About This Manual

This manual can be used with controllers labeled R-30iA or R-J3iC. If you have a controller labeled R-J3iC, you should read R-30iA as R-J3iC throughout this manual.

Copyrights and Trademarks

This new publication contains proprietary information of FANUC Robotics America Corporation, furnished for customer use only. No other uses are authorized without the express written permission of FANUC Robotics America Corporation.

FANUC Robotics America Corporation
3900 W. Hamlin Road
Rochester Hills, MI 48309-3253

The descriptions and specifications contained in this manual were in effect at the time this manual was approved. FANUC Robotics America Corporation, hereinafter referred to as FANUC Robotics, reserves the right to discontinue models at any time or to change specifications or design without notice and without incurring obligations.

FANUC Robotics manuals present descriptions, specifications, drawings, schematics, bills of material, parts, connections and/or procedures for installing, disassembling, connecting, operating and programming FANUC Robotics’ products and/or systems. Such systems consist of robots, extended axes, robot controllers, application software, the KAREL® programming language, INSIGHT® vision equipment, and special tools.

FANUC Robotics recommends that only persons who have been trained in one or more approved FANUC Robotics Training Course(s) be permitted to install, operate, use, perform procedures on, repair, and/or maintain FANUC Robotics’ products and/or systems and their respective components. Approved training necessitates that the courses selected be relevant to the type of system installed and application performed at the customer site.

Warning

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A computing devices pursuant to subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of the equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measure may be required to correct the interference.
FANUC Robotics conducts courses on its systems and products on a regularly scheduled basis at its headquarters in Rochester Hills, Michigan. For additional information contact

FANUC Robotics America Corporation
3900 W. Hamlin Road
Rochester Hills, MI 48309-3253

www.fanucrobotics.com

For customer assistance, including Technical Support, Service, Parts & Part Repair, and Marketing Requests, contact the Customer Resource Center, 24 hours a day, at 1-800-47-ROBOT (1-800-477-6268). International customers should call 011-1-248-377-7159.

Send your comments and suggestions about this manual to: product.documentation@fanucrobotics.com

The information illustrated or contained herein is not to be reproduced, copied, downloaded, translated into another language, distributed, or published in any physical or electronic format, including Internet, or transmitted in whole or in part in any way without the prior written consent of FANUC Robotics America, Inc.

AccuStat®, ArcTool®, iRVision®, KAREL®, PaintTool®, PalletTool®, SOCKETS®, SpotTool®, SpotWorks®, and TorchMate® are Registered Trademarks of FANUC Robotics.

FANUC Robotics reserves all proprietary rights, including but not limited to trademark and trade name rights, in the following names:

AccuAir™, AccuCal™, AccuChop™, AccuFlow™, AccuPath™, AccuSeal™, ARC Mate™, ARC Mate Sr.™, ARC Mate System 1™, ARC Mate System 2™, ARC Mate System 3™, ARC Mate System 4™, ARC Mate System 5™, ARCWorks Pro™, AssistTool™, AutoNormal™, AutoTCP™, BellTool™, BODYWorks™, Cal Mate™, Cell Finder™, Center Finder™, Clean Wall™, DualARM™, LR Tool™, MIG Eye™, MotionParts™, MultiARM™, NoBots™, Paint Stick™, PaintPro™, PaintTool 100™, PAINTWorks™, PAINTWorks II™, PAINTWorks III™, PalletMate™, PalletMate PC™, PalletTool PC™, PayloadID™, RecipTool™, RemovalTool™, Robo Chop™, Robo Spray™, S-420i™, S-430i™, ShapeGen™, SoftFloat™, SOFT PARTSTM, SpotTool+™, SR Mate™, SR ShotTool™, SureWeld™, SYSTEM R-J2 Controller™, SYSTEM R-J3 Controller™, SYSTEM R-J3/B Controller™, SYSTEM R-J3/C Controller™, SYSTEM R-30iA Controller™, SYSTEM R-30iB Controller™, TCP Mate™, TorchMate™, TripleARMTM, TurboMove™, visLOC™, visPRO-3D™, visTRAC™, WebServer™, WebTP™, and YagTool™.

Patents

One or more of the following U.S. patents might be related to the FANUC Robotics products described in this manual.
FRA Patent List

4,630,567 4,639,878 4,707,647 4,708,580 4,942,539 4,984,745 5,238,029 5,239,739 5,272,805 5,293,107 5,293,911 5,331,264 5,367,944 5,373,221 5,421,218 5,434,489 5,644,898 5,670,202 5,696,687 5,737,218 5,823,389 5,853,027 5,887,800 5,942,539 5,987,726 6,059,092 6,064,168 6,070,109 6,122,062 6,204,620 6,243,621 6,253,799 6,285,920 6,313,595 6,325,302 6,345,818 6,356,807 6,360,143 6,378,190 6,385,508 6,425,177 6,477,913 6,490,369 6,518,980 6,541,757 6,569,258 6,612,449 6,703,079 6,705,361 6,726,773 6,768,078 6,845,295 6,945,483 7,149,606 7,149,606 7,211,978 7,266,422 7,399,363

FANUC LTD Patent List


VersaBell, ServoBell and SpeedDock Patents Pending.

**Conventions**

This manual includes information essential to the safety of personnel, equipment, software, and data. This information is indicated by headings and boxes in the text.
Warning

Information appearing under WARNING concerns the protection of personnel. It is boxed and in bold type to set it apart from other text.

Caution

Information appearing under CAUTION concerns the protection of equipment, software, and data. It is boxed to set it apart from other text.

Note

Information appearing next to NOTE concerns related information or useful hints.
## Contents

### About This Manual

- About This Manual ........................................................................................................................................ i

### Safety

- Safety ....................................................................................................................................................... xxvii

### Chapter 1 KAREL LANGUAGE OVERVIEW

- 1.1 OVERVIEW ........................................................................................................................................ 1–2
- 1.2 KAREL PROGRAMMING LANGUAGE ..................................................................................... 1–2
  - 1.2.1 Overview ....................................................................................................................................... 1–2
  - 1.2.2 Creating a Program .................................................................................................................... 1–4
  - 1.2.3 Translating a Program ............................................................................................................... 1–4
  - 1.2.4 Loading Program Logic and Data ............................................................................................. 1–4
  - 1.2.5 Executing a Program ................................................................................................................ 1–5
  - 1.2.6 Execution History ..................................................................................................................... 1–5
  - 1.2.7 Program Structure .................................................................................................................... 1–5
- 1.3 SYSTEM SOFTWARE ................................................................................................................... 1–7
  - 1.3.1 Software Components ............................................................................................................... 1–7
  - 1.3.2 Supported Robots ..................................................................................................................... 1–7
- 1.4 CONTROLLER ........................................................................................................................ 1–8
  - 1.4.1 Memory ...................................................................................................................................... 1–8
  - 1.4.2 Input/Output System ................................................................................................................ 1–10
  - 1.4.3 User Interface Devices ............................................................................................................ 1–10

### Chapter 2 LANGUAGE ELEMENTS

- 2.1 LANGUAGE COMPONENTS ........................................................................................................ 2–1
  - 2.1.1 Character Set ............................................................................................................................ 2–2
  - 2.1.2 Operators ............................................................................................................................... 2–5
  - 2.1.3 Reserved Words ................................................................................................................... 2–6
  - 2.1.4 User-Defined Identifiers ....................................................................................................... 2–7
  - 2.1.5 Labels ...................................................................................................................................... 2–8
  - 2.1.6 Predefined Identifiers ............................................................................................................. 2–8
  - 2.1.7 System Variables .................................................................................................................. 2–10
  - 2.1.8 Comments ............................................................................................................................ 2–10
- 2.2 TRANSLATOR DIRECTIVES .................................................................................................... 2–11
- 2.3 DATA TYPES ............................................................................................................................ 2–13
- 2.4 USER-DEFINED DATA TYPES AND STRUCTURES .................................................................. 2–14
  - 2.4.1 User-Defined Data Types ........................................................................................................ 2–14
  - 2.4.2 User-Defined Data Structures ............................................................................................... 2–16
- 2.5 ARRAYS ........................................................................................................................................ 2–18
  - 2.5.1 Multi-Dimensional Arrays ...................................................................................................... 2–19
  - 2.5.2 Variable-Sized Arrays ............................................................................................................ 2–20

### Chapter 3 USE OF OPERATORS

- 3.1 USE OF OPERATORS ...................................................................................................................... 3–1
### Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 3</td>
<td>EXPRESSIONS AND ASSIGNMENTS</td>
<td>3–2</td>
</tr>
<tr>
<td></td>
<td>3.1.1 Rule for Expressions and Assignments</td>
<td>3–2</td>
</tr>
<tr>
<td></td>
<td>3.1.2 Evaluation of Expressions and Assignments</td>
<td>3–2</td>
</tr>
<tr>
<td></td>
<td>3.1.3 Variables and Expressions</td>
<td>3–4</td>
</tr>
<tr>
<td></td>
<td>3.2 OPERATIONS</td>
<td>3–4</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Arithmetic Operations</td>
<td>3–5</td>
</tr>
<tr>
<td></td>
<td>3.2.2 Relational Operations</td>
<td>3–6</td>
</tr>
<tr>
<td></td>
<td>3.2.3 Boolean Operations</td>
<td>3–7</td>
</tr>
<tr>
<td></td>
<td>3.2.4 Special Operations</td>
<td>3–8</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>PROGRAM CONTROL</td>
<td>4–1</td>
</tr>
<tr>
<td></td>
<td>4.1 OVERVIEW</td>
<td>4–2</td>
</tr>
<tr>
<td></td>
<td>4.2 PROGRAM CONTROL STRUCTURES</td>
<td>4–2</td>
</tr>
<tr>
<td></td>
<td>4.2.1 Alternation Control Structures</td>
<td>4–2</td>
</tr>
<tr>
<td></td>
<td>4.2.2 Looping Control Statements</td>
<td>4–3</td>
</tr>
<tr>
<td></td>
<td>4.2.3 Unconditional Branch Statement</td>
<td>4–3</td>
</tr>
<tr>
<td></td>
<td>4.2.4 Execution Control Statements</td>
<td>4–3</td>
</tr>
<tr>
<td></td>
<td>4.2.5 Condition Handlers</td>
<td>4–4</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>ROUTINES</td>
<td>5–1</td>
</tr>
<tr>
<td></td>
<td>5.1 ROUTINE EXECUTION</td>
<td>5–2</td>
</tr>
<tr>
<td></td>
<td>5.1.1 Declaring Routines</td>
<td>5–2</td>
</tr>
<tr>
<td></td>
<td>5.1.2 Invoking Routines</td>
<td>5–5</td>
</tr>
<tr>
<td></td>
<td>5.1.3 Returning from Routines</td>
<td>5–7</td>
</tr>
<tr>
<td></td>
<td>5.1.4 Scope of Variables</td>
<td>5–8</td>
</tr>
<tr>
<td></td>
<td>5.1.5 Parameters and Arguments</td>
<td>5–9</td>
</tr>
<tr>
<td></td>
<td>5.1.6 Stack Usage</td>
<td>5–13</td>
</tr>
<tr>
<td></td>
<td>5.2 BUILT-IN ROUTINES</td>
<td>5–15</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>CONDITION HANDLERS</td>
<td>6–1</td>
</tr>
<tr>
<td></td>
<td>6.1 CONDITION HANDLER OPERATIONS</td>
<td>6–3</td>
</tr>
<tr>
<td></td>
<td>6.1.1 Global Condition Handlers</td>
<td>6–3</td>
</tr>
<tr>
<td></td>
<td>6.2 CONDITIONS</td>
<td>6–6</td>
</tr>
<tr>
<td></td>
<td>6.2.1 Port_Id Conditions</td>
<td>6–7</td>
</tr>
<tr>
<td></td>
<td>6.2.2 Relational Conditions</td>
<td>6–7</td>
</tr>
<tr>
<td></td>
<td>6.2.3 System and Program Event Conditions</td>
<td>6–8</td>
</tr>
<tr>
<td></td>
<td>6.3 ACTIONS</td>
<td>6–11</td>
</tr>
<tr>
<td></td>
<td>6.3.1 Assignment Actions</td>
<td>6–11</td>
</tr>
<tr>
<td></td>
<td>6.3.2 Motion Related Actions</td>
<td>6–13</td>
</tr>
<tr>
<td></td>
<td>6.3.3 Routine Call Actions</td>
<td>6–13</td>
</tr>
<tr>
<td></td>
<td>6.3.4 Miscellaneous Actions</td>
<td>6–14</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>FILE INPUT/OUTPUT OPERATIONS</td>
<td>7–1</td>
</tr>
<tr>
<td></td>
<td>7.1 OVERVIEW</td>
<td>7–3</td>
</tr>
<tr>
<td></td>
<td>7.2 FILE VARIABLES</td>
<td>7–3</td>
</tr>
<tr>
<td></td>
<td>7.3 OPEN FILE STATEMENT</td>
<td>7–4</td>
</tr>
<tr>
<td></td>
<td>7.3.1 Setting File and Port Attributes</td>
<td>7–5</td>
</tr>
<tr>
<td></td>
<td>7.3.2 File String</td>
<td>7–10</td>
</tr>
<tr>
<td></td>
<td>7.3.3 Usage String</td>
<td>7–11</td>
</tr>
<tr>
<td></td>
<td>7.4 CLOSE FILE STATEMENT</td>
<td>7–14</td>
</tr>
</tbody>
</table>
Chapter 8 POSITION DATA ................................................................. 8–1
  8.1 OVERVIEW .................................................................................... 8–2
  8.2 POSITIONAL DATA .......................................................................... 8–2
  8.3 FRAMES OF REFERENCE ................................................................. 8–3
    8.3.1 World Frame .............................................................................. 8–4
    8.3.2 User Frame (UFRAME) ............................................................... 8–5
    8.3.3 Tool Definition (UTOOL) .......................................................... 8–5
    8.3.4 Using Frames in the Teach Pendant Editor (TP) ......................... 8–5
  8.4 JOG COORDINATE SYSTEMS ......................................................... 8–6

Chapter 9 FILE SYSTEM ........................................................................ 9–1
  9.1 OVERVIEW ..................................................................................... 9–2
  9.2 FILE SPECIFICATION ................................................................. 9–3
    9.2.1 Device Name ........................................................................... 9–3
    9.2.2 File Name ............................................................................... 9–4
    9.2.3 File Type ................................................................................ 9–5
  9.3 STORAGE DEVICE ACCESS ......................................................... 9–6
    9.3.1 Overview ............................................................................... 9–7
    9.3.2 Memory File Devices ............................................................. 9–13
    9.3.3 Virtual Devices ..................................................................... 9–14
    9.3.4 File Pipes .............................................................................. 9–16
  9.4 FILE ACCESS ............................................................................... 9–20
  9.5 FORMATTING XML INPUT ........................................................ 9–20
    9.5.1 Overview .............................................................................. 9–20
    9.5.2 Installation Sequence ............................................................ 9–21
    9.5.3 Example KAREL Program Referencing an XML File .............. 9–22
### Chapter 10 DICTIONARIES AND FORMS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1 OVERVIEW</td>
<td>10–1</td>
</tr>
<tr>
<td>10.2 CREATING USER DICTIONARIES</td>
<td>10–3</td>
</tr>
<tr>
<td>10.2.1 Dictionary Syntax</td>
<td>10–3</td>
</tr>
<tr>
<td>10.2.2 Dictionary Element Number</td>
<td>10–4</td>
</tr>
<tr>
<td>10.2.3 Dictionary Element Name</td>
<td>10–5</td>
</tr>
<tr>
<td>10.2.4 Dictionary Cursor Positioning</td>
<td>10–5</td>
</tr>
<tr>
<td>10.2.5 Dictionary Element Text</td>
<td>10–6</td>
</tr>
<tr>
<td>10.2.6 Dictionary Reserved Word Commands</td>
<td>10–8</td>
</tr>
<tr>
<td>10.2.7 Character Codes</td>
<td>10–10</td>
</tr>
<tr>
<td>10.2.8 Nesting Dictionary Elements</td>
<td>10–10</td>
</tr>
<tr>
<td>10.2.9 Dictionary Comment</td>
<td>10–11</td>
</tr>
<tr>
<td>10.2.10 Generating a KAREL Constant File</td>
<td>10–11</td>
</tr>
<tr>
<td>10.2.11 Compressing and Loading Dictionaries on the Controller</td>
<td>10–11</td>
</tr>
<tr>
<td>10.2.12 Accessing Dictionary Elements from a KAREL Program</td>
<td>10–12</td>
</tr>
<tr>
<td>10.3 CREATING USER FORMS</td>
<td>10–13</td>
</tr>
<tr>
<td>10.3.1 Form Syntax</td>
<td>10–14</td>
</tr>
<tr>
<td>10.3.2 Form Attributes</td>
<td>10–15</td>
</tr>
<tr>
<td>10.3.3 Form Title and Menu Label</td>
<td>10–16</td>
</tr>
<tr>
<td>10.3.4 Form Menu Text</td>
<td>10–17</td>
</tr>
<tr>
<td>10.3.5 Form Selectable Menu Item</td>
<td>10–18</td>
</tr>
<tr>
<td>10.3.6 Edit Data Item</td>
<td>10–19</td>
</tr>
<tr>
<td>10.3.7 Dynamic Forms using Tree View</td>
<td>10–26</td>
</tr>
<tr>
<td>10.3.8 Non-Selectable Text</td>
<td>10–27</td>
</tr>
<tr>
<td>10.3.9 Display Only Data Items</td>
<td>10–27</td>
</tr>
<tr>
<td>10.3.10 Cursor Position Attributes</td>
<td>10–27</td>
</tr>
<tr>
<td>10.3.11 Form Reserved Words and Character Codes</td>
<td>10–28</td>
</tr>
<tr>
<td>10.3.12 Form Function Key Element Name or Number</td>
<td>10–30</td>
</tr>
<tr>
<td>10.3.13 Form Function Key Using a Variable</td>
<td>10–31</td>
</tr>
<tr>
<td>10.3.14 Form Help Element Name or Number</td>
<td>10–31</td>
</tr>
<tr>
<td>10.3.15 Teach Pendant Form Screen</td>
<td>10–32</td>
</tr>
<tr>
<td>10.3.16 CRT/KB Form Screen</td>
<td>10–32</td>
</tr>
<tr>
<td>10.3.17 Form File Naming Convention</td>
<td>10–33</td>
</tr>
<tr>
<td>10.3.18 Compressing and Loading Forms on the Controller</td>
<td>10–34</td>
</tr>
<tr>
<td>10.3.19 Displaying a Form</td>
<td>10–36</td>
</tr>
</tbody>
</table>

### Chapter 11 SOCKET MESSAGING

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 OVERVIEW</td>
<td>11–1</td>
</tr>
<tr>
<td>11.2 SYSTEM REQUIREMENTS</td>
<td>11–2</td>
</tr>
<tr>
<td>11.2.1 Overview</td>
<td>11–2</td>
</tr>
<tr>
<td>11.2.2 Software Requirements</td>
<td>11–2</td>
</tr>
<tr>
<td>11.2.3 Hardware Requirements</td>
<td>11–2</td>
</tr>
<tr>
<td>11.3 CONFIGURING THE SOCKET MESSAGING OPTION</td>
<td>11–3</td>
</tr>
<tr>
<td>11.3.1 Overview</td>
<td>11–3</td>
</tr>
<tr>
<td>11.3.2 Setting up a Server Tag</td>
<td>11–3</td>
</tr>
<tr>
<td>11.3.3 Setting up a Client Tag</td>
<td>11–6</td>
</tr>
<tr>
<td>11.4 SOCKET MESSAGING AND KAREL</td>
<td>11–9</td>
</tr>
<tr>
<td>11.4.1 Overview</td>
<td>11–9</td>
</tr>
<tr>
<td>11.4.2 MSG_CONN( string, integer )</td>
<td>11–9</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>11</td>
<td>NETWORK PERFORMANCE</td>
</tr>
<tr>
<td>11.3</td>
<td>MSG_DISCO( string, integer )</td>
</tr>
<tr>
<td>11.4</td>
<td>MSG_PING( string, integer )</td>
</tr>
<tr>
<td>11.4.5</td>
<td>Exchanging Data during a Socket Messaging Connection</td>
</tr>
<tr>
<td>11.5</td>
<td>INTERPRETER ASSIGNMENT</td>
</tr>
<tr>
<td>11.5.1</td>
<td>Overview</td>
</tr>
<tr>
<td>11.5.2</td>
<td>Guidelines for a Good Implementation</td>
</tr>
<tr>
<td>11.6</td>
<td>PROGRAMMING EXAMPLES</td>
</tr>
<tr>
<td>11.6.1</td>
<td>A KAREL Client Application</td>
</tr>
<tr>
<td>11.6.2</td>
<td>A KAREL Server Application</td>
</tr>
<tr>
<td>11.6.3</td>
<td>ANSI C Loopback Client Example</td>
</tr>
<tr>
<td>11.6.4</td>
<td>A KAREL Client Application</td>
</tr>
<tr>
<td>12</td>
<td>SYSTEM VARIABLES</td>
</tr>
<tr>
<td>12.1</td>
<td>ACCESS RIGHTS</td>
</tr>
<tr>
<td>12.2</td>
<td>STORAGE</td>
</tr>
<tr>
<td>13</td>
<td>KAREL COMMAND LANGUAGE (KCL)</td>
</tr>
<tr>
<td>13.1</td>
<td>COMMAND FORMAT</td>
</tr>
<tr>
<td>13.1.1</td>
<td>Default Program</td>
</tr>
<tr>
<td>13.1.2</td>
<td>Variables and Data Types</td>
</tr>
<tr>
<td>13.2</td>
<td>PROGRAM CONTROL COMMANDS</td>
</tr>
<tr>
<td>13.3</td>
<td>ENTERING COMMANDS</td>
</tr>
<tr>
<td>13.3.1</td>
<td>Abbreviations</td>
</tr>
<tr>
<td>13.3.2</td>
<td>Error Messages</td>
</tr>
<tr>
<td>13.3.3</td>
<td>Subdirectories</td>
</tr>
<tr>
<td>13.4</td>
<td>COMMAND PROCEDURES</td>
</tr>
<tr>
<td>13.4.1</td>
<td>Command Procedure Format</td>
</tr>
<tr>
<td>13.4.2</td>
<td>Creating Command Procedures</td>
</tr>
<tr>
<td>13.4.3</td>
<td>Error Processing</td>
</tr>
<tr>
<td>13.4.4</td>
<td>Executing Command Procedures</td>
</tr>
<tr>
<td>14</td>
<td>INPUT/OUTPUT SYSTEM</td>
</tr>
<tr>
<td>14.1</td>
<td>USER-DEFINED SIGNALS</td>
</tr>
<tr>
<td>14.1.1</td>
<td>DIN and DOUT Signals</td>
</tr>
<tr>
<td>14.1.2</td>
<td>GIN and GOUT Signals</td>
</tr>
<tr>
<td>14.1.3</td>
<td>AIN and AOUT Signals</td>
</tr>
<tr>
<td>14.1.4</td>
<td>Hand Signals</td>
</tr>
<tr>
<td>14.2</td>
<td>SYSTEM-DEFINED SIGNALS</td>
</tr>
<tr>
<td>14.2.1</td>
<td>Robot Digital Input and Output Signals (RDI/RDO)</td>
</tr>
<tr>
<td>14.2.2</td>
<td>Operator Panel Input and Output Signals (OPIN/OPOUT)</td>
</tr>
<tr>
<td>14.2.3</td>
<td>Teach Pendant Input and Output Signals (TPIN/TPOUT)</td>
</tr>
<tr>
<td>14.3</td>
<td>Serial Input/Output</td>
</tr>
<tr>
<td>14.3.1</td>
<td>Serial Input/Output</td>
</tr>
<tr>
<td>15</td>
<td>MULTI-TASKING</td>
</tr>
<tr>
<td>15.1</td>
<td>MULTI-TASKING TERMINOLOGY</td>
</tr>
<tr>
<td>15.2</td>
<td>INTERPRETER ASSIGNMENT</td>
</tr>
<tr>
<td>15.3</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>15.4</td>
<td>TASK SCHEDULING</td>
</tr>
<tr>
<td>15.4.1</td>
<td>Priority Scheduling</td>
</tr>
</tbody>
</table>
Appendix A  KAREL LANGUAGE ALPHABETICAL DESCRIPTION ........................................... A–1

A.1 OVERVIEW .................................................................................................................. A–9

A.2 - A - KAREL LANGUAGE DESCRIPTION .................................................................... A–18
A.2.1 ABORT Action ......................................................................................................... A–18
A.2.2 ABORT Condition .................................................................................................. A–18
A.2.3 ABORT Statement .................................................................................................. A–19
A.2.4 ABORT_TASK Built-In Procedure ........................................................................ A–19
A.2.5 ABS Built-In Function .......................................................................................... A–20
A.2.6 ACOS Built-In Function ........................................................................................ A–20
A.2.7 ACT_SCREEN Built-In Procedure ......................................................................... A–21
A.2.8 ACT_TBL Built-In Procedure ............................................................................... A–22
A.2.9 ADD_BYNAMEPC Built-In Procedure .................................................................. A–24
A.2.10 ADD_DICT Built-In Procedure ............................................................................ A–25
A.2.11 ADD_INTPC Built-In Procedure .......................................................................... A–26
A.2.12 ADD_REALPC Built-In Procedure ........................................................................ A–27
A.2.13 ADD_STRINGPC Built-In Procedure ................................................................... A–29
A.2.14 %ALPHABETIZE Translator Directive ................................................................ A–30
A.2.15 APPEND_NODE Built-In Procedure .................................................................... A–30
A.2.16 APPEND_QUEUE Built-In Procedure ................................................................ A–31
A.2.17 APPROACH Built-In Function ............................................................................ A–32
A.2.18 ARRAY Data Type ................................................................................................ A–33
A.2.19 ARRAY_LEN Built-In Function ........................................................................... A–34
A.2.20 ASIN Built-In Function ....................................................................................... A–34
A.2.21 Assignment Action .............................................................................................. A–35
A.2.22 Assignment Statement ....................................................................................... A–36
A.2.23 ATAN2 Built-In Function .................................................................................. A–38
A.2.24 ATTACH Statement ............................................................................................ A–39
A.2.25 ATT_WINDOW_D Built-In Procedure ................................................................ A–39
A.2.26 ATT_WINDOW_S Built-In Procedure ................................................................ A–40
A.2.27 AVL_POS_NUM Built-In Procedure ..................................................................... A–41

A.3 - B - KAREL LANGUAGE DESCRIPTION .................................................................... A–42
A.3.1 BOOLEAN Data Type .............................................................................................. A–42
A.3.2 BYNAME Built-In Function ................................................................................ A–43
A.3.3 BYTE Data Type .................................................................................................. A–44
A.3.4 BYTES_AHEAD Built-In Procedure .................................................................... A–45
A.3.5 BYTES_LEFT Built-In Function .......................................................................... A–46

A.4 - C - KAREL LANGUAGE DESCRIPTION .................................................................... A–47
A.4.1 CALL_PROG Built-In Procedure ......................................................................... A–47
A.4.2 CALL_PROGLIN Built-In Procedure .................................................................. A–48
A.4.3 CANCEL Action ................................................................................................. A–49
A.4.4 CANCEL Statement ............................................................................................ A–49

15.5 STARTING TASKS ........................................................................................................ 15–6
15.5.1 Running Programs from the User Operator Panel (UOP) PNS Signal ....................... 15–7

Appendix A  KAREL LANGUAGE ALPHABETICAL DESCRIPTION ........................................... A–1

A.1 OVERVIEW .................................................................................................................. A–9

A.2 - A - KAREL LANGUAGE DESCRIPTION .................................................................... A–18
A.2.1 ABORT Action ......................................................................................................... A–18
A.2.2 ABORT Condition .................................................................................................. A–18
A.2.3 ABORT Statement .................................................................................................. A–19
A.2.4 ABORT_TASK Built-In Procedure ........................................................................ A–19
A.2.5 ABS Built-In Function .......................................................................................... A–20
A.2.6 ACOS Built-In Function ........................................................................................ A–20
A.2.7 ACT_SCREEN Built-In Procedure ......................................................................... A–21
A.2.8 ACT_TBL Built-In Procedure ............................................................................... A–22
A.2.9 ADD_BYNAMEPC Built-In Procedure .................................................................. A–24
A.2.10 ADD_DICT Built-In Procedure ............................................................................ A–25
A.2.11 ADD_INTPC Built-In Procedure .......................................................................... A–26
A.2.12 ADD_REALPC Built-In Procedure ........................................................................ A–27
A.2.13 ADD_STRINGPC Built-In Procedure ................................................................... A–29
A.2.14 %ALPHABETIZE Translator Directive ................................................................ A–30
A.2.15 APPEND_NODE Built-In Procedure .................................................................... A–30
A.2.16 APPEND_QUEUE Built-In Procedure ................................................................ A–31
A.2.17 APPROACH Built-In Function ............................................................................ A–32
A.2.18 ARRAY Data Type ................................................................................................ A–33
A.2.19 ARRAY_LEN Built-In Function ........................................................................... A–34
A.2.20 ASIN Built-In Function ....................................................................................... A–34
A.2.21 Assignment Action .............................................................................................. A–35
A.2.22 Assignment Statement ....................................................................................... A–36
A.2.23 ATAN2 Built-In Function .................................................................................. A–38
A.2.24 ATTACH Statement ............................................................................................ A–39
A.2.25 ATT_WINDOW_D Built-In Procedure ................................................................ A–39
A.2.26 ATT_WINDOW_S Built-In Procedure ................................................................ A–40
A.2.27 AVL_POS_NUM Built-In Procedure ..................................................................... A–41

A.3 - B - KAREL LANGUAGE DESCRIPTION .................................................................... A–42
A.3.1 BOOLEAN Data Type .............................................................................................. A–42
A.3.2 BYNAME Built-In Function ................................................................................ A–43
A.3.3 BYTE Data Type .................................................................................................. A–44
A.3.4 BYTES_AHEAD Built-In Procedure .................................................................... A–45
A.3.5 BYTES_LEFT Built-In Function .......................................................................... A–46

