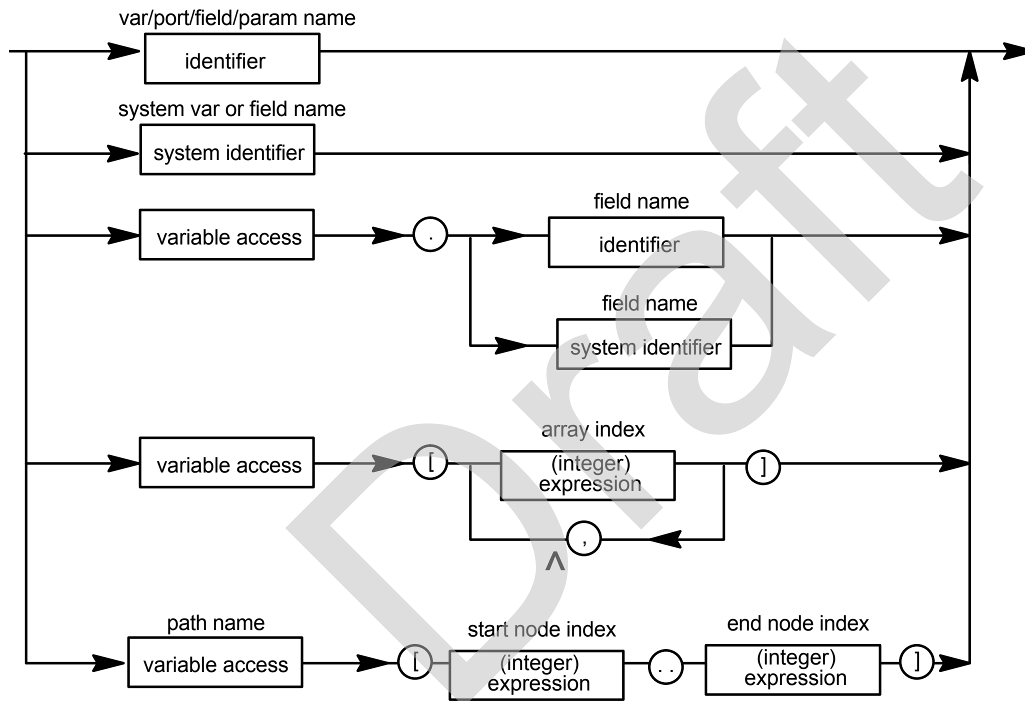
condition handler action continued



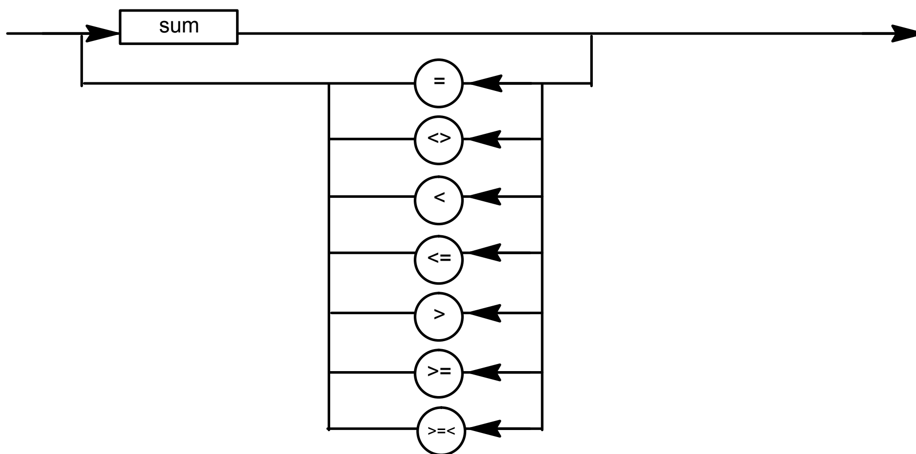
Draft

Figure E–23.

variable access



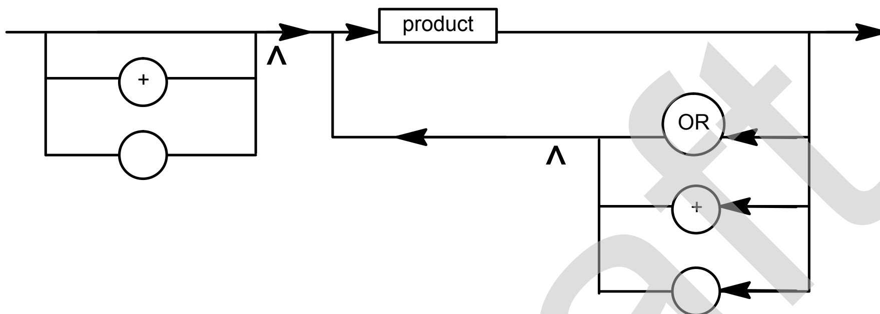
expression



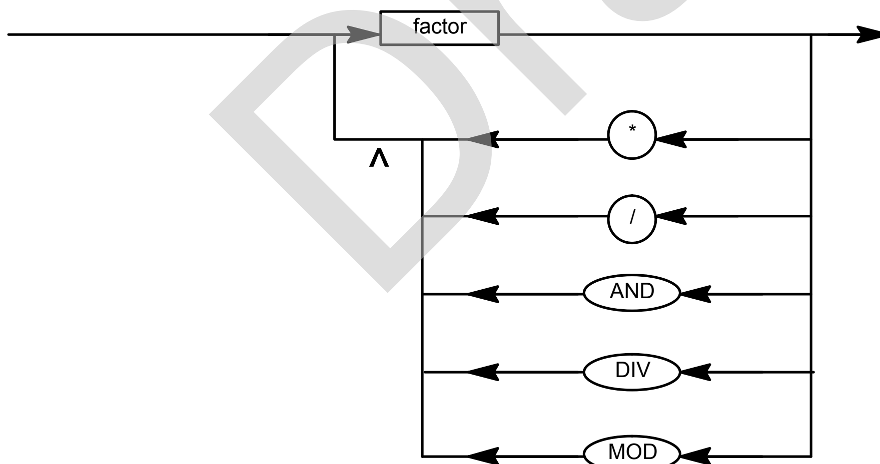
Draft

Figure E–24.