A.4 - C - KAREL LANGUAGE DESCRIPTION .................................................................... A–47
A.4.1 CALL_PROG Built-In Procedure ......................................................................... A–47
A.4.2 CALL_PROGLIN Built-In Procedure .................................................................. A–48
A.4.3 CANCEL Action ................................................................................................. A–49
A.4.4 CANCEL Statement ............................................................................................ A–49
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.4.5</td>
<td>CANCEL_FILE Statement</td>
<td>A-50</td>
</tr>
<tr>
<td>A.4.6</td>
<td>CHECK_DICT Built-In Procedure</td>
<td>A-51</td>
</tr>
<tr>
<td>A.4.7</td>
<td>CHECK_EPOS Built-In Procedure</td>
<td>A-52</td>
</tr>
<tr>
<td>A.4.8</td>
<td>CHECK_NAME Built-In Procedure</td>
<td>A-53</td>
</tr>
<tr>
<td>A.4.9</td>
<td>CHR Built-In Function</td>
<td>A-53</td>
</tr>
<tr>
<td>A.4.10</td>
<td>CLEAR Built-In Procedure</td>
<td>A-54</td>
</tr>
<tr>
<td>A.4.11</td>
<td>CLEAR_SEMA Built-In Procedure</td>
<td>A-55</td>
</tr>
<tr>
<td>A.4.12</td>
<td>CLOSE_FILE Statement</td>
<td>A-55</td>
</tr>
<tr>
<td>A.4.13</td>
<td>CLOSE_HAND Statement</td>
<td>A-56</td>
</tr>
<tr>
<td>A.4.14</td>
<td>CLOSE_TPE Built-In Procedure</td>
<td>A-56</td>
</tr>
<tr>
<td>A.4.15</td>
<td>CLR_IO_STAT Built-In Procedure</td>
<td>A-57</td>
</tr>
<tr>
<td>A.4.16</td>
<td>CLR_PORT_SIM Built-In Procedure</td>
<td>A-57</td>
</tr>
<tr>
<td>A.4.17</td>
<td>CLR_POS_REG Built-In Procedure</td>
<td>A-58</td>
</tr>
<tr>
<td>A.4.18</td>
<td>%CMOSVARS Translator Directive</td>
<td>A-59</td>
</tr>
<tr>
<td>A.4.19</td>
<td>%CMOS2SHADOW Translator Directive</td>
<td>A-59</td>
</tr>
<tr>
<td>A.4.20</td>
<td>CNC_DYN_DISB Built-In Procedure</td>
<td>A-59</td>
</tr>
<tr>
<td>A.4.21</td>
<td>CNC_DYN_DISE Built-In Procedure</td>
<td>A-60</td>
</tr>
<tr>
<td>A.4.22</td>
<td>CNC_DYN_DISI Built-In Procedure</td>
<td>A-61</td>
</tr>
<tr>
<td>A.4.23</td>
<td>CNC_DYN_DISP Built-In Procedure</td>
<td>A-62</td>
</tr>
<tr>
<td>A.4.24</td>
<td>CNC_DYN_DISR Built-In Procedure</td>
<td>A-62</td>
</tr>
<tr>
<td>A.4.25</td>
<td>CNC_DYN_DISS Built-In Procedure</td>
<td>A-63</td>
</tr>
<tr>
<td>A.4.26</td>
<td>CNCL_STP_MTN Built-In Procedure</td>
<td>A-64</td>
</tr>
<tr>
<td>A.4.27</td>
<td>CVN_CNFR_STRG Built-In Procedure</td>
<td>A-64</td>
</tr>
<tr>
<td>A.4.28</td>
<td>CVN_CONF_STR Built-In Procedure</td>
<td>A-65</td>
</tr>
<tr>
<td>A.4.29</td>
<td>CVN_INT_STR Built-In Procedure</td>
<td>A-66</td>
</tr>
<tr>
<td>A.4.30</td>
<td>CVN_JPOS_REL Built-In Procedure</td>
<td>A-67</td>
</tr>
<tr>
<td>A.4.31</td>
<td>CVN_REAL_STR Built-In Procedure</td>
<td>A-67</td>
</tr>
<tr>
<td>A.4.32</td>
<td>CVN_REL_JPOS Built-In Procedure</td>
<td>A-68</td>
</tr>
<tr>
<td>A.4.33</td>
<td>CVN_STR_CONF Built-In Procedure</td>
<td>A-69</td>
</tr>
<tr>
<td>A.4.34</td>
<td>CVN_STR_INT Built-In Procedure</td>
<td>A-69</td>
</tr>
<tr>
<td>A.4.35</td>
<td>CVN_STR_REAL Built-In Procedure</td>
<td>A-70</td>
</tr>
<tr>
<td>A.4.36</td>
<td>CVN_STR_TIME Built-In Procedure</td>
<td>A-71</td>
</tr>
<tr>
<td>A.4.37</td>
<td>CVN_TIME_STR Built-In Procedure</td>
<td>A-71</td>
</tr>
<tr>
<td>A.4.38</td>
<td>%COMMENT Translator Directive</td>
<td>A-73</td>
</tr>
<tr>
<td>A.4.39</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td>A-73</td>
</tr>
<tr>
<td>A.4.40</td>
<td>CONDITION...ENDCONDITION Statement</td>
<td>A-75</td>
</tr>
<tr>
<td>A.4.41</td>
<td>CONFIG Data Type</td>
<td>A-77</td>
</tr>
<tr>
<td>A.4.42</td>
<td>CONNECT TIMER Statement</td>
<td>A-78</td>
</tr>
<tr>
<td>A.4.43</td>
<td>CONTINUE Action</td>
<td>A-79</td>
</tr>
<tr>
<td>A.4.44</td>
<td>CONTINUE Condition</td>
<td>A-79</td>
</tr>
<tr>
<td>A.4.45</td>
<td>CONT_TASK Built-In Procedure</td>
<td>A-80</td>
</tr>
<tr>
<td>A.4.46</td>
<td>COPY_FILE Built-In Procedure</td>
<td>A-81</td>
</tr>
<tr>
<td>A.4.47</td>
<td>COPY_PATH Built-In Procedure</td>
<td>A-82</td>
</tr>
<tr>
<td>A.4.48</td>
<td>COPY_QUEUE Built-In Procedure</td>
<td>A-83</td>
</tr>
<tr>
<td>A.4.49</td>
<td>COPY_TPE Built-In Procedure</td>
<td>A-85</td>
</tr>
<tr>
<td>A.4.50</td>
<td>COS Built-In Function</td>
<td>A-86</td>
</tr>
<tr>
<td>A.4.51</td>
<td>CR Input/Output Item</td>
<td>A-86</td>
</tr>
<tr>
<td>A.4.52</td>
<td>CREATE_TPE Built-In Procedure</td>
<td>A-87</td>
</tr>
<tr>
<td>A.4.53</td>
<td>CREATE_VAR Built-In Procedure</td>
<td>A-88</td>
</tr>
<tr>
<td>A.4.54</td>
<td>%CRTDEVICE</td>
<td>A-90</td>
</tr>
<tr>
<td>A.4.55</td>
<td>CURRPOS Built-In Function</td>
<td>A-91</td>
</tr>
<tr>
<td>A.4.56</td>
<td>CURPOS Built-In Function</td>
<td>A-92</td>
</tr>
<tr>
<td>A.4.57</td>
<td>CURR_PROG Built-In Function</td>
<td>A-93</td>
</tr>
<tr>
<td>A.5</td>
<td>- D - KAREL LANGUAGE DESCRIPTION</td>
<td>A-93</td>
</tr>
<tr>
<td>A.5.1</td>
<td>DAQ_CHECKP Built-In Procedure</td>
<td>A-93</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A.8.10</td>
<td>GET_PORT_VAL Built-In Procedure</td>
<td>A–156</td>
</tr>
<tr>
<td>A.8.11</td>
<td>GET_POS_FRM Built-In Procedure</td>
<td>A–157</td>
</tr>
<tr>
<td>A.8.12</td>
<td>GET_POS_REG Built-In Procedure</td>
<td>A–158</td>
</tr>
<tr>
<td>A.8.13</td>
<td>GET_POS_TPE Built-In Function</td>
<td>A–158</td>
</tr>
<tr>
<td>A.8.14</td>
<td>GET_POS_TYP Built-In Procedure</td>
<td>A–159</td>
</tr>
<tr>
<td>A.8.15</td>
<td>GET_PREG_CMT Built-In-Procedure</td>
<td>A–160</td>
</tr>
<tr>
<td>A.8.16</td>
<td>GET_QUEUE Built-In Procedure</td>
<td>A–161</td>
</tr>
<tr>
<td>A.8.17</td>
<td>GET_REG Built-In Procedure</td>
<td>A–162</td>
</tr>
<tr>
<td>A.8.18</td>
<td>GET_REG_CMT</td>
<td>A–163</td>
</tr>
<tr>
<td>A.8.19</td>
<td>GET_SREG_CMT Built-in Procedure</td>
<td>A–163</td>
</tr>
<tr>
<td>A.8.20</td>
<td>GET_STR_REG Built-In Procedure</td>
<td>A–164</td>
</tr>
<tr>
<td>A.8.21</td>
<td>GET_TIME Built-In Procedure</td>
<td>A–164</td>
</tr>
<tr>
<td>A.8.22</td>
<td>GET_TPE_CMT Built-in Procedure</td>
<td>A–165</td>
</tr>
<tr>
<td>A.8.23</td>
<td>GET_TPE_PRM Built-in Procedure</td>
<td>A–166</td>
</tr>
<tr>
<td>A.8.24</td>
<td>GET_TSK_INFO Built-In Procedure</td>
<td>A–168</td>
</tr>
<tr>
<td>A.8.25</td>
<td>GET_USEC_SUB Built-In Procedure</td>
<td>A–170</td>
</tr>
<tr>
<td>A.8.26</td>
<td>GET_USEC_TIM Built-In Function</td>
<td>A–171</td>
</tr>
<tr>
<td>A.8.27</td>
<td>GET_VAR Built-In Procedure</td>
<td>A–171</td>
</tr>
<tr>
<td>A.8.28</td>
<td>GO TO Statement</td>
<td>A–175</td>
</tr>
<tr>
<td>A.9.1</td>
<td>HOLD Action</td>
<td>A–176</td>
</tr>
<tr>
<td>A.9.2</td>
<td>HOLD Statement</td>
<td>A–177</td>
</tr>
<tr>
<td>A.10.1</td>
<td>IF ... ENDIF Statement</td>
<td>A–178</td>
</tr>
<tr>
<td>A.10.2</td>
<td>IN Clause</td>
<td>A–179</td>
</tr>
<tr>
<td>A.10.3</td>
<td>%INCLUDE Translator Directive</td>
<td>A–180</td>
</tr>
<tr>
<td>A.10.4</td>
<td>INDEX Built-In Function</td>
<td>A–181</td>
</tr>
<tr>
<td>A.10.5</td>
<td>INI_DYN_DISB Built-In Procedure</td>
<td>A–181</td>
</tr>
<tr>
<td>A.10.6</td>
<td>INI_DYN_DISE Built-In Procedure</td>
<td>A–183</td>
</tr>
<tr>
<td>A.10.7</td>
<td>INI_DYN_DISI Built-In Procedure</td>
<td>A–184</td>
</tr>
<tr>
<td>A.10.8</td>
<td>INI_DYN_DISP Built-In Procedure</td>
<td>A–186</td>
</tr>
<tr>
<td>A.10.9</td>
<td>INI_DYN_DISR Built-In Procedure</td>
<td>A–187</td>
</tr>
<tr>
<td>A.10.10</td>
<td>INI_DYN_DISS Built-In Procedure</td>
<td>A–188</td>
</tr>
<tr>
<td>A.10.11</td>
<td>INIT_QUEUE Built-In Procedure</td>
<td>A–189</td>
</tr>
<tr>
<td>A.10.12</td>
<td>INIT_TBL Built-In Procedure</td>
<td>A–190</td>
</tr>
<tr>
<td>A.10.13</td>
<td>IN_RANGE Built-In Function</td>
<td>A–201</td>
</tr>
<tr>
<td>A.10.14</td>
<td>INSERT_NODE Built-In Procedure</td>
<td>A–202</td>
</tr>
<tr>
<td>A.10.15</td>
<td>INSERT_QUEUE Built-In Procedure</td>
<td>A–203</td>
</tr>
<tr>
<td>A.10.16</td>
<td>INTEGER Data Type</td>
<td>A–204</td>
</tr>
<tr>
<td>A.10.17</td>
<td>INV Built-In Function</td>
<td>A–205</td>
</tr>
<tr>
<td>A.10.18</td>
<td>IO_MODULE Built-In Procedure</td>
<td>A–206</td>
</tr>
<tr>
<td>A.10.19</td>
<td>IO_STATUS Built-In Function</td>
<td>A–207</td>
</tr>
<tr>
<td>A.11</td>
<td>- J - KAREL LANGUAGE DESCRIPTION</td>
<td>A–208</td>
</tr>
<tr>
<td>A.11.1</td>
<td>J_IN_RANGE Built-In Function</td>
<td>A–208</td>
</tr>
<tr>
<td>A.11.2</td>
<td>JOINTPOS Data Type</td>
<td>A–209</td>
</tr>
<tr>
<td>A.11.3</td>
<td>JOINT2POS Built-In Function</td>
<td>A–209</td>
</tr>
<tr>
<td>A.12</td>
<td>- K - KAREL LANGUAGE DESCRIPTION</td>
<td>A–211</td>
</tr>
<tr>
<td>A.12.1</td>
<td>KCL Built-In Procedure</td>
<td>A–211</td>
</tr>
<tr>
<td>A.12.2</td>
<td>KCL_NO_WAIT Built-In Procedure</td>
<td>A–212</td>
</tr>
<tr>
<td>A.12.3</td>
<td>KCL_STATUS Built-In Procedure</td>
<td>A–213</td>
</tr>
<tr>
<td>A.13</td>
<td>- L - KAREL LANGUAGE DESCRIPTION</td>
<td>A–213</td>
</tr>
<tr>
<td>A.13.1</td>
<td>LN Built-In Function</td>
<td>A–213</td>
</tr>
<tr>
<td>A.13.2</td>
<td>LOAD Built-In Procedure</td>
<td>A–214</td>
</tr>
<tr>
<td>A.13.3</td>
<td>LOAD_STATUS Built-In Procedure</td>
<td>A–215</td>
</tr>
<tr>
<td>A.13.4</td>
<td>LOCK_GROUP Built-In Procedure</td>
<td>A-216</td>
</tr>
<tr>
<td>A.13.5</td>
<td>%LOCKGROUP Translator Directive</td>
<td>A-217</td>
</tr>
<tr>
<td>A.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.14.1</td>
<td>MIRROR Built-In Function</td>
<td>A-218</td>
</tr>
<tr>
<td>A.14.2</td>
<td>MODIFY_QUEUE Built-In Procedure</td>
<td>A-219</td>
</tr>
<tr>
<td>A.14.3</td>
<td>MOTION_CTL Built-In Function</td>
<td>A-221</td>
</tr>
<tr>
<td>A.14.4</td>
<td>MOUNT_DEV Built-In Procedure</td>
<td>A-221</td>
</tr>
<tr>
<td>A.14.5</td>
<td>MOVE_FILE Built-In Procedure</td>
<td>A-222</td>
</tr>
<tr>
<td>A.14.6</td>
<td>MSG_CONNECT Built-In Procedure</td>
<td>A-223</td>
</tr>
<tr>
<td>A.14.7</td>
<td>MSG_DISCO Built-In Procedure</td>
<td>A-225</td>
</tr>
<tr>
<td>A.14.8</td>
<td>MSG_PING</td>
<td>A-225</td>
</tr>
<tr>
<td>A.17.2</td>
<td>PATH_LEN Built-In Function</td>
<td>A-238</td>
</tr>
<tr>
<td>A.17.3</td>
<td>PATH Data Type</td>
<td>A-236</td>
</tr>
<tr>
<td>A.17.4</td>
<td>PATH_LEN Built-In Function</td>
<td>A-238</td>
</tr>
<tr>
<td>A.17.5</td>
<td>PAUSE Action</td>
<td>A-238</td>
</tr>
<tr>
<td>A.17.6</td>
<td>PAUSE Condition</td>
<td>A-239</td>
</tr>
<tr>
<td>A.17.7</td>
<td>PAUSE Statement</td>
<td>A-240</td>
</tr>
<tr>
<td>A.17.8</td>
<td>PAUSE_TASK Built-In Procedure</td>
<td>A-241</td>
</tr>
<tr>
<td>A.17.9</td>
<td>PEND_SEMA Built-In Procedure</td>
<td>A-242</td>
</tr>
<tr>
<td>A.17.10</td>
<td>PIPE_CONFIG Built-In Procedure</td>
<td>A-242</td>
</tr>
<tr>
<td>A.17.11</td>
<td>POP_KEY_RD Built-In Procedure</td>
<td>A-243</td>
</tr>
<tr>
<td>A.17.12</td>
<td>Port_Id Action</td>
<td>A-244</td>
</tr>
<tr>
<td>A.17.13</td>
<td>Port_Id Condition</td>
<td>A-245</td>
</tr>
<tr>
<td>A.17.14</td>
<td>POS Built-In Function</td>
<td>A-246</td>
</tr>
<tr>
<td>A.17.15</td>
<td>POS2JOINT Built-In Function</td>
<td>A-246</td>
</tr>
<tr>
<td>A.17.16</td>
<td>POS_REG_TYPE Built-In Procedure</td>
<td>A-248</td>
</tr>
<tr>
<td>A.17.17</td>
<td>POSITION Data Type</td>
<td>A-249</td>
</tr>
<tr>
<td>A.17.18</td>
<td>POST_ERR Built-In Procedure</td>
<td>A-250</td>
</tr>
<tr>
<td>A.17.19</td>
<td>PRIORITY Translator Directive</td>
<td>A-252</td>
</tr>
<tr>
<td>A.17.20</td>
<td>PROC_BACKUP Built-In Procedure</td>
<td>A-255</td>
</tr>
<tr>
<td>A.17.21</td>
<td>PROC_CLEAR Built-In Procedure</td>
<td>A-258</td>
</tr>
<tr>
<td>A.17.22</td>
<td>PROC_RESTORE Built-In Procedure</td>
<td>A-260</td>
</tr>
<tr>
<td>A.17.23</td>
<td>PROC_LIST Built-In Procedure</td>
<td>A-262</td>
</tr>
<tr>
<td>A.17.24</td>
<td>PROGRAM Statement</td>
<td>A-263</td>
</tr>
<tr>
<td>A.17.25</td>
<td>PULSE Action</td>
<td>A-264</td>
</tr>
</tbody>
</table>
### A.19 - R - KAREL LANGUAGE DESCRIPTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.19.1</td>
<td>READ Statement</td>
<td>A–269</td>
</tr>
<tr>
<td>A.19.2</td>
<td>READ_DICTION Built-In Procedure</td>
<td>A–269</td>
</tr>
<tr>
<td>A.19.3</td>
<td>READ_DICTION_MACRO Built-In Procedure</td>
<td>A–271</td>
</tr>
<tr>
<td>A.19.4</td>
<td>READ_KEYBOARD Built-In Procedure</td>
<td>A–272</td>
</tr>
<tr>
<td>A.19.5</td>
<td>REAL Data Type</td>
<td>A–273</td>
</tr>
<tr>
<td>A.19.6</td>
<td>Relational Condition</td>
<td>A–277</td>
</tr>
<tr>
<td>A.19.7</td>
<td>RELAX_HAND Statement</td>
<td>A–279</td>
</tr>
<tr>
<td>A.19.8</td>
<td>RELEASE Statement</td>
<td>A–280</td>
</tr>
<tr>
<td>A.19.9</td>
<td>REMOVE_DICTION Built-In Procedure</td>
<td>A–281</td>
</tr>
<tr>
<td>A.19.10</td>
<td>RENAME_FILE Built-In Procedure</td>
<td>A–282</td>
</tr>
<tr>
<td>A.19.11</td>
<td>RENAME_VAR Built-In Procedure</td>
<td>A–283</td>
</tr>
<tr>
<td>A.19.12</td>
<td>RENAME_VARS Built-In Procedure</td>
<td>A–284</td>
</tr>
<tr>
<td>A.19.13</td>
<td>REPEAT ... UNTIL Statement</td>
<td>A–285</td>
</tr>
<tr>
<td>A.19.14</td>
<td>RESET Built-In Procedure</td>
<td>A–286</td>
</tr>
<tr>
<td>A.19.15</td>
<td>RESUME Action</td>
<td>A–286</td>
</tr>
<tr>
<td>A.19.16</td>
<td>RESUME Statement</td>
<td>A–287</td>
</tr>
<tr>
<td>A.19.17</td>
<td>RETURN Statement</td>
<td>A–287</td>
</tr>
<tr>
<td>A.19.18</td>
<td>ROUND Built-In Function</td>
<td>A–288</td>
</tr>
<tr>
<td>A.19.19</td>
<td>ROUTINE Statement</td>
<td>A–289</td>
</tr>
<tr>
<td>A.19.20</td>
<td>RUN_TASK Built-In Procedure</td>
<td>A–290</td>
</tr>
<tr>
<td>A.20</td>
<td>- S - KAREL LANGUAGE DESCRIPTION</td>
<td>A–291</td>
</tr>
<tr>
<td>A.20.1</td>
<td>SAVE Built-In Procedure</td>
<td>A–291</td>
</tr>
<tr>
<td>A.20.2</td>
<td>SAVE_DRAM Built-In Procedure</td>
<td>A–292</td>
</tr>
<tr>
<td>A.20.3</td>
<td>SELECT ... ENDSELECT Statement</td>
<td>A–293</td>
</tr>
<tr>
<td>A.20.4</td>
<td>SELECT_TPE Built-In Procedure</td>
<td>A–294</td>
</tr>
<tr>
<td>A.20.5</td>
<td>SEMA_COUNT Built-In Function</td>
<td>A–294</td>
</tr>
<tr>
<td>A.20.6</td>
<td>SEMAPHORE Condition</td>
<td>A–295</td>
</tr>
<tr>
<td>A.20.7</td>
<td>SEND_DATAPC Built-In Procedure</td>
<td>A–295</td>
</tr>
<tr>
<td>A.20.8</td>
<td>SEND_EVENTPC Built-In Procedure</td>
<td>A–296</td>
</tr>
<tr>
<td>A.20.9</td>
<td>SET_ATTR_PRG Built-In Procedure</td>
<td>A–297</td>
</tr>
<tr>
<td>A.20.10</td>
<td>SET_CURSOR Built-In Procedure</td>
<td>A–297</td>
</tr>
<tr>
<td>A.20.11</td>
<td>SET_EPOS_REG Built-In Procedure</td>
<td>A–300</td>
</tr>
<tr>
<td>A.20.12</td>
<td>SET_EPOS_TPE Built-In Procedure</td>
<td>A–301</td>
</tr>
<tr>
<td>A.20.13</td>
<td>SET_FILE_ATR Built-In Procedure</td>
<td>A–302</td>
</tr>
<tr>
<td>A.20.14</td>
<td>SET_FILE_POS Built-In Procedure</td>
<td>A–303</td>
</tr>
<tr>
<td>A.20.15</td>
<td>SET_INT_REG Built-In Procedure</td>
<td>A–304</td>
</tr>
<tr>
<td>A.20.16</td>
<td>SET_JPOS_REG Built-In Procedure</td>
<td>A–304</td>
</tr>
<tr>
<td>A.20.17</td>
<td>SET_JPOS_TPE Built-In Procedure</td>
<td>A–305</td>
</tr>
<tr>
<td>A.20.18</td>
<td>SET_LANG Built-In Procedure</td>
<td>A–306</td>
</tr>
<tr>
<td>A.20.19</td>
<td>SET_PERCH Built-In Procedure</td>
<td>A–307</td>
</tr>
<tr>
<td>A.20.20</td>
<td>SET_PORT_ASG Built-In Procedure</td>
<td>A–308</td>
</tr>
<tr>
<td>A.20.21</td>
<td>SET_PORT_ATR Built-In Function</td>
<td>A–309</td>
</tr>
<tr>
<td>A.20.22</td>
<td>SET_PORT_CMT Built-In Procedure</td>
<td>A–311</td>
</tr>
<tr>
<td>A.20.23</td>
<td>SET_PORT_MOD Built-In Procedure</td>
<td>A–312</td>
</tr>
<tr>
<td>A.20.24</td>
<td>SET_PORT_SIM Built-In Procedure</td>
<td>A–313</td>
</tr>
<tr>
<td>A.20.25</td>
<td>SET_PORT_VAL Built-In Procedure</td>
<td>A–314</td>
</tr>
<tr>
<td>A.20.26</td>
<td>SET_POS_VAL Built-In Procedure</td>
<td>A–315</td>
</tr>
</tbody>
</table>

---

### A.20 - S - KAREL LANGUAGE DESCRIPTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.20.1</td>
<td>SAVE Built-In Procedure</td>
<td>A–291</td>
</tr>
<tr>
<td>A.20.2</td>
<td>SAVE_DRAM Built-In Procedure</td>
<td>A–292</td>
</tr>
<tr>
<td>A.20.3</td>
<td>SELECT ... ENDSELECT Statement</td>
<td>A–293</td>
</tr>
<tr>
<td>A.20.4</td>
<td>SELECT_TPE Built-In Procedure</td>
<td>A–294</td>
</tr>
<tr>
<td>A.20.5</td>
<td>SEMA_COUNT Built-In Function</td>
<td>A–294</td>
</tr>
<tr>
<td>A.20.6</td>
<td>SEMAPHORE Condition</td>
<td>A–295</td>
</tr>
<tr>
<td>A.20.7</td>
<td>SEND_DATAPC Built-In Procedure</td>
<td>A–295</td>
</tr>
<tr>
<td>A.20.8</td>
<td>SEND_EVENTPC Built-In Procedure</td>
<td>A–296</td>
</tr>
<tr>
<td>A.20.9</td>
<td>SET_ATTR_PRG Built-In Procedure</td>
<td>A–297</td>
</tr>
<tr>
<td>A.20.10</td>
<td>SET_CURSOR Built-In Procedure</td>
<td>A–297</td>
</tr>
<tr>
<td>A.20.11</td>
<td>SET_EPOS_REG Built-In Procedure</td>
<td>A–300</td>
</tr>
<tr>
<td>A.20.12</td>
<td>SET_EPOS_TPE Built-In Procedure</td>
<td>A–301</td>
</tr>
<tr>
<td>A.20.13</td>
<td>SET_FILE_ATR Built-In Procedure</td>
<td>A–302</td>
</tr>
<tr>
<td>A.20.14</td>
<td>SET_FILE_POS Built-In Procedure</td>
<td>A–303</td>
</tr>
<tr>
<td>A.20.15</td>
<td>SET_INT_REG Built-In Procedure</td>
<td>A–304</td>
</tr>
<tr>
<td>A.20.16</td>
<td>SET_JPOS_REG Built-In Procedure</td>
<td>A–304</td>
</tr>
<tr>
<td>A.20.17</td>
<td>SET_JPOS_TPE Built-In Procedure</td>
<td>A–305</td>
</tr>
<tr>
<td>A.20.18</td>
<td>SET_LANG Built-In Procedure</td>
<td>A–306</td>
</tr>
<tr>
<td>A.20.19</td>
<td>SET_PERCH Built-In Procedure</td>
<td>A–307</td>
</tr>
<tr>
<td>A.20.20</td>
<td>SET_PORT_ASG Built-In Procedure</td>
<td>A–308</td>
</tr>
<tr>
<td>A.20.21</td>
<td>SET_PORT_ATR Built-In Function</td>
<td>A–309</td>
</tr>
<tr>
<td>A.20.22</td>
<td>SET_PORT_CMT Built-In Procedure</td>
<td>A–311</td>
</tr>
<tr>
<td>A.20.23</td>
<td>SET_PORT_MOD Built-In Procedure</td>
<td>A–312</td>
</tr>
<tr>
<td>A.20.24</td>
<td>SET_PORT_SIM Built-In Procedure</td>
<td>A–313</td>
</tr>
<tr>
<td>A.20.25</td>
<td>SET_PORT_VAL Built-In Procedure</td>
<td>A–314</td>
</tr>
<tr>
<td>A.20.26</td>
<td>SET_POS_VAL Built-In Procedure</td>
<td>A–315</td>
</tr>
</tbody>
</table>
A.20.27 SET_POS_TPE Built-In Procedure ................................................................. A–316
A.20.28 SET_PRG_CMT Built-In-Procedure .......................................................... A–317
A.20.29 SET_REAL_REG Built-In Procedure ......................................................... A–317
A.20.30 SET_REG_CMT Built-In Procedure ....................................................... A–317
A.20.31 SET_SREG_CMT Built-in Procedure ....................................................... A–318
A.20.32 SET_STR_REG Built-in Procedure ......................................................... A–319
A.20.33 SET_TIME Built-In Procedure ............................................................... A–319
A.20.34 SET_TPE_CMT Built-In Procedure ....................................................... A–321
A.20.35 SET_TRNS_TPE Built-In Procedure ...................................................... A–321
A.20.36 SET_TSK_ATTR Built-In Procedure ...................................................... A–322
A.20.37 SET_TSK_NAME Built-In Procedure .................................................... A–323
A.20.38 SET_VAR Built-In Procedure ............................................................... A–324
A.20.39 %SHADOWVARS Translator Directive ................................................ A–327
A.20.40 SHORT Data Type .................................................................................. A–327
A.20.41 SIGNAL EVENT Action .......................................................................... A–328
A.20.42 SIGNAL EVENT Statement ................................................................. A–328
A.20.43 SIGNAL SEMAPHORE Action ............................................................... A–329
A.20.44 SIN Built-In Function .......................................................................... A–329
A.20.45 SQRT Built-In Function ........................................................................ A–330
A.20.46 %STACKSIZE Translator Directive ..................................................... A–330
A.20.47 STD_PTH_NODE Data Type ................................................................. A–331
A.20.48 STOP Action ......................................................................................... A–331
A.20.49 STOP Statement ................................................................................... A–332
A.20.50 STRING Data Type ................................................................................ A–333
A.20.51 STR_LEN Built-In Function .................................................................. A–334
A.20.52 STRUCTURE Data Type ....................................................................... A–334
A.20.53 SUB_STR Built-In Function ................................................................... A–335
A.21 - T - KAREL LANGUAGE DESCRIPTION .................................................. A–336
A.21.1 TAN Built-In Function ........................................................................... A–336
A.21.2 %TIMESLICE Translator Directive ...................................................... A–337
A.21.3 %TPMOTION Translator Directive ...................................................... A–337
A.21.4 TRANSLATE Built-In Procedure ........................................................... A–337
A.21.5 TRUNC Built-In Function ...................................................................... A–339
A.22 - U - KAREL LANGUAGE DESCRIPTION .................................................. A–340
A.22.1 UNHOLD Action .................................................................................... A–340
A.22.2 UNHOLD Statement ............................................................................ A–340
A.22.3 UNINIT Built-In Function ..................................................................... A–341
A.22.4 %UNINITVARS Translator Directive ................................................ A–341
A.22.5 UNLOCK_GROUP Built-In Procedure ................................................ A–342
A.22.6 UNPAUSE Action ................................................................................ A–343
A.22.7 UNPOS Built-In Procedure ................................................................... A–344
A.22.8 USING ... ENDUSING Statement ....................................................... A–344
A.23 - V - KAREL LANGUAGE DESCRIPTION .................................................. A–345
A.23.1 V_CAM_CALIB /RVision Built-In Procedure ........................................ A–345
A.23.2 V_GET_OFFSET /RVision Built-In Procedure ........................................ A–347
A.23.3 V_GET_PASSFL /RVision Built-In Procedure ......................................... A–348
A.23.4 V_GET_QUEUE /RVision Built-in Procedure ....................................... A–350
A.23.5 V_INIT_QUEUE /RVision Built-in Procedure ....................................... A–350
A.23.6 V_RALC_QUEUE /RVision Built-in procedure ................................... A–351
A.23.7 V_RUN_FIND /RVision Built-In Procedure ........................................... A–351
A.23.8 V_SET_REF /RVision Built-In Procedure ............................................. A–353
A.23.9 V_START_VTRK /RVision Built-in Procedure ....................................... A–354
A.23.10 V_STOP_VTRK /RVision Built-In Procedure ........................................ A–354
A.23.11 VAR_INFO Built-In Procedure ............................................................... A–355
# MARRC75KR07091E Rev D Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.23.12</td>
<td>VAR_LIST Built-In Procedure</td>
</tr>
<tr>
<td>A.23.13</td>
<td>VECTOR Data Type</td>
</tr>
<tr>
<td>A.23.14</td>
<td>VOL_SPACE Built-In Procedure</td>
</tr>
<tr>
<td>A.23.15</td>
<td>VREG_FND_POS /Vison Built-in Procedure</td>
</tr>
<tr>
<td>A.23.16</td>
<td>VREG_OFFSET /Vision Built-in Procedure</td>
</tr>
<tr>
<td>A.24</td>
<td>-W- KAREL LANGUAGE DESCRIPTION</td>
</tr>
<tr>
<td>A.24.1</td>
<td>WAIT FOR Statement</td>
</tr>
<tr>
<td>A.24.2</td>
<td>WHEN Clause</td>
</tr>
<tr>
<td>A.24.3</td>
<td>WHILE...ENDWHILE Statement</td>
</tr>
<tr>
<td>A.24.4</td>
<td>WITH Clause</td>
</tr>
<tr>
<td>A.24.5</td>
<td>WRITE Statement</td>
</tr>
<tr>
<td>A.24.6</td>
<td>WRITE_DICT Built-In Procedure</td>
</tr>
<tr>
<td>A.24.7</td>
<td>WRITE_DICT_V Built-In Procedure</td>
</tr>
<tr>
<td>A.25</td>
<td>-X- KAREL LANGUAGE DESCRIPTION</td>
</tr>
<tr>
<td>A.25.1</td>
<td>XML_ADDTAG Built-In Procedure</td>
</tr>
<tr>
<td>A.25.2</td>
<td>XML_GETDATA Built-In Procedure</td>
</tr>
<tr>
<td>A.25.3</td>
<td>XML_REMTAG Built-In Procedure</td>
</tr>
<tr>
<td>A.25.4</td>
<td>XML_SCAN Built-In Procedure</td>
</tr>
<tr>
<td>A.25.5</td>
<td>XML_SETVAR Built-In Procedure</td>
</tr>
<tr>
<td>A.25.6</td>
<td>XYZWPR Data Type</td>
</tr>
<tr>
<td>A.25.7</td>
<td>XYZWPREXT Data Type</td>
</tr>
<tr>
<td>A.26</td>
<td>-Y- KAREL LANGUAGE DESCRIPTION</td>
</tr>
<tr>
<td>A.27</td>
<td>-Z- KAREL LANGUAGE DESCRIPTION</td>
</tr>
</tbody>
</table>

## Appendix B

### KAREL EXAMPLE PROGRAMS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>SETTING UP DIGITAL OUTPUT PORTS FOR PROCESS MONITORING</td>
</tr>
<tr>
<td>B.2</td>
<td>COPYING PATH VARIABLES</td>
</tr>
<tr>
<td>B.3</td>
<td>SAVING DATA TO THE DEFAULT DEVICE</td>
</tr>
<tr>
<td>B.4</td>
<td>STANDARD ROUTINES</td>
</tr>
<tr>
<td>B.5</td>
<td>USING REGISTER BUILT-INS</td>
</tr>
<tr>
<td>B.6</td>
<td>PATH VARIABLES AND CONDITION HANDLERS PROGRAM</td>
</tr>
<tr>
<td>B.7</td>
<td>LISTING FILES AND PROGRAMS AND MANIPULATING STRINGS</td>
</tr>
<tr>
<td>B.8</td>
<td>GENERATING AND MOVING ALONG A HEXAGON PATH</td>
</tr>
<tr>
<td>B.9</td>
<td>USING THE FILE AND DEVICE BUILT-INS</td>
</tr>
<tr>
<td>B.10</td>
<td>USING DYNAMIC DISPLAY BUILT-INS</td>
</tr>
<tr>
<td>B.11</td>
<td>MANIPULATING VALUES OF DYNAMICALLY DISPLAYED VARIABLES</td>
</tr>
<tr>
<td>B.12</td>
<td>Displaying a List from a Dictionary File</td>
</tr>
<tr>
<td>B.12.1</td>
<td>Dictionary Files</td>
</tr>
<tr>
<td>B.13</td>
<td>USING THE DISCTRL_ALPHA BUILT-IN</td>
</tr>
<tr>
<td>B.13.1</td>
<td>Dictionary Files</td>
</tr>
<tr>
<td>B.14</td>
<td>APPLYING OFFSETS TO A COPIED TEACH PENDANT PROGRAM</td>
</tr>
</tbody>
</table>

## Appendix C

### KCL COMMAND ALPHABETICAL DESCRIPTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>ABORT command</td>
</tr>
<tr>
<td>C.2</td>
<td>APPEND FILE command</td>
</tr>
<tr>
<td>C.3</td>
<td>APPEND NODE command</td>
</tr>
<tr>
<td>C.4</td>
<td>CHDIR command</td>
</tr>
</tbody>
</table>
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 V ARS 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 V ARS 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
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 V ARS 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
C.103 WAIT command ................................................................. C–52

Appendix D  CHARACTER CODES ......................................................... D–1
D.1  CHARACTER CODES ................................................................. D–2

Appendix E  SYNTAX DIAGRAMS ......................................................... E–1

Glossary ............................................................... GL–28

Index ............................................................... Index–1
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>Controller Memory</td>
<td>1–9</td>
</tr>
<tr>
<td>1–2</td>
<td>R-30/A Controller</td>
<td>1–11</td>
</tr>
<tr>
<td>3–1</td>
<td>Determining ( w\text{_handle} ) Relative to WORLD Frame</td>
<td>3–10</td>
</tr>
<tr>
<td>3–2</td>
<td>Determining ( b\text{_handle} ) Relative to BUMPER Frame</td>
<td>3–11</td>
</tr>
<tr>
<td>7–1</td>
<td>( t\text{_sc} ) Screen</td>
<td>7–39</td>
</tr>
<tr>
<td>7–2</td>
<td>( t\text{_sc} ) Screen with $TP_USESTAT = \text{TRUE}</td>
<td>7–39</td>
</tr>
<tr>
<td>7–3</td>
<td>( c\text{_sc} ) Screen</td>
<td>7–40</td>
</tr>
<tr>
<td>7–4</td>
<td>( c\text{_sc} ) Screen with $CRT_USERSTAT = \text{TRUE}</td>
<td>7–41</td>
</tr>
<tr>
<td>8–1</td>
<td>Referencing Positions in KAREL</td>
<td>8–4</td>
</tr>
<tr>
<td>9–1</td>
<td>XML File 1 Separated start and end tags</td>
<td>9–22</td>
</tr>
<tr>
<td>9–2</td>
<td>XML File 2 Combined start and end tags</td>
<td>9–22</td>
</tr>
<tr>
<td>9–3</td>
<td>XML File 3 GRID tag not registered or processed</td>
<td>9–22</td>
</tr>
<tr>
<td>9–4</td>
<td>KAREL Program</td>
<td>9–23</td>
</tr>
<tr>
<td>10–1</td>
<td>Dictionary Compressor and User Dictionary File</td>
<td>10–12</td>
</tr>
<tr>
<td>10–2</td>
<td>Teach Pendant Form Screen</td>
<td>10–32</td>
</tr>
<tr>
<td>10–3</td>
<td>CRT/KB Form Screen</td>
<td>10–33</td>
</tr>
<tr>
<td>10–4</td>
<td>Dictionary Compressor and Form Dictionary File</td>
<td>10–34</td>
</tr>
<tr>
<td>10–5</td>
<td>Example of Selectable Menu Items</td>
<td>10–36</td>
</tr>
<tr>
<td>10–6</td>
<td>Example of Edit Data Items</td>
<td>10–38</td>
</tr>
<tr>
<td>10–7</td>
<td>Example of Display Only Data Items</td>
<td>10–42</td>
</tr>
<tr>
<td>14–1</td>
<td>KAREL Logic for Converting Input to a Real Value Representing the Voltage</td>
<td>14–4</td>
</tr>
<tr>
<td>14–2</td>
<td>RSR Timing Diagram</td>
<td>14–16</td>
</tr>
<tr>
<td>14–3</td>
<td>PNS Timing Diagram</td>
<td>14–17</td>
</tr>
<tr>
<td>14–4</td>
<td>Location of Ports on the Controller</td>
<td>14–23</td>
</tr>
<tr>
<td>15–1</td>
<td>Task Synchronization Using a Semaphore</td>
<td>15–10</td>
</tr>
<tr>
<td>A–1</td>
<td>Example of COMPARE_FILE built-in Procedure</td>
<td>A–74</td>
</tr>
<tr>
<td>A–2</td>
<td>FRAME Built-In Function</td>
<td>A–143</td>
</tr>
<tr>
<td>A–3</td>
<td>GET_USEC_SUB Built-In Function</td>
<td>A–171</td>
</tr>
<tr>
<td>E–1</td>
<td></td>
<td>E–3</td>
</tr>
<tr>
<td>E–2</td>
<td></td>
<td>E–4</td>
</tr>
<tr>
<td>E–3</td>
<td></td>
<td>E–5</td>
</tr>
<tr>
<td>E–4</td>
<td></td>
<td>E–6</td>
</tr>
<tr>
<td>E–5</td>
<td></td>
<td>E–7</td>
</tr>
</tbody>
</table>
Table 2–1. ASCII Character Set .......................................................................................................... 2–2
Table 2–2. Multinational Character Set ................................................................................................. 2–4
Table 2–3. Graphics Character Set ....................................................................................................... 2–5
Table 2–4. KAREL Operators ............................................................................................................ 2–5
Table 2–5. KAREL Operator Precedence .............................................................................................. 2–6
Table 2–6. Reserved Word List ........................................................................................................... 2–6
Table 2–7. Predefined Identifier and Value Summary .............................................................................. 2–8
Table 2–8. Port and File Predefined Identifier Summary ........................................................................ 2–9
Table 2–9. Translator Directives ....................................................................................................... 2–11
Table 2–10. Simple and Structured Data Types ...................................................................................... 2–14
Table 3–1. Summary of Operation Result Types ..................................................................................... 3–3
Table 3–2. KAREL Operators ............................................................................................................ 3–5
Table 3–3. Arithmetic Operations Using +, -, and * Operators .................................................................. 3–5
Table 3–4. Arithmetic Operations Examples .......................................................................................... 3–5
Table 3–5. Arithmetic Operations Using Bitwise Operands ...................................................................... 3–6
Table 3–6. Relational Operation Examples ............................................................................................ 3–7
Table 3–7. BOOLEAN Operation Summary .......................................................................................... 3–8
Table 3–8. BOOLEAN Operations Using AND, OR, and NOT Operators ................................................... 3–8
Table 3–9. Examples of Vector Operations .......................................................................................... 3–13
Table 5–1. Stack Usage ................................................................................................................... 5–13
Table 5–2. KAREL Built-in Routine Summary .................................................................................. 5–16
Table 6–1. Conditions ....................................................................................................................... 6–2
Table 6–2. Actions ........................................................................................................................... 6–2
Table 6–3. Condition Handler Operations ............................................................................................. 6–3
Table 6–4. Interval Between Global Condition Handler Scans ................................................................ 6–4
Table 6–5. Port_Id Conditions ........................................................................................................ 6–7
Table 6–6. Relational Conditions ........................................................................................................ 6–8
Table 6–7. System and Program Event Conditions ................................................................................ 6–9
Table 6–8. Assignment Actions ........................................................................................................ 6–12
Table 6–9. Motion Related Actions .................................................................................................... 6–13
Table 6–10. Miscellaneous Actions ..................................................................................................... 6–14
Table 7–1. Predefined File Variables .................................................................................................... 7–3
| Table A–2. | Actions | A–11 |
| Table A–3. | Clauses | A–11 |
| Table A–4. | Conditions | A–12 |
| Table A–5. | Data Types | A–12 |
| Table A–6. | Directives | A–13 |
| Table A–7. | KAREL Built—In Routine Summary | A–13 |
| Table A–8. | Items | A–16 |
| Table A–9. | Statements | A–17 |
| Table A–10. | Valid and Invalid BOOLEAN Values | A–42 |
| Table A–11. | INTEGER Representation of Current Time | A–165 |
| Table A–12. | Conversion Characters | A–185 |
| Table A–13. | Conversion Characters | A–193 |
| Table A–14. | Valid and Invalid INTEGER Literals | A–204 |
| Table A–15. | IO_STATUS Errors | A–207 |
| Table A–16. | Group_mask Setting | A–216 |
| Table A–17. | Group_mask Setting | A–221 |
| Table A–18. | Valid and Invalid REAL operators | A–278 |
| Table A–19. | Group_mask Setting | A–291 |
| Table A–20. | Attribute Values | A–310 |
| Table A–21. | 32–Bit INTEGER Format of Time | A–320 |
| Table A–22. | Example STRING Literals | A–333 |
| Table A–23. | Group_mask Settings | A–342 |
| Table A–24. | Function Code Values | A–346 |
| Table A–25. | Valid Data Types | A–356 |
| Table A–26. | Valid Data Types | A–358 |
| Table B–1. | KAREL Example Programs | B–2 |
| Table D–1. | ASCII Character Codes | D–2 |
| Table D–2. | Special ASCII Character Codes | D–3 |
| Table D–3. | Multinational Character Codes | D–4 |
| Table D–4. | Graphics Character Codes (not available in R-30iB) | D–6 |
| Table D–5. | Teach Pendant Input Codes | D–7 |
| Table D–6. | European Character Codes | D–8 |
| Table D–7. | Graphics Characters | D–10 |
FANUC Robotics is not and does not represent itself as an expert in safety systems, safety equipment, or the specific safety aspects of your company and/or its work force. It is the responsibility of the owner, employer, or user to take all necessary steps to guarantee the safety of all personnel in the workplace.

The appropriate level of safety for your application and installation can best be determined by safety system professionals. FANUC Robotics therefore, recommends that each customer consult with such professionals in order to provide a workplace that allows for the safe application, use, and operation of FANUC Robotic systems.

According to the industry standard ANSI/RIA R15-06, the owner or user is advised to consult the standards to ensure compliance with its requests for Robotics System design, usability, operation, maintenance, and service. Additionally, as the owner, employer, or user of a robotic system, it is your responsibility to arrange for the training of the operator of a robot system to recognize and respond to known hazards associated with your robotic system and to be aware of the recommended operating procedures for your particular application and robot installation.

Ensure that the robot being used is appropriate for the application. Robots used in classified (hazardous) locations must be certified for this use.

FANUC Robotics therefore, recommends that all personnel who intend to operate, program, repair, or otherwise use the robotics system be trained in an approved FANUC Robotics training course and become familiar with the proper operation of the system. Persons responsible for programming the system-including the design, implementation, and debugging of application programs-must be familiar with the recommended programming procedures for your application and robot installation.

The following guidelines are provided to emphasize the importance of safety in the workplace.

**CONSIDERING SAFETY FOR YOUR ROBOT INSTALLATION**

Safety is essential whenever robots are used. Keep in mind the following factors with regard to safety:

- The safety of people and equipment
- Use of safety enhancing devices
- Techniques for safe teaching and manual operation of the robot(s)
- Techniques for safe automatic operation of the robot(s)
- Regular scheduled inspection of the robot and workcell
- Proper maintenance of the robot
Keeping People Safe

The safety of people is always of primary importance in any situation. When applying safety measures to your robotic system, consider the following:

- External devices
- Robot(s)
- Tooling
- Workpiece

Using Safety Enhancing Devices

Always give appropriate attention to the work area that surrounds the robot. The safety of the work area can be enhanced by the installation of some or all of the following devices:

- Safety fences, barriers, or chains
- Light curtains
- Interlocks
- Pressure mats
- Floor markings
- Warning lights
- Mechanical stops
- EMERGENCY STOP buttons
- DEADMAN switches

Setting Up a Safe Workcell

A safe workcell is essential to protect people and equipment. Observe the following guidelines to ensure that the workcell is set up safely. These suggestions are intended to supplement and not replace existing federal, state, and local laws, regulations, and guidelines that pertain to safety.

- Sponsor your personnel for training in approved FANUC Robotics training course(s) related to your application. Never permit untrained personnel to operate the robots.
- Install a lockout device that uses an access code to prevent unauthorized persons from operating the robot.
- Use anti-tie-down logic to prevent the operator from bypassing safety measures.
- Arrange the workcell so the operator faces the workcell and can see what is going on inside the cell.
- Clearly identify the work envelope of each robot in the system with floor markings, signs, and special barriers. The work envelope is the area defined by the maximum motion range of the robot, including any tooling attached to the wrist flange that extend this range.

- Position all controllers outside the robot work envelope.

- Never rely on software or firmware based controllers as the primary safety element unless they comply with applicable current robot safety standards.

- Mount an adequate number of EMERGENCY STOP buttons or switches within easy reach of the operator and at critical points inside and around the outside of the workcell.

- Install flashing lights and/or audible warning devices that activate whenever the robot is operating, that is, whenever power is applied to the servodrive system. Audible warning devices shall exceed the ambient noise level at the end-use application.

- Wherever possible, install safety fences to protect against unauthorized entry by personnel into the work envelope.

- Install special guarding that prevents the operator from reaching into restricted areas of the work envelope.

- Use interlocks.

- Use presence or proximity sensing devices such as light curtains, mats, and capacitance and vision systems to enhance safety.

- Periodically check the safety joints or safety clutches that can be optionally installed between the robot wrist flange and tooling. If the tooling strikes an object, these devices dislodge, remove power from the system, and help to minimize damage to the tooling and robot.

- Make sure all external devices are properly filtered, grounded, shielded, and suppressed to prevent hazardous motion due to the effects of electro-magnetic interference (EMI), radio frequency interference (RFI), and electro-static discharge (ESD).

- Make provisions for power lockout/tagout at the controller.

- Eliminate *pinch points* . Pinch points are areas where personnel could get trapped between a moving robot and other equipment.

- Provide enough room inside the workcell to permit personnel to teach the robot and perform maintenance safely.

- Program the robot to load and unload material safely.

- If high voltage electrostatics are present, be sure to provide appropriate interlocks, warning, and beacons.

- If materials are being applied at dangerously high pressure, provide electrical interlocks for lockout of material flow and pressure.
Staying Safe While Teaching or Manually Operating the Robot

Advise all personnel who must teach the robot or otherwise manually operate the robot to observe the following rules:

• Never wear watches, rings, neckties, scarves, or loose clothing that could get caught in moving machinery.
• Know whether or not you are using an intrinsically safe teach pendant if you are working in a hazardous environment.
• Before teaching, visually inspect the robot and work envelope to make sure that no potentially hazardous conditions exist. The work envelope is the area defined by the maximum motion range of the robot. These include tooling attached to the wrist flange that extends this range.
• The area near the robot must be clean and free of oil, water, or debris. Immediately report unsafe working conditions to the supervisor or safety department.
• FANUC Robotics recommends that no one enter the work envelope of a robot that is on, except for robot teaching operations. However, if you must enter the work envelope, be sure all safeguards are in place, check the teach pendant DEADMAN switch for proper operation, and place the robot in teach mode. Take the teach pendant with you, turn it on, and be prepared to release the DEADMAN switch. Only the person with the teach pendant should be in the work envelope.

Warning

Never bypass, strap, or otherwise deactivate a safety device, such as a limit switch, for any operational convenience. Deactivating a safety device is known to have resulted in serious injury and death.

• Know the path that can be used to escape from a moving robot; make sure the escape path is never blocked.
• Isolate the robot from all remote control signals that can cause motion while data is being taught.
• Test any program being run for the first time in the following manner:

Warning

Stay outside the robot work envelope whenever a program is being run. Failure to do so can result in injury.

— Using a low motion speed, single step the program for at least one full cycle.
— Using a low motion speed, test run the program continuously for at least one full cycle.
— Using the programmed speed, test run the program continuously for at least one full cycle.
• Make sure all personnel are outside the work envelope before running production.
Staying Safe During Automatic Operation

Advise all personnel who operate the robot during production to observe the following rules:

- Make sure all safety provisions are present and active.
- Know the entire workcell area. The workcell includes the robot and its work envelope, plus the area occupied by all external devices and other equipment with which the robot interacts.
- Understand the complete task the robot is programmed to perform before initiating automatic operation.
- Make sure all personnel are outside the work envelope before operating the robot.
- Never enter or allow others to enter the work envelope during automatic operation of the robot.
- Know the location and status of all switches, sensors, and control signals that could cause the robot to move.
- Know where the EMERGENCY STOP buttons are located on both the robot control and external control devices. Be prepared to press these buttons in an emergency.
- Never assume that a program is complete if the robot is not moving. The robot could be waiting for an input signal that will permit it to continue activity.
- If the robot is running in a pattern, do not assume it will continue to run in the same pattern.
- Never try to stop the robot, or break its motion, with your body. The only way to stop robot motion immediately is to press an EMERGENCY STOP button located on the controller panel, teach pendant, or emergency stop stations around the workcell.

Staying Safe During Inspection

When inspecting the robot, be sure to

- Turn off power at the controller.
- Lock out and tag out the power source at the controller according to the policies of your plant.
- Turn off the compressed air source and relieve the air pressure.
- If robot motion is not needed for inspecting the electrical circuits, press the EMERGENCY STOP button on the operator panel.
- Never wear watches, rings, neckties, scarves, or loose clothing that could get caught in moving machinery.
- If power is needed to check the robot motion or electrical circuits, be prepared to press the EMERGENCY STOP button, in an emergency.
- Be aware that when you remove a servomotor or brake, the associated robot arm will fall if it is not supported or resting on a hard stop. Support the arm on a solid support before you release the brake.
Staying Safe During Maintenance

When performing maintenance on your robot system, observe the following rules:

- Never enter the work envelope while the robot or a program is in operation.
- Before entering the work envelope, visually inspect the workcell to make sure no potentially hazardous conditions exist.
- Never wear watches, rings, neckties, scarves, or loose clothing that could get caught in moving machinery.
- Consider all or any overlapping work envelopes of adjoining robots when standing in a work envelope.
- Test the teach pendant for proper operation before entering the work envelope.
- If it is necessary for you to enter the robot work envelope while power is turned on, you must be sure that you are in control of the robot. Be sure to take the teach pendant with you, press the DEADMAN switch, and turn the teach pendant on. Be prepared to release the DEADMAN switch to turn off servo power to the robot immediately.
- Whenever possible, perform maintenance with the power turned off. Before you open the controller front panel or enter the work envelope, turn off and lock out the 3-phase power source at the controller.
- Be aware that an applicator bell cup can continue to spin at a very high speed even if the robot is idle. Use protective gloves or disable bearing air and turbine air before servicing these items.
- Be aware that when you remove a servomotor or brake, the associated robot arm will fall if it is not supported or resting on a hard stop. Support the arm on a solid support before you release the brake.

**Warning**

Lethal voltage is present in the controller WHENEVER IT IS CONNECTED to a power source. Be extremely careful to avoid electrical shock. HIGH VOLTAGE IS PRESENT at the input side whenever the controller is connected to a power source. Turning the disconnect or circuit breaker to the OFF position removes power from the output side of the device only.

- Release or block all stored energy. Before working on the pneumatic system, shut off the system air supply and purge the air lines.
- Isolate the robot from all remote control signals. If maintenance must be done when the power is on, make sure the person inside the work envelope has sole control of the robot. The teach pendant must be held by this person.
• Make sure personnel cannot get trapped between the moving robot and other equipment. Know the path that can be used to escape from a moving robot. Make sure the escape route is never blocked.

• Use blocks, mechanical stops, and pins to prevent hazardous movement by the robot. Make sure that such devices do not create pinch points that could trap personnel.

**Warning**

Do not try to remove any mechanical component from the robot before thoroughly reading and understanding the procedures in the appropriate manual. Doing so can result in serious personal injury and component destruction.

• Be aware that when you remove a servomotor or brake, the associated robot arm will fall if it is not supported or resting on a hard stop. Support the arm on a solid support before you release the brake.

• When replacing or installing components, make sure dirt and debris do not enter the system.

• Use only specified parts for replacement. To avoid fires and damage to parts in the controller, never use nonspecified fuses.

• Before restarting a robot, make sure no one is inside the work envelope; be sure that the robot and all external devices are operating normally.

**KEEPING MACHINE TOOLS AND EXTERNAL DEVICES SAFE**

Certain programming and mechanical measures are useful in keeping the machine tools and other external devices safe. Some of these measures are outlined below. Make sure you know all associated measures for safe use of such devices.

**Programming Safety Precautions**

Implement the following programming safety measures to prevent damage to machine tools and other external devices.

• Back-check limit switches in the workcell to make sure they do not fail.

• Implement “failure routines” in programs that will provide appropriate robot actions if an external device or another robot in the workcell fails.

• Use *handshaking* protocol to synchronize robot and external device operations.

• Program the robot to check the condition of all external devices during an operating cycle.
Mechanical Safety Precautions

Implement the following mechanical safety measures to prevent damage to machine tools and other external devices.

- Make sure the workcell is clean and free of oil, water, and debris.
- Use DCS (Dual Check Safety), software limits, limit switches, and mechanical hardstops to prevent undesired movement of the robot into the work area of machine tools and external devices.

KEEPING THE ROBOT SAFE

Observe the following operating and programming guidelines to prevent damage to the robot.

Operating Safety Precautions

The following measures are designed to prevent damage to the robot during operation.

- Use a low override speed to increase your control over the robot when jogging the robot.
- Visualize the movement the robot will make before you press the jog keys on the teach pendant.
- Make sure the work envelope is clean and free of oil, water, or debris.
- Use circuit breakers to guard against electrical overload.

Programming Safety Precautions

The following safety measures are designed to prevent damage to the robot during programming:

- Establish *interference zones* to prevent collisions when two or more robots share a work area.
- Make sure that the program ends with the robot near or at the home position.
- Be aware of signals or other operations that could trigger operation of tooling resulting in personal injury or equipment damage.
- In dispensing applications, be aware of all safety guidelines with respect to the dispensing materials.

**Note** Any deviation from the methods and safety practices described in this manual must conform to the approved standards of your company. If you have questions, see your supervisor.
ADDITIONAL SAFETY CONSIDERATIONS FOR PAINT ROBOT INSTALLATIONS

Process technicians are sometimes required to enter the paint booth, for example, during daily or routine calibration or while teaching new paths to a robot. Maintenance personal also must work inside the paint booth periodically.

Whenever personnel are working inside the paint booth, ventilation equipment must be used. Instruction on the proper use of ventilating equipment usually is provided by the paint shop supervisor.

Although paint booth hazards have been minimized, potential dangers still exist. Therefore, today’s highly automated paint booth requires that process and maintenance personnel have full awareness of the system and its capabilities. They must understand the interaction that occurs between the vehicle moving along the conveyor and the robot(s), hood/deck and door opening devices, and high-voltage electrostatic tools.

**Caution**

Ensure that all ground cables remain connected. Never operate the paint robot with ground provisions disconnected. Otherwise, you could injure personnel or damage equipment.

Paint robots are operated in three modes:

- Teach or manual mode
- Automatic mode, including automatic and exercise operation
- Diagnostic mode

During both teach and automatic modes, the robots in the paint booth will follow a predetermined pattern of movements. In teach mode, the process technician teaches (programs) paint paths using the teach pendant.

In automatic mode, robot operation is initiated at the System Operator Console (SOC) or Manual Control Panel (MCP), if available, and can be monitored from outside the paint booth. All personnel must remain outside of the booth or in a designated safe area within the booth whenever automatic mode is initiated at the SOC or MCP.

In automatic mode, the robots will execute the path movements they were taught during teach mode, but generally at production speeds.
When process and maintenance personnel run diagnostic routines that require them to remain in the paint booth, they must stay in a designated safe area.

Paint System Safety Features

Process technicians and maintenance personnel must become totally familiar with the equipment and its capabilities. To minimize the risk of injury when working near robots and related equipment, personnel must comply strictly with the procedures in the manuals.

This section provides information about the safety features that are included in the paint system and also explains the way the robot interacts with other equipment in the system.

The paint system includes the following safety features:

- Most paint booths have red warning beacons that illuminate when the robots are armed and ready to paint. Your booth might have other kinds of indicators. Learn what these are.
- Some paint booths have a blue beacon that, when illuminated, indicates that the electrostatic devices are enabled. Your booth might have other kinds of indicators. Learn what these are.
- EMERGENCY STOP buttons are located on the robot controller and teach pendant. Become familiar with the locations of all E-STOP buttons.
- An intrinsically safe teach pendant is used when teaching in hazardous paint atmospheres.
- A DEADMAN switch is located on each teach pendant. When this switch is held in, and the teach pendant is on, power is applied to the robot servo system. If the engaged DEADMAN switch is released during robot operation, power is removed from the servo system, all axis brakes are applied, and the robot comes to an EMERGENCY STOP. Safety interlocks within the system might also E-STOP other robots.

Warning

An EMERGENCY STOP will occur if the DEADMAN switch is released on a bypassed robot.

- Overtravel by robot axes is prevented by software limits. All of the major and minor axes are governed by software limits. DCS (Dual Check Safety), limit switches and hardstops also limit travel by the major axes.
- EMERGENCY STOP limit switches and photoelectric eyes might be part of your system. Limit switches, located on the entrance/exit doors of each booth, will EMERGENCY STOP all equipment in the booth if a door is opened while the system is operating in automatic or manual mode. For some systems, signals to these switches are inactive when the switch on the SOC is
in teach mode. When present, photoelectric eyes are sometimes used to monitor unauthorized intrusion through the entrance/exit silhouette openings.

- System status is monitored by computer. Severe conditions result in automatic system shutdown.

Staying Safe While Operating the Paint Robot

When you work in or near the paint booth, observe the following rules, in addition to all rules for safe operation that apply to all robot systems.

**Warning**

Observe all safety rules and guidelines to avoid injury.

**Warning**

Never bypass, strap, or otherwise deactivate a safety device, such as a limit switch, for any operational convenience. Deactivating a safety device is known to have resulted in serious injury and death.

**Warning**

Enclosures shall not be opened unless the area is known to be nonhazardous or all power has been removed from devices within the enclosure. Power shall not be restored after the enclosure has been opened until all combustible dusts have been removed from the interior of the enclosure and the enclosure purged. Refer to the Purge chapter for the required purge time.

- Know the work area of the entire paint station (workcell).
- Know the work envelope of the robot and hood/deck and door opening devices.
- Be aware of overlapping work envelopes of adjacent robots.
- Know where all red, mushroom-shaped EMERGENCY STOP buttons are located.
- Know the location and status of all switches, sensors, and/or control signals that might cause the robot, conveyor, and opening devices to move.
- Make sure that the work area near the robot is clean and free of water, oil, and debris. Report unsafe conditions to your supervisor.
- Become familiar with the complete task the robot will perform BEFORE starting automatic mode.
• Make sure all personnel are outside the paint booth before you turn on power to the robot servo system.
• Never enter the work envelope or paint booth before you turn off power to the robot servo system.
• Never enter the work envelope during automatic operation unless a safe area has been designated.
• Never wear watches, rings, neckties, scarves, or loose clothing that could get caught in moving machinery.
• Remove all metallic objects, such as rings, watches, and belts, before entering a booth when the electrostatic devices are enabled.
• Stay out of areas where you might get trapped between a moving robot, conveyor, or opening device and another object.
• Be aware of signals and/or operations that could result in the triggering of guns or bells.
• Be aware of all safety precautions when dispensing of paint is required.
• Follow the procedures described in this manual.

Special Precautions for Combustible Dusts (powder paint)

When the robot is used in a location where combustible dusts are found, such as the application of powder paint, the following special precautions are required to insure that there are no combustible dusts inside the robot.

• Purge maintenance air should be maintained at all times, even when the robot power is off. This will insure that dust can not enter the robot.
• A purge cycle will not remove accumulated dusts. Therefore, if the robot is exposed to dust when maintenance air is not present, it will be necessary to remove the covers and clean out any accumulated dust. Do not energize the robot until you have performed the following steps.

1. Before covers are removed, the exterior of the robot should be cleaned to remove accumulated dust.
2. When cleaning and removing accumulated dust, either on the outside or inside of the robot, be sure to use methods appropriate for the type of dust that exists. Usually lint free rags dampened with water are acceptable. Do not use a vacuum cleaner to remove dust as it can generate static electricity and cause an explosion unless special precautions are taken.
3. Thoroughly clean the interior of the robot with a lint free rag to remove any accumulated dust.
4. When the dust has been removed, the covers must be replaced immediately.
5. Immediately after the covers are replaced, run a complete purge cycle. The robot can now be energized.
Staying Safe While Operating Paint Application Equipment

When you work with paint application equipment, observe the following rules, in addition to all rules for safe operation that apply to all robot systems.

**Warning**

> When working with electrostatic paint equipment, follow all national and local codes as well as all safety guidelines within your organization. Also reference the following standards: NFPA 33 Standards for Spray Application Using Flammable or Combustible Materials, and NFPA 70 National Electrical Code.

- **Grounding**: All electrically conductive objects in the spray area must be grounded. This includes the spray booth, robots, conveyors, workstations, part carriers, hooks, paint pressure pots, as well as solvent containers. Grounding is defined as the object or objects shall be electrically connected to ground with a resistance of not more than 1 megohms.
- **High Voltage**: High voltage should only be on during actual spray operations. Voltage should be off when the painting process is completed. Never leave high voltage on during a cap cleaning process.
- Avoid any accumulation of combustible vapors or coating matter.
- Follow all manufacturer recommended cleaning procedures.
- Make sure all interlocks are operational.
- No smoking.
- Post all warning signs regarding the electrostatic equipment and operation of electrostatic equipment according to NFPA 33 Standard for Spray Application Using Flammable or Combustible Material.
- Disable all air and paint pressure to bell.
- Verify that the lines are not under pressure.

Staying Safe During Maintenance

When you perform maintenance on the painter system, observe the following rules, and all other maintenance safety rules that apply to all robot installations. Only qualified, trained service or maintenance personnel should perform repair work on a robot.

- Paint robots operate in a potentially explosive environment. Use caution when working with electric tools.
- When a maintenance technician is repairing or adjusting a robot, the work area is under the control of that technician. All personnel not participating in the maintenance must stay out of the area.
• For some maintenance procedures, station a second person at the control panel within reach of the EMERGENCY STOP button. This person must understand the robot and associated potential hazards.

• Be sure all covers and inspection plates are in good repair and in place.

• Always return the robot to the “home” position before you disarm it.

• Never use machine power to aid in removing any component from the robot.

• During robot operations, be aware of the robot’s movements. Excess vibration, unusual sounds, and so forth, can alert you to potential problems.

• Whenever possible, turn off the main electrical disconnect before you clean the robot.

• When using vinyl resin observe the following:
  — Wear eye protection and protective gloves during application and removal
  — Adequate ventilation is required. Overexposure could cause drowsiness or skin and eye irritation.
  — If there is contact with the skin, wash with water.
  — Follow the Original Equipment Manufacturer’s Material Safety Data Sheets.

• When using paint remover observe the following:
  — Eye protection, protective rubber gloves, boots, and apron are required during booth cleaning.
  — Adequate ventilation is required. Overexposure could cause drowsiness.
  — If there is contact with the skin or eyes, rinse with water for at least 15 minutes. Then, seek medical attention as soon as possible.
  — Follow the Original Equipment Manufacturer’s Material Safety Data Sheets.
Chapter 1

KAREL LANGUAGE OVERVIEW

Contents

Chapter 1  KAREL LANGUAGE OVERVIEW ................................................................. 1–1
1.1  OVERVIEW ...................................................................................................... 1–2
1.2  KAREL PROGRAMMING LANGUAGE ............................................................ 1–2
  1.2.1 Overview ................................................................................................... 1–2
  1.2.2 Creating a Program ................................................................................... 1–4
  1.2.3 Translating a Program .............................................................................. 1–4
  1.2.4 Loading Program Logic and Data ............................................................... 1–4
  1.2.5 Executing a Program ................................................................................ 1–5
  1.2.6 Execution History ..................................................................................... 1–5
  1.2.7 Program Structure .................................................................................... 1–5
1.3  SYSTEM SOFTWARE ...................................................................................... 1–7
  1.3.1 Software Components .............................................................................. 1–7
  1.3.2 Supported Robots ..................................................................................... 1–7
1.4  CONTROLLER ............................................................................................... 1–8
  1.4.1 Memory ................................................................................................... 1–8
  1.4.2 Input/Output System ............................................................................... 1–10
  1.4.3 User Interface Devices ............................................................................. 1–10
1. KAREL LANGUAGE OVERVIEW

1.1 OVERVIEW

FANUC Robotics’ KAREL system consists of a robot, a controller and system software. It accomplishes industrial tasks using programs written in the KAREL programming language. KAREL can manipulate data, control and communicate with related equipment and interact with an operator.

The SYSTEM R-30iA controller with KAREL works with a wide range of robot models to handle a variety of applications. This means common operating, programming, and troubleshooting procedures, as well as fewer spare parts. KAREL systems expand to include a full line of support products such as integral vision, off-line programming, and application-specific software packages.

The KAREL programming language is a practical blend of the logical, English-like features of high-level languages, such as Pascal and PL/1, and the proven factory-floor effectiveness of machine control languages. KAREL incorporates structures and conventions common to high-level languages as well as features developed especially for robotics applications. These KAREL features include:

- Simple and structured data types
- Arithmetic, relational, and Boolean operators
- Control structures for loops and selections
- Condition handlers
- Procedure and function routines
- Input and output operations
- Multi-programming support

This chapter summarizes the KAREL programming language, and describes the KAREL system software and the controller.

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.

1.2 KAREL PROGRAMMING LANGUAGE

1.2.1 Overview

A KAREL program is made up of declarations and executable statements stored in a source code file. The variable data values associated with a program are stored in a variable file.
KAREL programs are created and edited using OLPC PRO, or another editor such as Word Pad.

The KAREL language translator turns the source code into an internal format called p-code and generates a p-code file. The translator is provided with OLPC PRO. After being translated, the resulting p-code program can be loaded onto the controller using the KAREL Command Language (KCL) or the FILE menu.

During loading, the system will create any required variables that are not in RAM and set them uninitialized. When you run the program, the KAREL interpreter executes the loaded p-code instructions.

A KAREL program is composed of the program logic and the program data. Program logic defines a sequence of steps to be taken to perform a specific task. Program data is the task-related information that the program logic uses. In KAREL the program logic is separate from the program data.

Program logic is defined by KAREL executable statements between the BEGIN and the END statements in a KAREL program. Program data includes variables that are identified in the VAR declaration section of a KAREL program by name, data type and storage area in RAM.

Values for program data can be taught using the teach pendant to jog the robot, computed by the program, read from data files, set from within the CRT/KB or teach pendant menu structure, or accepted as input to the program during execution. The data values can change from one execution to the next, but the same program logic is used to manipulate the data.

Program logic and program data are separate in a KAREL program for the following reasons:

- To allow data to be referenced from several places in the same program
- To allow more than one program to reference or share the same data
- To allow a program to use alternative data
- To facilitate the building of data files by an off-line computer-aided design (CAD) system

The executable section of the program contains the data manipulation statements, I/O statements, and routine calls.

The program development cycle is described briefly in the following list. Section 1.2.2 - Section 1.2.6 that follow provide details on each phase.

- Create a program source code file
- Translate the program file.
- Load the program logic and data.
- Execute the program.
- Maintain the execution history of the program.

A log or history of programs that have been executed is maintained by the controller and can be viewed.
1.2.2 Creating a Program

You can create a KAREL program using an off-line editor such as OLPC PRO or any text editor such as WordPad. The resulting file is called the source file or source code.

1.2.3 Translating a Program

KAREL source files must be translated into internal code, called p-code, before they are executed. The KAREL language translator performs this function and also checks for errors in the source code.

The KAREL language translator starts at the first line of the source code and continues until it encounters an error or translates the program successfully. If an error is encountered, the translator tries to continue checking the program, but no p-code will be generated.

You can invoke the translator from OLPC PRO, and the source code you were editing will be translated. After a successful translation, the translator displays a successful translation message and creates a p-code file. The p-code file will use the source code file name and a .pc file type. This file contains an internal representation of the source code and information the system needs to link the program to variable data and routines.

If the translator detects any errors, it displays the error messages and the source lines that were being translated. After you have corrected the errors, you can translate the program again.

1.2.4 Loading Program Logic and Data

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.

The p-code for a program is loaded onto a controller where it can be executed. When a program is loaded, a variable data table, containing all the static variables in the program, is created in RAM. The variable data table contains the program identifier, all of the variable identifiers, and the name of the storage area in RAM where the variables are located.

Loading a program also establishes the links between statements and variables. Initially, the values in the variable data table will be uninitialized. If a variable file (.vr) is loaded successfully, the values of any variables will be stored in the variable data storage area (CMOS, DRAM, SHADOW).
Multiple programs are often used to break a large application or problem into smaller pieces that can be developed and tested separately. The KAREL system permits loading of multiple programs. Each program that is loaded has its own p-code structure.

Variable data can be shared among multiple programs. In this case, the KAREL language FROM clause must be specified in the VAR declarations so that the system can perform the link when the program is loaded. This saves the storage required to include multiple copies of the data.

The following limits apply to the number and size of KAREL programs that can be loaded:

- Number of programs is limited to 2704 or available RAM.
- Number of variables per program is limited to 2704 or available RAM.

### 1.2.5 Executing a Program

After you have selected a program from the program list and the p-code and variable files are loaded into RAM, test and debug the program to make sure that it operates as intended.

Program execution begins at the first executable line. A stack of 300 words is allocated unless you specify a stack size. The stack is allocated from available user RAM. Stack usage is described in Section 5.1.6.

### 1.2.6 Execution History

Each time a program is executed, a log of the nested routines and the line numbers that have been executed can be displayed from KCL with the SHOW HISTORY command.

This is useful when a program has paused or been aborted unexpectedly. Execution history displays the sequence of events that led to the disruption.

### 1.2.7 Program Structure

A KAREL program is composed of declaration and executable sections made up of KAREL language statements, as shown in Structure of a KAREL Program.

Structure of a KAREL Program

```
PROGRAM prog_name
  Translator Directives
  CONST, TYPE, and/or VAR Declarations
  ROUTINE Declarations
BEGIN
```
End Executable Statements
END prog_name

ROUTINE Declarations

In Structure of a KAREL Program, the words shown in uppercase letters are KAREL reserved words, which have dedicated meanings. PROGRAM, CONST, TYPE, VAR, and ROUTINE indicate declaration sections of the program. BEGIN and END mark the executable section. Reserved words are described in Section 2.1.3.

The PROGRAM statement, which identifies the program, must be the first statement in any KAREL program. The PROGRAM statement consists of the reserved word PROGRAM and an identifier of your choice (prog_name in Structure of a KAREL Program). Identifiers are described in Section 2.1.4.

Note Your program must reside in a file. The file can, but does not have to, have the same name as the program. This distinction is important because you invoke the translator and load programs with the name of the file containing your program, but you initiate execution of the program and clear the program with the program name.

For example, if a program named mover was contained in a file named transfer, you would reference the file by transfer to translate it, but would use the program name mover to execute the program. If both the program and the file were named mover, you could use mover to translate the file and also to execute the program.

A task is created to execute the program and the task name is the name of the program you initiate. The program can call a routine in another program, but the task name does not change.

The identifier used to name the program cannot be used in the program for any other purpose, such as to identify a variable or constant.

The CONST (constant), TYPE (type), and VAR (variable) declaration sections come after the PROGRAM statement. A program can contain any number of CONST, TYPE, and VAR sections. Each section can also contain any number of individual declaration statements. Also, multiple CONST, TYPE, and VAR sections can appear in any order. The number of CONST, TYPE, and VAR sections, and declaration statements are limited only by the amount of memory available.

ROUTINE declarations can follow the CONST, TYPE, and VAR sections. Each routine begins with the reserved word ROUTINE and is similar in syntax to a program. ROUTINE declarations can also follow the executable section of the main program after the END statement.

The executable section must be marked by BEGIN at the beginning and END, followed by the program identifier (prog_name in Structure of a KAREL Program), at the end. The same program identifier must be used in the END statement as in the PROGRAM statement. The executable section can contain any number of executable statements, limited only by the amount of memory available.

See Also: Chapter 2 LANGUAGE ELEMENTS, Chapter 3 USE OF OPERATORS, and Chapter 5 ROUTINES.
1.3 SYSTEM SOFTWARE

The R-30iA system includes a robot and controller electronics. Hardware interfaces and system software support programming, daily operation, maintenance, and troubleshooting.

This section provides an overview of the supported system software and robot models.

Hardware topics are covered in greater detail in the Maintenance Manual specific for your robot and controller model.

1.3.1 Software Components

R-30iA system software is the FANUC Robotics-supplied software that is executed by the controller CPU, which allows you to operate the R-30iA system. You use the system software to run programs, as well as to perform daily operations, maintenance, and troubleshooting.

The components of the system software include:

- **Motion Control** - movement of the tool center point (TCP) from an initial position to a desired destination position
- **File System** - storage of data on the RAM disk or peripheral storage devices
- **System Variables** - permanently defined variables declared as part of the KAREL system software
- **CRT/KB or Teach Pendant Screens** - screens that facilitate operation of the KAREL system
- **KCL** - KAREL Command Language
- **KAREL Interpreter** - executes KAREL programs

See Also: application-specific FANUC Robotics Setup and Operations Manual for detailed operation procedures using the CRT/KB and teach pendant screens.

1.3.2 Supported Robots

The robot, using the appropriate tooling, performs application tasks directed by the system software and controller. The R-30iA system supports a variety of robots, each designed for a specific type of application.

For a current list of supported robot models, consult your FANUC Robotics technical representative.

See Also: The Maintenance Manual for your specific robot type, for more information on your robot.
1.4 CONTROLLER

The R-30iA controller contains the electronic circuitry and memory required to operate the R-30iA system. The electronic circuitry, supported by the system software, directs the operation and motion of the robot and allows communication with peripheral devices.

Controller electronics includes a central processing unit (CPU), several types of memory, an input/output (I/O) system, and user interface devices. A cabinet houses the controller electronics and the ports to which remote user interface devices and other peripheral devices are connected.

1.4.1 Memory

There are three kinds of controller memory:

- Dynamic Random Access Memory (DRAM)
- A limited amount of battery-backed static/random access memory (SRAM)
- Flash Programmable Read Only Memory (FROM)

In addition, the controller is capable of storing information externally.

DRAM

DRAM memory is volatile. Memory contents do not retain their stored values when power is removed. DRAM memory is also referred to as temporary memory (TEMP). The system software is executed in DRAM memory. KAREL programs and most KAREL variables are loaded into DRAM and executed from here also.

Note Even though DRAM variables are in volatile memory, you can control their value at startup. Any time that a the program .VR or .PC file is loaded, the values in DRAM for that program are set to the value in the .VR file. This means that there is not a requirement to re-load the VR file itself at every startup to set initial values. If the value of that variable changes during normal operation it will revert to the value it was set to the last time the .VR or .PC file was loaded.

If you want the DRAM variables to be uninitialized at start up you can use the IN UNINIT_DRAM clause on any variable you want to insure is uninitialized at startup. You can use the %UNINITDRAM directive to specify that all the variables in a program are to be uninitialized at startup.

If you have a SHADOW variables and DRAM variables in the same KAREL program, there is a possibility that the power up settings of the DRAM variables could change without loading a .PC/.VR File. In this case the programmer must pay particular attention to the reliance of KAREL software on a particular setting of a DRAM variable at startup. Specifically, the DRAM startup values will always retain the values that they had at the end of controlled start. If SHADOW memory is full, the DRAM startup values could be set during normal system operation.
SRAM

SRAM memory is nonvolatile. Memory contents retain their stored values when power is removed. SRAM memory is also referred to as CMOS or as permanent memory (PERM).

The TPP memory pool (used for teach pendant programs) is allocated from PERM. KAREL programs can designate variables to be stored in CMOS. A portion of SRAM memory can be defined as a user storage device called RAM Disk (RD:).

Flash memory (FROM)

FROM memory is nonvolatile. Memory contents retain their stored values when power is removed. FROM is used for permanent storage of the system software. FROM is also available for user storage as the FROM device (FR:).

SHADOW

Shadow memory provides the same capabilities as SRAM. Any values set in shadow are non-volatile and will maintain their state through power cycle. Shadow memory is intended for data which tends to be static. Storing dynamic variables in shadow memory, such as FOR loop indexes or other rapidly changing data, is not efficient.

Figure 1–1. Controller Memory
External Storage

You can back up and store files on external devices. You can use the following devices:

- Memory card
- Ethernet via FTP
- USB Memory Stick

1.4.2 Input/Output System

The controller can support a modular I/O structure, allowing you to add I/O boards as required by your application. Both digital and analog input and output modules are supported. In addition, you can add optional process I/O boards for additional I/O. The type and number of I/O signals you have depends on the requirements of your application.

See Also: Chapter 14 INPUT/OUTPUT SYSTEM, for more information

1.4.3 User Interface Devices

The user interface devices enable you to program and operate the KAREL system. The common user interface devices supported by KAREL include the operator panel, the teach pendant or the CRT/KB.

Figure 1–2 illustrates these user interface devices. The operator panel and teach pendant have the same basic functions for all models; however, different configurations are also available.

The operator panel, located on the front of the controller cabinet, provides buttons for performing daily operations such as powering up, running a program, and powering down. Lights on the operator panel indicate operating conditions such as when the power is on and when the robot is in cycle.

The system also supports I/O signals for a user operator panel (UOP), which is a user-supplied device such as a custom control panel, a programmable controller, or a host computer. Refer to Chapter 14 INPUT/OUTPUT SYSTEM.
The CRT/KB is a software option on the controller that allows an external terminal such as a PC running TelNet to display a Menu System that looks similar to the one seen on the teach pendant.

The teach pendant consists of an LCD display, menu-driven function keys, keypad keys, and status LEDs. It is connected to the controller cabinet via a cable, allowing you to perform operations away from the controller.

Internally, the teach pendant connects to the controller’s Main CPU board. It is used to jog the robot, teach program data, test and debug programs, and adjust variables. It can also be used to monitor and control I/O, to control end-of-arm tooling, and to display information such as the current position of the robot or the status of an application program.

The application-specific FANUC Robotics Setup and Operations Manual provides descriptions of each of the user interface devices, as well as procedures for operating each device.
## Chapter 2

### LANGUAGE ELEMENTS

#### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 2 LANGUAGE ELEMENTS</td>
<td>2–1</td>
</tr>
<tr>
<td>2.1 LANGUAGE COMPONENTS</td>
<td>2–2</td>
</tr>
<tr>
<td>2.1.1 Character Set</td>
<td>2–2</td>
</tr>
<tr>
<td>2.1.2 Operators</td>
<td>2–5</td>
</tr>
<tr>
<td>2.1.3 Reserved Words</td>
<td>2–6</td>
</tr>
<tr>
<td>2.1.4 User-Defined Identifiers</td>
<td>2–7</td>
</tr>
<tr>
<td>2.1.5 Labels</td>
<td>2–8</td>
</tr>
<tr>
<td>2.1.6 Predefined Identifiers</td>
<td>2–8</td>
</tr>
<tr>
<td>2.1.7 System Variables</td>
<td>2–10</td>
</tr>
<tr>
<td>2.1.8 Comments</td>
<td>2–10</td>
</tr>
<tr>
<td>2.2 TRANSLATOR DIRECTIVES</td>
<td>2–11</td>
</tr>
<tr>
<td>2.3 DATA TYPES</td>
<td>2–13</td>
</tr>
<tr>
<td>2.4 USER-DEFINED DATA TYPES AND STRUCTURES</td>
<td>2–14</td>
</tr>
<tr>
<td>2.4.1 User-Defined Data Types</td>
<td>2–14</td>
</tr>
<tr>
<td>2.4.2 User-Defined Data Structures</td>
<td>2–16</td>
</tr>
<tr>
<td>2.5 ARRAYS</td>
<td>2–18</td>
</tr>
<tr>
<td>2.5.1 Multi-Dimensional Arrays</td>
<td>2–19</td>
</tr>
<tr>
<td>2.5.2 Variable-Sized Arrays</td>
<td>2–20</td>
</tr>
</tbody>
</table>
The KAREL language provides the elements necessary for programming effective robotics applications. This chapter lists and describes each of the components of the KAREL language, the available translator directives and the available data types.

## 2.1 LANGUAGE COMPONENTS

This section describes the following basic components of the KAREL language:

- Character set
- Operators
- Reserved words
- User-defined Identifiers
- Labels
- Predefined Identifiers
- System Variables
- Comments

### 2.1.1 Character Set

The ASCII character set is available in the KAREL language. Table 2–1 lists the elements in the ASCII character set. Three character sets are available in the KAREL language:

- ASCII Character Set
- Multinational Character Set
- Graphics Character Set (not available in R-30iB)

All of the characters recognized by the KAREL language are listed in Table 2–1, Table 2–2, and Table 2–3. The default character set is ASCII. The multinational and graphics character sets are permitted only in literals, data, and comments.

**See Also:** CHR Built-In Procedure, Appendix A.

<table>
<thead>
<tr>
<th>Table 2–1. ASCII Character Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Digits</td>
</tr>
</tbody>
</table>
The following rules are applicable for the ASCII character set:

- Blanks or spaces are:
  - Required to separate reserved words and identifiers. For example, the statement `PROGRAM prog_name` must include a blank between `PROGRAM` and `prog_name`.
  - Allowed but are not required within expressions between symbolic operators and their operands. For example, the statement `a = b` is equivalent to `a=b`.
  - Used to indent lines in a program.