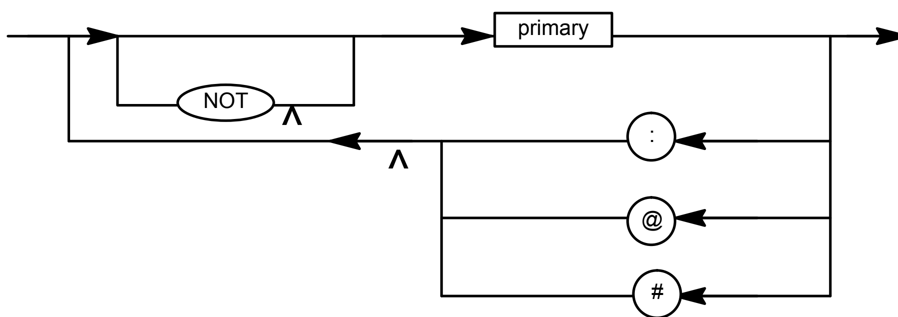
sum



product

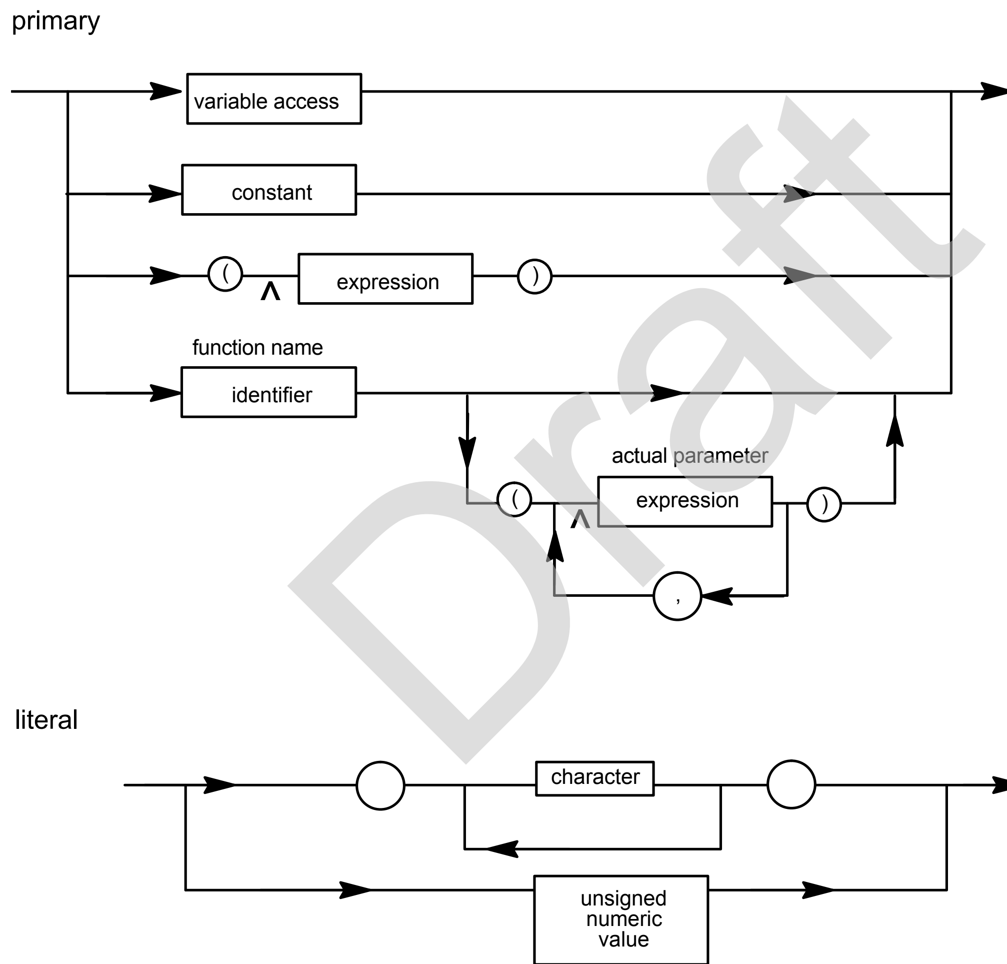


factor



Draft

Figure E–25.



Draft

Glossary

Draft

A

abort

Abnormal termination of a computer program caused by hardware or software malfunction or operator cancellation.

absolute pulse code system

A positional information system for servomotors that relies on battery-backed RAM to store encoder pulse counts when the robot is turned off. This system is calibrated when it is turned on.

A/D value

An analog to digital-value. Converts a multilevel analog electrical system pattern into a digital bit.

AI

Analog input.

AO

Analog output.

alarm

The difference in value between actual response and desired response in the performance of a controlled machine, system or process. Alarm=Error.

algorithm

A fixed step-by-step procedure for accomplishing a given result.

alphanumeric

Data that are both alphabetical and numeric.

Draft

AMPS

Amperage amount.

analog

The representation of numerical quantities by measurable quantities such as length, voltage or resistance. Also refers to analog type I/O blocks and distinguishes them from discrete I/O blocks. Numerical data that can vary continuously, for example, voltage levels that can vary within the range of -10 to +10 volts.

AND

An operation that places two contacts or groups of contacts in series. All contacts in series control the resulting status and also mathematical operator.

ANSI

American National Standard Institute, the U.S. government organization with responsibility for the development and announcement of technical data standards.

APC

See absolute pulse code system.

APC motor

See servomotor.

application program

The set of instructions that defines the specific intended tasks of robots and robot systems to make them reprogrammable and multifunctional. You can initiate and change these programs.

arm

A robot component consisting of an interconnecting set of links and powered joints that move and support the wrist socket and end effector.

articulated arm

A robot arm constructed to simulate the human arm, consisting of a series of rotary motions and joints, each powered by a motor.

ASCII

Abbreviation for American Standard Code for Information Interchange. An 8-level code (7 bits plus 1 parity bit) commonly used for the exchange of data.

automatic mode

The robot state in which automatic operation can be initiated.

automatic operation

The time during which robots are performing programmed tasks through unattended program execution.

axis

1. A straight line about which a robot joint rotates or moves. 2. One of the reference lines or a coordinate system. 3. A single joint on the robot arm.

B

backplane

A group of connectors mounted at the back of a controller rack to which printed circuit boards are mated.

BAR

A unit of pressure equal to 100,000 pascals.

barrier

A means of physically separating persons from the restricted work envelope; any physical boundary to a hazard or electrical device/component.

battery low alarm

A programmable value (in engineering units) against which the analog input signal automatically is compared on Genius I/O blocks. A fault is indicated if the input value is equal to or less than the low alarm value.

baud

A unit of transmission speed equal to the number of code elements (bits) per second.

big-endian

The adjectives big-endian and little-endian refer to which bytes are most significant in multi-byte data types and describe the order in which a sequence of bytes is stored in a computer's memory. In a big-endian system, the most significant value in the sequence is stored at the lowest storage address (i.e., first). In a little-endian system, the least significant value in the sequence is stored first.

binary

A numbering system that uses only 0 and 1.

bit

Contraction of binary digit. 1. The smallest unit of information in the binary numbering system, represented by a 0 or 1. 2. The smallest division of a programmable controller word.

bps

Bits per second.

buffer

A storage area in the computer where data is held temporarily until the computer can process it.

bus

A channel along which data can be sent.

Draft

bus controller
A Genius bus interface board for a programmable controller.

bus scan
One complete communications cycle on the serial bus.

Bus Switching Module
A device that switches a block cluster to one bus or the other of a dual bus.

byte
A sequence of binary digits that can be used to store a value from 0 to 255 and usually operated upon as a unit. Consists of eight bits used to store two numeric or one alpha character.

C

calibration
The process whereby the joint angle of each axis is calculated from a known reference point.

Cartesian coordinate system
A coordinate system whose axes (x, y, and z) are three intersecting perpendicular straight lines. The origin is the intersection of the axes.

Cartesian coordinates
A set of three numbers that defines the location of a point within a rectilinear coordinate system and consisting of three perpendicular axes (x, y, z).

cathode ray tube
A device, like a television set, for displaying information.

central processing unit
The main computer component that is made up of a control section and an arithmetic-logic section. The other basic units of a computer system are input/output units and primary storage.

channel
The device along which data flow between the input/output units of a computer and primary storage.

character
One of a set of elements that can be arranged in ordered groups to express information. Each character has two forms: 1. a man-intelligible form, the graphic, including the decimal digits 0-9, the letters A-Z, punctuation marks, and other formatting and control symbols; 2. a computer intelligible form, the code, consisting of a group of binary digits (bits).

circular
A MOTYPE option in which the robot tool center point moves in an arc defined by three points. These points can be positions or path nodes.

Draft

clear

To replace information in a storage unit by zero (or blank, in some machines).

closed loop

A control system that uses feedback. An open loop control system does not use feedback.

C-MOS RAM

Complementary metal-oxide semiconductor random-access memory. A read/write memory in which the basic memory cell is a pair of MOS (metal-oxide semiconductor) transistors. It is an implementation of S-RAM that has very low power consumption, but might be less dense than other S-RAM implementations.

coaxial cable

A transmission line in which one conductor is centered inside and insulated from an outer metal tube that serves as the second conductor. Also known as coax, coaxial line, coaxial transmission line, concentric cable, concentric line, concentric transmission line.

component

An inclusive term used to identify a raw material, ingredient, part or subassembly that goes into a higher level of assembly, compound or other item.

computer

A device capable of accepting information, applying prescribed processes to the information, and supplying the results of these processes.

configuration

The joint positions of a robot and turn number of wrist that describe the robot at a specified position. Configuration is designated by a STRING value and is included in positional data.

continuous path

A trajectory control system that enables the robot arm to move at a constant tip velocity through a series of predefined locations. A rounding effect of the path is required as the tip tries to pass through these locations.

continuous process control

The use of transducers (sensors) to monitor a process and make automatic changes in operations through the design of appropriate feedback control loops. While such devices historically have been mechanical or electromechanical, microcomputers and centralized control is now used, as well.

continuous production

A production system in which the productive equipment is organized and sequenced according to the steps involved to produce the product. Denotes that material flow is continuous during the production process. The routing of the jobs is fixed and set-ups are seldom changed.

controlled stop

A controlled stop controls robot deceleration until it stops. When a safety stop input such as a safety fence signal is opened, the robot decelerates in a controlled manner and then stops. After the robot stops, the Motor Control Contactor opens and drive power is removed.

Draft

controller

A hardware unit that contains the power supply, operator controls, control circuitry, and memory that directs the operation and motion of the robot and communications with external devices. See control unit.

controller memory

A medium in which data are retained. Primary storage refers to the internal area where the data and program instructions are stored for active use, as opposed to auxiliary or external storage (magnetic tape, disk, diskette, and so forth.)

control, open-loop

An operation where the computer applies control directly to the process without manual intervention.

control unit

The portion of a computer that directs the automatic operation of the computer, interprets computer instructions, and initiates the proper signals to the other computer circuits to execute instructions.

coordinate system

See Cartesian coordinate system.

CPU

See central processing unit.

CRT

See cathode ray tube.

cps (viscosity)

Centipoises per second.

CRT/KB

Cathode ray tube/keyboard. An optional interface device for the robot system. The CRT/KB is used for some robot operations and for entering programs. It can be a remote device that attaches to the robot via a cable.

cycle

1. A sequence of operations that is repeated regularly. The time it takes for one such sequence to occur. 2. The interval of time during which a system or process, such as seasonal demand or a manufacturing operation, periodically returns to similar initial conditions. 3. The interval of time during which an event or set of events is completed. In production control, a cycle is the length of time between the release of a manufacturing order and shipment to the customer or inventory.

cycle time

1. In industrial engineering, the time between completion of two discrete units of production. 2. In materials management, the length of time from when material enters a production facility until it exits. See throughput.

Draft

cursor

An indicator on a teach pendant or CRT display screen at which command entry or editing occurs. The indicator can be a highlighted field or an arrow (> or ^).

cylindrical

Type of work envelope that has two linear major axes and one rotational major axis. Robotic device that has a predominantly cylindrical work envelope due to its design. Typically has fewer than 6 joints and typically has only 1 linear axis.

D

D/A converter

A digital-to-analog converter. A device that transforms digital data into analog data.

D/A value

A digital-to-analog value. Converts a digital bit pattern into a multilevel analog electrical system.

daisy chain

A means of connecting devices (readers, printers, etc.) to a central processor by party-line input/output buses that join these devices by male and female connectors. The last female connector is shorted by a suitable line termination.

daisy chain configuration

A communications link formed by daisy chain connection of twisted pair wire.

data

A collection of facts, numeric and alphabetical characters, or any representation of information that is suitable for communication and processing.

data base

A data file philosophy designed to establish the independence of computer program from data files. Redundancy is minimized and data elements can be added to, or deleted from, the file designs without changing the existing computer programs.