- Carriage return or a semi-colon (;) separate statements. Carriage returns can also appear in other places.

- A carriage return or a semi-colon is required after the BEGIN statement.

- A line is truncated after 252 characters. It can be continued on the next line by using the concatenation character &.
## Table 2–2. Multinational Character Set

<table>
<thead>
<tr>
<th>Symbols</th>
<th>₤</th>
<th>€</th>
<th>£</th>
<th>¥</th>
<th>§</th>
<th>№</th>
</tr>
</thead>
<tbody>
<tr>
<td>©</td>
<td>a</td>
<td>«</td>
<td>О</td>
<td>±</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>²</td>
<td>µ</td>
<td>¶</td>
<td>•</td>
<td>1</td>
<td>₀</td>
<td></td>
</tr>
<tr>
<td>»</td>
<td>¼</td>
<td>½</td>
<td>¿</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special Characters</th>
<th>Á</th>
<th>Â</th>
<th>Æ</th>
<th>ą</th>
<th>ã</th>
<th>Æ</th>
<th>Æ</th>
<th>Æ</th>
<th>Æ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ä</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
</tr>
<tr>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
<th>ɪ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
</tr>
</tbody>
</table>
### Graphics Character Set

<table>
<thead>
<tr>
<th>Letters</th>
<th>ABCDEFGHIJKLMNOPQRSTUVWXYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>0123456789</td>
</tr>
<tr>
<td>Symbols</td>
<td>@ &lt; &gt; = / * + - , . # $ ` ] ( ) &amp; % ! ^ ♦ ♦ ♦ ♦</td>
</tr>
</tbody>
</table>

#### Operators

KAREL provides operators for standard arithmetic operations, relational operations, and Boolean (logical) operations. KAREL also includes special operators that can be used with positional and VECTOR data types as operands.

Table 2–4 lists all of the operators available for use with KAREL.

#### Table 2–4. KAREL Operators

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>/</th>
<th>DIV</th>
<th>MOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>&lt;</td>
<td>&lt;=</td>
<td>=</td>
<td>&lt; &gt;</td>
<td>&gt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND</td>
<td>OR</td>
<td>NOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special</td>
<td>&gt;=</td>
<td>&lt;</td>
<td>:</td>
<td>#</td>
<td>@</td>
<td></td>
</tr>
</tbody>
</table>

The precedence rules for these operators are as follows:

- Expressions within parentheses are evaluated first.
- Within a given level of parentheses, operations are performed starting with those of highest precedence and proceeding to those of lowest precedence.
- Within the same level of parentheses and operator precedence, operations are performed from left to right.

Table 2–5 lists the precedence levels for the KAREL operators.

---

See Also: Appendix D for a listing of the character codes for each character set
2. LANGUAGE ELEMENTS

Table 2–5. KAREL Operator Precedence

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>PRECEDENCE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>High</td>
</tr>
<tr>
<td>;, @, #</td>
<td>↓</td>
</tr>
<tr>
<td>*, /, AND, DIV, MOD</td>
<td>↓</td>
</tr>
<tr>
<td>Unary + and -, OR, +, -</td>
<td>↓</td>
</tr>
<tr>
<td>&lt;, &gt;, =, &lt; &gt;, &lt; =, &gt; =, &gt; = &lt;</td>
<td>Low</td>
</tr>
</tbody>
</table>

See Also: Chapter 3 USE OF OPERATORS, for descriptions of functions operators perform

2.1.3 Reserved Words

Reserved words have a dedicated meaning in KAREL. They can be used only in their prescribed contexts. All KAREL reserved words are listed in Table 2–6.

Table 2–6. Reserved Word List

<table>
<thead>
<tr>
<th>ABORT</th>
<th>CONST</th>
<th>GET_VAR</th>
<th>NOPAUSE</th>
<th>STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOUT</td>
<td>CONTINUE</td>
<td>GO</td>
<td>NOT</td>
<td>STRING</td>
</tr>
<tr>
<td>ABS</td>
<td>COORDINATED</td>
<td>GOTO</td>
<td>NOWAIT</td>
<td>STRUCTURE</td>
</tr>
<tr>
<td>AFTER</td>
<td>CR</td>
<td>GROUP</td>
<td>OF</td>
<td>THEN</td>
</tr>
<tr>
<td>ALONG</td>
<td>DELAY</td>
<td>GROUP_ASSOC</td>
<td>OPEN</td>
<td>TIME</td>
</tr>
<tr>
<td>ALSO</td>
<td>DISABLE</td>
<td>HAND</td>
<td>OR</td>
<td>TIMER</td>
</tr>
<tr>
<td>AND</td>
<td>DISCONNECT</td>
<td>HOLD</td>
<td>PATH</td>
<td>TO</td>
</tr>
<tr>
<td>ARRAY</td>
<td>DIV</td>
<td>IF</td>
<td>PATHHEADER</td>
<td>TPENABLE</td>
</tr>
<tr>
<td>ARRAY_LEN</td>
<td>DO</td>
<td>IN</td>
<td>PAUSE</td>
<td>TYPE</td>
</tr>
<tr>
<td>AT</td>
<td>DOWNTO</td>
<td>INDEPENDENT</td>
<td>POSITION</td>
<td>UNHOLD</td>
</tr>
<tr>
<td>ATTACH</td>
<td>DRAM</td>
<td>INTEGER</td>
<td>POWERUP</td>
<td>UNINIT</td>
</tr>
<tr>
<td>AWAY</td>
<td>ELSE</td>
<td>JOINTPOS</td>
<td>PROGRAM</td>
<td>UNPAUSE</td>
</tr>
<tr>
<td>AXIS</td>
<td>ENABLE</td>
<td>JOINTPOS1</td>
<td>PULSE</td>
<td>UNTIL</td>
</tr>
<tr>
<td>BEFORE</td>
<td>END</td>
<td>JOINTPOS2</td>
<td>PURGE</td>
<td>USING</td>
</tr>
<tr>
<td>BEGIN</td>
<td>ENDCONDITION</td>
<td>JOINTPOS3</td>
<td>READ</td>
<td>VAR</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>ENDFOR</td>
<td>JOINTPOS4</td>
<td>REAL</td>
<td>VECTOR</td>
</tr>
<tr>
<td>BY</td>
<td>ENDFIF</td>
<td>JOINTPOS5</td>
<td>RELATIVE</td>
<td>VIA</td>
</tr>
</tbody>
</table>
Table 2–6. Reserved Word List (Cont’d)

<table>
<thead>
<tr>
<th>WORD</th>
<th>WORD</th>
<th>WORD</th>
<th>WORD</th>
<th>WORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYNAME</td>
<td>ENDMOVE</td>
<td>JOINTPOS6</td>
<td>RELAX</td>
<td>VIS_PROCESS</td>
</tr>
<tr>
<td>BYTE</td>
<td>ENDSELECT</td>
<td>JOINTPOS7</td>
<td>RELEASE</td>
<td>WAIT</td>
</tr>
<tr>
<td>CAM_SETUP</td>
<td>ENDSTRUCTURE</td>
<td>JOINTPOS8</td>
<td>REPEAT</td>
<td>WHEN</td>
</tr>
<tr>
<td>CANCEL</td>
<td>ENDUSING</td>
<td>JOINTPOS9</td>
<td>RESTORE</td>
<td>WHILE</td>
</tr>
<tr>
<td>CASE</td>
<td>ENDMETHOD</td>
<td>MOD</td>
<td>RESUME</td>
<td>WITH</td>
</tr>
<tr>
<td>CLOSE</td>
<td>ERROR</td>
<td>MODEL</td>
<td>RETURN</td>
<td>WRITE</td>
</tr>
<tr>
<td>CMOS</td>
<td>EVAL</td>
<td>MOVE</td>
<td>ROUTINE</td>
<td>XYZWPR</td>
</tr>
<tr>
<td>COMMAND</td>
<td>EVENT</td>
<td>NEAR</td>
<td>SELECT</td>
<td>XYZWPREXT</td>
</tr>
<tr>
<td>COMMON_ASSOC</td>
<td>END</td>
<td>NOABORT</td>
<td>SEMAPHORE</td>
<td></td>
</tr>
<tr>
<td>CONDITION</td>
<td>FILE</td>
<td>NODE</td>
<td>SET_VAR</td>
<td></td>
</tr>
<tr>
<td>CONFIG</td>
<td>FOR</td>
<td>NODEDATA</td>
<td>SHORT</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>FROM</td>
<td>NOMESSAGE</td>
<td>SIGNAL</td>
<td></td>
</tr>
</tbody>
</table>

See Also: Index for references to descriptions of KAREL reserved words

2.1.4 User-Defined Identifiers

User-defined identifiers represent constants, data types, statement labels, variables, routine names, and program names. Identifiers

- Start with a letter
- Can include letters, digits, and underscores
- Can have a maximum of 12 characters
- Can have only one meaning within a particular scope. Refer to Section 5.1.4.
- Cannot be reserved words
- Must be defined before they can be used.

For example, the program excerpt in Declaring Identifiers shows how to declare program, variable, and constant identifiers.

Declaring Identifiers

```karel
PROGRAM mover --program identifier (mover)
    VAR
        original : POSITION --variable identifier (original)
    CONST
```
no_of_parts = 10  --constant identifier (no_of_parts)

2.1.5 Labels

Labels are special identifiers that mark places in the program to which program control can be transferred using the GOTO Statement.

- Are immediately followed by two colons (::). Executable statements are permitted on the same line and subsequent lines following the two colons.
- Cannot be used to transfer control into or out of a routine.

In Using Labels, weld: : denotes the section of the program in which a part is welded. When the statement go to weld is executed, program control is transferred to the weld section.

Using Labels

weld: :  --label
    .  --additional program statements
    .
    GOTO weld

2.1.6 Predefined Identifiers

Predefined identifiers within the KAREL language have a predefined meaning. These can be constants, types, variables, or built-in routine names. Table 2–7 and Table 2–8 list the predefined identifiers along with their corresponding values. Either the identifier or the value can be specified in the program statement. For example, $MOTYPE = 7 is the same as $MOTYPE = LINEAR. However, the predefined identifier MININT is an exception to this rule. This identifier must always be used in place of its value, -2147483648. The value or number itself can not be used.

<table>
<thead>
<tr>
<th>Predefined Identifier</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>BOOLEAN</td>
<td>ON</td>
</tr>
<tr>
<td>FALSE</td>
<td>BOOLEAN</td>
<td>OFF</td>
</tr>
<tr>
<td>ON</td>
<td>BOOLEAN</td>
<td>ON</td>
</tr>
<tr>
<td>OFF</td>
<td>BOOLEAN</td>
<td>OFF</td>
</tr>
<tr>
<td>MAXINT</td>
<td>INTEGER</td>
<td>+2147483647</td>
</tr>
<tr>
<td>MININT</td>
<td>INTEGER</td>
<td>-2147483648</td>
</tr>
</tbody>
</table>
Table 2–7. Predefined Identifier and Value Summary (Cont’d)

<table>
<thead>
<tr>
<th>Predefined Identifier</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSWORLD</td>
<td>Orientation Type: $ORIENT_TYPE</td>
<td>1</td>
</tr>
<tr>
<td>AESWORLD</td>
<td>Orientation Type: $ORIENT_TYPE</td>
<td>2</td>
</tr>
<tr>
<td>WRISTJOINT</td>
<td>Orientation Type: $ORIENT_TYPE</td>
<td>3</td>
</tr>
<tr>
<td>JOINT</td>
<td>Motion Type: $MOTYPE</td>
<td>6</td>
</tr>
<tr>
<td>LINEAR (or STRAIGHT)</td>
<td>Motion Type: $MOTYPE</td>
<td>7</td>
</tr>
<tr>
<td>CIRCULAR</td>
<td>Motion Type: $MOTYPE</td>
<td>8</td>
</tr>
<tr>
<td>FINE</td>
<td>Termination Types: $TERMTYPE and $SEGTERMTYPE</td>
<td>1</td>
</tr>
<tr>
<td>COARSE</td>
<td>Termination Types: $TERMTYPE and $SEGTERMTYPE</td>
<td>2</td>
</tr>
<tr>
<td>NOSETTLE</td>
<td>Termination Types: $TERMTYPE and $SEGTERMTYPE</td>
<td>3</td>
</tr>
<tr>
<td>NODECEL</td>
<td>Termination Types: $TERMTYPE and $SEGTERMTYPE</td>
<td>4</td>
</tr>
<tr>
<td>VARDECEL</td>
<td>Termination Types: $TERMTYPE and $SEGTERMTYPE</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2–8. Port and File Predefined Identifier Summary

<table>
<thead>
<tr>
<th>Predefined Identifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIN (Digital input)</td>
<td>Boolean port array</td>
</tr>
<tr>
<td>DOUT (Digital output)</td>
<td></td>
</tr>
<tr>
<td>GIN (Group input)</td>
<td>Integer port array</td>
</tr>
<tr>
<td>GOUT (Group output)</td>
<td></td>
</tr>
<tr>
<td>AIN (Analog input)</td>
<td></td>
</tr>
<tr>
<td>AOUT (Analog output)</td>
<td></td>
</tr>
<tr>
<td>TPIN (Teach pendant input)</td>
<td>Boolean port array</td>
</tr>
<tr>
<td>TPOUT (Teach pendant output)</td>
<td></td>
</tr>
<tr>
<td>RDI (Robot digital input)</td>
<td></td>
</tr>
<tr>
<td>RDO (Robot digital output)</td>
<td></td>
</tr>
<tr>
<td>OPIN (Operator panel input)</td>
<td></td>
</tr>
<tr>
<td>OPOUT (Operator panel output)</td>
<td></td>
</tr>
<tr>
<td>WDI (Weld input)</td>
<td></td>
</tr>
<tr>
<td>WDOUT (Weld output)</td>
<td></td>
</tr>
<tr>
<td>UIN (User operator panel input)</td>
<td></td>
</tr>
<tr>
<td>UOUT (User operator panel output)</td>
<td></td>
</tr>
<tr>
<td>LDI (Laser digital input)</td>
<td></td>
</tr>
<tr>
<td>LDO (Laser digital output)</td>
<td></td>
</tr>
<tr>
<td>FLG (Flag)</td>
<td></td>
</tr>
<tr>
<td>MRK (Marker)</td>
<td></td>
</tr>
</tbody>
</table>
Table 2–8. Port and File Predefined Identifier Summary (Cont’d)

<table>
<thead>
<tr>
<th>Predefined Identifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAI (Laser analog input)</td>
<td>Integer port array</td>
</tr>
<tr>
<td>LAO (Laser analog output)</td>
<td></td>
</tr>
<tr>
<td>TPDISPLAY (Teach pendant KAREL display)*</td>
<td>File</td>
</tr>
<tr>
<td>TERROR (Teach pendant message line)</td>
<td></td>
</tr>
<tr>
<td>TPPROMPT (Teach pendant function key line)*</td>
<td></td>
</tr>
<tr>
<td>TPFUNC (Teach pendant function key line)*</td>
<td></td>
</tr>
<tr>
<td>TPSTATUS (Teach pendant status line)*</td>
<td></td>
</tr>
<tr>
<td>INPUT (CRT/KB KAREL keyboard)*</td>
<td></td>
</tr>
<tr>
<td>OUTPUT (CRT/KB KAREL screen)*</td>
<td></td>
</tr>
<tr>
<td>CRERROR (CRT/KB message line)</td>
<td></td>
</tr>
<tr>
<td>CRFUNC (CRT function key line)*</td>
<td></td>
</tr>
<tr>
<td>CRTSTATUS (CRT status line)*</td>
<td></td>
</tr>
<tr>
<td>CRTPROMPT (CRT prompt line)*</td>
<td></td>
</tr>
<tr>
<td>VIS_MONITOR (Vision Monitor Screen)</td>
<td></td>
</tr>
</tbody>
</table>

*Input and output occurs on the USER menu of the teach pendant or CRT/KB.

2.1.7 System Variables

System variables are variables that are declared as part of the KAREL system software. They have permanently defined variable names, that begin with a dollar sign ($). Many are robot specific, meaning their values depend on the type of robot that is attached to the system.

Some system variables are not accessible to KAREL programs. Access rights govern whether or not a KAREL program can read from or write to system variables.

See Also: *FANUC Robotics Software Reference Manual* for a complete list and description of all available system variables.

2.1.8 Comments

Comments are lines of text within a program used to make the program easier for you or another programmer to understand. For example, Comments From Within a Program contains some comments from %INCLUDE Directive in a KAREL Program and Include File mover_decs for a KAREL Program.

Comments From Within a Program

--This program, called mover, picks up 10 objects
-- from an original POSITION and puts them down
-- at a destination POSITION.
original : POSITION -- POSITION of objects
destination : POSITION -- Destination of objects
count : INTEGER -- Number of objects moved

A comment is marked by a pair of consecutive hyphens (- -). On a program line, anything to the right of these hyphens is treated as a comment.

Comments can be inserted on lines by themselves or at the ends of lines containing any program statement. They are ignored by the translator and have absolutely no effect on a running program.

### 2.2 TRANSLATOR DIRECTIVES

Translator directives provide a mechanism for directing the translation of a KAREL program. Translator directives are special statements used within a KAREL program to

- Include other files into a program at translation time
- Specify program and task attributes

All directives except %INCLUDE must be after the program statement but before any other statements. Table 2–9 lists and briefly describes each translator directive. Refer to Appendix A for a complete description of each translator directive.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ALPHABETIZE</td>
<td>Specifies that variables will be created in alphabetical order when p-code is loaded.</td>
</tr>
<tr>
<td>%CMOSVARS</td>
<td>Specifies the default storage for KAREL variables is CMOS RAM.</td>
</tr>
<tr>
<td>%CMOS2SHADOW</td>
<td>Instructs the translator to put all CMOS variables in SHADOW memory.</td>
</tr>
<tr>
<td>%COMMENT = 'comment'</td>
<td>Specifies a comment of up to 16 characters. During load time, the comment is stored as a program attribute and can be displayed on the SELECT screen of the teach pendant or CRT/KB.</td>
</tr>
<tr>
<td>%CRTDEVICE</td>
<td>Specifies that the CRT/KB user window will be the default in the READ and WRITE statements instead of the TPDISPLAY window.</td>
</tr>
<tr>
<td>%DEFGROUP = n</td>
<td>Specifies the default motion group to be used by the translator.</td>
</tr>
<tr>
<td>%DELAY</td>
<td>Specifies the amount of time the program will be delayed out of every 250 milliseconds.</td>
</tr>
</tbody>
</table>
### Table 2-9. Translator Directives (Cont’d)

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ENVIRONMENT filename</td>
<td>Used by the off-line translator to specify that a particular environment file should be loaded.</td>
</tr>
<tr>
<td>%INCLUDE filename</td>
<td>Specifies files to insert into a program at translation time.</td>
</tr>
<tr>
<td>%LOCKGROUP = n,n</td>
<td>Specifies the motion group(s) locked by this task.</td>
</tr>
<tr>
<td>%NOABORT = option</td>
<td>Specifies a set of conditions which will be prevented from aborting the program.</td>
</tr>
<tr>
<td>%NOBUSYLAMP</td>
<td>Specifies that the busy lamp will be OFF during execution.</td>
</tr>
<tr>
<td>%NOLOCKGROUP</td>
<td>Specifies that no motion groups will be locked by this task.</td>
</tr>
<tr>
<td>%NOPAUSE = option</td>
<td>Specifies a set of conditions which will be prevented from pausing the program.</td>
</tr>
<tr>
<td>%NOPAUSESHFT</td>
<td>Specifies that the task is not paused if the teach pendant shift key is released.</td>
</tr>
<tr>
<td>%PRIORITY = n</td>
<td>Specifies the task priority.</td>
</tr>
<tr>
<td>%SHADOWVARS</td>
<td>Specifies that all variables by default are created in SHADOW.</td>
</tr>
<tr>
<td>%STACKSIZE = n</td>
<td>Specifies the stack size in long words.</td>
</tr>
<tr>
<td>%TIMESLICE = n</td>
<td>Supports round-robin type time slicing for tasks with the same priority.</td>
</tr>
<tr>
<td>%TPMOTION</td>
<td>Specifies that task motion is enabled only when the teach pendant is enabled.</td>
</tr>
<tr>
<td>%UNINITVARS</td>
<td>Specifies that all variables are by default uninitialized.</td>
</tr>
</tbody>
</table>

%INCLUDE Directive in a KAREL Program illustrates the %INCLUDE directive. Include File mover_decs for a KAREL Program shows the included file.

### %INCLUDE Directive in a KAREL Program

```karel
PROGRAM mover
    -- This program, called mover, picks up 10 objects
    -- from an original position and puts them down
    -- at a destination position.
    %INCLUDE mover_decs
    -- Uses %INCLUDE directive to include the file
    -- called mover_decs containing declarations
BEGIN
    OPEN HAND gripper
    -- Loop to move total number of objects
    FOR count = 1 TO num_of_parts DO
        -- Put position in Position Register 1
        SET_POS_REG(1,original,status)
        -- Call TP program to move to Position Register
        move_to_pr
    CLOSE HAND gripper
```

2–12
include File mover_decs for a KAREL Program

-- Declarations for program mover in file mover_decs
VAR
original : XYZWPR -- POSITION of objects
destination : XYZWPR -- Destination of objects
count : INTEGER -- Number of objects moved
CONST
gripper = 1 -- Hand number 1
num_of_parts = 10 -- Number of objects to move

2.3 DATA TYPES

Three forms of data types are provided by KAREL to define data items in a program:

• Simple type data items
  — Can be assigned constants or variables in a KAREL program
  — Can be assigned actual (literal) values in a KAREL program
  — Can assume only single values

• Structured type data items
  — Are defined as data items that permit or require more than a single value
  — Are composites of simple data and structured data

• User-defined type data items
  — Are defined in terms of existing data types including other user-defined types
  — Can be defined as structures consisting of several KAREL variable data types
  — Cannot include itself

Table 2–10 lists the simple and structured data types available in KAREL. User-defined data types are described in Section 2.4.
2. LANGUAGE ELEMENTS

2.4 USER-DEFINED DATA TYPES AND STRUCTURES

User-defined data types are data types you define in terms of existing data types. User-defined data structures are data structures in which you define a new data type as a structure consisting of several KAREL variable data types, including previously defined user data types.

2.4.1 User-Defined Data Types

User-defined data types are data types you define in terms of existing data types. With user-defined data types, you

- Include their declarations in the TYPE sections of a KAREL program.
- Define a KAREL name to represent a new data type, described in terms of other data types.
- Can use predefined data types required for specific applications.

User-defined data types can be defined as structures, consisting of several KAREL variable data types.

The continuation character, "&", can be used to continue a declaration on a new line.

User-Defined Data Type Example shows an example of user-defined data type usage and continuation character usage.
User-Defined Data Type Example

```c
CONST
    n_pages = 20
    n_lines = 40
    std_str_lng = 8

TYPE
    std_string_t = STRING [std_str_lng]
    std_table_t = ARRAY [n_pages] & --continuation character
                   OF ARRAY [n_lines] OF std_string_t
    path_hdr_t FROM main_prog = STRUCTURE --user defined data type
        ph_uframe: POSITION
        ph_utool: POSITION
    ENDSTRUCTURE
    node_data_t FROM main_prog = STRUCTURE
        gun_on: BOOLEAN
        air_flow: INTEGER
    ENDSTRUCTURE
    std_path_t FROM main_prog =
        PATH PATHDATA = path_hdr_t NODEDATA = node_data_t

VAR
    msg_table_1: std_table_t
    msg_table_2: std_table_t
    temp_string: std_string_t
    seam_1_path: std_path_t
```

**Usage**

User-defined type data can be

- Assigned to other variables of the same type
- Passed as a parameter
- Returned as a function

Assignment between variables of different user-defined data types, even if identically declared, is not permitted. In addition, the system provides the ability to load and save variables of user-defined data types, checking consistency during the load with the current declaration of the data type.

**Restrictions**

A user-defined data type cannot

- Include itself
- Include any type that includes it, either directly or indirectly
- Be declared within a routine
2.4.2 User-Defined Data Structures

A structure is used to store a collection of information that is generally used together. User-defined data structures are data structures in which you define a new data type as a structure consisting of several KAREL variable data types.

When a program containing variables of user-defined data types is loaded, the definitions of these types are checked against a previously created definition. If a previously created definition does not exist, a new one is created.

With user-defined data structures, you

- Define a data type as a structure consisting of a list of component fields, each of which can be a standard data type or another, previously defined, user data type. See Defining a Data Type as a User-Defined Structure.

**Defining a Data Type as a User-Defined Structure**

```
new_type_name = STRUCTURE
    field_name_1: type_name_1
    field_name_2: type_name_2
.. 
ENDSTRUCTURE
```

- Access elements of a data type defined as a structure in a KAREL program. The continuation character, "&", can be used to continue access of the structure elements. See Accessing Elements of a User-Defined Structure in a KAREL Program.

**Accessing Elements of a User-Defined Structure in a KAREL Program**

```
var_name = new_type_name.field_nam_1
new_type_name.field_name_2 = expression
outer_struct_name.inner_struct_name&
    .field_name = expression
```

- Access elements of a data type defined as a structure from the CRT/KB and at the teach pendant.
- Define a range of executable statements in which fields of a STRUCTURE type variable can be accessed without repeating the name of the variable. See Defining a Range of Executable Statements.
Defining a Range of Executable Statements

```
USING struct_var, struct_var2 DO
  statements
. .
ENDUSING
```

In the above example, struct_var and struct_var2 are the names of structure type variables.

**Note** If the same name is both a field name and a variable name, the field name is assumed. If the same field name appears in more than one variable, the right-most variable in the USING statement is used.

**Restrictions**

User-defined data structures have the following restrictions:

- The following data types are **not valid** as part of a data structure:
  - STRUCTURE definitions; types that are declared structures are permitted. See **Valid STRUCTURE Definitions**.

**Valid STRUCTURE Definitions**

The following is valid:

```
TYPE
  sub_struct = STRUCTURE
    subs_field_1: INTEGER
    subs_field_2: BOOLEAN
  ENDSTRUCTURE
  big_struct = STRUCTURE
    bigs_field_1: INTEGER
    bigs_field_2: sub_struct
  ENDSTRUCTURE
```

The following is not valid:

```
big_struct = STRUCTURE
  bigs_field_1: INTEGER
  bigs_field_2: STRUCTURE
    subs_field_1: INTEGER
    subs_field_2: BOOLEAN
  ENDSTRUCTURE
ENDSTRUCTURE
```

- **PATH types**
— FILE types
— VISION types
— Variable length arrays
— The data structure itself, or any type that includes it, either directly or indirectly
— Any structure not previously defined.

• A variable can not be defined as a structure, but can be defined as a data type previously defined as a structure. See Defining a Variable as a Type Previously Defined as a Structure.

Defining a Variable as a Type Previously Defined as a Structure

The following is valid:

```plaintext
TYPE
  struct_t = STRUCTURE
  st_1: BOOLEAN
  st_2: REAL
ENDSTRUCTURE
VAR
  var_name: struct_t
```

The following is not valid:

```plaintext
VAR
  var_name: STRUCTURE
  vn_1: BOOLEAN
  vn_2: REAL
ENDSTRUCTURE
```

2.5 ARRAYS

You can declare arrays of any data type except PATH.

You can access elements of these arrays in a KAREL program, from the CRT/KB, and from the teach pendant.

In addition, you can define two types of arrays:

• Multi-dimensional arrays
• Variable-sized arrays
2.5.1 Multi-Dimensional Arrays

Multi-dimensional arrays are arrays of elements with two or three dimensions. These arrays allow you to identify an element using two or three subscripts.

Multi-dimensional arrays allow you to

- Declare variables as arrays with two or three (but not more) dimensions. See Declaring Variables as Arrays with Two or Three Dimensions.

Declaring Variables as Arrays with Two or Three Dimensions

```karel
VAR
    name: ARRAY [size_1] OF ARRAY [size_2] .., OF element_type
OR
VAR
    name: ARRAY [size_1, size_2,...] OF element_type
```

- Access elements of these arrays in KAREL statements. See Accessing Elements of Multi-Dimensional Arrays in KAREL Statements.

Accessing Elements of Multi-Dimensional Arrays in KAREL Statements

```karel
name [subscript_1, subscript_2,...] = value
value = name [subscript_1, subscript_2,...]
```

- Declare routine parameters as multi-dimensional arrays. See Declaring Routine Parameters as Multi-Dimensional Arrays.

Declaring Routine Parameters as Multi-Dimensional Arrays

```
Routine expects 2-dimensional array of INTEGER.
    ROUTINE array_user(array_param:ARRAY [*,*] OF INTEGER)
The following are equivalent:
    ROUTINE rtn_name(array_param: ARRAY[*] OF INTEGER)
    and
    ROUTINE rtn_name(array_param: ARRAY OF INTEGER)
```
• Access elements with KCL commands and the teach pendant.
• Save and load multi-dimensional arrays to and from variable files.

Restrictions

The following restrictions apply to multi-dimensional arrays:

• A subarray can be passed as a parameter or assigned to another array by omitting one or more of the right-most subscripts only if it was defined as a separate type. See Using a Subarray.

Using a Subarray

```plaintext
TYPE
  array_30 = ARRAY[30] OF INTEGER
  array_20_30 = ARRAY[20] OF array_30
VAR
  array_1: array_30
  array_2: array_20_30
ROUTINE array_user(array_data: ARRAY OF INTEGER FROM other-prog
BEGIN
  array_user(array_3[5,3]) -- passes elements array_3[5,3,1] through array_3[5,3,30] to array_user
```

• The element type cannot be any of the following:
  — Array (but it can be a user-defined type that is an array)
  — Path

2.5.2 Variable-Sized Arrays

Variable-sized arrays are arrays whose actual size is not known, and that differ from one use of the program to another. Variable-sized arrays allow you to write KAREL programs without establishing dimensions of the array variables. In all cases, the dimension of the variable must be established before the .PC file is loaded.
Variable-sized arrays allow you to

- Declare an array size as “to-be-determined” (*). See Indicates that the Size of an Array is "To-Be-Determined".

**Indicates that the Size of an Array is "To-Be-Determined"**

```pascal
VAR
  one_d_array: ARRAY[*] OF type
  two_d_array: ARRAY[*,*] OF type
```

- Determine an array size from that in a variable file or from a KCL CREATE VAR command rather than from the KAREL source code.

The actual size of a variable-sized array will be determined by the actual size of the array if it already exists, the size of the array in a variable file if it is loaded first, or the size specified in a KCL CREATE VAR command executed before the program is loaded. Dimensions explicitly specified in a program must agree with those specified from the .VR file or specified in the KCL CREATE VAR command.

**Restrictions**

Variable-sized arrays have the following restrictions:

- The variable must be loaded or created in memory (in a .VR file or using KCL), with a known length, before it can be used.
- When the .PC file is loaded, it uses the established dimension, otherwise it uses 0.
- Variable-sized arrays are only allowed in the VAR section and not the TYPE section of a program.
- Variable-sized arrays are only allowed for static variables.
This chapter describes how operators are used with other language elements to perform operations within a KAREL application program. Expressions and assignments, which are program statements that include operators and operands, are explained first. Next, the kinds of operations that can be performed using each available KAREL operator are discussed.

### 3.1 EXPRESSIONS AND ASSIGNMENTS

Expressions are values defined by a series of operands, connected by operators and cause desired computations to be made. For example, \( 4 + 8 \) is an expression in which 4 and 8 are the *operands* and the plus symbol (+) is the *operator*.

Assignments are statements that set the value of variables to the result of an evaluated expression.

#### 3.1.1 Rule for Expressions and Assignments

The following rules apply to expressions and assignments:

- Each operand of an expression has a data type determined by the nature of the operator.
- Each KAREL operator requires a particular operand type and causes a computation that produces a particular result type.
- Both operands in an expression must be of the same data type. For example, the AND operator requires that both its operands are INTEGER values or that both are BOOLEAN values. The expression \( i \text{ AND } b \), where \( i \) is an INTEGER and \( b \) is a BOOLEAN, is invalid.
- Five special cases in which the operands can be mixed provide an exception to this rule. These five cases include the following:
  - INTEGER and REAL operands to produce a REAL result
  - INTEGER and REAL operands to produce a BOOLEAN result
  - INTEGER and VECTOR operands to produce a VECTOR
  - REAL and VECTOR operands to produce a VECTOR
  - POSITION and VECTOR operands to produce a VECTOR
- Any positional data type can be substituted for the POSITION data type.

#### 3.1.2 Evaluation of Expressions and Assignments

Table 3–1 summarizes the data types of the values that result from the evaluation of expressions containing KAREL operators and operands.
### Table 3–1. Summary of Operation Result Types

| Operator          | + | - | * | / | DIV MOD | <,>,>=,<=,|= | >,=< | AND OR | NOT | # | @ | : |
|-------------------|---|---|---|---|---------|-------------|------|--------|-----|---|---|---|
| **Types of Operators** |   |   |   |   |         |             |      |        |     |   |   |   |
| INTEGER           | I | I | I | R | I       | B           | –    | I      | –   | – | – | – |
| REAL              | R | R | R | R | –       | B           | –    | –      | –   | – | – | – |
| Mixed** INTEGER-REAL | R | R | R | R | –       | B           | –    | –      | –   | – | – | – |
| BOOLEAN           | – | – | – | – | –       | B           | –    | B      | –   | – | – | – |
| STRING            | S | – | – | – | –       | B           | –    | –      | –   | – | – | – |
| Mixed** INTEGER-VECTOR | – | – | V | V | –       | –           | –    | –      | –   | – | – | – |
| Mixed** REAL-VECTOR | – | – | V | V | –       | –           | –    | –      | –   | – | – | – |
| VECTOR            | V | V | – | – | –       | B***        | –    | –      | V   | R | – | – |
| POSITION          | – | – | – | – | –       | B           | –    | –      | –   | P | – | V |
| Mixed** POSITION-VECTOR | – | – | – | – | –       | –           | –    | –      | –   | – | – | V |

**Mixed means one operand of each type**

***VECTOR values can be compared using =<> only***

– Operation not allowed

I INTEGER
R REAL
B BOOLEAN
V VECTOR
P POSITION
### 3.1.3 Variables and Expressions

Assignment statements contain variables and expressions. The variables can be any user-defined variable, a system variable with write access, or an output port array with write access. The expression can be any valid KAREL expression. The following examples are acceptable assignments:

```plaintext
$SPEED = 200.00  -- assigns a REAL value to a system variable

count = count + 1  -- assigns an INTEGER value to an INTEGER variable
```

The data types of `variable` and `expression` must match with three exceptions:

- INTEGER variables can be assigned to REAL variables. In this case, the INTEGER is treated as a REAL number during evaluation of the expression. However, a REAL number cannot be used where an INTEGER value is expected.
- If required, a REAL number can be converted to an INTEGER using the ROUND or TRUNC built-in functions.
- INTEGER, BYTE, and SHORT types can be assigned to each other, although a run-time error will occur if the assigned value is out of range.
- Any positional type can be assigned to any other positional type. A run-time error will result if a JOINTPOS from a group without kinematics is assigned to an XYZWPR.

**See Also:** Relational Operations, ROUND and TRUNC built-in functions, Appendix A, “KAREL Language Alphabetical Description”

### 3.2 OPERATIONS

Operations include the manipulation of variables, constants, and literals to compute values using the available KAREL operators. The following operations are discussed:

- Arithmetic Operations
- Relational Operations
- Boolean Operations
- Special Operations

*Table 3–2* lists all of the operators available for use with KAREL.
3. USE OF OPERATORS

### Table 3–2. KAREL Operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+, -, * / DIV MOD</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt; &lt; = = &lt; &gt; &gt; = &gt;</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND OR NOT</td>
</tr>
<tr>
<td>Special</td>
<td>&gt;= &lt; : # @</td>
</tr>
</tbody>
</table>

#### 3.2.1 Arithmetic Operations

The addition (+), subtraction (-), and multiplication (*) operators, along with the DIV and MOD operators, can be used to compute values within arithmetic expressions. Refer to Table 3–3.

### Table 3–3. Arithmetic Operations Using +, -, and * Operators

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 2</td>
<td>5</td>
</tr>
<tr>
<td>3 - 2</td>
<td>1</td>
</tr>
<tr>
<td>3 * 2</td>
<td>6</td>
</tr>
</tbody>
</table>

- The DIV and MOD operators are used to perform INTEGER division. Refer to Table 3–4.

### Table 3–4. Arithmetic Operations Examples

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 DIV 2</td>
<td>5</td>
</tr>
<tr>
<td>11 MOD 2</td>
<td>1</td>
</tr>
</tbody>
</table>

- The DIV operator truncates the result of an equation if it is not a whole number.
- The MOD operator returns the remainder of an equation that results from dividing the left-side operand by the right-side operand.
- If the right-side operand of a MOD equation is a negative number, the result is also negative.
- If the divisor in a DIV equation or the right-side operand of a MOD equation is zero, the KAREL program is aborted with the “Divide by zero” error.

- The INTEGER bitwise operators, AND, OR, and NOT, produce the result of a binary AND, OR, or NOT operation on two INTEGER values. Refer to Table 3–5.
### Table 3–5. Arithmetic Operations Using Bitwise Operands

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>BINARY EQUIVALENT</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 AND 8</td>
<td>0101 AND 1000</td>
<td>0000 = 0</td>
</tr>
<tr>
<td>5 OR 8</td>
<td>0101 OR 1000</td>
<td>1101 = 13</td>
</tr>
<tr>
<td>-4 AND 8</td>
<td>1100 AND 1000</td>
<td>1000 = 8</td>
</tr>
<tr>
<td>-4 OR 8</td>
<td>1100 OR 1000</td>
<td>1100 = -4</td>
</tr>
<tr>
<td>NOT 5</td>
<td>NOT 0101</td>
<td>1010 = -6*</td>
</tr>
<tr>
<td>NOT -15</td>
<td>NOT 110001</td>
<td>1110 = 14*</td>
</tr>
</tbody>
</table>

*Because negative INTEGER values are represented in the two’s complement form, NOT i is not the same as -i.

- If an INTEGER or REAL equation results in a value exceeding the limit for INTEGER or REAL variables, the program is aborted with an error. If the result is too small to represent, it is set to zero.

Table 3–6 lists the precedence levels for the KAREL operators.