DC

Abbreviation for direct current.

DEADMAN switch

A control switch on the teach pendant that is used to enable servo power. Pressing the DEADMAN switch while the teach pendant is on activates servo power and releases the robot brakes; releasing the switch deactivates servo power and applies the robot brakes.

debugging

The process of detecting, locating and removing mistakes from a computer program, or manufacturing control system. See diagnostic routine.

Draft

deceleration tolerance
The specification of the percentage of deceleration that must be completed before a motion is considered finished and another motion can begin.

default
The value, display, function or program automatically selected if you have not specified a choice.

deviation
Usually, the absolute difference between a number and the mean of a set of numbers, or between a forecast value and the actual data.

device
Any type of control hardware, such as an emergency-stop button, selector switch, control pendant, relay, solenoid valve, or sensor.

diagnostic routine
A test program used to detect and identify hardware/software malfunctions in the controller and its associated I/O equipment. See debugging.

diagnostics
Information that permits the identification and evaluation of robot and peripheral device conditions.

digital
A description of any data that is expressed in numerical format. Also, having the states On and Off only.

digital control
The use of a digital computer to perform processing and control tasks in a manner that is more accurate and less expensive than an analog control system.

digital signal
A single point control signal sent to or from the controller. The signal represents one of two states: ON (TRUE, 1. or OFF (FALSE, 0).

directory
A listing of the files stored on a device.

discrete
Consisting of individual, distinct entities such as bits, characters, circuits, or circuit components. Also refers to ON/OFF type I/O blocks.

disk
A secondary memory device in which information is stored on a magnetically sensitive, rotating disk.

disk memory
A non-programmable, bulk-storage, random-access memory consisting of a magnetized coating on one or both sides of a rotating thin circular plate.

Draft

drive power

The energy source or sources for the robot servomotors that produce motion.

DRAM

Dynamic Random Access Memory. A read/write memory in which the basic memory cell is a capacitor. DRAM (or D-RAM) tends to have a higher density than SRAM (or S-RAM). Due to the support circuitry required, and power consumption needs, it is generally impractical to use. A battery can be used to retain the content upon loss of power.

E

edit

1. A software mode that allows creation or alteration of a program. 2. To modify the form or format of data, for example, to insert or delete characters.

emergency stop

The operation of a circuit using hardware-based components that overrides all other robot controls, removes drive power from the actuators, and causes all moving parts of to stop. The operator panel and teach pendant are each equipped with EMERGENCY STOP buttons.

enabling device

A manually operated device that, when continuously activated, permits motion. Releasing the device stops the motion of the robot and associated equipment that might present a hazard.

encoder

1. A device within the robot that sends the controller information about where the robot is. 2. A transducer used to convert position data into electrical signals. The robot system uses an incremental optical encoder to provide position feedback for each joint. Velocity data is computed from the encoder signals and used as an additional feedback signal to assure servo stability.

end effector

An accessory device or tool specifically designed for attachment to the robot wrist or tool mounting plate to enable the robot to perform its intended tasks. Examples include gripper, spot weld gun, arc weld gun, spray paint gun, etc.

end-of-arm tooling

Any of a number of tools, such as welding guns, torches, bells, paint spraying devices, attached to the faceplate of the robot wrist. Also called end effector or EOAT.

engineering units

Units of measure as applied to a process variable, for example, psi, Degrees F., etc.

envelope, maximum

The volume of space encompassing the maximum designed movements of all robot parts including the end effector, workpiece, and attachments.

Draft

EOAT

See end of arm tooling, tool.

EPROM

Erasable Programmable Read Only Memory. Semiconductor memory that can be erased and reprogrammed. A non-volatile storage memory.

error

The difference in value between actual response and desired response in the performance of a controlled machine, system or process. Alarm=Error.

error message

A numbered message, displayed on the CRT/KB and teach pendant, that indicates a system problem or warns of a potential problem.

Ethernet

A Local Area Network (LAN) bus-oriented, hardware technology that is used to connect computers, printers, terminal concentrators (servers), and many other devices together. It consists of a master cable and connection devices at each machine on the cable that allow the various devices to "talk" to each other. Software that can access the Ethernet and cooperate with machines connected to the cable is necessary. Ethernets come in varieties such as baseband and broadband and can run on different media, such as coax, twisted pair and fiber. Ethernet is a trademark of Xerox Corporation.

execute

To perform a specific operation, such as one that would be accomplished through processing one statement or command, a series of statements or commands, or a complete program or command procedure.

extended axis

An optional, servo-controlled axis that provides extended reach capability for a robot, including in-booth rail, single- or double-link arm, also used to control motion of positioning devices.

F

faceplate

The tool mounting plate of the robot.

feedback

1. The signal or data fed back to a commanding unit from a controlled machine or process to denote its response to the command signal. The signal representing the difference between actual response and desired response that is used by the commanding unit to improve performance of the controlled machine or process. 2. The flow of information back into the control system so that actual performance can be compared with planned performance, for instance in a servo system.

Draft

field

A specified area of a record used for a particular category of data. 2. A group of related items that occupy the same space on a CRT/KB screen or teach pendant LCD screen. Field name is the name of the field; field items are the members of the group.

field devices

User-supplied devices that provide information to the PLC (inputs: push buttons, limit switches, relay contacts, an so forth) or perform PLC tasks (outputs: motor starters, solenoids, indicator lights, and so forth.)

file

1. An organized collection of records that can be stored or retrieved by name. 2. The storage device on which these records are kept, such as bubble memory or disk.

filter

A device to suppress interference that would appear as noise.

Flash File Storage

A portion of FROM memory that functions as a separate storage device. Any file can be stored on the FROM disk.

Flash ROM

Flash Read Only Memory. Flash ROM is not battery-backed memory but it is non-volatile. All data in Flash ROM is saved even after you turn off and turn on the robot.

flow chart

A systems analysis tool to graphically show a procedure in which symbols are used to represent operations, data, flow, and equipment. See block diagram, process chart.

flow control

A specific production control system that is based primarily on setting production rates and feeding work into production to meet the planned rates, then following it through production to make sure that it is moving. This concept is most successful in repetitive production.

format

To set up or prepare a memory card or floppy disk (not supported with version 7.20 and later) so it can be used to store data in a specific system.

FR

See Flash ROM.

F-ROM

See Flash ROM.

FROM disk

See Flash ROM.

Draft

G

general override stat

A percentage value that governs the maximum robot jog speed and program run speed.