### Table 3–6. KAREL Operator Precedence

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>PRECEDENCE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>High</td>
</tr>
<tr>
<td>;, @, #</td>
<td>↓</td>
</tr>
<tr>
<td>*, /, AND, DIV, MOD</td>
<td>↓</td>
</tr>
<tr>
<td>Unary + and -, OR, +, -</td>
<td>↓</td>
</tr>
<tr>
<td>&lt;, &gt;, =, &lt;&gt;, &lt;=, &gt;=, &gt; = &lt;</td>
<td>Low</td>
</tr>
</tbody>
</table>

#### 3.2.2 Relational Operations

Relational operators (<>, =, >, <=, >=) produce a BOOLEAN (TRUE/FALSE) result corresponding to whether or not the values of the operands are in the relation specified. In a relational expression, both operands must be of the same simple data type. Two exceptions to this rule exist:

- REAL and INTEGER expressions can be mixed where the INTEGER operand is converted to a REAL number.
  
  For example, in the expression 1 > .56, the number 1 is converted to 1.0 and the result is TRUE.
- VECTOR operands, which are a structured data type, can be compared in a relational expression but only by using the equality (=) or inequality (<>) operators.
The relational operators function with INTEGER and REAL operands to evaluate standard mathematical equations. Refer to Table 3–7.

**Note** Performing equality (=) or inequality (<> tests between REAL values might not yield the results you expect. Because of the way REAL values are stored and manipulated, two values that would appear to be equal might not be exactly equal. This is also true of VECTOR values which are composed of REAL values. Use >= or <= where appropriate instead of =.

Relational operators can also have STRING values as operands. STRING values are compared lexically character by character from left to right until one of the following occurs. Refer to Table 3–7.

- The character code for a character in one STRING is greater than the character code for the corresponding character in the other STRING. The result in this case is that the first string is greater. For example, the ASCII code for A is 65 and for a is 97. Therefore, a > A = TRUE.
- One STRING is exhausted while characters remain in the other STRING. The result is that the first STRING is less than the other STRING.
- Both STRING expressions are exhausted without finding a mismatch. The result is that the STRINGs are equal.

### Table 3–7. Relational Operation Examples

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A' &lt; 'AA'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'A' = 'a'</td>
<td>FALSE</td>
</tr>
<tr>
<td>4 &gt; 2</td>
<td>TRUE</td>
</tr>
<tr>
<td>17.3&lt; &gt; 5.6</td>
<td>TRUE</td>
</tr>
<tr>
<td>(3 <em>4) &lt; &gt; (4</em> 3)</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

With BOOLEAN operands, TRUE > FALSE is defined as a true statement. Thus the expression FALSE >= TRUE is a false statement. The statements FALSE >= FALSE and TRUE >= FALSE are also true statements.

### 3.2.3 Boolean Operations

The Boolean operators AND, OR, and NOT, with BOOLEAN operands, can be used to perform standard mathematical evaluations. Table 3–8 summarizes the results of evaluating Boolean expressions, and some examples are listed in Table 3–9.
### 3. USE OF OPERATORS

#### Table 3–8. BOOLEAN Operation Summary

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>OPERAND 1</th>
<th>OPERAND 2</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>TRUE</td>
<td>-</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>-</td>
<td>TRUE</td>
</tr>
<tr>
<td>OR</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>AND</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

#### Table 3–9. BOOLEAN Operations Using AND, OR, and NOT Operators

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &lt; y) OR (y &gt; z)</td>
<td>TRUE if x &lt; y or if y &gt; z; otherwise FALSE</td>
</tr>
<tr>
<td>(i = 2) OR (i = 753)</td>
<td>TRUE if i = 2 or if i = 753; otherwise FALSE</td>
</tr>
</tbody>
</table>

#### 3.2.4 Special Operations

The KAREL language provides special operators to perform functions such as testing the value of approximately equal POSITION variables, relative POSITION variables, VECTOR variables, and STRING variables. This section describes their operations and gives examples of their usage.

The following rules apply to approximately equal operations:

- The relational operator (>=<) determines if two POSITION operands are approximately equal and produces a BOOLEAN result. The comparison is similar to the equality (=) relation except that the operands compared need not be identical. Extended axis values are not considered.

- Approximately equal operations must be used in conjunction with the system variables, SLOCTOL, SORIENTTOL, and SCHECKCONFIG to determine how close two positions must be. Refer to the FANUC Robotics Software Reference Manual for a description of these variables.
• The relational operator (>=<) is allowed only in normal program use and cannot be used as a condition in a condition handler statement.

In the following example the relational operator (>=<) is used to determine if the current robot position (determined by using the CURPOS built-in procedure) is near the designated perch position:

**Relational Operator**

```plaintext
IF perch >=< CURPOS (0,0) THEN
    — Call move to perch program
    move_to_perch
ELSE
    ABORT
ENDIF
```

**Relative Position Operations**

To locate a position in space, you must reference it to a specific coordinate frame. In KAREL, reference frames have the POSITION data type. The relative position operator (:) allows you to reference a position or vector with respect to the coordinate frame of another position (that is, the coordinate frame that has the other position as its origin point).

The relative position operator (:) is used to transform a position from one reference frame to another frame.

In the example shown in Figure 3–1, a vision system is used to locate a target on a car such as a bolt head on a bumper. The relative position operator is used to calculate the position of the door handle based on data from the car drawings. The equation shown in Figure 3–1 is used to calculate the position of \texttt{w\_handle} in the WORLD frame.
Figure 3–1. Determining w_handle Relative to WORLD Frame

\[ w_{\text{handle}} = \text{bolt} : b_{\text{handle}} \]
\[ (\text{world} \ (\text{world} \ (\text{bumper} \ \text{frame}) \ \text{frame}) \ \text{frame}) \]

where:
- \text{bolt} is the position of the BUMPER frame origin referenced in the WORLD frame.
- \text{w_handle} is the handle position referenced in the WORLD frame.
- \text{b_handle} is the handle position referenced in the BUMPER frame.

The KAREL INV Built-In Function reverses the direction of the reference.

For example, to determine the position of the door handle target (b_handle) relative to the position of the bolt, use the equation shown in Figure 3–2.
3. USE OF OPERATORS

Figure 3–2. Determining b_handle Relative to BUMPER Frame

\[ b_{\text{handle}} = \text{INV(bolt)} : w_{\text{handle}} \]

(bumper (bumper (world frame) frame) frame)

where:

- \( \text{INV(bolt)} \) is the position of the WORLD frame origin referenced in the BUMPER frame.
- \( w_{\text{handle}} \) is the handle position referenced in the WORLD frame.
- \( b_{\text{handle}} \) is the handle position referenced in the BUMPER frame.

**Note** The order of the relative operator (:) is important. \( b_{\text{handle}} = \text{bolt} : w_{\text{handle}} \) is NOT the same as \( b_{\text{handle}} = w_{\text{handle}} : \text{bolt} \)

**See Also:** Chapter 8 POSITION DATA, INV Built-In Function, Appendix A.

**Vector Operations**

The following rules apply to VECTOR operations:

- A VECTOR expression can perform addition (+) and subtraction (-) equations on VECTOR operands. The result is a VECTOR whose components are the sum or difference of the corresponding components of the operands. For example, the components of the VECTOR \( \text{vect}_3 \) will equal (5, 10, 9) as a result of the following program statements:

\[
\begin{align*}
\text{vect}_1.x &= 4; \\
\text{vect}_1.y &= 8; \\
\text{vect}_1.z &= 5 \\
\text{vect}_2.x &= 1; \\
\text{vect}_2.y &= 2; \\
\text{vect}_2.z &= 4 \\
\text{vect}_3 &= \text{vect}_1 + \text{vect}_2
\end{align*}
\]

- The multiplication (*) and division (/) operators can be used with either
— A VECTOR and an INTEGER operand
— A VECTOR and a REAL operand

The product of a VECTOR and an INTEGER or a VECTOR and a REAL is a scaled version of the VECTOR. Each component of the VECTOR is multiplied by the INTEGER (treated as a REAL number) or the REAL.

For example, the VECTOR (8, 16, 10) is produced as a result of the following operation:

\[(4, 8, 5) \times 2\]

VECTOR components can be on the left or right side of the operator.

• A VECTOR divided by an INTEGER or a REAL causes each component of the VECTOR to be divided by the INTEGER (treated as a REAL number) or REAL. For example, \((4, 8, 5) / 2\) results in (2, 4, 2.5).

If the divisor is zero, the program is aborted with the “Divide by zero” error.

• An INTEGER or REAL divided by a VECTOR causes the INTEGER (treated as a REAL number) or REAL to be multiplied by the reciprocal of each element of the VECTOR, thus producing a new VECTOR. For example, \(3.5 / \text{VEC}(7.0, 8.0, 9.0)\) results in (0.5, 0.4375, 0.38889).

If any of the elements of the VECTOR are zero, the program is aborted with the “Divide by zero” error.

• The cross product operator (#) produces a VECTOR that is normal to the two operands in the direction indicated by the right hand rule and with a magnitude equal to the product of the magnitudes of the two vectors and \(\sin(\Theta)\), where \(\Theta\) is the angle between the two vectors. For example, \(\text{VEC}(3.0, 4.0, 5.0) \# \text{VEC}(6.0, 7.0, 8.0)\) results in (-3.0, 6.0, -3.0).

If either vector is zero, or the vectors are exactly parallel, an error occurs.

• The inner product operator (@) results in a REAL number that is the sum of the products of the corresponding elements of the two vectors. For example, \(\text{VEC}(3.0, 4.0, 5.0) @ \text{VEC}(6.0, 7.0, 8.0)\) results in 86.0.

• If the result of any of the above operations is a component of a VECTOR with a magnitude too large for a KAREL REAL number, the program is aborted with the “Real overflow” error.

Table 3–10 lists additional examples of vector operations.
Table 3–10. Examples of Vector Operations

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEC(3.0,7.0,6.0) + VEC(12.6,3.2,7.5)</td>
<td>(15.6,10.2,13.5)</td>
</tr>
<tr>
<td>VEC(7.6,9.0,7.0) - VEC(14.0,3.5,17.0)</td>
<td>(-6.4,5.5,-10)</td>
</tr>
<tr>
<td>4.5 * VEC(3.2,7.6,4.0)</td>
<td>(14.4,34.2,18.0)</td>
</tr>
<tr>
<td>VEC(12.7,2.0,8.3) * 7.6</td>
<td>(96.52,15.2,63.08)</td>
</tr>
<tr>
<td>VEC(17.3,1.5,0.23) / 2</td>
<td>(8.65,0.75,0.115)</td>
</tr>
</tbody>
</table>

**String Operations**

The following rules apply to STRING operations:

- You can specify that a KAREL routine returns a STRING as its value. See [Specifying a KAREL Routine to Return a STRING Value](#).

**Specifying a KAREL Routine to Return a STRING Value**

```
ROUTINE name(parameter_list): STRING
    declares name as returning a STRING value
```

- An operator can be used between strings to indicate the concatenation of the strings. See [Using an Operator to Concatenate Strings](#).

**Using an Operator to Concatenate Strings**

```
string_1 = string_2 + string_3 + 'ABC' + 'DEF'
```

- STRING expressions can be used in WRITE statements. See [Using a STRING Expression in a WRITE Statement](#).

**Using a STRING Expression in a WRITE Statement**

```
WRITE(CHR(13) + string_1 + string_2)
    writes a single string consisting of a return character followed by string_1 and string_2
```
• During STRING assignment, the string will be truncated if the target string is not large enough to hold the same string.

• You can compare or extract a character from a string. For example if \texttt{string\_1 = 'ABCDE'}. Your output would be \texttt{'D'}. See String Comparison.

**String Comparison**

```plaintext
IF SUB\_STR(string\_1, 4, 1) = 'D' THEN
```

• You can build a string from another string. See Building a String from Another String.

**Building a String from Another String**

```plaintext
ROUTINE toupper(p\_char: INTEGER): STRING
BEGIN
  IF (p\_char > 96) AND (p\_char < 123) THEN
    p\_char = p\_char - 32
  ENDIF
  RETURN (CHR(p\_char))
END toupper
```

```plaintext
BEGIN
  WRITE OUTPUT ('Enter string: ')
  READ INPUT (string\_1)
  string\_2 = '
  FOR idx = 1 TO STR\_LEN(string\_1) DO
    string\_2 = string\_2 + toupper(ORD(string\_1, idx))
  ENDFOR
```
Chapter 4

PROGRAM CONTROL

Contents

Chapter 4  PROGRAM CONTROL  ................................................................. 4–1
  4.1 OVERVIEW ........................................................................................ 4–2
  4.2 PROGRAM CONTROL STRUCTURES .............................................. 4–2
    4.2.1 Alternation Control Structures .................................................. 4–2
    4.2.2 Looping Control Statements ...................................................... 4–3
    4.2.3 Unconditional Branch Statement .............................................. 4–3
    4.2.4 Execution Control Statements ................................................... 4–3
    4.2.5 Condition Handlers .................................................................... 4–4
4.1 OVERVIEW

Program control structures define the flow of execution within a program or routine and include alternation, looping, and unconditional branching as well as execution control.

4.2 PROGRAM CONTROL STRUCTURES

Program control structures can be used to define the flow of execution within a program or routine. By default, execution starts with the first statement following the BEGIN statement and proceeds sequentially until the END statement (or a RETURN statement) is encountered. The following control structures are available in KAREL:

- Alternation
- Looping
- Unconditional Branching
- Execution Control
- Condition Handlers

For detailed information on each type of control structure, refer to Appendix A, “KAREL Language Alphabetical Description.”

4.2.1 Alternation Control Structures

An alternation control structure allows you to include alternative sequences of statements in a program or routine. Each alternative can consist of several statements.

During program execution, an alternative is selected based on the value of one or more data items. Program execution then proceeds through the selected sequence of statements.

Two types of alternation control structures can be used:

- **IF Statement** - provides a means of specifying one of two alternatives based on the value of a BOOLEAN expression.

- **SELECT Statement** - used when a choice is to be made between several alternatives. An alternative is chosen depending on the value of the specified INTEGER expression.

See Also: IF...THEN Statement, Appendix A, SELECT Statement, Appendix A.
4.2.2 Looping Control Statements

A looping control structure allows you to specify that a set of statements be repeated an arbitrary number of times, based on the value of data items in the program. KAREL supports three looping control structures:

- **The FOR statement** - used when a set of statements is to be executed a specified number of times. The number of times is determined by INTEGER data items in the FOR statement. At the beginning of the FOR loop, the initial value in the range is assigned to an INTEGER counter variable. Each time the cycle is repeated, the counter is reevaluated.

- **The REPEAT statement** - allows execution of a sequence of statements to continue as long as some BOOLEAN expression remains FALSE. The sequence of executable statements within the REPEAT statement will always be executed once.

- **The WHILE statement** - used when an action is to be executed as long as a BOOLEAN expression remains TRUE. The boolean expression is tested at the start of each iteration, so it is possible for the action to be executed zero times.

See Also: FOR Statement, Appendix A, REPEAT Statement, Appendix A, WHILE Statement, Appendix A

4.2.3 Unconditional Branch Statement

Unconditional branching allows you to use a GO TO Statement to transfer control from one place in a program to a specified label in another area of the program, without being dependent upon a condition or BOOLEAN expression.

**Warning**

Never include a GO TO Statement into or out of a FOR loop. The program might be aborted with a "Run time stack overflow" error.

See Also: GO TO Statement, Appendix A.

4.2.4 Execution Control Statements

The KAREL language provides the following program control statements, which are used to terminate or suspend program execution:

- **ABORT** - causes the execution of the program, including any motion in progress, to be terminated. The program cannot be continued after being aborted.

- **DELAY** - causes execution to be suspended for a specified time, expressed in milliseconds.
• **PAUSE** - causes execution to be suspended until a CONTINUE operation is executed.

• **WAIT FOR** - causes execution to be suspended until a specified condition or list of conditions is satisfied.

**See Also:** ABORT Statement, DELAY Statement, PAUSE Statement, WAIT FOR Statement, all in Appendix A, Chapter 6 *CONDITION HANDLERS*

### 4.2.5 Condition Handlers

A condition handler defines a series of actions which are to be performed whenever a specified condition is satisfied. Once defined, a condition handler can be ENABLED or DISABLED.
Routines, similar in structure to a program, provide a method of modularizing KAREL programs. Routines can include VAR and/or CONST declarations and executable statements. Unlike programs, however, a routine must be declared within an upper case program, and cannot include other routine declarations.

KAREL supports two types of routines:

- Procedure Routines - do not return a value
- Function Routines - return a value

KAREL routines can be predefined routines called built-in routines or they can be user-defined.

The following rules apply to all KAREL routines:

- Parameters can be included in the declaration of a routine. This allows you to pass data to the routine at the time it is called, and return the results to the calling program.
- Routines can be called or invoked:
  - By the program in which they are declared
  - By any routine contained in that program
  - With declarations by another program, refer to Section 5.1.1

## 5.1 ROUTINE EXECUTION

This section explains the execution of procedure and function routines:

- Declaring routines
- Invoking routines
- Returning from routines
- Scope of variables
- Parameters and Arguments

### 5.1.1 Declaring Routines

The following rules apply to routine declarations:

- A routine cannot be declared in another routine.
- The ROUTINE statement is used to declare both procedure and function routines.
- Both procedure and function routines must be declared before they are called.
• Routines that are local to the program are completely defined in the program. Declarations of local routines include:
  — The ROUTINE statement
  — Any VAR and/or CONST declarations for the routine
  — The executable statements of the routine

• While the VAR and CONST sections in a routine are identical in syntax to those in a program, the following restrictions apply:
  — PATH, FILE, and vision data types cannot be specified.
  — FROM clauses are not allowed.
  — IN clauses are not allowed.

• Routines that are local to the program can be defined after the executable section if the routine is declared using a FROM clause with the same program name. The parameters should only be defined once. See Defining Local Routines Using a FROM Clause.

Defining Local Routines Using a FROM Clause

```
PROGRAM funct_lib
ROUTINE done_yet(x: REAL; s1, s2: STRING): BOOLEAN FROM funct_lib
BEGIN
  IF done_yet(3.2, 'T', '')
  --
  END funct_lib
ROUTINE done_yet
BEGIN
  --
END done_yet
```

• Routines that are external to the program are declared in one program but defined in another.
  — Declarations of external routines include only the ROUTINE statement and a FROM clause.
  — The FROM clause identifies the name of the program in which the routine is defined.
  — The routine must be defined local to the program named in the FROM clause.

• You can include a list of parameters in the declaration of a routine. A parameter list is an optional part of the ROUTINE statement.

• If a routine is external to the program, the names in the parameter list are of no significance but must be included to specify the parameters. If there are no parameters, the parentheses used to enclose the list must be omitted for both external and local routines.

The examples in Local and External Procedure Declarations illustrate local and external procedure routine declarations.
Local and External Procedure Declarations

PROGRAM procs_lib
ROUTINE wait_a_bit
--local procedure, no parameters
BEGIN
  DELAY 20
END wait_a_bit
ROUTINE toggle_out(i: INTEGER)
--local procedure, one parameter
BEGIN
  DOUT[i] = ON --reference to parameter i
  DELAY 1000
  DOUT[i] = OFF
END toggle_out
ROUTINE calc_dist(p1,p2: POSITION; dist: REAL) & FROM math_lib
--external procedure defined in math_lib.kL
BEGIN
END procs_lib

The example in Function Declarations illustrate local and external function routine declarations.

Function Declarations

PROGRAM funct_lib
ROUTINE done_yet(x: REAL; s1, s2 :STRING): BOOLEAN & FROM bool_lib
--external function routine defined in bool_lib.kL
--returns a BOOLEAN value
ROUTINE xy_dist(x1,y1,x2,y2: REAL): REAL
--local function, returns a REAL value
VAR
  sum_square: REAL --dynamic local variable
  dx,dy: REAL --dynamic local variables
BEGIN
  dx = x2-x1 --references parameters x2 and x1
  dy = y2-y1 --references parameters y2 and y1
  sum_square = dx * dx + dy * dy
  RETURN(SQRT(sum_square)) --SQRT is a built-in
END xy_dist
BEGIN
END funct_lib

See Also: FROM Clause, Appendix A, ROUTINE Statement, Appendix A.
5.1.2 Invoking Routines

Routines that are declared in a program can be called within the executable section of the program, or within the executable section of any routine contained in the program. Calling a routine causes the routine to be invoked. A routine is invoked according to the following procedure:

- When a routine is invoked, control of execution passes to the routine.
- After execution of a procedure is finished, control returns to the next statement after the point where the procedure was called.
- After execution of a function is finished, control returns to the assignment statement where the function was called.

The following rules apply when invoking procedure and function routines:

- Procedure and function routines are both called with the routine name followed by an argument for each parameter that has been declared for the routine.
- The argument list is enclosed in parentheses.
- Routines without parameters are called with only the routine name.
- A procedure is invoked as though it were a statement. Consequently, a procedure call constitutes a complete executable statement.

Procedure Calls shows the declarations for two procedures followed by the procedure calls to invoke them.

Procedure Calls

ROUTINE wait_a_bit FROM proc_lib
   --external procedure with no parameters
ROUTINE calc_dist(p1,p2: POSITION; dist: REAL)&
FROM math_lib
   --external procedure with three parameters
BEGIN
   ...
   wait_a_bit    --invokes wait_a_bit procedure
   calc_dist (start_pos, end_pos, distance)
   --invokes calc_dist using three arguments for
   --the three declared parameters

- Because a function returns a value, a function call must appear as part or all of an expression.
- When control returns to the calling program or routine, execution of the statement containing the function call is resumed using the returned value.

Function Calls shows the declarations for two functions followed by the function calls to invoke them.
Function Calls

ROUTINE error_check : BOOLEAN FROM error_prog
--external function with no parameters returns a BOOLEAN value

ROUTINE distance(p1, p2: POSITION) : REAL &
FROM funct_lib
--external function with two parameters returns a REAL value

BEGIN  --Main program
--the function error_check is invoked and returns a BOOLEAN
--expression in the IF statement
  IF error_check THEN
    ...
  ENDIF
  travel_time = distance(prev_pos, next_pos)/current_spd
--the function distance is invoked as part of an expression in
--an assignment statement

• Routines can call other routines as long as the other routine is declared in the program containing the initial routine. For example, if a program named master_prog contains a routine named call_proc, that routine can call any routine that is declared in the program, master_prog.

• A routine that calls itself is said to be recursive and is allowed in KAREL. For example, the routine factorial, shown in Recursive Function, calls itself to calculate a factorial value.

Recursive Function

ROUTINE factorial(n: INTEGER) : INTEGER
--calculates the factorial value of the integer n
BEGIN
  IF n = 0 THEN RETURN (1)
  ELSE RETURN (n * factorial(n-1))
  --recursive call to factorial
  ENDIF
END factorial

• The only constraint on the depth of routine calling is the use of the KAREL stack, an area used for storage of temporary and local variables and for parameters. Routine calls cause information to be placed in memory on the stack. When the RETURN or END statement is executed in the routine, this information is taken off of the stack. If too many routine calls are made without this information being removed from the stack, the program will run out of stack space.

See Also: Section 5.1.6 for information on how much space is used on the stack for routine calls
5.1.3 Returning from Routines

The RETURN statement is used in a routine to restore execution control from a routine to the calling routine or program.

The following rules apply when returning from a routine:

- In a procedure, the RETURN statement cannot include a value.
- If no RETURN statement is executed, the END statement restores control to the calling program or routine.

Procedure RETURN Statements illustrates some examples of using the RETURN statement in a procedure.

Procedure RETURN Statements

ROUTINE gun_on (error_flag: INTEGER)
  --performs some operation while a "gun" is turned on
  --returns from different statements depending on what,
  --if any, error occurs.
VAR gun: INTEGER
BEGIN
  IF error_flag = 1 THEN RETURN
  --abnormal exit from routine, returns before
  --executing WHILE loop
  ENDIF
  WHILE DIN[gun] DO
  --continues until gun is off
  ...
  IF error_flag = 2 THEN RETURN
  --abnormal exit from routine, returns from
  --within WHILE loop
  ENDIF
ENDWHILE
END gun_on

- In a function, the RETURN statement must specify a value to be passed back when control is restored to the calling routine or program.
- The function routine can return any data type except
  - FILE
  - PATH
  - Vision types
- If the return type is an ARRAY, you cannot specify a size. This allows an ARRAY of any length to be returned by the function. The returned ARRAY, from an ARRAY valued function, can
be used only in a direct assignment statement. ARRAY valued functions cannot be used as parameters to other routines. Refer to Correct Passage of an ARRAY, for an example of an ARRAY passed between two function routines.

- If no value is provided in the RETURN statement of a function, a translator error is generated.
- If no RETURN statement is executed in a function, execution of the function terminates when the END statement is reached. No value can be passed back to the calling routine or program, so the program aborts with an error.

Function RETURN Statements illustrates some examples using the RETURN statement in function routines.

Function RETURN Statements

ROUTINE index_value (table: ARRAY of INTEGER;
    table_size: INTEGER): INTEGER
    --Returns index value of FOR loop (i) depending on
    --condition of IF statement. Returns 0 in cases where
    --IF condition is not satisfied.
    VAR i: INTEGER
    BEGIN
        FOR i = 1 TO table_size DO
            IF table[i] = 0 THEN RETURN (i) --returns index
                ENDIF
        ENDFOR
        RETURN (0) --returns 0
    END index_value

ROUTINE compare (test_var_1: INTEGER;
     test_var_2: INTEGER): BOOLEAN
    --Returns TRUE value in cases where IF test is
    --satisfied. Otherwise, returns FALSE value.
    BEGIN
        IF test_var_1 = test_var_2 THEN
            RETURN (TRUE) --returns TRUE
        ELSE
            RETURN (FALSE) --returns FALSE
        ENDIF
    END compare

See Also: ROUTINE Statement, Appendix A.

5.1.4 Scope of Variables

The scope of a variable declaration can be

- Global
• Local

Global Declarations and Definitions

The following rules apply to global declarations and definitions:

• Global declarations are recognized throughout a program.

• Global declarations are referred to as static because they are given a memory location that does not change during program execution, even if the program is cleared or reloaded (unless the variables themselves are cleared.)

• Declarations made in the main program, as well as predefined identifiers, are global.

• The scope rules for predefined and user-defined routines, types, variables, constants, and labels are as follows:
  — All predefined identifiers are recognized throughout the entire program.
  — Routines, types, variables, and constants declared in the declaration section of a program are recognized throughout the entire program, including routines that are in the program.

Local Declarations and Definitions

The following rules apply to local declarations and definitions:

• Local declarations are recognized only within the routines where they are declared.

• Local data is created when a routine is invoked. Local data is destroyed when the routine finishes executing and returns.

• The scope rules for predefined and user-defined routines, variables, constants, and labels are as follows:
  — Variables and constants, declared in the declaration section of a routine, and parameters, declared in the routine parameter list, are recognized only in that routine.
  — Labels defined in a program (not in a routine of the program) are local to the body of the program and are not recognized within any routines of the program.
  — Labels defined in a routine are local to the routine and are recognized only in that routine.

• Types cannot be declared in a routine, so are never local.

5.1.5 Parameters and Arguments

Identifiers that are used in the parameter list of a routine declaration are referred to as parameters. A parameter declared in a routine can be referenced throughout the routine. Parameters are used to pass data between the calling program and the routine. The data supplied in a call, referred to as arguments, can affect the way in which the routine is executed.

The following rules apply to the parameter list of a routine call:
As part of the routine call, you must supply a data item, referred to as an argument, for each parameter in the routine declaration.

An argument can be a variable, constant, or expression. There must be one argument corresponding to each parameter.

Arguments must be of the same data type as the parameters to which they correspond, with three exceptions:

- An INTEGER argument can be passed to a REAL parameter. In this case, the INTEGER value is treated as type REAL, and the REAL equivalent of the INTEGER is passed by value to the routine.
- A BYTE or SHORT argument can be passed by value to an INTEGER or REAL parameter.
- Any positional types can be passed to any other positional type. If they are being passed to a user-defined routine, the argument positional type is converted and passed by value to the parameter type.
- ARRAY or STRING arguments of any length can be passed to parameters of the same data type.

Corresponding Parameters and Arguments shows an example of a routine declaration and three calls to that routine.

**Corresponding Parameters and Arguments**

```
PROGRAM params
VAR
  long_string: STRING[10]; short_string: STRING[5]
  exact_dist: REAL; rough_dist: INTEGER
ROUTINE label_dist (strg: STRING; dist: REAL) &
  FROM procs_lib
BEGIN
  ...
  label_dist(long_string, exact_dist)
  --long_string corresponds to strg;
  --exact_dist corresponds to dist
  label_dist(short_string, rough_dist)
  --short_string, of a different length,
  --corresponds to strg; rough_dist, an
  --INTEGER, corresponds to REAL dist
  label_dist('new distance', (exact_dist * .75))
  --literal constant and REAL expression
  --arguments correspond to the parameters
END params
```

When the routine is invoked, the argument used in the routine call is passed to the corresponding parameter. Two methods are used for passing arguments to parameters:

- **Passing Arguments By Reference**
If an argument is passed by reference, the corresponding parameter shares the same memory location as the argument. Therefore, changing the value of the parameter changes the value of the corresponding argument.

— **Passing Arguments By Value**

If an argument is passed by value, a temporary copy of the argument is passed to the routine. The corresponding parameter uses this temporary copy. Changing the parameter does not affect the original argument.

- Constant and expression arguments are always passed to the routine by value. Variables are normally passed by reference. The following variable arguments, however, are passed by value:
  - Port array variables
  - INTEGER variables passed to REAL parameters
  - BYTE and SHORT arguments passed to INTEGER or REAL parameters
  - System variables with read only (RO) access
  - Positional parameters that need to be converted

- While variable arguments are normally passed by reference, you can pass them by value by enclosing the variable identifier in parentheses. The parentheses, in effect, turn the variable into an expression.

- PATH, FILE, and vision variables cannot be passed by value. ARRAY elements (indexed form of an ARRAY variable) can be passed by value, but entire ARRAY variables cannot.

**Passing Variable Arguments** shows a routine that affects the argument being passed to it differently depending on how the variable argument is passed.

**Passing Variable Arguments**

```plaintext
PROGRAM reference
  VAR arg : INTEGER
  ROUTINE test(param : INTEGER)
  BEGIN
    param = param * 3
    WRITE ('value of param:', param, CR)
  END test
  BEGIN
    arg = 5
    test(arg) -- arg passed to param by value
    WRITE('value of arg:', arg, CR)
    test((arg)) -- arg passed to param by value
    WRITE('value of arg:', arg, CR)
  END reference
```

The output from the program in **Passing Variable Arguments** is as follows:

value of param: 15
If the routine calls from Passing Variable Arguments were made in reverse order, first passing arg by reference using "test(arg)" and then passing it by value using "test((arg))," the output would be affected as follows:

- value of param: 15
- value of arg: 15
- value of param: 45
- value of arg: 15

- To pass a variable as a parameter to a KAREL routine you can use one of two methods:
  - You can specify the name of the variable in the parameter list. For example, `other_rtn(param_var)` passes the variable `param_var` to the routine `other_rtn`. To write this statement, you have to know the name of the variable to be passed.
  - You can use BYNAME. The BYNAME feature allows a program to pass as a parameter to a routine a variable whose name is contained in a string. For example, if the string variables `prog_name` and `var_name` contain the name of a program and variable the operator has entered, this variable is passed to a routine using this syntax:

    `other_rtn(BYNAME(prog_name,var_name, entry))`

    Refer to Appendix A for more information about BYNAME.

- If a function routine returns an ARRAY, a call to this function cannot be used as an argument to another routine. If an incorrect pass is attempted, a translation error is detected.

Correct Passage of an ARRAY shows the correct use of an ARRAY passed between two function routines.

**Correct Passage of an ARRAY**

```
PROGRAM correct
VAR a : ARRAY[8] of INTEGER
ROUTINE rtn_ary : ARRAY of INTEGER FROM util_prog
ROUTINE print_ary(arg : ARRAY of INTEGER)
    VAR i : INTEGER
    BEGIN
        FOR i = 1 to ARRAY_LEN(arg) DO
            WRITE(arg[i],cr)
        ENDFOR
    END print_ary
BEGIN
    a = rtn_ary
    print_ary(a)
```

Incorrect Passage of an ARRAY shows the incorrect use of an ARRAY passed between two function routines.

Incorrect Passage of an ARRAY

PROGRAM wrong
ROUTINE rtn_ary : ARRAY of INTEGER FROM util_prog
ROUTINE print_ary(arg : ARRAY of INTEGER)
  VAR i : INTEGER
  BEGIN
    FOR i = 1 to ARRAY_LEN(arg) DO
      WRITE(arg[i],cr)
    ENDFOR
  END print_ary
BEGIN
  print_ary(rtn_ary)
END wrong

See Also: ARRAY_LEN Built-In Function, Appendix A, STR_LEN Built-In Function, Appendix A, Appendix E, "Syntax Diagrams"

5.1.6 Stack Usage

When a program is executed, a stack of 300 words is allocated unless you specify a stack size. The stack is allocated from available user RAM.

Stack usage can be calculated as follows:

- Each call (or function reference) uses at least five words of stack.
- In addition, for each parameter and local variable in the routine, additional space on the stack is used, depending on the variable or parameter type as shown in Table 5–1.

Table 5–1. Stack Usage

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter Passed by Reference</th>
<th>Parameter Passed by Value</th>
<th>Local Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ARRAY OF BOOLEAN</td>
<td>not allowed</td>
<td></td>
<td>1 + array size</td>
</tr>
<tr>
<td>ARRAY OF BYTE</td>
<td>1</td>
<td>not allowed</td>
<td>1 + (array size)/4</td>
</tr>
<tr>
<td>Type</td>
<td>Parameter Passed by Reference</td>
<td>Parameter Passed by Value</td>
<td>Local Variable</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>CAM_SETUP</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF CAM_SETUP</td>
<td></td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>CONFIG</td>
<td>1</td>
<td>2</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF CONFIG</td>
<td></td>
<td>1 + array size</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF INTEGER</td>
<td></td>
<td>1 + array size</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF FILE</td>
<td></td>
<td>not allowed</td>
<td></td>
</tr>
<tr>
<td>JOINTPOS</td>
<td>2</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 10 * array size</td>
</tr>
<tr>
<td>JOINTPOS1</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS1</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 2 * array size</td>
</tr>
<tr>
<td>JOINTPOS2</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS2</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 3 * array size</td>
</tr>
<tr>
<td>JOINTPOS3</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS3</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 4 * array size</td>
</tr>
<tr>
<td>JOINTPOS4</td>
<td>2</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS4</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 5 * array size</td>
</tr>
<tr>
<td>JOINTPOS5</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS5</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 6 * array size</td>
</tr>
<tr>
<td>JOINTPOS6</td>
<td>2</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS6</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 7 * array size</td>
</tr>
<tr>
<td>JOINTPOS7</td>
<td>2</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS7</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 8 * array size</td>
</tr>
<tr>
<td>JOINTPOS8</td>
<td>2</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS8</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 9 * array size</td>
</tr>
<tr>
<td>JOINTPOS9</td>
<td>2</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>ARRAY OF JOINTPOS9</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 10 * array size</td>
</tr>
<tr>
<td>MODEL</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF MODEL</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>PATH</td>
<td>2</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>POSITION</td>
<td>2</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>ARRAY OF POSITION</td>
<td>1</td>
<td>not allowed</td>
<td>1 + 14 * array size</td>
</tr>
</tbody>
</table>
### Table 5–1. Stack Usage (Cont’d)

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter Passed by Reference</th>
<th>Parameter Passed by Value</th>
<th>Local Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ARRAY OF REAL</td>
<td>1</td>
<td>not allowed</td>
<td>1 + array size</td>
</tr>
<tr>
<td>ARRAY OF SHORT</td>
<td>1</td>
<td>not allowed</td>
<td>1 + (array size)/2</td>
</tr>
<tr>
<td>STRING</td>
<td>2</td>
<td>(string length+2)/4</td>
<td>1+(string length+2) *array size/4</td>
</tr>
<tr>
<td>ARRAY OF STRING</td>
<td>1</td>
<td>not allowed</td>
<td></td>
</tr>
<tr>
<td>VECTOR</td>
<td>4</td>
<td>3</td>
<td>1 + 3 * array size</td>
</tr>
<tr>
<td>ARRAY OF VECTOR</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>VIS_PROCESS</td>
<td>1</td>
<td>not allowed</td>
<td>not allowed</td>
</tr>
<tr>
<td>ARRAY OF VIS_PROCESS</td>
<td>1</td>
<td>not allowed</td>
<td></td>
</tr>
<tr>
<td>XYZWPR</td>
<td>10</td>
<td>8</td>
<td>1 + 8 * array size</td>
</tr>
<tr>
<td>ARRAY OF XYZWPR</td>
<td>1</td>
<td>not allowed</td>
<td></td>
</tr>
<tr>
<td>XYZWPREXT</td>
<td>13</td>
<td>11</td>
<td>1 + 11 * array size</td>
</tr>
<tr>
<td>ARRAY [m,n] OF some_type</td>
<td>not allowed</td>
<td>m(ele size/4 * n + 1)+1</td>
<td></td>
</tr>
<tr>
<td>ARRAY [l,m,n] OF some_type</td>
<td>not allowed</td>
<td>l(m(ele size/4 * n + 1)+1)+1</td>
<td></td>
</tr>
</tbody>
</table>

### 5.2 BUILT-IN ROUTINES

The KAREL language includes predefined routines referred to as KAREL built-in routines, or built-ins. Predefined routines can be either procedure or function built-ins. They are provided as a programming convenience and perform commonly needed services.

Many of the built-ins return a status parameter that signifies an error if not equal to 0. The error returned can be any of the error codes defined in the application-specific *FANUC Robotics Setup and Operations Manual*. These errors can be posted to the error log and displayed on the error line by calling the POST_ERR built-in routine with the returned status parameter.

*Table A–7* is a summary list of all the predefined built-in routines included in the KAREL language. A detailed description of all the KAREL built-in routines is provided in *Appendix A*.