Genius I/O bus

The serial bus that provides communications between blocks, controllers, and other devices in the system especially with respect to GE FANUC Genius I/O.

gripper

The "hand" of a robot that picks up, holds and releases the part or object being handled. Sometimes referred to as a manipulator. See EOAT, tool.

group signal

An input/output signal that has a variable number of digital signals, recognized and taken as a group.

gun

See applicator.

H

Hand Model.

Used in Interference Checking, the Hand Model is the set of virtual model elements (spheres and cylinders) that are used to represent the location and shape of the end of arm tooling with respect to the robot's faceplate.

hardware

1. In data processing, the mechanical, magnetic, electrical and electronic devices of which a computer, controller, robot, or panel is built. 2. In manufacturing, relatively standard items such as nuts, bolts, washers, clips, and so forth.

hard-wire

To connect electric components with solid metallic wires.

hard-wired

1. Having a fixed wired program or control system built in by the manufacturer and not subject to change by programming. 2. Interconnection of electrical and electronic devices directly through physical wiring.

hazardous motion

Unintended or unexpected robot motion that can cause injury.

hexadecimal

A numbering system having 16 as the base and represented by the digits 0 through 9, and A through F.

Draft

hold

A smoothly decelerated stopping of all robot movement and a pause of program execution. Power is maintained on the robot and program execution generally can be resumed from a hold.

HTML.

Hypertext Markup Language. A markup language that is used to create hypertext and hypermedia documents incorporating text, graphics, sound, video, and hyperlinks.

http.

Hypertext transfer protocol. The protocol used to transfer HTML files between web servers.

I

impedance

A measure of the total opposition to current flow in an electrical circuit.

incremental encoder system

A positional information system for servomotors that requires calibrating the robot by moving it to a known reference position (indicated by limit switches) each time the robot is turned on or calibration is lost due to an error condition.

index

An integer used to specify the location of information within a table or program.

index register

A memory device containing an index.

industrial robot

A reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions in order to perform a variety of tasks.

industrial robot system

A system that includes industrial robots, end effectors, any equipment devices and sensors required for the robot to perform its tasks, as well as communication interfaces for interlocking, sequencing, or monitoring the robot.

information

The meaning derived from data that have been arranged and displayed in a way that they relate to that which is already known. See data.

initialize

1. Setting all variable areas of a computer program or routine to their desired initial status, generally done the first time the code is executed during each run. 2. A program or hardware circuit that returns a program a system, or hardware device to an original state. See startup, initial.

Draft

input

The data supplied from an external device to a computer for processing. The device used to accomplish this transfer of data.

input device

A device such as a terminal keyboard that, through mechanical or electrical action, converts data from the form in which it has been received into electronic signals that can be interpreted by the CPU or programmable controller. Examples are limit switches, push buttons, pressure switches, digital encoders, and analog devices.

input processing time

The time required for input data to reach the microprocessor.

input/output

Information or signals transferred between devices, discreet electrical signals for external control.

input/output control

A technique for controlling capacity where the actual output from a work center is compared with the planned output developed by CRP. The input is also monitored to see if it corresponds with plans so that work centers will not be expected to generate output when jobs are not available to work on.

integrated circuit

A solid-state micro-circuit contained entirely within a chip of semiconductor material, generally silicon. Also called chip.

interactive

Refers to applications where you communicate with a computer program via a terminal by entering data and receiving responses from the computer.

interface

1. A concept that involves the specifications of the inter-connection between two equipments having different functions. 2. To connects a PLC with the application device, communications channel, and peripherals through various modules and cables. 3. The method or equipment used to communicate between devices.

interference zone

An area that falls within the work envelope of a robot, in which there is the potential for the robot motion to coincide with the motion of another robot or machine, and for a collision to occur.

interlock

An arrangement whereby the operation of one control or mechanism brings about, or prevents, the operations of another.

interrupt

A break in the normal flow of a system or program that occurs in a way that the flow can be resumed from that point at a later time. Interrupts are initiated by two types of signals: 1. signals originating within the computer system to synchronize the operation of the computer system with the outside

Draft

world; 2. signals originating exterior to the computer system to synchronize the operation of the computer system with the outside world.

I/O

Abbreviation for input/output or input/output control.

I/O block

A microprocessor-based, configurable, rugged solid state device to which field I/O devices are attached.

I/O electrical isolation

A method of separating field wiring from logic level circuitry. This is typically done through optical isolation devices.

I/O module

A printed circuit assembly that is the interface between user devices and the Series Six PLC.

I/O scan

A method by which the CPU monitors all inputs and controls all outputs within a prescribed time. A period during which each device on the bus is given a turn to send information and listen to all of the broadcast data on the bus.

ISO

The International Standards Organization that establishes the ISO interface standards.

isolation

1. The ability of a logic circuit having more than one inputs to ensure that each input signal is not affected by any of the others. 2. A method of separating field wiring circuitry from logic level circuitry, typically done optically.

item

1. A category displayed on the teach pendant on a menu. 2. A set of adjacent digits, bits, or characters that is treated as a unit and conveys a single unit of information. 3. Any unique manufactured or purchased part or assembly: end product, assembly, subassembly, component, or raw material.

J

jog coordinate systems

Coordinate systems that help you to move the robot more effectively for a specific application. These systems include JOINT, WORLD, TOOL, and USER.

JOG FRAME

A jog coordinate system you define to make the robot jog the best way possible for a specific application. This can be different from world coordinate frame.

jogging

Pressing special keys on the teach pendant to move the robot.

Draft

jog speed

Is a percentage of the maximum speed at which you can jog the robot.

joint

1. A single axis of rotation. There are up to six joints in a robot arm (P-155 swing arm has 8). 2. A jog coordinate system in which one axis is moved at a time.

JOINT

A motion type in which the robot moves the appropriate combination of axes independently to reach a point most efficiently. (Point to point, non-linear motion).

joint interpolated motion

A method of coordinating the movement of the joints so all joints arrive at the desired location at the same time. This method of servo control produces a predictable path regardless of speed and results in the fastest cycle time for a particular move. Also called joint motion.

K

K

Abbreviation for kilo, or exactly 1024 in computer jargon. Related to 1024 words of memory.

KAREL

The programming language developed for robots by the FANUC Robotics America, Inc.

L

label

An ordered set of characters used to symbolically identify an instruction, a program, a quantity, or a data area.