**See Also:** *Appendix A*, which lists optional KAREL built-ins and where they are documented.
<table>
<thead>
<tr>
<th>Category</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byname</strong></td>
<td>CALL_PROG</td>
</tr>
<tr>
<td></td>
<td>CALL_PROGLIN</td>
</tr>
<tr>
<td></td>
<td>CURR_PROG</td>
</tr>
<tr>
<td></td>
<td>FILE_LIST</td>
</tr>
<tr>
<td></td>
<td>PROG_LIST</td>
</tr>
<tr>
<td></td>
<td>VAR_INFO</td>
</tr>
<tr>
<td></td>
<td>VAR_LIST</td>
</tr>
<tr>
<td><strong>Data Acquisition</strong></td>
<td>DAQ_CHECKP</td>
</tr>
<tr>
<td></td>
<td>DAQ_REGPIPE</td>
</tr>
<tr>
<td></td>
<td>DAQ_START</td>
</tr>
<tr>
<td></td>
<td>DAQ_STOP</td>
</tr>
<tr>
<td></td>
<td>DAQ_UNREG</td>
</tr>
<tr>
<td></td>
<td>DAQ_WRITE</td>
</tr>
<tr>
<td><strong>Error Code Handling</strong></td>
<td>ERR_DATA</td>
</tr>
<tr>
<td></td>
<td>POST_ERR</td>
</tr>
<tr>
<td></td>
<td>POST_ERR_L</td>
</tr>
<tr>
<td><strong>File and Device Operation</strong></td>
<td>CHECK_NAME</td>
</tr>
<tr>
<td></td>
<td>COMPARE_FILE</td>
</tr>
<tr>
<td></td>
<td>COPY_FILE</td>
</tr>
<tr>
<td></td>
<td>DELETE_FILE</td>
</tr>
<tr>
<td></td>
<td>DISMOUNT_DEV</td>
</tr>
<tr>
<td></td>
<td>DOSFILE_INF</td>
</tr>
<tr>
<td></td>
<td>FORMAT_DEV</td>
</tr>
<tr>
<td></td>
<td>MOUNT_DEV</td>
</tr>
<tr>
<td></td>
<td>MOVE_FILE</td>
</tr>
<tr>
<td></td>
<td>PRINT_FILE</td>
</tr>
<tr>
<td></td>
<td>PURGE_DEV</td>
</tr>
<tr>
<td></td>
<td>RENAME_FILE</td>
</tr>
<tr>
<td></td>
<td>XML_ADDTAG</td>
</tr>
<tr>
<td></td>
<td>XML_GETDATA</td>
</tr>
<tr>
<td></td>
<td>XML_REMTAG</td>
</tr>
<tr>
<td></td>
<td>XML_SCAN</td>
</tr>
<tr>
<td></td>
<td>XML_SETVAR</td>
</tr>
<tr>
<td><strong>Serial I/O, File Usage</strong></td>
<td>BYTES_AHEAD</td>
</tr>
<tr>
<td></td>
<td>BYTES_LEFT</td>
</tr>
<tr>
<td></td>
<td>CLR_IO_STAT</td>
</tr>
<tr>
<td></td>
<td>GET_FILE_POS</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_ATR</td>
</tr>
<tr>
<td></td>
<td>IO_STATUS</td>
</tr>
<tr>
<td></td>
<td>MSG_CONNECT</td>
</tr>
<tr>
<td></td>
<td>MSG_DISCO</td>
</tr>
<tr>
<td></td>
<td>MSG_PING</td>
</tr>
<tr>
<td></td>
<td>PIPE_CONFIG</td>
</tr>
<tr>
<td></td>
<td>SET_FILE_ATR</td>
</tr>
<tr>
<td></td>
<td>SET_FILE_POS</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_ATR</td>
</tr>
<tr>
<td></td>
<td>VOL_SPACE</td>
</tr>
<tr>
<td><strong>Process I/O Setup</strong></td>
<td>CLR_PORT_SIM</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_ASG</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_MOD</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_SIM</td>
</tr>
<tr>
<td></td>
<td>GET_PORT_VAL</td>
</tr>
<tr>
<td></td>
<td>IO_MOD_TYPE</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_ASG</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_CMT</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_MOD</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_SIM</td>
</tr>
<tr>
<td></td>
<td>SET_PORT_VAL</td>
</tr>
<tr>
<td><strong>KCL Operation</strong></td>
<td>KCL</td>
</tr>
<tr>
<td></td>
<td>KCL_NO_WAIT</td>
</tr>
<tr>
<td></td>
<td>KCL_STATUS</td>
</tr>
<tr>
<td><strong>Memory Operation</strong></td>
<td>CLEAR</td>
</tr>
<tr>
<td></td>
<td>CREATE_VAR</td>
</tr>
<tr>
<td></td>
<td>LOAD</td>
</tr>
<tr>
<td></td>
<td>LOAD_STATUS</td>
</tr>
<tr>
<td></td>
<td>PROG_BACKUP</td>
</tr>
<tr>
<td></td>
<td>PROG_CLEAR</td>
</tr>
<tr>
<td></td>
<td>PROG_RESTORE</td>
</tr>
<tr>
<td></td>
<td>RENAME_VARS</td>
</tr>
<tr>
<td></td>
<td>SAVE</td>
</tr>
<tr>
<td></td>
<td>SAVE_DRAM</td>
</tr>
<tr>
<td><strong>Mirror</strong></td>
<td>MIRROR</td>
</tr>
<tr>
<td><strong>Motion and Program Control</strong></td>
<td>CNCL_STP_MTN</td>
</tr>
<tr>
<td></td>
<td>MOTION_CTL</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
</tr>
<tr>
<td><strong>Multi-programming</strong></td>
<td>ABORT_TASK</td>
</tr>
<tr>
<td></td>
<td>CLEAR_SEMA</td>
</tr>
<tr>
<td></td>
<td>CONT_TASK</td>
</tr>
<tr>
<td></td>
<td>GET_TSK_INFO</td>
</tr>
<tr>
<td></td>
<td>LOCK_GROUP</td>
</tr>
<tr>
<td></td>
<td>PAUSE_TASK</td>
</tr>
<tr>
<td></td>
<td>PEND_SEMA</td>
</tr>
<tr>
<td></td>
<td>POST_SEMA</td>
</tr>
<tr>
<td></td>
<td>RUN_TASK</td>
</tr>
<tr>
<td></td>
<td>SEMA_COUNT</td>
</tr>
<tr>
<td></td>
<td>SET_TSK_ATTR</td>
</tr>
<tr>
<td></td>
<td>SET_TSK_NAME</td>
</tr>
<tr>
<td></td>
<td>UNLOCK_GROUP</td>
</tr>
<tr>
<td><strong>Path Operation</strong></td>
<td>APPEND_NODE</td>
</tr>
<tr>
<td></td>
<td>COPY_PATH</td>
</tr>
<tr>
<td></td>
<td>DELETE_NODE</td>
</tr>
<tr>
<td></td>
<td>INSERT_NODE</td>
</tr>
<tr>
<td></td>
<td>NODE_SIZE</td>
</tr>
<tr>
<td></td>
<td>PATH_LEN</td>
</tr>
<tr>
<td>Category</td>
<td>Identifier</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Personal Computer Communications</td>
<td>ADD_BYNAMETPC</td>
</tr>
<tr>
<td></td>
<td>ADD_INTPC</td>
</tr>
<tr>
<td></td>
<td>ADD_REALPC</td>
</tr>
<tr>
<td></td>
<td>ADD_STRINGPC</td>
</tr>
<tr>
<td></td>
<td>SEND_DATAPC</td>
</tr>
<tr>
<td></td>
<td>SEND_EVENTPC</td>
</tr>
<tr>
<td>Position</td>
<td>CHECK_EPOS</td>
</tr>
<tr>
<td></td>
<td>CNV_JPOS_REL</td>
</tr>
<tr>
<td></td>
<td>CNV_JPOS_REL</td>
</tr>
<tr>
<td></td>
<td>CURPOS</td>
</tr>
<tr>
<td></td>
<td>CURJPOS</td>
</tr>
<tr>
<td></td>
<td>FRAME</td>
</tr>
<tr>
<td></td>
<td>IN_RANGE</td>
</tr>
<tr>
<td></td>
<td>J_IN_RANGE</td>
</tr>
<tr>
<td></td>
<td>JOINT2POS</td>
</tr>
<tr>
<td></td>
<td>POS</td>
</tr>
<tr>
<td></td>
<td>POS2JOINT</td>
</tr>
<tr>
<td></td>
<td>SET_PERCH</td>
</tr>
<tr>
<td></td>
<td>UNPOS</td>
</tr>
<tr>
<td>Queue Manager</td>
<td>APPEND_QUEUE</td>
</tr>
<tr>
<td></td>
<td>COPY_QUEUE</td>
</tr>
<tr>
<td></td>
<td>DELETE_QUEUE</td>
</tr>
<tr>
<td></td>
<td>GET_QUEUE</td>
</tr>
<tr>
<td></td>
<td>INIT_QUEUE</td>
</tr>
<tr>
<td></td>
<td>INSERT_QUEUE</td>
</tr>
<tr>
<td></td>
<td>MODIFY_QUEUE</td>
</tr>
<tr>
<td>Register Operation</td>
<td>CLR_POS_REG</td>
</tr>
<tr>
<td></td>
<td>GET_JPOS_REG</td>
</tr>
<tr>
<td></td>
<td>GET_POS_REG</td>
</tr>
<tr>
<td></td>
<td>GET_PREG_REG</td>
</tr>
<tr>
<td></td>
<td>GET_REG</td>
</tr>
<tr>
<td></td>
<td>GET_REG_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_SREG_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_STR_REG</td>
</tr>
<tr>
<td></td>
<td>POS_REG_TYPE</td>
</tr>
<tr>
<td></td>
<td>SET_EPOS_REG</td>
</tr>
<tr>
<td></td>
<td>SET_INT_REG</td>
</tr>
<tr>
<td></td>
<td>SET_JPOS_REG</td>
</tr>
<tr>
<td>String Operation</td>
<td>CNV_CNF_STRG</td>
</tr>
<tr>
<td></td>
<td>CNV_CONF_STR</td>
</tr>
<tr>
<td></td>
<td>CNV_INT_STR</td>
</tr>
<tr>
<td></td>
<td>CNV_REAL_STR</td>
</tr>
<tr>
<td></td>
<td>CNV_STR_CONF</td>
</tr>
<tr>
<td></td>
<td>CNV_STR_INT</td>
</tr>
<tr>
<td></td>
<td>CNV_STR_REAL</td>
</tr>
<tr>
<td></td>
<td>STR_LEN</td>
</tr>
<tr>
<td></td>
<td>SUB_STR</td>
</tr>
<tr>
<td>System</td>
<td>ABS</td>
</tr>
<tr>
<td></td>
<td>ACOS</td>
</tr>
<tr>
<td></td>
<td>ARRAY_LEN</td>
</tr>
<tr>
<td></td>
<td>ASIN</td>
</tr>
<tr>
<td></td>
<td>ATAN2</td>
</tr>
<tr>
<td></td>
<td>BYNAME</td>
</tr>
<tr>
<td></td>
<td>CHR</td>
</tr>
<tr>
<td></td>
<td>COS</td>
</tr>
<tr>
<td></td>
<td>EXP</td>
</tr>
<tr>
<td></td>
<td>GET_VAR</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>INV</td>
</tr>
<tr>
<td></td>
<td>LN</td>
</tr>
<tr>
<td></td>
<td>ORD</td>
</tr>
<tr>
<td></td>
<td>ROUND</td>
</tr>
<tr>
<td></td>
<td>SET_VAR</td>
</tr>
<tr>
<td></td>
<td>SIN</td>
</tr>
<tr>
<td></td>
<td>SQRT</td>
</tr>
<tr>
<td></td>
<td>TAN</td>
</tr>
<tr>
<td></td>
<td>TRUNC</td>
</tr>
<tr>
<td></td>
<td>UNINIT</td>
</tr>
<tr>
<td>Time-of-Day Operation</td>
<td>CNV_STR_TIME</td>
</tr>
<tr>
<td></td>
<td>CNV_TIME_STR</td>
</tr>
<tr>
<td></td>
<td>GET_TIME</td>
</tr>
<tr>
<td></td>
<td>GET_USEC_SUB</td>
</tr>
<tr>
<td></td>
<td>GET_USEC_TIM</td>
</tr>
<tr>
<td></td>
<td>SET_TIME</td>
</tr>
<tr>
<td>TPE Program</td>
<td>AVL_POS_NUM</td>
</tr>
<tr>
<td></td>
<td>CLOSE_TPE</td>
</tr>
<tr>
<td></td>
<td>COPY_TPE</td>
</tr>
<tr>
<td></td>
<td>CREATE_TPE</td>
</tr>
<tr>
<td></td>
<td>DEL_INST_TPE</td>
</tr>
<tr>
<td></td>
<td>GET_ATTR_PRG</td>
</tr>
<tr>
<td></td>
<td>GET_JPOS_TPE</td>
</tr>
<tr>
<td></td>
<td>GET_POS_FRM</td>
</tr>
<tr>
<td></td>
<td>GET_POS_TPE</td>
</tr>
<tr>
<td></td>
<td>GET_POS_TYP</td>
</tr>
<tr>
<td></td>
<td>GET_TPE_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_TPE_PRM</td>
</tr>
<tr>
<td></td>
<td>OPEN_TPE</td>
</tr>
<tr>
<td></td>
<td>SELECT_TPE</td>
</tr>
<tr>
<td></td>
<td>SET_ATTR_PRG</td>
</tr>
<tr>
<td></td>
<td>SET_EPOS_TPE</td>
</tr>
<tr>
<td></td>
<td>SET_JPOS_TPE</td>
</tr>
<tr>
<td></td>
<td>SET_POS_TPE</td>
</tr>
<tr>
<td></td>
<td>SET_TPE_CMT</td>
</tr>
<tr>
<td></td>
<td>SET_TRNS_TPE</td>
</tr>
<tr>
<td>Translate</td>
<td>TRANSLATE</td>
</tr>
<tr>
<td>Category</td>
<td>Identifier</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>User Interface</td>
<td>ACT_SCREEN</td>
</tr>
<tr>
<td></td>
<td>ADD_DICT</td>
</tr>
<tr>
<td></td>
<td>ATT_WINDOW_D</td>
</tr>
<tr>
<td></td>
<td>ATT_WINDOW_S</td>
</tr>
<tr>
<td></td>
<td>CHECK_DICT</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISB</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISI</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISP</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISR</td>
</tr>
<tr>
<td></td>
<td>DEF_SCREEN</td>
</tr>
<tr>
<td></td>
<td>DEF_WINDOW</td>
</tr>
<tr>
<td></td>
<td>DET_WINDOW</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_ALPH</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_FORM</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_LIST</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_PLMN</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_SBMN</td>
</tr>
<tr>
<td></td>
<td>DISCTRL_TBL</td>
</tr>
<tr>
<td></td>
<td>FORCE_SPMENU</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISB</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISE</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISI</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISP</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISR</td>
</tr>
<tr>
<td></td>
<td>INI_DYN_DISS</td>
</tr>
<tr>
<td></td>
<td>INIT_TBL</td>
</tr>
<tr>
<td></td>
<td>POP_KEY_RD</td>
</tr>
<tr>
<td></td>
<td>PUSH_KEY_RD</td>
</tr>
<tr>
<td></td>
<td>READ_DICT</td>
</tr>
<tr>
<td></td>
<td>READ_DICT_V</td>
</tr>
<tr>
<td></td>
<td>READ_KB</td>
</tr>
<tr>
<td></td>
<td>REMOVE_DICT</td>
</tr>
<tr>
<td></td>
<td>SET_CURSOR</td>
</tr>
<tr>
<td></td>
<td>SET_LANG</td>
</tr>
<tr>
<td></td>
<td>WRITE_DICT</td>
</tr>
<tr>
<td></td>
<td>WRITE_DICT_V</td>
</tr>
<tr>
<td>Vector</td>
<td>APPROACH</td>
</tr>
<tr>
<td></td>
<td>ORIENT</td>
</tr>
<tr>
<td>Vision Operation</td>
<td>V_CAM_CALIB</td>
</tr>
<tr>
<td></td>
<td>V_GET_OFFSET</td>
</tr>
<tr>
<td></td>
<td>V_GET_PASSFL</td>
</tr>
<tr>
<td></td>
<td>V_GET_QUEUE</td>
</tr>
<tr>
<td></td>
<td>V_INIT_QUEUE</td>
</tr>
<tr>
<td></td>
<td>V_RALC_QUEUE</td>
</tr>
<tr>
<td></td>
<td>V_RUN_FIND</td>
</tr>
<tr>
<td></td>
<td>V_SET_REF</td>
</tr>
<tr>
<td></td>
<td>V_START_VTRK</td>
</tr>
<tr>
<td></td>
<td>V_STOP_VTRK</td>
</tr>
<tr>
<td></td>
<td>VREG_FND_POS</td>
</tr>
<tr>
<td></td>
<td>VREG_OFFSET</td>
</tr>
</tbody>
</table>
# Condition Handlers

## Contents

<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>CONDITION HANDLERS</th>
<th>6–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>CONDITION HANDLER OPERATIONS</td>
<td>6–3</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Global Condition Handlers</td>
<td>6–3</td>
</tr>
<tr>
<td>6.2</td>
<td>CONDITIONS</td>
<td>6–6</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Port_Id Conditions</td>
<td>6–7</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Relational Conditions</td>
<td>6–7</td>
</tr>
<tr>
<td>6.2.3</td>
<td>System and Program Event Conditions</td>
<td>6–8</td>
</tr>
<tr>
<td>6.3</td>
<td>ACTIONS</td>
<td>6–11</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Assignment Actions</td>
<td>6–11</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Motion Related Actions</td>
<td>6–13</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Routine Call Actions</td>
<td>6–13</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Miscellaneous Actions</td>
<td>6–14</td>
</tr>
</tbody>
</table>
The condition handler feature of the KAREL language allows a program to respond to external conditions more efficiently than conventional program control structures allow.

These condition handlers, also known as Global condition handlers, allow specified conditions to be monitored in parallel with normal program execution and, if the conditions occur, corresponding actions to be taken in response.

For a condition handler to be monitored, it must be defined first and then enabled. Disabling a condition handler removes it from the group being scanned. Purging condition handlers deletes their definition.

Table 6–1 lists the conditions that can be monitored by condition handlers.

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_id[n]</td>
<td>ERROR[n]</td>
</tr>
<tr>
<td>NOT port_id[n]</td>
<td>EVENT[n]</td>
</tr>
<tr>
<td>port_id[n]+</td>
<td>ABORT</td>
</tr>
<tr>
<td>port_id[n]-</td>
<td>PAUSE</td>
</tr>
<tr>
<td>operand = operand</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>operand &lt;&gt; operand</td>
<td>SEMAPHORE[n]</td>
</tr>
<tr>
<td>operand &lt; operand</td>
<td></td>
</tr>
<tr>
<td>operand &lt;= operand</td>
<td></td>
</tr>
<tr>
<td>operand &gt; operand</td>
<td></td>
</tr>
<tr>
<td>operand &gt;= operand</td>
<td></td>
</tr>
</tbody>
</table>

Table 6–2 lists the actions that can be taken.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable = expression</td>
<td>NOABORT</td>
</tr>
<tr>
<td>port_id[n] = expression</td>
<td>NOMESSAGE</td>
</tr>
<tr>
<td>STOP</td>
<td>NOPAUSE</td>
</tr>
<tr>
<td>CANCEL</td>
<td>ENABLE CONDITION[n]</td>
</tr>
<tr>
<td>RESUME</td>
<td>DISABLE CONDITION[n]</td>
</tr>
<tr>
<td>HOLD</td>
<td>PULSE DOUT[n] FOR t</td>
</tr>
<tr>
<td>UNHOLD</td>
<td>UNPAUSE</td>
</tr>
<tr>
<td>routine_name</td>
<td>ABORT</td>
</tr>
<tr>
<td>SIGNAL EVENT[n]</td>
<td>CONTINUE</td>
</tr>
</tbody>
</table>
6. CONDITION HANDLERS

Table 6–2. Actions (Cont’d)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PAUSE</td>
</tr>
<tr>
<td></td>
<td>SIGNAL SEMAPHORE[n]</td>
</tr>
</tbody>
</table>

### 6.1 CONDITION HANDLE OPERATIONS

Table 6–3 summarizes condition handler operations.

Table 6–3. Condition Handler Operations

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>GLOBAL CONDITION HANDLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define</td>
<td>CONDITION[n]:&lt;WITH $SCAN_TIME = n&gt;. WHEN conditions DO actions ENDCONDITION</td>
</tr>
<tr>
<td>Enable</td>
<td>ENABLE CONDITION[n] (statement or action)</td>
</tr>
<tr>
<td>Disable</td>
<td>DISABLE CONDITION[n] (statement or action) or conditions satisfied</td>
</tr>
<tr>
<td>Purge</td>
<td>PURGE CONDITION[n] (statement), program terminated</td>
</tr>
</tbody>
</table>

#### 6.1.1 Global Condition Handlers

Global condition handlers are defined by executing a CONDITION statement in the executable section of a program. The definition specifies conditions/actions pairs. The following rules apply to global condition handlers.

- Each global condition handler is referenced throughout the program by a specified number, from 1 to 1000. If a condition handler with the specified number was previously defined, it must be purged before it is replaced by the new one.
- The conditions/actions pairs of a global condition handler are specified in the WHEN clauses of a CONDITION statement. All WHEN clauses for a condition handler are enabled, disabled, and purged together.
- The condition list represents a list of conditions to be monitored when the condition handler is scanned.
- By default, each global condition handler is scanned at a rate based on the value of $SCR.$cond_time. If the “WITH $SCAN_TIME = n” clause is used in a CONDITION statement, the condition will be scanned roughly every “n” milliseconds. The actual interval between the scans is determined as shown in Table 6–4.
6. CONDITION HANDLERS

Table 6–4. Interval Between Global Condition Handler Scans

<table>
<thead>
<tr>
<th>&quot;n&quot;</th>
<th>Interval Between Scans</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \leq $COND_TIME</td>
<td>$$COND_TIME$</td>
</tr>
<tr>
<td>$$COND_TIME &lt; n \leq (2 \times $COND_TIME)$</td>
<td>$(2 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(2 \times $COND_TIME) &lt; n \leq (4 \times $COND_TIME)$</td>
<td>$(4 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(4 \times $COND_TIME) &lt; n \leq (8 \times $COND_TIME)$</td>
<td>$(8 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(8 \times $COND_TIME) &lt; n \leq (16 \times $COND_TIME)$</td>
<td>$(16 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(16 \times $COND_TIME) &lt; n \leq (32 \times $COND_TIME)$</td>
<td>$(32 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(32 \times $COND_TIME) &lt; n \leq (64 \times $COND_TIME)$</td>
<td>$(64 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(64 \times $COND_TIME) &lt; n \leq (128 \times $COND_TIME)$</td>
<td>$(128 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(128 \times $COND_TIME) &lt; n \leq (256 \times $COND_TIME)$</td>
<td>$(256 \times $COND_TIME)$</td>
</tr>
<tr>
<td>$(256 \times $COND_TIME) &lt; n</td>
<td>$(512 \times $COND_TIME)$</td>
</tr>
</tbody>
</table>

- Multiple conditions must all be separated by the AND operator or the OR operator. Mixing of AND and OR is not allowed.
- If AND is used, all of the conditions of a single WHEN clause must be satisfied simultaneously for the condition handler to be triggered.
- If OR is used, the actions are triggered when any of the conditions are TRUE.
- The action list represents a list of actions to be taken when the corresponding conditions of the WHEN clause are simultaneously satisfied.
- Multiple actions must be separated by a comma or a new line.

**Global Condition Handler Definitions** shows three examples of defining global condition handlers.

**See Also:** $\$SCR.$\$cond\_time System Variable, *FANUC Robotics Software Reference Manual* $\$SCAN\_TIME Condition Handler Qualifier, *FANUC Robotics Software Reference Manual*

**Global Condition Handler Definitions**

```plaintext
CONDITION[1]:  --defines condition handler number 1
WHEN DIN[1] DO DOUT[1] = TRUE  --triggered if any one
ENDCONDITION

CONDITION[2]:  --defines condition handler number 2
WHEN PAUSE DO  --one condition triggers
  AOUT[speed_out] = 0  --multiple actions
  DOUT[pause_light] = TRUE
ENABLE CONDITION [2]  --enables this condition
```
You can enable, disable, and purge global condition handlers as needed throughout the program. Whenever a condition handler is triggered, it is automatically disabled, unless an ENABLE action is included in the action list. (See condition handler 2 in Global Condition Handler Definitions.)

- The ENABLE statement or action enables the specified condition handler. The condition handler will be scanned during the next scan operation and will continue to be scanned until it is disabled.
- The DISABLE statement or action removes the specified condition handler from the group of scanned condition handlers. The condition handler remains defined and can be enabled again with the ENABLE statement or action.
- The PURGE statement deletes the definition of the specified condition handler.

ENABLE, DISABLE, and PURGE have no effect if the specified condition handler is not defined. If the specified condition handler is already enabled, ENABLE has no effect; if it is already disabled, DISABLE has no effect.

Using Global Condition Handlers shows examples of enabling, disabling, and purging global condition handlers.

Using Global Condition Handlers

CONDITION[1]: --defines condition handler number 1
  WHEN line_stop = TRUE DO DOUT[1] = FALSE
ENDCONDITION
CONDITION[2]: --defines condition handler number 2
  WHEN line_go = TRUE DO
    DOUT[1] = TRUE, ENABLE CONDITION [1]
ENDCONDITION
ENABLE CONDITION[2] --condition handler 2 is enabled
... IF ready THEN line_go = TRUE; ENDIF
--If ready is TRUE condition handler 2 is triggered (and --disabled) and condition handler 1 is enabled.
--Otherwise, condition handler 2 is not triggered (and is --still enabled), condition handler 1 is not yet enabled, --and the next two statements will have no effect.
DISABLE CONDITION[1]
ENABLE CONDITION[2]
ENABLE CONDITION[1] --condition handler 1 is enabled

line_stop = TRUE --triggers (and disables) condition handler 1

PURGE CONDITION[2] --definition of condition handler 2 deleted
ENABLE CONDITION[2] --no longer has any effect
line_go = TRUE --no longer a monitored condition

### 6.2 CONDITIONS

One or more conditions are specified in the condition list of a WHEN or UNTIL clause, defining the conditions portion of a conditions/actions pair. Conditions can be

- **States** - which remain satisfied as long as the state exists. Examples of states are DIN[1] and (VAR1 > VAR2).
- **Events** - which are satisfied only at the instant the event occurs. Examples of events are ERROR[n], DIN[n]+, and PAUSE.

The following rules apply to system and program event conditions:

- After a condition handler is enabled, the specified conditions are monitored.
  - If all of the conditions of an AND, WHEN, or UNTIL clause are simultaneously satisfied, the condition handler is triggered and corresponding actions are performed.
  - If all of the conditions of an OR, WHEN, or UNTIL clause are satisfied, the condition handler is triggered and corresponding actions are performed.
- Event conditions very rarely occur simultaneously. Therefore, you should never use AND between two event conditions in a single WHEN or UNTIL clause because, both conditions will not be satisfied simultaneously.
- While many conditions are similar in form to BOOLEAN expressions in KAREL, and are similar in meaning, only the forms listed in this section, not general BOOLEAN expressions, are permitted.
- Expressions are permitted within an EVAL clause. More general expressions may be used on the right side of comparison conditions, by enclosing the expression in an EVAL clause: EVAL (expression). However, expressions in an EVAL clause are evaluated when the condition handler is defined. They are not evaluated dynamically.
- The value of an EVAL clause expression must be INTEGER, REAL, or BOOLEAN.

See Also: EVAL Clause, Appendix A.
6. CONDITION HANDLERS

6.2.1 Port_Id Conditions

Port_id conditions are used to monitor digital port signals. Port_id must be one of the predefined BOOLEAN port array identifiers (DIN, DOUT, OPIN, OPOUT, TPIN, TPOUT, RDI, RDO, WDI, or WDO). The value of n specifies the port array signal to be monitored. Table 6–5 lists the available port_id conditions.

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>SATISFIED (TRUE) WHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_id[n]</td>
<td>Digital port n is TRUE. (state)</td>
</tr>
<tr>
<td>NOT port_id[n]</td>
<td>Digital port n is FALSE. (state)</td>
</tr>
<tr>
<td>port_id[n]+</td>
<td>Digital port n changes from FALSE to TRUE. (event)</td>
</tr>
<tr>
<td>port_id[n]-</td>
<td>Digital port n changes from TRUE to FALSE. (event)</td>
</tr>
</tbody>
</table>

• For the state conditions, port_id[n] and NOT port_id[n], the port is tested during every scan. The following conditions would be satisfied if, during a scan, DIN[1] was TRUE and DIN[2] was FALSE:


  Note that an input signal should remain ON or OFF for the minimum scan time to ensure that its state is detected.

• For the event condition port_id[n]+, the initial port value is tested when the condition handler is enabled. Each scan tests for the specified change in the signal. The change must occur while the condition handler is enabled.

  The following condition would only be satisfied if, while the condition handler was enabled, DIN[1] changed from TRUE to FALSE since the last scan.

  WHEN DIN[1]- DO . . .

6.2.2 Relational Conditions

Relational conditions are used to test the relationship between two operands. They are satisfied when the specified relationship is TRUE. Relational conditions are state conditions, meaning the relationship is tested during every scan. Table 6–6 lists the relational conditions.
### Table 6–6. Relational Conditions

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>SATISFIED (TRUE) WHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>operand = operand</td>
<td>Relationship specified is TRUE. Operands on the left can be a port array element, referenced as port_id[n], or a variable. Operands on the right can be a variable, a constant, or an EVAL clause. (state)</td>
</tr>
<tr>
<td>operand &lt; &gt; operand</td>
<td></td>
</tr>
<tr>
<td>operand &lt; operand</td>
<td></td>
</tr>
<tr>
<td>operand &lt;= operand</td>
<td></td>
</tr>
<tr>
<td>operand &gt; operand</td>
<td></td>
</tr>
<tr>
<td>operand &gt;= operand</td>
<td></td>
</tr>
</tbody>
</table>

The following rules apply to relational conditions:

- Both operands must be of the same data type and can only be of type INTEGER, REAL, or BOOLEAN. (As in other situations, INTEGER constants can be used where REAL values are required, and will be treated as REAL values.)
- The operand on the left side of the condition can be any of the port array signals, a user-defined variable, a static variable, or a system variable that can be read by a KAREL program.
- The operand on the right side of the condition can be a user-defined variable, a static variable, a system variable that can be read by a KAREL program, any constant, or an EVAL clause. For example:

  WHEN DIN[1] = ON DO ... --port_id and constant
  WHEN flag = TRUE DO ... --variable and constant
  WHEN AIN[1] >= temp DO ... --port_id and variable
  WHEN flag_1 <> flag_2 DO ... --variable and variable
  WHEN AIN[1] <= EVAL(temp * scale) DO ... --port_id and EVAL clause
  WHEN dif > EVAL(max_count - count) DO ... --variable and EVAL clause

- The EVAL clause allows you to include expressions in relational conditions. However, it is evaluated only when the condition handler is defined. The expression in the EVAL clause cannot include any routine calls.

See Also: EVAL Clause, Appendix A.

### 6.2.3 System and Program Event Conditions

System and program event conditions are used to monitor system and program generated events. The specified condition is satisfied only if the event occurs when the condition handler is enabled.
Enabled condition handlers containing ERROR, EVENT, PAUSE, ABORT, POWERUP, or CONTINUE conditions are scanned only if the specified type of event occurs. For example, an enabled condition handler containing an ERROR condition will be scanned only when an error occurs. Table 6–7 lists the available system and program event conditions.

### Table 6–7. System and Program Event Conditions

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>SATISFIED (TRUE) WHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR [n]</td>
<td>The error specified by n is reached or, if n = *, any error occurs. (event)</td>
</tr>
<tr>
<td>EVENT[n]</td>
<td>The event specified by n is signaled. (event)</td>
</tr>
<tr>
<td>ABORT</td>
<td>The program is aborted. (event)</td>
</tr>
<tr>
<td>PAUSE</td>
<td>The program is paused. (event)</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>The program is continued. (event)</td>
</tr>
<tr>
<td>POWERUP</td>
<td>The program is continued. (event)</td>
</tr>
<tr>
<td>SEMAPHORE[n]</td>
<td>The value of the semaphore specified by n is posted.</td>
</tr>
</tbody>
</table>

The following rules apply to these conditions:

**ERROR Condition**

- The ERROR condition can be used to monitor the occurrence of a particular error by specifying the error code for that error. For example, ERROR[15018] monitors the occurrence of the error represented by the error code 15018.

  The error codes are listed in the following format:

  \[ ffccc \text{ (decimal)} \]

  where

  \( ff \) represents the facility code of the error
  \( ccc \) represents the error code within the specified facility

  For example, 15018 is MOTN-018, which is "Position not reachable." The facility code is 15 and the error code is 018. Refer to the *FANUC Robotics Error Code Manual* for a complete listing of error codes.

- The ERROR condition can also be used to monitor the occurrence of any error by specifying an asterisk (*), the wildcard character, in place of a specific error code. For example, ERROR[*] monitors the occurrence of any error.

- The ERROR condition is satisfied only for the scan performed when the error was detected. The error is not remembered in subsequent scans.
EVENT Condition

- The EVENT condition monitors the occurrence of the specified program event. The SIGNAL statement or action in a program indicates that an event has occurred.
- The EVENT condition is satisfied only for the scan performed when the event was signaled. The event is not remembered in subsequent scans.

ABORT Condition

- The ABORT condition monitors the aborting of program execution. If an ABORT occurs, the corresponding actions are performed. However, if one of the actions is a routine call, the routine will not be executed because program execution has been aborted.

If an ABORT condition is used in a condition handler all actions, except routine calls, will be performed even though the program has aborted.

PAUSE Condition

- The PAUSE condition monitors the pausing of program execution. If one of the corresponding actions is a routine call, it is also necessary to specify a NOPAUSE or UNPAUSE action.

CONTINUE Condition

- The CONTINUE condition monitors the resumption of program execution. If program execution is paused, the CONTINUE action, the KCL> CONTINUE command, a CYCLE START from the operator panel, or the teach pendant FWD key will continue program execution and satisfy the CONTINUE condition.

POWERUP Condition

- The POWERUP condition monitors the resumption of program execution after a power failure recovery. The controller must be able to recover successfully from a power failure before the program can be resumed.

SEMAPHORE Condition

- The SEMAPHORE condition monitors the specified semaphore. The CLEAR_SEMA built-in can be used to set the semaphore value to 0. The POST_SEMA built-in or the SIGNAL SEMAPHORE action can be used to increment the semaphore value and satisfy the SEMAPHORE condition.

See Also: In Appendix A:

ABORT Condition
CONTINUE Condition
ERROR Condition
EVENT Condition
PAUSE Condition
POWERUP Condition
SEMAPHORE Condition


6.3 ACTIONS

Actions are specified in the action list of a WHEN clause. Actions can be

• Specially defined KAREL actions that are executed in parallel with the program
• A routine call, which will interrupt program execution

When the conditions of a condition handler are satisfied, the condition handler is triggered. The actions corresponding to the satisfied conditions are performed in the sequence in which they appear in the condition handler definition, except for routine calls. Routines are executed after all of the other actions have been performed.

Note that, although many of the actions are similar in form to KAREL statements and the effects are similar to corresponding KAREL statements, the actions are not executable statements. Only the forms indicated in this section are permitted.

See Also: Actions and Statements, Appendix A.

6.3.1 Assignment Actions

The available assignment actions are given in Table 6–8.
The following rules apply to assignment actions:

- The assignment actions, “variable = expression” and “port_id[n] = expression” can be used to assign values to variables and port array elements.
  - The variable must be either a user-defined variable, a static variable, or a system variable without a minimum/maximum range and that can be written to by a KAREL program.
  - The port array, if on the left, must be an output port array that can be set by a KAREL program.
  - The expression can be a user-defined variable, a static variable, a system variable that can be read by a KAREL program, any constant, or an EVAL clause.
- If a variable is on the left side of the assignment, the expression can also be a port array element. However, you cannot assign a port array element to a port array element directly. For example, the first assignment shown is invalid, but the next two are valid:

\[
\begin{align*}
DOUT[1] &= DOUT[2] \quad \text{--invalid action} \\
\text{port_var} &= DOUT[2] \quad \text{--valid action, where port_var is a variable} \\
DOUT[1] &= \text{port_var} \quad \text{--another valid action, which if executed} \\
& \quad \text{--after port_var = DOUT[2], would in effect} \\
& \quad \text{--assign DOUT[2] to DOUT[1]} \\
\end{align*}
\]

- If the expression is a variable, it must be a global 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 the expression is an EVAL clause, it is evaluated when the condition handler is defined and that value is assigned when the action is taken.
- Both sides of the assignment action must be of the same data type. An INTEGER or EVAL clause is permitted on the right side of the assignment with an INTEGER, REAL, or BOOLEAN on the left.
6.3.2 Motion Related Actions

Motion related actions affect the current motion and might affect subsequent motions. They are given in Table 6–9.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>RESULT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td>Current motion is stopped.</td>
<td></td>
</tr>
<tr>
<td>RESUME</td>
<td>The last stopped motion is resumed.</td>
<td></td>
</tr>
<tr>
<td>CANCEL</td>
<td>Current motion is canceled.</td>
<td></td>
</tr>
<tr>
<td>HOLD</td>
<td>Current motion is held. Subsequent motions are not started.</td>
<td></td>
</tr>
<tr>
<td>UNHOLD</td>
<td>Held motion is released.</td>
<td></td>
</tr>
</tbody>
</table>

The following rules apply to motion related actions:

- If a STOP is issued, the current motion and any queued motions are pushed as a set on a stopped motion stack. If no motion is in progress, an empty entry is pushed on the stack.
- If a RESUME is issued, the newest stopped motion set on the stopped motion stack is queued for execution.
- If a CANCEL is issued, the motion currently in progress is canceled. Any motions queued to the same group behind the current motion are also canceled. If no motion is in progress, the action has no effect.
- If a HOLD is issued, the current motion is held and subsequent motions are prevented from starting. The UNHOLD action releases held motion.

6.3.3 Routine Call Actions

Routine call actions, or interrupt routines, are specified by

\(<\text{WITH } $\text{PRIORITY} = n> \text{ routine\_name} \>\)

The following restrictions apply to routine call actions or interrupt routines:

- The interrupt routine cannot have parameters and must be a procedure (not a function).
- If the interrupted program is using READ statements, the interrupt routine cannot read from the same file variable. If an interrupted program is reading and the interrupt routine attempts a read from the same file variable, the program is aborted.
- When an interrupt routine is started, the interrupted KAREL program is suspended until the routine returns.
Interrupt routines, like KAREL programs, can be interrupted by other routines. The maximum depth of interruption is limited only by stack memory size.

Routines are started in the sequence in which they appear in the condition handler definition, but since they interrupt each other, they will actually execute in reverse order.

Interrupts can be prioritized so that certain interrupt routines cannot be interrupted by others. The \$PRIORITY condition handler qualifier can be used to set the priority of execution for an indicated routine action. \$PRIORITY values must be 0-255 where the lower value represents a lower priority. If a low priority routine is called while a routine with a higher priority is running, it will be executed only when the higher priority routine has completed. If \$PRIORITY is not specified, the routine’s priority will default to the current value of the \$PRIORITY system variable.