LCD

See liquid crystal display.

lead time

The span of time needed to perform an activity. In the production and inventory control context, this activity is normally the procurement of materials and/or products either from an outside supplier or from one’s own manufacturing facility. Components of lead time can include order preparation time, queue time, move or transportation time, receiving and inspection time.

LED

See Light Emitting Diode.

LED display

An alphanumeric display that consists of an array of LEDs.

Light Emitting Diode

A solid-state device that lights to indicate a signal on electronic equipment.

Draft

limiting device

A device that restricts the work envelope by stopping or causing to stop all robot motion and that is independent of the control program and the application programs.

limit switch

A switch that is actuated by some part or motion of a machine or equipment to alter the electrical circuit associated with it. It can be used for position detection.

linear

A motion type in which the appropriate combination of axes move in order to move the robot TCP in a straight line while maintaining tool center point orientation.

liquid crystal display

A digital display on the teach pendant that consists of two sheets of glass separated by a sealed-in, normally transparent, liquid crystal material. Abbreviated LCD.

little-endian

The adjectives big-endian and little-endian refer to which bytes are most significant in multi-byte data types and describe the order in which a sequence of bytes is stored in a computer's memory. In a big-endian system, the most significant value in the sequence is stored at the lowest storage address (i.e., first). In a little-endian system, the least significant value in the sequence is stored first.

load

1. The weight (force) applied to the end of the robot arm. 2. A device intentionally placed in a circuit or connected to a machine or apparatus to absorb power and convert it into the desired useful form. 3. To copy programs or data into memory storage.

location

1. A storage position in memory uniquely specified by an address. 2. The coordinates of an object used in describing its x, y, and z position in a Cartesian coordinate system.

lockout/tagout

The placement of a lock and/or tag on the energy isolating device (power disconnecting device) in the off or open position. This indicates that the energy isolating device or the equipment being controlled will not be operated until the lock/tag is removed.

log

A record of values and/or action for a given function.

logic

A fixed set of responses (outputs) to various external conditions (inputs). Also referred to as the program.

loop

The repeated execution of a series of instructions for a fixed number of times, or until interrupted by the operator.

Draft

M

mA

See milliampere.

machine language

A language written in a series of bits that are understandable by, and therefore instruct, a computer. This is a "first level" computer language, as compared to a "second level" assembly language, or a "third level" compiler language.

machine lock

A test run option that allows the operator to run a program without having the robot move.

macro

A source language instruction from which many machine-language instructions can be generated.

magnetic disk

A metal or plastic floppy disk (not supported on version 7.10 and later) that looks like a phonograph record whose surface can store data in the form of magnetized spots.

magnetic disk storage

A storage device or system consisting of magnetically coated metal disks.

magnetic tape

Plastic tape, like that used in tape recorder, on which data is stored in the form of magnetized spots.

maintenance

Keeping the robots and system in their proper operating condition.

MC

See memory card.

mechanical unit

The robot arm, including auxiliary axis, and hood/deck and door openers.

medium

plural **media** . The physical substance upon which data is recorded, such as a memory card (or floppy disk which is not supported on version 7.10 and later).

memory

A device or media used to store information in a form that can be retrieved and is understood by the computer or controller hardware. Memory on the controller includes C-MOS RAM, Flash ROM and D-RAM.

memory card

A C-MOS RAM memory card or a flash disk-based PC card.

Draft

menu

A list of options displayed on the teach pendant screen.

message

A group of words, variable in length, transporting an item of information.

microprocessor

A single integrated circuit that contains the arithmetic, logic, register, control and memory elements of a computer.

microsecond

One millionth (0.000001) of a second

milliampere

One one-thousandth of an ampere. Abbreviated mA.

millisecond

One thousandth of a second. Abbreviated msec.

module

A distinct and identifiable unit of computer program for such purposes as compiling, loading, and linkage editing. It is eventually combined with other units to form a complete program.

motion type

A feature that allows you to select how you want the robot to move from one point to the next. MOTYPES include joint, linear, and circular.

mode

1. One of several alternative conditions or methods of operation of a device. 2. The most common or frequent value in a group of values.

N

network

1. The interconnection of a number of devices by data communication facilities. "Local networking" is the communications network internal to a robot."Global networking" is the ability to provide communications connections outside of the robot's internal system. 2. Connection of geographically separated computers and/or terminals over communications lines. The control of transmission is managed by a standard protocol.

non-volatile memory

Memory capable of retaining its stored information when power is turned off.

O

Draft

Obstacle Model.

Used in Interference Checking, the Obstacle Model is the set of virtual model elements (spheres, cylinders, and planes) that are used to represent the shape and the location of a given obstacle in space.

off-line

Equipment or devices that are not directly connected to a communications line.

off-line operations

Data processing operations that are handled outside of the regular computer program. For example, the computer might generate a report off-line while the computer was doing another job.

off-line programming

The development of programs on a computer system that is independent of the "on-board" control of the robot. The resulting programs can be copied into the robot controller memory.

offset

The count value output from a A/D converter resulting from a zero input analog voltage. Used to correct subsequent non-zero measurements also incremental position or frame adjustment value.

on-line

A term to describe equipment or devices that are connected to the communications line.

on-line processing

A data processing approach where transactions are entered into the computer directly, as they occur.

operating system

Lowest level system monitor program.

operating work envelope

The portion of the restricted work envelope that is actually used by the robot while it is performing its programmed motion. This includes the maximum the end-effector, the workpiece, and the robot itself.

operator

A person designated to start, monitor, and stop the intended productive operation of a robot or robot system.

operator box

A control panel that is separate from the robot and is designed as part of the robot system. It consists of the buttons, switches, and indicator lights needed to operate the system.

operator panel

A control panel designed as part of the robot system and consisting of the buttons, switches, and indicator lights needed to operate the system.

optional features

Additional capabilities available at a cost above the base price.

Draft

OR

An operation that places two contacts or groups of contacts in parallel. Any of the contacts can control the resultant status, also a mathematical operation.

orientation

The attitude of an object in space. Commonly described by three angles: rotation about x (w), rotation about y (p), and rotation about z (r).

origin

The point in a Cartesian coordinate system where axes intersect; the reference point that defines the location of a frame.

OT

See overtravel.

output

Information that is transferred from the CPU for control of external devices or processes.

output device

A device, such as starter motors, solenoids, that receive data from the programmable controller.

output module

An I/O module that converts logic levels within the CPU to a usable output signal for controlling a machine or process .

outputs

Signals, typically on or off, that controls external devices based upon commands from the CPU.

override

See general override.

overtravel

A condition that occurs when the motion of a robot axis exceeds its prescribed limits.

overwrite

To replace the contents of one file with the contents of another file when copying.