See Also: WITH Clause, Appendix A, “KAREL Language Alphabetical Description,” for more information on \$PRIORITY

### 6.3.4 Miscellaneous Actions

Table 6–10 describes other allowable actions.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL EVENT[n]</td>
<td>The event specified by n is signaled.</td>
</tr>
<tr>
<td>NOMESSAGE</td>
<td>The error message that otherwise would have been generated is not displayed or logged.</td>
</tr>
<tr>
<td>NOPAUSE</td>
<td>Program execution is resumed if the program was paused, or is prevented from pausing.</td>
</tr>
<tr>
<td>NOABORT</td>
<td>Program execution is resumed if the program was aborted, or is prevented from aborting.</td>
</tr>
<tr>
<td>ABORT</td>
<td>Program execution is aborted.</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Program execution is continued.</td>
</tr>
<tr>
<td>PAUSE</td>
<td>Program execution is paused.</td>
</tr>
<tr>
<td>SIGNAL SEMAPHORE[n]</td>
<td>Specified semaphore is signaled.</td>
</tr>
<tr>
<td>ENABLE CONDITION[n]</td>
<td>Condition handler n is enabled.</td>
</tr>
<tr>
<td>DISABLE CONDITION[n]</td>
<td>Condition handler n is disabled.</td>
</tr>
<tr>
<td>PULSE DOUT[n] FOR t</td>
<td>Specified port n is pulsed for the time interval t (in milliseconds).</td>
</tr>
<tr>
<td>UNPAUSE</td>
<td>If a routine name is specified as an action, but program execution is paused, execution is resumed only for the duration of the routine and then is paused again.</td>
</tr>
</tbody>
</table>
See Also: Appendix A for more information on each miscellaneous action.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.10</td>
<td>USER INTERFACE TIPS</td>
<td>7–38</td>
</tr>
<tr>
<td>7.10.1</td>
<td>USER Menu on the Teach Pendant</td>
<td>7–38</td>
</tr>
<tr>
<td>7.10.2</td>
<td>USER Menu on the CRT/KB</td>
<td>7–40</td>
</tr>
</tbody>
</table>
The KAREL language facilities allow you to perform the following serial input/output (I/O) operations:

- Open data files and serial communication ports using the OPEN FILE Statement
- Close data files and serial communication ports using the CLOSE FILE Statement
- Read from files, communication ports, and user interface devices using the READ Statement
- Write to files, communication ports, and user interface devices using the WRITE Statement
- Cancel read or write operations

File variables are used to indicate the file, communication port, or device on which a serial I/O operation is to be performed.

Buffers are used to hold data that has not yet been transmitted. The use of data items in READ and WRITE statements and their format specifiers depend on whether the data is text (ASCII) or binary, and on the data type.

### 7.2 FILE VARIABLES

A KAREL program can perform serial I/O operations on the following:

- Data files residing in the KAREL file system
- Serial communication ports associated with connectors on the KAREL controller
- User interface devices including the CRT/KB and teach pendant

A file variable is used to indicate the file, communication port, or device on which you want to perform a particular serial I/O operation.

Table 7–1 lists the predefined file variables for user interface devices. These file variables are already opened and can be used in the READ or WRITE statements.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DEVICE</th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPFUNC*</td>
<td>Teach pendant function key line</td>
<td>Both</td>
</tr>
<tr>
<td>TPDISPLAY*</td>
<td>Teach pendant KAREL display</td>
<td>Both</td>
</tr>
<tr>
<td>TPPROMPT*</td>
<td>Teach pendant prompt line</td>
<td>Both</td>
</tr>
<tr>
<td>TPERROR</td>
<td>Teach pendant message line</td>
<td>Write</td>
</tr>
<tr>
<td>TPSTATUS*</td>
<td>Teach pendant status line</td>
<td>Write</td>
</tr>
</tbody>
</table>
7. FILE INPUT/OUTPUT OPERATIONS

Table 7–1. Predefined File Variables (Cont’d)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRTFUNC*</td>
<td>CRT/KB function key line</td>
<td>Both</td>
</tr>
<tr>
<td>INPUT</td>
<td>CRT/KB keyboard</td>
<td>Read</td>
</tr>
<tr>
<td>OUTPUT*</td>
<td>CRT/KB KAREL screen</td>
<td>Write</td>
</tr>
<tr>
<td>CRTPROMPT*</td>
<td>CRT/KB prompt line</td>
<td>Both</td>
</tr>
<tr>
<td>CRTEXCEPTION</td>
<td>CRT/KB message line</td>
<td>Write</td>
</tr>
<tr>
<td>CRTSTATUS*</td>
<td>CRT/KB status line</td>
<td>Write</td>
</tr>
</tbody>
</table>

* Only displayed when teach pendant or CRT is in the user menu.

A file variable can be specified in a KAREL statement as a FILE variable. Using FILE in a KAREL Program shows an example of declaring a FILE variable and of using FILE in the executable section of a program.

Using FILE in a KAREL Program

```karel
PROGRAM lun_prog
  VAR
    currnt_file : FILE
  ROUTINE input_data(file_spec:FILE) FROM util_prog
  BEGIN
    OPEN FILE currnt_file ('RW','text.dt') --variable FILE
    input_data(currnt_file) --file variable argument
    WRITE TPERROR ('Error has occurred')
  END lun_prog
```

Sharing FILE variables between programs is allowed as long as a single task is executing the programs. Sharing file variables between tasks is not allowed.

7.3 OPEN FILE STATEMENT

The OPEN FILE statement associates the file variable with a particular data file or communication port.

The association remains in effect until the file is closed, either explicitly by a CLOSE FILE statement or implicitly when program execution terminates or is aborted.

The OPEN FILE statement specifies how the file is to be used (usage string), and which file or port (file string) is used.
7. FILE INPUT/OUTPUT OPERATIONS

7.3.1 Setting File and Port Attributes

Attributes specify the details of operation of a serial port, or KAREL FILE variable. The SET_PORT_ATR and SET_FILE_ATR built-ins are used to set these attributes. SET_FILE_ATR must be called before the FILE is opened. SET_PORT_ATR can be called before or after the FILE that is using a serial port, is opened.

Table 7–2 lists each attribute type, its function and whether the attribute is intended for use with teach pendant and CRT/KB devices, serial ports, data files, or pipes. Refer to Appendix A for more information.

### Table 7–2. Predefined Attribute Types

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>FUNCTION</th>
<th>SET_PORT_ATR OR SET_FILE_ATR</th>
<th>TP/ CRT</th>
<th>SERIAL PORTS</th>
<th>DATA FILES</th>
<th>PIPES</th>
<th>SOCKET MESSAGING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR_BAUD</td>
<td>Baud rate</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_DBITS</td>
<td>Data length</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_EOL</td>
<td>End of line</td>
<td>SET_FILE_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_FIELD</td>
<td>Field</td>
<td>SET_FILE_ATR</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_IA</td>
<td>Interactively write</td>
<td>SET_FILE_ATR</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_MODEM</td>
<td>Modem line</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_PARITY</td>
<td>Parity</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_PASSALL</td>
<td>Passall</td>
<td>SET_FILE_ATR</td>
<td>valid</td>
<td>valid</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_READAHD</td>
<td>Read ahead buffer</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_REVERSE</td>
<td>Reverse transfer</td>
<td>SET_FILE_ATR</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_SBITS</td>
<td>Stop bits</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_TIMEOUT</td>
<td>Timeout</td>
<td>SET_FILE_ATR</td>
<td>valid</td>
<td>valid</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_UF</td>
<td>Unformatted transfer</td>
<td>SET_FILE_ATR</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
</tbody>
</table>
Table 7–2. Predefined Attribute Types (Cont’d)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>FUNCTION</th>
<th>SET_PORT_ATR OR SET_FILE_ATR</th>
<th>TP/CRT</th>
<th>SERIAL PORTS</th>
<th>DATA FILES</th>
<th>PIPES</th>
<th>SOCKET MESSAGING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR_XONOFF</td>
<td>XON/XOFF</td>
<td>SET_PORT_ATR</td>
<td>not used</td>
<td>valid</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
</tr>
<tr>
<td>ATR_PIPOVADV</td>
<td>Pipe Overflow</td>
<td>SET_FILE_ATR</td>
<td>not used</td>
<td>not used</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>ATR_PIPWAIT</td>
<td>Wait for data</td>
<td>SET_FILE_ATR</td>
<td>not used</td>
<td>not used</td>
<td>valid</td>
<td>valid</td>
<td></td>
</tr>
</tbody>
</table>

Table 7–3 contains detailed explanations of each attribute.

Table 7–3. Attribute Values

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Description</th>
<th>Valid Device</th>
<th>Usage Mode</th>
<th>Valid Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR_BAUD</td>
<td>Baud rate</td>
<td>PORT</td>
<td>Read/Write</td>
<td>BAUD_9600: 9600 baud, BAUD_4800: 4800 baud, BAUD_2400: 2400 baud, BAUD_1200: 1200 baud</td>
<td>BAUD_9600</td>
</tr>
<tr>
<td>ATR_DBITS</td>
<td>Data length</td>
<td>PORT</td>
<td>Read/Write</td>
<td>DBITS_5: 5 bits, DBITS_6: 6 bits, DBITS_7: 7 bits, DBITS_8: 8 bits</td>
<td>DBITS_8</td>
</tr>
<tr>
<td>ATR_EOL</td>
<td>End of line</td>
<td>PORT</td>
<td>Read/Write</td>
<td>Any ASCII character code</td>
<td>13 (carriage return)</td>
</tr>
<tr>
<td>ATR_FIELD</td>
<td>Field</td>
<td>TP/CRT, PORT, FILE</td>
<td>Read only</td>
<td>Ignored</td>
<td></td>
</tr>
</tbody>
</table>

7–6
### Table 7–3. Attribute Values (Cont’d)

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Description</th>
<th>Valid Device</th>
<th>Usage Mode</th>
<th>Valid Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR_IA</td>
<td>If specified, the contents of the buffer are output when each write operation to the buffer is complete. (Interactive) If not specified, the contents of the buffer are output only when the buffer becomes full or when CR is specified. The size of the output buffer is 256 bytes. (Not interactive)</td>
<td>TP/CRT, PORT, FILE</td>
<td>Write only</td>
<td>Ignored</td>
<td>TP/CRT is interactive, PORT, FILE are not interactive</td>
</tr>
<tr>
<td>ATR_MODEM</td>
<td>Modem line</td>
<td>Refer to &quot;Modem Line&quot; section that follows for information.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATR_PARITY</td>
<td>Parity</td>
<td>The parity for a serial port can be changed to one of the valid attribute values.</td>
<td>PORT</td>
<td>Read/ Write</td>
<td>PARITY_NONE: No parity, PARITY_ODD: Odd parity, PARITY_EVEN: Even parity</td>
</tr>
<tr>
<td>ATR_PASSALL</td>
<td>Passall</td>
<td>If specified, input is read without interpretation or transaction. Since the terminator character (EOL) will not terminate the read, the field attribute automatically assumes the &quot;field&quot; option.</td>
<td>TP/CRT, PORT</td>
<td>Read only</td>
<td>Ignored</td>
</tr>
<tr>
<td>ATR_PIPOVADV</td>
<td>Configures the behavior of the read when an overflow occurs. By default the behavior is to signal an end of file (EOF) when the overflow occurs.</td>
<td>PIPE</td>
<td>Read</td>
<td>The value must be between 0 and the total number of bytes in the pipe. The value will be rounded up to the nearest binary record.</td>
<td>The value parameter is either OVF_EOF (sets the default behavior) or the number of bytes to advance when an overflow occurs.</td>
</tr>
</tbody>
</table>
Table 7-3. Attribute Values (Cont’d)

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Description</th>
<th>Valid Device</th>
<th>Usage Mode</th>
<th>Valid Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR_PIPWAIT</td>
<td>The read operation waits for data to arrive in the pipe.</td>
<td>PIPE</td>
<td>Read</td>
<td>WAIT_USED or WAIT_NOTUSED</td>
<td>The default is snapshot which means that the system returns an EOF when all the data in the pipe has been read.</td>
</tr>
<tr>
<td>ATR_READAHD</td>
<td>The attribute value is specified in units of 128 bytes, and allocates a read ahead buffer of the indicated size.</td>
<td>PORT</td>
<td>Read/Write</td>
<td>any positive integer</td>
<td>1 (128 byte buffer)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=128 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=256 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=disable bytes</td>
<td></td>
</tr>
<tr>
<td>ATR_REVERSE</td>
<td>The bytes will be swapped.</td>
<td>PORT, FILE</td>
<td>Read/Write</td>
<td>Ignored</td>
<td>Not reverse transfer</td>
</tr>
<tr>
<td>ATR_SBITS</td>
<td>This specifies the number of stop bits for the serial port.</td>
<td>PORT</td>
<td>Read/Write</td>
<td>SBITS_1:1 bit</td>
<td>SBITS_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SBITS_15: 1.5 bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SBITS_2:2 bits</td>
<td></td>
</tr>
<tr>
<td>ATR_TIMEOUT</td>
<td>If specified, an error will be returned by IO_STATUS if the read takes longer than the specified attribute value.</td>
<td>TP/CRT, PORT</td>
<td>Read only</td>
<td>Any integer value (units are in msec)</td>
<td>0 (external)</td>
</tr>
<tr>
<td>ATR_UF</td>
<td>If specified, a binary transfer is performed. For read operations, the terminator character (EOL) will not terminate the read, and therefore automatically assumes the “field” option. If not specified, ASCII transfer is performed.</td>
<td>PORT, FILE</td>
<td>Read/Write</td>
<td>Ignored</td>
<td>ASCII transfer</td>
</tr>
<tr>
<td>ATR_XONOFF</td>
<td>If specified, the XON/XOFF for a serial port is changed to the specified attribute value.</td>
<td>PORT</td>
<td>Read/Write</td>
<td>XF_NOT_USED: Not used</td>
<td>XF_USED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XF_USED: Used</td>
<td></td>
</tr>
</tbody>
</table>
Modem line

Valid device: PORT

Usage mode: Read/Write

Default value: MD_NOT_USED: DSR, DTR, and RTS not used

Valid attribute values:
- MD_NOT_USED: DSR, DTR, and RTS not used
- MD_USE_DSR: DSR used
- MD_NOUSE_DSR: DSR not used
- MD_USE_DTR: DTR used
- MD_NOUSE_DTR: DTR not used
- MD_USE_RTS: RTS used
- MD_NOUSE_RTS: RTS not used

- This attribute controls the operation of the modem line. The control is based on the following binary mask, where the flag bits are used to indicate what bit value you are changing.

<table>
<thead>
<tr>
<th>RTS value</th>
<th>DSR value</th>
<th>DTR value</th>
<th>RTS flag</th>
<th>DSR flag</th>
<th>DTR flag</th>
</tr>
</thead>
</table>
- RTS (request to send) and DTR (data terminal ready) are both outputs.
- DSR (data set ready) is an input.

- Set the modem line attribute by doing the following.
  - To indicate RTS is used (HIGH/ON): status = SET_PORT_ATR (port_name, ATR_MODEM, MD_USE_RTS)
  - To indicate RTS is NOT used (LOW/OFF): status = SET_PORT_ATR (port_name, ATR_MODEM, MD_NOUSE_RTS)
  - To indicate RTS is used (HIGH/ON) and DTR is not used (LOW/OFF): status = SET_PORT_ATR (port_name, ATR_MODEM, MD_USE_RTS or MD_NOUSE_DTR)

- The following examples demonstrate how to use the returned attribute value from the GET_PORT_ATR built-in.

  status = GET_PORT_ATR (port, ATR_MODEM, atr_value)

  - To determine if DTR is used:

    IF ((atr_value AND MD_USE_DTR) = MD_USE_DTR) THEN
write ('DTR is in use', cr)
ENDIF

— To determine if DTR is not used (LOW/OFF)

IF (atr_value AND MD_USE_DTR) = MD_NOUSE_DTR) THEN
    write ('DTR is not in use', cr)
ENDIF

For more information on GET_PORT_ATR Built-in, refer to Appendix A.

7.3.2 File String

The file string in an OPEN FILE statement specifies a data file name and type, or a communication port.

- The OPEN FILE statement associates the data file or port specified by the file string with the file variable. For example, OPEN FILE file_var ('RO', 'data_file.dt') associates the data file called 'data_file.dt' with the file file_var.
- If the file string is enclosed in single quotes, it is treated as a literal. Otherwise, it is treated as a STRING variable or constant identifier.
- When specifying a data file, you must include both a file name and a valid KAREL file type (any 1, 2, or 3 character file extension).
- The following STRING values can be used to associate file variables with serial communication ports on the KAREL controller. Defaults for are:
  - 'P2:' - Debug console connector on the outside of the operator panel
  - 'P3:' - RS-232-C, JD17 connector on the Main CPU board (CRT/KB)
  - 'P4:' - RS-422, JD17 connector on the Main CPU board
  - 'KB:tp kb' - Input from numeric keypad on the teach pendant. TPDISPLAY or TPPROMPT are generally used, so OPEN FILE is not required.
  - 'KB:cr kb' - Input from CRT/KB. INPUT or CRTPROMPT are generally used, so OPEN FILE is not required.
  - 'WD:window_name' - Writes to a window.
  - 'WD:window_name</keyboard_name>' , where keyboard_name is either 'tpkb' or 'crkb' - Writes to the specified window. Inputs are from the TP keypad (tpkb) or the CRT keyboard (crkb). Inputs will be echoed in the specified window.
See Also: Chapter 9 FILE SYSTEM, for a description of file names and file types.

7.3.3 Usage String

The usage string in an OPEN FILE statement indicates how the file is to be used.

- It is composed of one usage specifier.
- It applies only to the file specified by the OPEN FILE statement and has no effect on other FILEs.
- It must be enclosed in single quotes if it is expressed as a literal.
- It can be expressed as a variable or a constant.

Table 7–4 lists each usage specifier, its function, and the devices or ports for which it is intended.

- “TP/CRT” indicates teach pendant and CRT/KB.
- “Ports” indicates serial ports.
- “Files” indicates data files.
- “Pipes” indicates pipe devices.
- “Valid” indicates a permissible use.
- “No use” indicates a permissible use that might have unpredictable side effects.

Table 7–4. Usage Specifiers

<table>
<thead>
<tr>
<th>SPECIFIER</th>
<th>FUNCTION</th>
<th>TP/CRT</th>
<th>PORTS</th>
<th>FILES</th>
<th>PIPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO</td>
<td>— Permits only read operations</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td></td>
<td>— Sets file position to beginning of file</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— File must already exist</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RW</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>— Rewrites over existing data in a file, deleting existing data</td>
<td></td>
<td></td>
<td>No use on FRx:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Permits read and write operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Sets file position to beginning of file</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— File will be created if it does not exist</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AP</td>
<td>no use</td>
<td>valid</td>
<td>valid -RAM disk* no use on FRx:</td>
<td>valid</td>
</tr>
<tr>
<td></td>
<td>— Appends to end of existing data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Permits read and write (First operation must be a write.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Sets file position to end of file</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— File will be created if it does not exist</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UD</td>
<td>no use</td>
<td>valid</td>
<td>valid -RAM disk* no use on FRx:</td>
<td>no use</td>
</tr>
<tr>
<td></td>
<td>— Updates from beginning of existing data. (Number of characters to be written must equal number of characters to be replaced.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Overwrites the existing data with the new data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Permits read and write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Sets file position to beginning of existing file</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
* AP and UD specifiers can only be used with uncompressed files on the RAM disk. Refer to Chapter 9 FILE SYSTEM, for more information on the RAM disk and Pipe devices.

File String Examples shows a program that includes examples of various file strings in OPEN FILE statements. The CONST and VAR sections are included to illustrate how file and port strings are declared.

**File String Examples**

```plaintext
PROGRAM open_luns
    CONST
        part_file_c = 'parts.dt' -- data file STRING constant
        comm_port = 'P3:' -- port STRING constant
    VAR
        file_var1 : FILE
        file_var2 : FILE
        file_var3 : FILE
        file_var4 : FILE
        file_var5 : FILE
        file_var12 : FILE
        temp_file : STRING[19]
            -- a STRING size of 19 accommodates 4 character device names,
            -- 12 character file names, the period, and 2 character,
            -- file types.
        port_var : STRING[3]
    BEGIN
        -- literal file name and type
        OPEN FILE file_var1 ('RO', 'log_file.dt')
            -- constant specifying parts.dt
        OPEN FILE file_var2 ('RW', part_file_c)
            -- variable specifying new_file.dt
        temp_file = 'RD:new_file.dt'
        OPEN FILE file_var3 ('AP', temp_file)
            -- literal communication port
        OPEN FILE file_var4 ('RW', 'P2:')
            -- constant specifying C0:
        OPEN FILE file_var5 ('RW', comm_port)
            -- variable specifying C3:
        port_var = 'C3:'
        OPEN FILE file_var12 ('RW', port_var)
    END open_luns
```

**See Also:** Chapter 9 FILE SYSTEM, for more information on the available storage devices

Chapter 14 INPUT/OUTPUT SYSTEM, for more information on the C0: and C3: ports
7.4 CLOSE FILE STATEMENT

The CLOSE FILE statement is used to break the association between a specified file variable and its data file or communication port. It accomplishes two objectives:

- Any buffered data is written to the file or port.
- The file variable is freed for another use.

CLOSE FILE Example shows a program that includes an example of using the CLOSE FILE statement in a FOR loop, where several files are opened, read, and then closed. The same file variable is used for each file.

CLOSE FILE Example

PROGRAM read_files
VAR
  file_var : FILE
  loop_count : INTEGER
  loop_file : STRING[15]
ROUTINE read_ops(file_spec:FILE) FROM util_prog
  --performs some read operations
ROUTINE get_names(names:ARRAY OF STRING) FROM util_prog
  --gets file names and types
BEGIN
  get_names(file_names)
  FOR loop_count = 1 TO 10 DO
    loop_file = file_names[loop_count]
    OPEN FILE file_var ('RO', loop_file)
    read_ops(file_var) --call routine for read operations
    CLOSE FILE file_var
  ENDFOR
END read_files

See Also: CLOSE FILE Statement, Appendix A.

IO_STATUS Built-In Function, Appendix A for a description of errors.

7.5 READ STATEMENT

The READ statement is used to read one or more specified data items from the indicated device. The data items are listed as part of the READ statement. The following rules apply to the READ statement:
• The OPEN FILE statement must be used to associate the file variable with the file opened in
the statement before any read operations can be performed unless one of the predefined files is
used (refer to Table 7–1).

• If the file variable is omitted from the READ statement, then TPDISPLAY is used as the default.

• Using the %CRTDEVICE directive will change the default to INPUT (CRT input window).

• Format specifiers can be used to control the amount of data that is 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.

• When the READ statement is executed (for ASCII files), 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.

• With STRING values, the input field begins with the next character and continues to the end of
the line or end of the file. If a STRING is read from the same line following a nonstring field, any
separating blanks are included in the STRING.

• ARRAY variables must be read element by element; they cannot be read in unsubscripted form.
Frequently, they are read using a READ statement in a FOR loop.

• PATH variables can be specified as follows in a READ statement, where “path_name” is a PATH
variable and “n” and “m” are PATH node indexes:

  — path_name : specifies that the entire path, starting with a header and including all of the
    nodes and their associated data, is to be read. The header consists of the path length and the
    associated data description in effect when the PATH was written.

  — path_name [0] : specifies that only the header is to be read. The path header consists of
    the path length and the associated data description in effect when the PATH was written.
    Nodes are deleted or created to make the path the correct length, and all new nodes are set
    uninitialized.

  — path_name [n] : specifies that data is to be read into node[n] from the current file position.
    The value of n must be in the range from 0 to the length of the PATH.

  — path_name [n .. m] : specifies that data is to be read into nodes n through m. The value of n
    must be in the range from 0 to the length of the PATH and can be less than, equal to, or greater
    than the value of m. The value of m must be in the range from 1 to the length of the PATH.

If an error occurs while reading node n (where n is greater than 0), it is handled as follows:

If n > original path length (prior to the read operation), the nodes from n to the new path length
are set uninitialized.

If n <= original path length , the nodes from n to the original path length remain as they were prior
to the read operation and any new nodes (greater than the original path length) are set uninitialized.

• If the associated data description that is read from the PATH does not agree with the current user
associated data, the read operation is terminated and the path will remain as it was prior to the
read operation. The IO_STATUS built-in function will return an error if this occurs.
7. FILE INPUT/OUTPUT OPERATIONS

- PATH data must be read in binary (unformatted) form.

**READ Statement Examples** shows several examples of the READ statement using a variety of file variables and data lists.

**READ Statement Examples**

```fortran
READ (next_part_no) --uses default TPDISPLAY
OPEN FILE file_var ('RO','data_file.dt')
READ file_var (color, style, option)
READ host_line (color, style, option, CR)
FOR i = 1 TO array_size DO
  READ data (data_array[i])
ENDFOR
```

If any errors occur during input, the variable being read and all subsequent variables up to CR in the data list are set uninitialized unless the file variable is open to a window device.

If reading from a window device, 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.

The built-in function IO_STATUS can be used to determine the success or failure (and the reason for the failure) of a READ operation.

**See Also:** READ Statement, Appendix A.

**IO_STATUS Built-In Functions, Appendix A** for a list of I/O error messages

%CRTDEVICE Translator Directive, Appendix A.

### 7.6 WRITE STATEMENT

The WRITE statement is used to write one or more specified data items to the indicated device. The data items are listed as part of the WRITE statement. The following rules apply to the WRITE statement:

- The OPEN FILE statement must be used to associate the file variable with the file opened in the statement before any write operations can be performed unless one of the predefined files is used (refer to Table 7–1).
- If the file variable is omitted from the WRITE statement, then TPDISPLAY is used as the default.
- Using the %CRTDEVICE directive will change the default to OUTPUT (CRT output window).
• Format specifiers can be used to control the format of data that is written for each data_item. The effect of format specifiers depends on the data type of the item being written and on whether the data is in text (ASCII) or binary (unformatted) form.

• ARRAY variables must be written element by element; they cannot be written in unsubscripted form. Frequently, they are written using a WRITE statement in a FOR loop.

• PATH variables can be specified as follows in a WRITE statement, where “path_name” is a PATH variable and “n” and “m” are PATH node indexes:
  — path_name: specifies that the entire path is to be written, starting with a header that provides the path length and associated data table, and followed by all of the nodes, including their associated data.
  — path_name[0]: specifies that only the header is to be written. The path header consists of the path length and a copy of the associated data table.
  — path_name[n]: specifies that node[n] is to be written.
  — path_name[n..m]: specifies that nodes n through m are to be written. The value of n must be in the range from 0 to the length of the PATH and can be less than, equal to, or greater than the value of m. The value of m must be in the range from 1 to the length of the PATH.

• PATH data must be written in binary (unformatted) form.

WRITE Statement Examples shows several examples of the WRITE statement using a variety of file variables and data lists.

WRITE Statement Examples

WRITE TPPROMPT('Press T.P. key "GO" when ready')
WRITE TPFUNC (' GO RECD QUIT BACK1 FWD-1')
WRITE log_file (part_no:5, good_count:5, bad_count:5, operator:3, CR)
WRITE ('This is line 1', CR, 'This is line 2', CR)
--uses default TPDISPLAY
FOR i = 1 TO array_size DO
  WRITE data (data_array[i])
ENDFOR

See Also: WRITE Statement, Appendix A.

IO_STATUS Built-In Functions, Appendix A.

7. INPUT/OUTPUT BUFFER

An area of RAM, called a buffer, is used to hold up to 256 bytes of data that has not yet been transmitted during a read or write operation.
Buffers are used by the READ and WRITE statements as follows:

- During the execution of a READ statement, if more data was read from the file than required by the READ statement, the remaining data is kept in a buffer for subsequent read operations. For example, if you enter more data in a keyboard input line than is required to satisfy the READ statement the extra data is kept in a buffer.
- If a WRITE statement is executed to a non-interactive file and the last data item was not a CR, the data is left in a buffer until a subsequent WRITE either specifies a CR or the buffer is filled.
- The total data that can be processed in a single READ or WRITE statement is limited to 127 bytes.

### 7.8 FORMATTING TEXT (ASCII) INPUT/OUTPUT

This section explains the format specifiers used to read and write ASCII (formatted) text for each data type.

The following rules apply to formatting data types:

- For text files, data items in READ and WRITE statements can be of any of the simple data types (INTEGER, REAL, BOOLEAN, and STRING).
- Positional and VECTOR variables cannot be read from text files but can be used in WRITE statements.
- ARRAY variables cannot be read or written in unsubscripted form. The elements of an ARRAY are read or written in the format that corresponds to the data type of the ARRAY.
- PATH variables cannot be read or written.
- Some formats and data combinations are not read in the same manner as they were written or become invalid if read with the same format.

The amount of data that is read or written can be controlled using zero, one, or two format specifiers for each data item in a READ or WRITE statement. Each format specifier, represented as an INTEGER literal, is preceded by double colons (::).

Table 7–5 summarizes the input format specifiers that can be used with the data items in a READ statement. The default format of each data type and the format specifiers that can affect each data type are explained in Section 7.8.1, through Section 7.8.6.

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>1ST FORMAT SPECIFIER</th>
<th>2ND FORMAT SPECIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Total number of characters read</td>
<td>Number base in range 2 - 16</td>
</tr>
<tr>
<td>REAL</td>
<td>Total number of characters read</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

Table 7–5. Text (ASCII) Input Format Specifiers
Table 7–5. Text (ASCII) Input Format Specifiers (Cont’d)

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>1ST FORMAT SPECIFIER</th>
<th>2ND FORMAT SPECIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Total number of characters read</td>
<td>Ignored</td>
</tr>
<tr>
<td>STRING</td>
<td>Total number of characters read</td>
<td>0 - unquoted STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - quoted STRING</td>
</tr>
</tbody>
</table>

Table 7–6 summarizes the output format specifiers that can be used with the data items in a WRITE statement. The default format of each data type and the format specifiers that can affect each data type are explained in Section 7.8.1 through Section 7.8.6.

Table 7–6. Text (ASCII) Output Format Specifiers

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>1ST FORMAT SPECIFIER</th>
<th>2ND FORMAT SPECIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Total number of characters written</td>
<td>Number base in range 2-16</td>
</tr>
<tr>
<td>REAL</td>
<td>Total number of characters written</td>
<td>Number of digits to the right of decimal point to be written</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If negative, uses scientific notation</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Total number of characters written</td>
<td>0 - Left justified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Right justified</td>
</tr>
<tr>
<td>STRING</td>
<td>Total number of characters written</td>
<td>0 - Left justified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Right justified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Left justified in quotes (leading blank)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - Right justified n quotes (leading blank)</td>
</tr>
<tr>
<td>VECTOR</td>
<td>Uses REAL format for each component</td>
<td>Uses REAL format for each component</td>
</tr>
<tr>
<td>POSITION</td>
<td>Uses REAL format for each component</td>
<td>Uses REAL format for each component</td>
</tr>
<tr>
<td>XYZWPR</td>
<td>Uses REAL format for each component</td>
<td>Uses REAL format for each component</td>
</tr>
<tr>
<td>XYZWPREXT</td>
<td>Uses REAL format for each component</td>
<td>Uses REAL format for each component</td>
</tr>
<tr>
<td>JOINTPOSn</td>
<td>Uses REAL format for each component</td>
<td>Uses REAL format for each component</td>
</tr>
</tbody>
</table>

7.8.1 Formatting INTEGER Data Items

INTEGER data items in a READ statement are processed as follows:

**Default:** Read as a decimal (base 10) INTEGER, starting with the next nonblank character on the input line and continuing until a blank or end of line is encountered. If the characters read do not form a valid INTEGER, the read operation fails.
First Format Specifier: Indicates the total number of characters to be read. The input field must be entirely on the current input line and can include leading, but not trailing, blanks.

Second Format Specifier: Indicates the number base used for the input and must be in the range of 2 (binary) to 16 (hexadecimal).

For bases over 10, the letters A, B, C, D, E, and F are used as input for the digits with values 10, 11, 12, 13, 14, and 15, respectively. Lowercase letters are accepted.

Table 7–7 lists examples of INTEGER input data items and their format specifiers. The input data and the resulting value of the INTEGER data items are included in the table. (The symbol [eol] indicates end of line.)

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>INPUT DATA</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>int_var</td>
<td>-2[eol]</td>
<td>int_var = -2</td>
</tr>
<tr>
<td>int_var</td>
<td>20 30 ...</td>
<td>int_var = 20</td>
</tr>
<tr>
<td>int_var::3</td>
<td>10000</td>
<td>int_var = 100</td>
</tr>
<tr>
<td>int_var::5::2</td>
<td>10101 (base 2 input)</td>
<td>int_var = 21 (base 10 value)</td>
</tr>
<tr>
<td>int_var</td>
<td>1.00</td>
<td>format error (invalid INTEGER)</td>
</tr>
<tr>
<td>int_var::5</td>
<td>100[eol]</td>
<td>format error (too few digits)</td>
</tr>
</tbody>
</table>

INTEGER data items in a WRITE statement are formatted as follows:

Default: Written as a decimal (base 10) INTEGER using the required number of digits and one leading blank. A minus sign precedes the digits if the INTEGER is a negative value.

First Format Specifier: Indicates the total number of characters to be written, including blanks and minus sign. If the format specifier is larger than required for the data, leading blanks are added. If it is smaller than required, the field is extended as required.

The specifier must be in the range of 1 to 127 for a file or 1 to 126 for other output devices.

Second Format Specifier: Indicates the number base used for the output and must be in the range of 2 (binary) to 16 (hexadecimal).

If a number base other than 10 (decimal) is specified, the number of characters specified in the first format specifier (minus one for the leading blank) is written, with leading zeros added if needed.

For bases over 10, the letters A, B, C, D, E, and F are used as input for the digits with values 10, 11, 12, 13, 14, and 15, respectively.
Table 7–8 lists examples of INTEGER output data items and their format specifiers. The output values of the INTEGER data items are also included in the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Leading blank</td>
</tr>
<tr>
<td>-5</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Leading blank</td>
</tr>
<tr>
<td>123::6</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Right justified (leading blanks)</td>
</tr>
<tr>
<td>-123::2</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Expanded as required</td>
</tr>
<tr>
<td>1024::0::16</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>400</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Hexadecimal output</td>
</tr>
</tbody>
</table>
7.8.2 Formatting REAL Data Items

REAL data items in a READ statement are processed as follows:

**Default:** Read starting with the next nonblank character on the input line and continuing until a blank or end of line is encountered.

Data can be supplied with or without a fractional part. The E used for scientific notation can be in upper or lower case. If the characters do not form a valid REAL, the read operation fails.

**First Format Specifier:** Indicates the total number of characters to be read. The input field must be entirely on the current input line and can include leading, but not trailing, blanks.

**Second Format Specifier:** Ignored for REAL data items.

Table 7–9 lists examples of REAL input data items and their format specifiers. The input data and the resulting value of the REAL data items are included in the table. The symbol [eol] indicates end of line and X indicates extraneous data on the input line.
### Examples of REAL Input Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>INPUT DATA</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>real_var</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>real_var</td>
<td>1.000</td>
<td>1.0</td>
</tr>
<tr>
<td>real_var</td>
<td>2.5 XX</td>
<td>2.50</td>
</tr>
<tr>
<td>real_var</td>
<td>1E5 XX</td>
<td>100000.0</td>
</tr>
<tr>
<td>real_var::7</td>
<td>2.5 XX</td>
<td>format error (trailing blank)</td>
</tr>
<tr>
<td>real_var</td>
<td>1E</td>
<td>format error (no exponent)</td>
</tr>
<tr>
<td>real_var::4</td>
<td>1E 2</td>
<td>format error (embedded blank)</td>
</tr>
</tbody>
</table>

REAL data items in a WRITE statement are formatted as follows:

**Default:** Written in scientific notation in the following form:

(\text{blank})(\text{msign})(\text{d})(\text{d})(\text{d})(\text{d})(\text{d})E(\text{esign})(\text{d})(\text{d})

where:

(\text{blank}) is a single blank

(\text{msign}) is a minus sign, if required

(\text{d}) is a digit

(\text{esign}) is a plus or minus sign

**First Format Specifier:** Indicates the total number of characters to be written, including all the digits, blanks, signs, and a decimal point. If the format specifier is larger than required for the data, leading blanks are added. If it is smaller than required, the field is extended as required.

In the case of scientific notation, character length should be greater than \((8 + 2\text{nd format specifier})\) to write the data completely.

The specifier must be in the range of 1 to 127 for a file or 1 to 126 for other output devices.

**Second Format Specifier:** Indicates the number of digits to be output to the right of the decimal point, whether or not scientific notation is to be used.

The absolute value of the second format specifier indicates the number of digits to be output to the right of the decimal point.

If the format specifier is positive, the data is displayed in fixed format (that is, without an exponent). If it is negative, scientific notation is used.
Table 7–10 lists examples of REAL output data items and their format specifiers. The output values of the REAL data items are also included in the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.

Table 7–10. Examples of REAL Output Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.0</td>
<td>&quot;</td>
<td>Scientific notation (default format)</td>
</tr>
<tr>
<td></td>
<td>1.23000E+02</td>
<td>&quot;</td>
</tr>
<tr>
<td>123.456789</td>
<td>&quot;</td>
<td>Rounded to 5 digits in fractional part</td>
</tr>
<tr>
<td></td>
<td>1.23457E+02</td>
<td>&quot;</td>
</tr>
<tr>
<td>.00123</td>
<td>&quot;</td>
<td>Negative exponent</td>
</tr>
<tr>
<td></td>
<td>1.23000E-03</td>
<td>&quot;</td>
</tr>
<tr>
<td>-1.00</td>
<td>&quot;</td>
<td>Negative value</td>
</tr>
<tr>
<td></td>
<td>-1.00000E+00</td>
<td>&quot;</td>
</tr>
<tr>
<td>-123.456::9</td>
<td>&quot;</td>
<td>Field expanded</td>
</tr>
<tr>
<td></td>
<td>-1.234560E+02</td>
<td>&quot;</td>
</tr>
<tr>
<td>123.456::12</td>
<td>&quot;</td>
<td>Leading blank added</td>
</tr>
<tr>
<td></td>
<td>1.234560E+02</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Table 7–10. Examples of REAL Output Data Items (Cont’d)

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.456::9::2</td>
<td>&quot;</td>
<td>Right justified and rounded</td>
</tr>
<tr>
<td></td>
<td>123.46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>123::12::-3</td>
<td>1.230E+02</td>
<td>Scientific notation</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
</tbody>
</table>

7.8.3 Formatting BOOLEAN Data Items

BOOLEAN data items in a READ statement are formatted as follows:

**Default:** Read starting with the next nonblank character on the input line and continuing until a blank or end of line is encountered.