P

parity

The anticipated state, odd or even, of a set of binary digits.

parity bit

A binary digit added to an array of bits to make the sum of all bits always odd or always even.

parity check

A check that tests whether the number of ones (or zeros) in an array of binary digits is odd or even.

Draft

parity error

A condition that occurs when a computed parity check does not agree with the parity bit.

part

A material item that is used as a component and is not an assembly or subassembly.

pascal

A unit of pressure in the meter-kilogram-second system equivalent to one newton per square meter.

path

1. A variable type available in the KAREL system that consists of a list of positions. Each node includes positional information and associated data. 2. The trajectory followed by the TCP in a move.

PCB

See printed circuit board.

PC Interface

The PC Interface software uses Ethernet connections to provide file transfer protocol (FTP) functions, PC send macros, telnet interface, TCP/IP interface web server functions, and host communications.

pendant

See teach pendant.

PLC

See programmable logic controller or cell controller.

PMC

The programmable machine controller (PMC) functions provide a ladder logic programming environment to create PMC functions. This provides the capability to use the robot I/O system to run PLC programs in the background of normal robot operations. This function can be used to control bulk supply systems, fixed automation that is part of the robot workcell, or other devices that would normally require basic PLC controls.

printed circuit board

A flat board whose front contains slots for integrated circuit chips and connections for a variety of electronic components, and whose back is printed with electrically conductive pathways between the components.

production mode

See automatic mode.

program

1. A plan for the solution of a problem. A complete program includes plans for the transcription of data, coding for the computer, and plans for the absorption of the results into the system. 2. A sequence of instructions to be executed by the computer or controller to control a robot/robot system. 3. To furnish a computer with a code of instructions. 4. To teach a robot system a specific set of movements and instructions to do a task.

Draft

programmable controller

See programmable logic controller or cell controller.

programmable logic controller

A solid-state industrial control device that receives inputs from user-supplied control devices, such as switches and sensors, implements them in a precise pattern determined by ladder diagram-based programs stored in the user memory, and provides outputs for control of processes or user-supplied devices such as relays and motor starters.

Program ToolBox

The Program ToolBox software provides programming utilities such as mirror image and flip wrist editing capabilities.

protocol

A set of hardware and software interfaces in a terminal or computer that allows it to transmit over a communications network, and that collectively forms a communications language.

psi

Pounds per square inch.

Draft

Q

queue.

1. Waiting lines resulting from temporary delays in providing service. 2. The amount of time a job waits at a work center before set-up or work is performed on the job. See also job queue.

R

RAM

See Random Access Memory.

random access

A term that describes files that do not have to be searched sequentially to find a particular record but can be addressed directly.

Random Access Memory

1. Volatile, solid-state memory used for storage of programs and locations; battery backup is required. 2. The working memory of the controller. Programs and variable data must be loaded into RAM before the program can execute or the data can be accessed by the program.

range

1. A characterization of a variable or function. All the values that a function can possess. 2. In statistics, the spread in a series of observations. 3. A programmable voltage or current spectrum of values to which input or output analog signals can be limited.

RI

Robot input.

RO

Robot output.

read

To copy, usually from one form of storage to another, particularly from external or secondary storage to internal storage. To sense the meaning of arrangements of hardware. To sense the presence of information on a recording medium.

Read Only Memory

A digital memory containing a fixed pattern of bits that you cannot alter.

record

To store the current set or sets of information on a storage device.

recovery

The restoration of normal processing after a hardware or software malfunction through detailed procedures for file backup, file restoration, and transaction logging.

register

1. A special section of primary storage in a computer where data is held while it is being worked on.
2. A memory device capable of containing one or more computer bits or words.

remote/local

A device connection to a given computer, with remote devices being attached over communications lines and local devices attached directly to a computer channel; in a network, the computer can be a remote device to the CPU controlling the network.

repair

To restore robots and robot systems to operating condition after damage, malfunction, or wear.

repeatability

The closeness of agreement among the number of consecutive movements made by the robot arm to a specific point.

reset

To return a register or storage location to zero or to a specified initial condition.

restricted work envelope

That portion of the work envelope to which a robot is restricted by limiting devices that establish limits that will not be exceeded in the event of any reasonably foreseeable failure of the robot or its controls. The maximum distance the robot can travel after the limited device is actuated defines the restricted work envelope of the robot.

RIA

Robotic Industries Association Subcommittee of the American National Standards Institute, Inc.

Draft

robot

A reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks.

Robot Model.

Used in Interference Checking, the Robot Model is the set of virtual model elements (sphere and cylinders) that are used to represent the location and shape of the robot arm with respect to the robot's base. Generally, the structure of a six axes robot can be accurately modeled as a series of cylinders and spheres. Each model element represents a link or part of the robot arm.

ROM

See Read Only Memory.

routine

1. A list of coded instructions in a program. 2. A series of computer instructions that performs a specific task and can be executed as often as needed during program execution.

S

saving data.

Storing program data in Flash ROM, to a floppy disk (not supported on version 7.10 and later), or memory card.

scfm

Standard cubic feet per minute.

scratch start

Allows you to enable and disable the automatic recovery function.

sensor

A device that responds to physical stimuli, such as heat, light, sound pressure, magnetism, or motion, and transmits the resulting signal or data for providing a measurement, operating a control or both. Also a device that is used to measure or adjust differences in voltage in order to control sophisticated machinery dynamically.

serial communication

A method of data transfer within a PLC whereby the bits are handled sequentially rather than simultaneously as in parallel transmission.

serial interface

A method of data transmission that permits transmitting a single bit at a time through a single line. Used where high speed input is not necessary.

Server Side Include (SSI)

A method of calling or "including" code into a web page.

Draft

servomotor

An electric motor that is controlled to produce precision motion. Also called a "smart" motor.

SI

System input.

signal

The event, phenomenon, or electrical quantity that conveys information from one point to another.

significant bit

A bit that contributes to the precision of a number. These are counted starting with the bit that contributes the most value, of "most significant bit", and ending with the bit that contributes the least value, or "least significant bit".

singulating

Separating parts into a single layer.

slip sheet

A sheet of material placed between certain layers of a unit load. Also known as tier sheet.

SO

System output.

specific gravity

The ratio of a mass of solid or liquid to the mass of an equal volume of water at 45C. You must know the specific gravity of the dispensing material to perform volume signal calibration. The specific gravity of a dispensing material is listed on the MSDS for that material.

SRAM

A read/write memory in which the basic memory cell is a transistor. SRAM (or S-RAM) tends to have a lower density than DRAM. A battery can be used to retain the content upon loss of power.

slpm

Standard liters per minute.

Standard Operator Panel (SOP).

A panel that is made up of buttons, keyswitches, and connector ports.

state

The on or off condition of current to and from and input or output device.

statement

See instruction.

storage device

Any device that can accept, retain, and read back one or more times. The available storage devices are SRAM, Flash ROM (FROM or F-ROM), floppy disks (not available on version 7.10 and later), memory cards, or a USB memory stick.

Draft

system variable

An element that stores data used by the controller to indicate such things as robot specifications, application requirements, and the current status of the system.

T

Tare

The difference between the gross weight of an object and its contents, and the object itself. The weight of an object without its contents.

TCP

See tool center point.

teaching

Generating and storing a series of positional data points effected by moving the robot arm through a path of intended motions.

teach mode

1. The mode of operation in which a robot is instructed in its motions, usually by guiding it through these motions using a teach pendant. 2. The generation and storage of positional data. Positional data can be taught using the teach pendant to move the robot through a series of positions and recording those positions for use by an application program.

teach pendant

1. A hand-held device used to instruct a robot, specifying the character and types of motions it is to undertake. Also known as teach box, teach gun. 2. A portable device, consisting of an LCD display and a keypad, that serves as a user interface to the KAREL system and attaches to the operator box or operator panel via a cable. The teach pendant is used for robot operations such as jogging the robot, teaching and recording positions, and testing and debugging programs.

telemetry

The method of transmission of measurements made by an instrument or a sensor to a remote location.

termination type

Feature that controls the blending of robot motion between segments.

tool

A term used loosely to define something mounted on the end of the robot arm, for example, a hand, gripper, or an arc welding torch.

tool center point

1. The location on the end-effector or tool of a robot hand whose position and orientation define the coordinates of the controlled object. 2. Reference point for position control, that is, the point on the tool that is used to teach positions. Abbreviated TCP.

Draft

TOOL Frame

The Cartesian coordinate system that has the position of the TCP as its origin to stet. The z-axis of the tool frame indicates the approach vector for the tool.

TP.

See teach pendant.

transducer

A device for converting energy from one form to another.

U

UOP

See user operator panel.

URL

Universal Resource Locator. A standard addressing scheme used to locate or reference files on web servers.

USB memory stick

The controller USB memory stick interface supports a USB 1.1 interface. The USB Organization specifies standards for USB 1.1 and 2.0. Most memory stick devices conform to the USB 2.0 specification for operation and electrical standards. USB 2.0 devices as defined by the USB Specification must be backward compatible with USB 1.1 devices. However, FANUC Robotics does not support any security or encryption features on USB memory sticks. The controller supports most widely-available USB Flash memory sticks from 32MB up to 1GB in size.

USER Frame

The Cartesian coordinate system that you can define for a specific application. The default value of the User Frame is the World Frame. All positional data is recorded relative to User Frame.

User Operator Panel

User-supplied control device used in place of or in parallel with the operator panel or operator box supplied with the controller. Abbreviated UOP .

V

variable

A quantity that can assume any of a given set of values.

variance

The difference between the expected (or planned) and the actual, also statistics definitions.

vision system

A device that collects data and forms an image that can be interpreted by a robot computer to determine the position or to “see” an object.

Draft

volatile memory
Memory that will lose the information stored in it if power is removed from the memory circuit device.

W

web server
An application that allows you to access files on the robot using a standard web browser.

warning device
An audible or visible device used to alert personnel to potential safety hazards.

work envelope
The volume of space that encloses the maximum designed reach of the robot manipulator including the end effector, the workpiece, and the robot itself. The work envelope can be reduced or restricted by limiting devices. The maximum distance the robot can travel after the limit device is actuated is considered the basis for defining the restricted work envelope.

write
To deliver data to a medium such as storage.

Draft

Draft

Index

D

DOSFILE_INF
    built-in procedure, A–127
DRAM, 1–8

E

Ethernet Device, 9–7, 9–11

F

F-ROM Disk, 9–2
file system, 9–2
filtered memory device, 9–7
flash file storage disk, 1–9
Flash File Storage disk (FR), 9–7, 9–9
Flash Rom (F-ROM), 1–8
FMD device, *see* filtered memory device

I

*i*RVision, A–345, A–347 to A–348, A–350 to A–351, A–353 to A–354, A–362 to A–363

J

J957 option, 9–9

M

memory, 1–8
memory card, 9–2
memory card (MC), 9–7
memory device, 9–7, 9–11
    filtered, 9–7
memory device backup, 9–11
memory device binary, 9–7
memory stick
    USB, 9–3

P

personal computer, 9–2

R

R545
    order number, 9–12
R632
    order number, 1–2
R648
    order number, 11–2
RAM Disk, 9–2
RAM Disk (RD), 9–7, 9–10

S

semaphore, 15–9
socket messaging
    configuring, 11–3
    KAREL, 11–9
    network performance, 11–10
    overview, 11–2
    programming examples, 11–11
    system requirements, 11–2
SRAM, 1–8 to 1–9
storage
    file system, 9–2
storage devices
    overview, 9–7

U

USB memory stick, 9–3
USB Memory Stick Device (UD1), 9–7 to 9–8
USB Memory Stick Device (UT1), 9–7

V

V_CAM_CALIB

Draft

built-in procedure, A–345
V_GET_OFFSET
built-in procedure, A–347
V_GET_PASSFL
built-in procedure, A–348
V_GET_QUEUE
built-in procedure, A–350
V_INIT_QUEUE
built-in procedure, A–350
V_RALC_QUEUE
built-in procedure, A–351
V_RUN_FIND
built-in procedure, A–351
V_SET_REF
built-in procedure, A–353
V_START_VTRK
built-in procedure, A–354
V_STOP_VTRK
built-in procedure, A–354
VAR_INFO
built-in procedure, A–355
VAR_LIST
built-in procedure, A–357
VECTOR
data type, A–360
VOL_SPACE
built-in procedure, A–361
VREG_FND_POS
built-in procedure, A–362
VREG_OFFSET
built-in procedure, A–363

Draft