Valid input values for TRUE include TRUE, TRU, TR, T, and ON. Valid input values for FALSE include FALSE, FALS, FAL, FA, F, OFF, and OF. If the characters read do not form a valid BOOLEAN, the read operation fails.

**First Format Specifier:** Indicates the total number of characters to be read. The input field must be entirely on the current input line and can include leading, but not trailing, blanks.

**Second Format Specifier:** Ignored for BOOLEAN data items.

Table 7–11 lists examples of BOOLEAN input data items and their format specifiers. The input data and the resulting value of the BOOLEAN data items are included in the table. (The symbol [eol] indicates end of line and X indicates extraneous data on the input line.)
### Table 7–11. Examples of BOOLEAN Input Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>INPUT DATA</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_var</td>
<td>FALSE[eol]</td>
<td>FALSE</td>
</tr>
<tr>
<td>bool_var</td>
<td>FAL 3...</td>
<td>FALSE</td>
</tr>
<tr>
<td>bool_var</td>
<td>T[eol]</td>
<td>TRUE</td>
</tr>
<tr>
<td>bool_var::1</td>
<td>FXX</td>
<td>FALSE (only reads “F”)</td>
</tr>
<tr>
<td>bool_var</td>
<td>O[eol]</td>
<td>format error (ambiguous)</td>
</tr>
<tr>
<td>bool_var</td>
<td>1.2[eol]</td>
<td>format error (not BOOLEAN)</td>
</tr>
<tr>
<td>bool_var::3</td>
<td>F [eol]</td>
<td>format error (trailing blanks)</td>
</tr>
<tr>
<td>bool_var::6</td>
<td>TRUE[eol]</td>
<td>format error (not enough data)</td>
</tr>
</tbody>
</table>

BOOLEAN data items in a WRITE statement are formatted as follows:

**Default:** Written as either “TRUE” or “FALSE”. (Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.)

**First Format Specifier:** Indicates the total number of characters to be written, including blanks (a leading blank is always included). If the format specifier is larger than required for the data, trailing blanks are added. If it is smaller than required, the field is truncated on the right.

The specifier must be in the range of 1 to 127 for a file or 1 to 126 for other output devices.

**Second Format Specifier:** Indicates whether the data is left or right justified. If the format specifier is equal to 0, the output word is left justified in the output field with one leading blank, and trailing blanks as required. If it is equal to 1, the output word is right justified in the output field, with leading blanks as required.

Table 7–12 lists examples of BOOLEAN output data items and their format specifiers. The output values of the BOOLEAN data items are also included in the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.
### Table 7–12. Examples of BOOLEAN Output Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
<th>COMMENT</th>
</tr>
</thead>
</table>
| FALSE     | "FALSE"
|           | "     |
|           | "     | Default includes a leading blank |
| TRUE      | "TRUE"
|           | "     |
|           | "     | TRUE is shorter than FALSE |
| FALSE::8  | "FALSE"
|           | "     |
|           | "     | Left justified (default) |
| FALSE::8::1| "FALSE"
|           | "     |
|           | "     | Right justified |
| TRUE::2   | "T"
|           | "     |
|           | "     | Truncated |

#### 7.8.4 Formatting STRING Data Items

STRING data items in a READ statement are formatted as follows:

**Default:** Read starting at the current position and continuing to the end of the line. If the length of the data obtained is longer than the declared length of the STRING, the data is truncated on the right. If it is shorter, the current length of the STRING is set to the actual length.
**First Format Specifier:** Indicates the total field length of the input data. If the field length is longer than the declared length of the STRING, the input data is truncated on the right. If it is shorter, the current length of the STRING is set to the specified field length.

**Second Format Specifier:** Indicates whether or not the input STRING is enclosed in single quotes. If the format specifier is equal to 0, the input is not enclosed in quotes. If it is equal to 2, the input must be enclosed in quotes. The input is scanned for the next nonblank character. If the character is not a quote, the STRING is not valid and the read operation fails.

If the character is a quote, the remaining characters are scanned until another quote or the end of line is found. If another quote is not found, the STRING is not valid and the read operation fails.

If both quotes are found, all of the characters between them are read into the STRING variable, unless the declared length of the STRING is too short, in which case the data is truncated on the right.

Table 7–13 lists examples of STRING input data items and their format specifiers, where str_var has been declared as a STRING[5]. The input data and the resulting value of the STRING data items are included in the table. The symbol [eol] indicates end of line and X indicates extraneous data on the input line.

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>INPUT DATA</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_var</td>
<td>&quot;ABC[eol]&quot;</td>
<td>&quot;ABC&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;ABCDEFG[eol]&quot;</td>
<td>&quot;ABCDE&quot;</td>
</tr>
<tr>
<td></td>
<td>(FG is read but the STRING is truncated to 5 characters)</td>
<td></td>
</tr>
</tbody>
</table>
Table 7–13. Examples of STRING Input Data Items (Cont’d)

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>INPUT DATA</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_var</td>
<td>&quot;'ABC'XX</td>
<td>&quot;'AB&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>(blanks and quote are read as data)</td>
<td></td>
</tr>
<tr>
<td>str_var::0::2</td>
<td>&quot;'ABC'XX</td>
<td>&quot;'ABC'</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>(read ends with second quote)</td>
<td></td>
</tr>
</tbody>
</table>

STRING data items in a WRITE statement are formatted as follows:

**Default:** Content of the STRING is written with no trailing or leading blanks or quotes.

The STRING must not be over 127 bytes in length for files or 126 bytes in length for other output devices. Otherwise, the program will be aborted with the “STRING TOO LONG” error.

**First Format Specifier:** Indicates the total number of characters to be written, including blanks. If the format specifier is larger than required for the data, the data is left justified and trailing blanks are added. If the format specifier is smaller than required, the STRING is truncated on the right.

The specifier must be in the range of 1 to 127 for a file or 1 to 126 for other output devices.

**Second Format Specifier:** Indicates whether the output is to be left or right justified and whether the STRING is to be enclosed in quotes using the following values:

0 left justified, no quotes
1 right justified, no quotes
2 left justified, quotes
3 right justified, quotes

Quoted STRING values, even if left justified, are preceded by a blank. Unquoted STRING values are not automatically preceded by a blank.
Table 7–14 lists examples of STRING output data items and their format specifiers. The output values of the STRING data items are also included in the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.

Table 7–14. Examples of STRING Output Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ABC'</td>
<td>&quot;</td>
<td>No leading blanks</td>
</tr>
<tr>
<td></td>
<td>ABC &quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>'ABC'::2</td>
<td>&quot;</td>
<td>Truncated on right</td>
</tr>
<tr>
<td></td>
<td>ABC &quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>'ABC'::8</td>
<td>&quot;</td>
<td>Left justified</td>
</tr>
<tr>
<td></td>
<td>ABC &quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>'ABC'::8:0</td>
<td>&quot;</td>
<td>Same as previous</td>
</tr>
<tr>
<td></td>
<td>ABC &quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>'ABC'::8:1</td>
<td>&quot;</td>
<td>Right justified</td>
</tr>
<tr>
<td></td>
<td>ABC &quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>'ABC'::8:2</td>
<td>&quot;</td>
<td>Note leading blank</td>
</tr>
<tr>
<td></td>
<td>'ABC'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td></td>
</tr>
</tbody>
</table>
### Formatting VECTOR Data Items

VECTOR data items cannot be read from text (ASCII) files. However, you can read three REAL values and assign them to the elements of a VECTOR variable. VECTOR data items in a WRITE statement are formatted as three REAL values on the same line.

**Table 7–15** lists examples of VECTOR output data items and their format specifiers, where `vect.x = 1.0`, `vect.y = 2.0`, `vect.z = 3.0`. The output values of the VECTOR data items are also included in...
the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.

See Also: Section 7.8.2, “Formatting REAL Data Items,” for information on the default output format and format specifiers used with REAL data items

Table 7–15. Examples of VECTOR Output Data Items

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>vect</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>1. 2. 3.</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td>vect:6::2</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>1.00 2.00 3.00</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td>vect:12::-3</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>1.000E+00 2.000E+00 3.000E+00</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
</tbody>
</table>

7.8.6 Formatting Positional Data Items

Positional data items cannot be read from text (ASCII) files. However, you can read six REAL values and a STRING value and assign them to the elements of an XYZWPR variable or use the POS built-in function to compose a POSITION. The CNV_STR_CONF built-in can be used to convert a STRING to a CONFIG data type.

POSITION and XYZWPR data items in a WRITE statement are formatted in three lines of output. The first line contains the location (x,y,z) component of the POSITION, the second line contains the orientation (w,p,r), and the third line contains the configuration string.

The location and orientation components are formatted as six REAL values. The default format for the REAL values in a POSITION is the default format for REAL(s). Refer to Section 7.8.2.

The configuration string is not terminated with a CR, meaning you can follow it with other data on the same line.
Table 7–16 lists examples of POSITION output data items and their format specifiers, where \( p = \text{POS}(2.0,-4.0,8.0,0.0,90.0,0.0,\text{config\_var}) \). The output values of the POSITION data items are also included in the table. Double quotes are used in the table as delimiters to show leading blanks; however, double quotes are not written by KAREL programs.

Table 7–16. Examples of POSITION Output Data Items (\( p = \text{POS}(2.0,-4.0,8.0,0.0,90.0,0.0,\text{config\_var}) \))

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>2.</td>
</tr>
<tr>
<td></td>
<td>-4.</td>
</tr>
<tr>
<td></td>
<td>8.</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>0.</td>
</tr>
<tr>
<td></td>
<td>9.</td>
</tr>
<tr>
<td></td>
<td>0.</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>( N, 127, \ , -1 \</td>
</tr>
<tr>
<td>p::7::2</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>2.00-4.00 8.00</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>0.0090.00 0.00</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>( N, 127, \ , -1 \</td>
</tr>
</tbody>
</table>

JOINTPOS data items in a WRITE statement are formatted similarly to POSITION types with three values on one line.

See Also: Section 7.8.2, for information on format specifiers used with REAL data items
This section explains the format specifier used in READ and WRITE statements to read and write binary (unformatted) data for each data item. Binary input/output operations are sometimes referred to as unformatted, as opposed to text (ASCII) input/output operations that are referred to as formatted.

The built-in SET_FILE_ATR with the ATR_UF attribute is used to designate a file variable for binary operations. If not specified, ASCII text operations will be used.

Data items in READ and WRITE statements can be any of the following data types for binary files:

- INTEGER
- REAL
- BOOLEAN
- STRING
- VECTOR
- POSITION
- XYZWPR
- XYZWPREXT
- JOINTPOS

Vision and array variables cannot be read or written in unsubscripted form. The elements of an ARRAY are read or written in the format that corresponds to the data type of the ARRAY.

Entire PATH variables can be read or written, or you can specify that only node[0] (containing the PATH header), a specific node, or a range of nodes be read or written. Format specifiers have no effect on PATH data. PATH data can be read or written only to a file and not to a serial port, CRT/KB, or teach pendant.

Binary I/O is preferred to text I/O when creating files that are to be read only by KAREL programs for the following reasons:

- Positional, VECTOR, and PATH variables cannot be read directly from text input.
- Some formats and data combinations are not read in the same manner as they were written in text files or they become invalid if read with the same format.
- Binary data is generally more compact, reducing both the file size and the I/O time.
- There is some inevitable loss of precision when converting from REAL data to its ASCII representation and back.

Generally, no format specifiers need to be used with binary I/O. If this rule is followed, all input data can be read exactly as it was before it was written.
However, if large numbers of INTEGER values are to be written and their values are known to be small, writing these with format specifiers reduces both storage space and I/O time.

For example, INTEGER values in the range of -128 to +127 require only one byte of storage space, and INTEGER values in the range of -32768 to +32767 require two bytes of storage space. Writing INTEGER values in these ranges with a first format specifier of 1 and 2, respectively, results in reduced storage space and I/O time requirements, with no loss of significant digits.

Table 7–17 summarizes input and output format specifiers that can be used with the data items in READ and WRITE statements. The default format of each data type is also included. Section 7.8.1 through Section 7.8.6 explain the effects of format specifiers on each data type in more detail.

See Also: SET_FILE_ATR Built-In Routine, Appendix A.

### Table 7–17. Binary Input/Output Format Specifiers

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>DEFAULT</th>
<th>1ST FORMAT SPECIFIER</th>
<th>2ND FORMAT SPECIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Four bytes read or written</td>
<td>Specified number of least significant bytes read or written, starting with most significant (1-4)</td>
<td>Ignored</td>
</tr>
<tr>
<td>REAL</td>
<td>Four bytes read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Four bytes read or written</td>
<td>Specified number of least significant bytes read or written, starting with most significant (1-4)</td>
<td>Ignored</td>
</tr>
<tr>
<td>STRING</td>
<td>Current length of string (1 byte), followed by data bytes</td>
<td>Number of bytes read or written</td>
<td>Ignored</td>
</tr>
<tr>
<td>VECTOR</td>
<td>Three 4-byte REAL numbers read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>POSITION</td>
<td>56 bytes read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>XYZWPR</td>
<td>32 bytes read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>XYZWPREXT</td>
<td>44 bytes read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>JOINTPOSn</td>
<td>4 + n*4 bytes read or written</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>PATH</td>
<td>Depends on size of structure</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

### 7.9.1 Formatting INTEGER Data Items

INTEGER data items in a READ or WRITE statement are formatted as follows:

**Default:** Four bytes of data are read or written starting with the most significant byte.
First Format Specifier: Indicates the number of least significant bytes of the INTEGER to read or write, with the most significant of these read or written first. The sign of the most significant byte read is extended to unread bytes. The format specifier must be in the range from 1 to 4.

For example, if an INTEGER is written with a format specifier of 2, bytes 3 and 4 (where byte 1 is the most significant byte) will be written. There is no check for loss of significant bytes when INTEGER values are formatted in binary I/O operations.

Note Formatting of INTEGER values can result in undetected loss of high order digits.

Second Format Specifier: Ignored for INTEGER data items.

7.9.2 Formatting REAL Data Items

REAL data items in a READ or WRITE statement are formatted as follows:

Default: Four bytes of data are read or written starting with the most significant byte.

First Format Specifier: Ignored for REAL data items.

Second Format Specifier: Ignored for REAL data items.

7.9.3 Formatting BOOLEAN Data Items

BOOLEAN data items in a READ or WRITE statement are formatted as follows:

Default: Four bytes of data are read or written. In a read operation, the remainder of the word, which is never used, is set to 0.

First Format Specifier: Indicates the number of least significant bytes of the BOOLEAN to read or write, the most significant of these first. The format specifier must be in the range from 1 to 4. Since BOOLEAN values are always 0 or 1, it is always safe to use a field width of 1.

Second Format Specifier: Ignored for BOOLEAN data items.

7.9.4 Formatting STRING Data Items

STRING data items in a READ or WRITE statement are formatted as follows:

Default: The current length of the STRING (not the declared length) is read or written as a single byte, followed by the content of the STRING. STRING values written without format specifiers have their lengths as part of the output, while STRING values written with format specifiers do not.
Likewise, if a STRING is read without a format specifier, the length is expected in the data, while if a STRING is read with a format specifier, the length is not expected.

This means that, if you write and then read STRING data, you must make sure your use of format specifiers is consistent.

**First Format Specifier:** Indicates the number of bytes to be read or written.

**Second Format Specifier:** Ignored for STRING data items.

In a read operation, if the first format specifier is greater than the declared length of the STRING, the data is truncated on the right. If it is less than the declared length of the STRING, the current length of the STRING is set to the number of bytes read.

In a write operation, if the first format specifier indicates a shorter field than the current length of the STRING, the STRING data is truncated on the right. If it is longer than the current length of the STRING, the output is padded on the right with blanks.

Writing STRING values with format specifiers can cause truncation of the original STRING values or padding blanks on the end of the STRING values when reread.

### 7.9.5 Formatting VECTOR Data Items

VECTOR data items in a READ or WRITE statement are formatted as follows:

**Default:** Data is read or written as three 4-byte binary REAL numbers.

**First Format Specifier:** Ignored for VECTOR data items.

**Second Format Specifier:** Ignored for VECTOR data items.

### 7.9.6 Formatting POSITION Data Items

POSITION data items in a READ or WRITE statement are formatted as follows:

**Default:** Read or written in the internal format of the controller, which is 56 bytes long.

### 7.9.7 Formatting XYZWPR Data Items

XYZWPR data items in a READ or WRITE statement are formatted as follows:

**Default:** Read or written in the internal format of the controller, which is 32 bytes long.
7.9.8 Formatting XYZWPREXT Data Items

XYZWPREXT data items in a READ or WRITE statement are formatted as follows:

**Default:** Read or written in the internal format of the controller, which is 44 bytes long.

7.9.9 Formatting JOINTPOS Data Items

JOINTPOS data items in a READ or WRITE statement are formatted as follows:

**Default:** Read or written in the internal format of the controller, which is 4 bytes plus 4 bytes for each axis.

7.10 USER INTERFACE TIPS

Input and output to the teach pendant or CRT/KB is accomplished by executing "READ" and "WRITE" statements within a KAREL program. If the USER menu is not the currently selected menu, the input will remain pending until the USER menu is selected. The output will be written to the "saved" windows that will be displayed when the USER menu is selected. You can have up to eight saved windows.

7.10.1 USER Menu on the Teach Pendant

The screen that is activated when the USER menu is selected from the teach pendant is named "t_sc". The windows listed in Table 7–18 are defined for "t_sc".

<table>
<thead>
<tr>
<th>Window Name</th>
<th>Lines</th>
<th>Predefined FILE Name</th>
<th>Scrolled</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;t_fu&quot;</td>
<td>10</td>
<td>TPDISPLAY</td>
<td>yes</td>
<td>5-14</td>
</tr>
<tr>
<td>&quot;t_pr&quot;</td>
<td>1</td>
<td>TPPROMPT</td>
<td>no</td>
<td>15</td>
</tr>
<tr>
<td>&quot;t_st&quot;</td>
<td>3</td>
<td>TPSTATUS</td>
<td>no</td>
<td>2-4</td>
</tr>
<tr>
<td>&quot;t_fk&quot;</td>
<td>1</td>
<td>TPFUNC</td>
<td>no</td>
<td>16</td>
</tr>
<tr>
<td>&quot;err&quot;</td>
<td>1</td>
<td>TPERROR</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>&quot;stat&quot;</td>
<td>1</td>
<td></td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>&quot;full&quot;</td>
<td>2</td>
<td></td>
<td>no</td>
<td>3-4</td>
</tr>
<tr>
<td>&quot;motn&quot;</td>
<td>1</td>
<td></td>
<td>no</td>
<td>3</td>
</tr>
</tbody>
</table>
By default, the USER menu will attach the "err", "stat", "full", "motn", "t_fu", "t_pr", and "t_fk" windows to the "t_sc" screen. See Figure 7–1.

**Figure 7–1. "t_sc" Screen**

![Diagram of "t_sc" Screen](image)

The following system variables affect the teach pendant USER menu:

- **$TP_DEFPROG: STRING** - Identifies the teach pendant default program. This is automatically set when a program is selected from the teach pendant SELECT menu.

- **$TP_INUSER: BOOLEAN** - Set to TRUE when the USER menu is selected from the teach pendant.

- **$TP_LCKUSER: BOOLEAN** - Locks the teach pendant in the USER menu while $TP_DEFPROG is running and $TP_LCKUSER is TRUE.

- **$TP_USESTAT: BOOLEAN** - Causes the user status window "t_st" (TPSTATUS) to be attached to the user screen while $TP_USESTAT is TRUE. While "t_st" is attached, the "stat", "motn", and "full" windows will be detached. See Figure 7–2.

**Figure 7–2. "t_sc" Screen with $TP_USESTAT = TRUE**

![Diagram showing "t_sc" Screen with $TP_USESTAT = TRUE](image)
7. FILE INPUT/OUTPUT OPERATIONS

7.10.2 USER Menu on the CRT/KB

The screen that is activated when the USER menu is selected from the CRT is named "c_sc". The windows listed in Table 7–19 are defined for "c_sc".

Table 7–19. Defined Windows for c_sc*

<table>
<thead>
<tr>
<th>Window Name</th>
<th>Lines</th>
<th>Predefined FILE Name</th>
<th>Scrolled</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;c_fu&quot;</td>
<td>17</td>
<td>INPUT and OUTPUT</td>
<td>yes</td>
<td>5-21</td>
</tr>
<tr>
<td>&quot;c_pr&quot;</td>
<td>1</td>
<td>CRTPROMPT</td>
<td>no</td>
<td>22</td>
</tr>
<tr>
<td>&quot;c_st&quot;</td>
<td>3</td>
<td>CRTSTATUS</td>
<td>no</td>
<td>2-4</td>
</tr>
<tr>
<td>&quot;c_fk&quot;</td>
<td>2</td>
<td>CRTFUNC</td>
<td>no</td>
<td>23-24</td>
</tr>
<tr>
<td>&quot;err&quot;</td>
<td>1</td>
<td>CRTERRORE</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>&quot;ct01&quot;</td>
<td>1</td>
<td></td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>&quot;uful&quot;</td>
<td>2</td>
<td></td>
<td>no</td>
<td>3-4</td>
</tr>
<tr>
<td>&quot;motn&quot;</td>
<td>1</td>
<td></td>
<td>no</td>
<td>3</td>
</tr>
</tbody>
</table>

By default, the USER menu will attach the "err", "ct01", "uful", "motn", "c_fu", "c_fk", and "uftn" windows to the "c_sc" screen. The "c_fk" window will label the function keys and show FCTN and MENUS for F9 and F10. See Figure 7–3.

Figure 7–3. "c_sc" Screen

The following system variables affect the CRT USER menu:

- **SCRT_DEFPROG: STRING** - This variable identifies the CRT default program. This is automatically set when a program is selected from the CRT SELECT menu.
- **SCRT_INUSER: BOOLEAN** - This variable is set to TRUE when the USER menu is selected from the CRT.
• **$CRT_LCKUSER: BOOLEAN** - This variable locks the CRT in the USER menu while $CRT_DEFPROG is running and $CRT_LCKUSER is TRUE.

• **$CRT_USERSTAT: BOOLEAN** - This variable causes the user status window "c_st" (CRTSTATUS) to be attached to the user screen while $CRT_USERSTAT is TRUE. While "c_st" is attached, the "ct01", "motn", and "uful" windows will be detached. See Figure 7–4.

**Figure 7–4. "c_sc" Screen with $CRT_USERSTAT = TRUE**

<table>
<thead>
<tr>
<th>err  (CRTERRO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_st (CRTSTATUS)</td>
</tr>
<tr>
<td>c_st (CRTSTATUS)</td>
</tr>
<tr>
<td>c_st (CRTSTATUS)</td>
</tr>
<tr>
<td>c_fu (INPUT and OUTPUT)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_pr (CRTPROMPT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_fk (CRTFUNC)</td>
</tr>
<tr>
<td>c_fk</td>
</tr>
</tbody>
</table>
Chapter 8

POSITION DATA

Contents

Chapter 8  POSITION DATA ................................................................. 8–1
8.1  OVERVIEW .............................................................................. 8–2
8.2  POSITIONAL DATA ................................................................. 8–2
8.3  FRAMES OF REFERENCE ....................................................... 8–3
  8.3.1  World Frame .................................................................. 8–4
  8.3.2  User Frame (UFRAME) .................................................... 8–5
  8.3.3  Tool Definition (UTOOL) ............................................... 8–5
  8.3.4  Using Frames in the Teach Pendant Editor (TP) ............. 8–5
8.4  JOG COORDINATE SYSTEMS ............................................... 8–6
8. POSITION DATA

8.1 OVERVIEW

In robotic applications, single segment motion is the movement of the tool center point (TCP) from an initial position to a desired destination position. The KAREL system represents positional data in terms of location (x, y, z), orientation (w, p, r), and configuration. The location and orientation are defined relative to a Cartesian coordinate system (user frame), making them independent of the robot joint angles. Configuration represents the unique set of joint angles at a particular location and orientation.

Note The KAREL system provides a way to create and manipulate position data but it does not support motion instructions. All motion must be initiated from a teach pendant program. Instructions and builtins are available for setting KAREL position data into teach pendant programs.

8.2 POSITIONAL DATA

The KAREL language uses the POSITION, XYZWPR, XYZWPREXT, JOINTPOS, and PATH data types to represent positional data. The POSITION data type is composed of the following:

- Three REAL values representing an x, y, z location expressed in millimeters
- Three REAL values representing a w, p, r orientation expressed in degrees
- One CONFIG Data Type, consisting of 4 booleans and 3 integers, which represent the configuration in terms of joint placement and turn number. Before you specify the config data type, make sure it is valid for the robot being used. Valid joint placement values include:
  - ‘R’ or ‘L’ (shoulder right or left)
  - ‘U’ or ‘D’ (elbow up or down)
  - ‘N’ or ‘F’ (wrist no-flip or flip)
  - ‘T’ or ‘B’ (config front or back)

A turn number is the number of complete turns a multiple turn joint makes beyond the required rotation to reach a position. Table 8–1 lists the valid turn number definitions.

<table>
<thead>
<tr>
<th>Turn Number</th>
<th>Rotation (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-2700 to -3059</td>
</tr>
<tr>
<td>-7</td>
<td>-2340 to -2699</td>
</tr>
</tbody>
</table>

Table 8–1. Turn Number Definitions
### Turn Number Definitions (Cont’d)

<table>
<thead>
<tr>
<th>Turn Number</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6</td>
<td>-1980 to -2339</td>
</tr>
<tr>
<td>-5</td>
<td>-1620 to -1979</td>
</tr>
<tr>
<td>-4</td>
<td>-1260 to -1619</td>
</tr>
<tr>
<td>-3</td>
<td>-900 to -1259</td>
</tr>
<tr>
<td>-2</td>
<td>-540 to -899</td>
</tr>
<tr>
<td>-1</td>
<td>-180 to -539</td>
</tr>
<tr>
<td>0</td>
<td>-179 to 179</td>
</tr>
<tr>
<td>1</td>
<td>180 to 539</td>
</tr>
<tr>
<td>2</td>
<td>540 to 899</td>
</tr>
<tr>
<td>3</td>
<td>900 to 1259</td>
</tr>
<tr>
<td>4</td>
<td>1260 to 1619</td>
</tr>
<tr>
<td>5</td>
<td>1620 to 1979</td>
</tr>
<tr>
<td>6</td>
<td>1980 to 2339</td>
</tr>
<tr>
<td>7</td>
<td>2340 to 2699</td>
</tr>
</tbody>
</table>

The PATH data type consists of a varying-length list of elements called path nodes.

**See Also:** The appropriate application-specific *FANUC Robotics Setup and Operations Manual* for configuration information on each supported robot model. The POSITION, XYZWPR, XYZWPREXT, JOINTPOS, and PATH Data Types, Appendix A, “KAREL Language Alphabetical Description.”

### 8.3 FRAMES OF REFERENCE

The KAREL system defines the location and orientation of positional data relative to a user-defined frame of reference, called user frame, as shown in Figure 8–1.
Three frames of reference exist:

- **WORLD** - predefined
- **UFRAME** - determined by the user
- **UTOOL** - defined by the tool

Using kinematic equations, the controller computes its positional information based on the known world frame and the data stored in the system variables $UFRAME$ (for user frame) and $UTOOL$ (for tool frame).

### 8.3.1 World Frame

The world frame is predefined for each robot. It is used as the default frame of reference. The location of world frame differs for each robot model.
8.3.2 User Frame (UFRAME)

The programmer defines user frame relative to the world frame by assigning a value to the system variable $UFRAME.

**Warning**

Be sure $UFRAME is set to the same value whether you are teaching positional data or running a program with that data, or damage to the tool could occur.

The location of UFRAME represents distances along the x-axis, y-axis, and z-axis of the world coordinate system; the orientation represents rotations around those axes.

By default, the system assigns a (0,0,0) location value and a (0,0,0) orientation value to $UFRAME, meaning the user frame is identical to that of the world coordinate system. All positions are recorded relative to UFRAME.

8.3.3 Tool Definition (UTOOL)

The *tool center point (TCP)* is the origin of the UTOOL frame of reference. The programmer defines the position of the TCP relative to the faceplate of the robot by assigning a value to the system variable $UTOOL. By default, the system assigns a (0,0,0) location and a (0,0,0) orientation to $UTOOL, meaning $UTOOL is identical to the faceplate coordinate system. The positive z-axis of UTOOL defines the *approach vector* of the tool.

**Warning**

Be sure $UTOOL correctly defines the position of the TCP for the tool you are using, or damage to the tool could occur.

The faceplate coordinate system has its origin at the center of the faceplate surface. Its orientation is defined with the plane of the x-axis and y-axis on the faceplate and the positive z-axis pointing straight out from the faceplate.

8.3.4 Using Frames in the Teach Pendant Editor (TP)

The system variable $USEUFRAME defines whether the current value of $MNUFRAMENUM[group_no] will be assigned to the position’s user frame when it is being recorded or touched up.
• When $USEUFRAME = FALSE$, the initial recording of positions and the touching up of positions is done with the user frame number equal to 0, regardless of the value of $MNUFRAMENUM[group_no]$.

• When $USEUFRAME = TRUE$, the initial recording of positions is done with the position’s user frame equal to the user frame defined by $MNUFRAMENUM[group_no]$. The touching up of positions must also be done with the position’s user frame equal to the user frame defined by $MNUFRAMENUM[group_no]$.

When a position is recorded in the teach pendant editor, the value of the position’s tool frame will always equal the value of $MNUTOOLNUM[group_no]$ at the time the position was recorded. When a teach pendant program is executed, you must make sure that the user frame and the tool frame of the position equal the values of $MNUFRAMENUM[group_no]$ and $MNUTOOLNUM[group_no]$; otherwise, an error will occur. Set the values of $MNUFRAMENUM[1]$ and $MNUTOOLNUM[1]$ using the UFRAME_NUM = n and UTOOL_NUM = n instructions in the teach pendant editor before you record the position to guarantee that the user and tool frame numbers match during program execution.

### 8.4 JOG COORDINATE SYSTEMS

The KAREL system provides five different jog coordinate systems:

• **JOINT** - a joint coordinate system in which individual robot axes move. The motion is joint interpolated.

• **WORLD** - a Cartesian coordinate system in which the TCP moves parallel to, or rotates around, the x, y, and z-axes of the predefined WORLD frame. The motion is linearly interpolated.

• **TOOLFRAME** - a Cartesian coordinate system in which the TCP moves parallel to, or rotates around, the x, y, and z-axes of the currently selected tool frame. The motion is linearly interpolated. The tool frame is normally selected using the SETUP Frames menu. To jog using $GROUP[group_no].$utool, set $MNUTOOLNUM[group_no] = 30$.

• **JOGFRAME** - a Cartesian coordinate system in which the TCP moves parallel to, or rotates around, the x, y, and z-axes of the coordinate system defined by the $JOG_GROUP[group_no].$jogframe system variable. The motion is linearly interpolated.

• **USER FRAME** - a Cartesian coordinate system in which the TCP moves parallel to, or rotates around, the x, y, and z-axes of the currently selected user frame. The motion is linearly interpolated. The user frame is normally selected using the SETUP Frames menu. To jog using $GROUP[group_no].$uframe, set $MNUFRAMENUM[group_no] = 62$.

The robot can be jogged in any one of these jog coordinate systems to reach a destination position. Once that position is reached, however, the positional data is recorded with reference to the user frame as discussed in Section 8.3.

**See Also:** The application-specific *FANUC Robotics Setup and Operations Manual* for step-by-step explanations of how to jog and define frames.
9.1 OVERVIEW

The file system provides a means of storing data on CMOS RAM, FROM, or external storage devices. The data is grouped into units, with each unit representing a file. For example, a file can contain the following:

- Source code statements for a KAREL program
- A sequence of KCL commands for a command procedure
- Variable data for a program

Files are identified by file specifications that include the following:

- The name of the device on which the file is stored
- The name of the file
- The type of data included in the file

The KAREL system includes five types of storage devices where files can be stored:

- RAM Disk
- FROM Disk
- IBM PC
- Memory Card
- USB Memory Stick Device

**RAM Disk** is a portion of SRAM (formerly CMOS RAM) or DRAM memory that functions as a separate storage device. Any file can be stored on the RAM Disk. RAM Disk files should be copied to disks for permanent storage.

**FROM Disk** is a portion of FROM memory that functions as a separate storage device. Any file can be stored on the F-ROM disk. However, the hardware supports a limited number of read and write cycles. Therefore, if a file needs to store dynamically changing data, the RAM disk should be used instead.

**IBM PC or compatible computers** can be used to store files off-line. You can use OLPC, the FANUC Robotics off-line storage software for the PC, to store files on an external storage device. The files on these storage devices are accessible in the following ways:

- Through the FILE menu on the teach pendant and CRT/KB
- Through KAREL programs

**Memory Card** refers to the ATA Flash File storage. The memory card interface is located on the MAIN CPU.

For more information on storage devices and memory, refer to Section 9.3.1.
**USB Memory Stick Device** 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. The USB interface is located on the controller operator panel.

### 9.2 FILE SPECIFICATION

File specifications identify files. The specification indicates:

- The name of the device on which the file is stored, refer to **Section 9.2.1**.
- The name of the file, refer to **Section 9.2.2**.
- The type of data the file contains, refer to **Section 9.2.3**.

The general form of a file specification is:

```
device_name:file_name.file_type
```

#### 9.2.1 Device Name

A device name consists of at least two characters that indicate the device on which a file is stored. Files can be stored on RAM disk, F-ROM disk, disk drive units, off-line on a PC, Memory Card, or PATH Composite Device. The device name always ends with a colon (:). The following is a list of valid storage devices.

- **RD**: (RAM Disk)
  
The RD: device name refers to files stored on the RAM Disk of the controller. RD: is used as the default device name.

- **FR**: (F-ROM Disk)
  
The FR: device name refers to files stored on the F-ROM disk of the controller.

- **MC**: (Memory Card Device)
  
The memory card can be formatted and used as an MS-DOS file system. It can be read from and written to on the controller and an IBM PC equipped with the proper hardware and software. If the memory card is used as an MS-DOS file system, it should be formatted only on the controller. Refer to the application-specific *FANUC Robotics Setup and Operations Manual* for information on formatting the memory card on the controller.
• **UD1: (USB Memory Stick Device)**

The USB memory stick can be formatted and used as an MS-DOS file system. It can be read from and written to on the controller and an IBM PC equipped with the proper hardware and software. If the USB memory stick is used as an MS-DOS file system, it should be formatted only on the controller. Refer to the application-specific *FANUC Robotics Setup and Operations Manual* for information on formatting the USB memory stick on the controller.

• **MD: (Memory Device)**

The memory device treats the controller’s program memory as if it were a file device. You can access all teach pendant programs, KAREL programs, KAREL variables, system variables, and error logs that are loaded in the controller. See Section 9.6 for further details.

• **MDB: (Memory Device Backup)**

The memory device backup device (MDB:) allows the user to copy the same files as provided by the Backup function on the File Menu. This allows the user to back up the controller remotely.

• **CONS: (Console Device)**

The console device provides access to the console log text files CONSLOG.LS and CONSTAIL.LS. It is used for diagnostic and debug purposes and not as a storage device.

• **MF: (Memory File Device)**

The MF: device name refers to files stored on both the RAM and F-ROM disks. Since a file cannot be on both disks at the same time, there will be no duplicate file names.

• **PATH: (Composite Device)**

The PATH: device is a read-only device that searches the F-ROM disk (FD:), memory card (MC:0, and floppy disk (FLPY:) in that order, for a specified file. The PATH: device eliminates the user’s need to know on which storage device the specified file exists.

• **PIP: (File Pipe Device)**

The PIP: device provides a way to write data from one application and, at the same time, read it from another application. The PIP: device also allows the last set of data written to be retained for analysis. The PIP: device allows you to access any number of pipe files. This access is to files that are in the controller’s memory. This means that the access to these files is very efficient. The size of the files and number of files are limited by available controller memory. This means that the best use of a file pipe is to buffer data or temporarily hold it.

### 9.2.2 File Name

A file name is an identifier that you choose to represent the contents of a file.

The following rules apply to file names:
• File names are limited to 36 characters.
• File names can include letters, digits, and underscores.
• File names cannot include these characters: .:*;/'
• Spaces are not allowed in the file name.
• Other special characters can be used with caution.
• Subdirectories can be used. These are also called a “path.” These begin and end with the “\” character. The rules for file names also apply to paths. Below is an example of a file name with a device and a path:

```
MC:\mypath\myfile.txt
```

### 9.2.3 File Type

A file type consists of two or three characters that indicate what type of data a file contains. A file type always begins with a period (.). Table 9–1 is an alphabetical list of each available file type and its function.

#### Table 9–1. File Type Descriptions

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BMP</td>
<td>Bit map files contain bit map images used in robot vision systems.</td>
</tr>
<tr>
<td>.CF</td>
<td>KCL command files are ASCII files that contain a sequence of KCL commands for a command procedure.</td>
</tr>
<tr>
<td>.CH</td>
<td>Condition handler files are used as part of the condition monitor feature.</td>
</tr>
<tr>
<td>.DF</td>
<td>Default file are binary files that contain the default motion instructions for each teach pendant programming.</td>
</tr>
<tr>
<td>.DG</td>
<td>Diagnostic files are ASCII files that provide status or diagnostic information about various functions of the controller.</td>
</tr>
<tr>
<td>.DT</td>
<td>KAREL data file An ASCII or binary file that can contain any data that is needed by the user.</td>
</tr>
<tr>
<td>.IO</td>
<td>Binary files that contain I/O configuration data - generated when an I/O screen is displayed and the data is saved.</td>
</tr>
<tr>
<td>.KL</td>
<td>KAREL source code files are ASCII files that contain the KAREL language statements for a KAREL program.</td>
</tr>
</tbody>
</table>
### Table 9–1. File Type Descriptions (Cont’d)

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.LS</td>
<td><strong>KAREL listing files</strong> are ASCII files that contain the listing of a KAREL language program and line number for each KAREL statement.</td>
</tr>
<tr>
<td>.MN</td>
<td><strong>Mnemonic</strong> program files are supported in previous versions of KAREL.</td>
</tr>
<tr>
<td>.ML</td>
<td><strong>Part model files</strong> contain part model information used in robot vision systems.</td>
</tr>
<tr>
<td>.PC</td>
<td><strong>KAREL p-code files</strong> are binary files that contain the p-code produced by the translator upon translation of a .KL file.</td>
</tr>
<tr>
<td>.SV</td>
<td><strong>System files</strong> are binary files that contain data for tool and user frames (SYSFRAME.SV), system variables (SYSVARS.SV), mastering (SYSMAST.SV), servo parameters (SYSSERVO.SV), and macros (SYSMACRO.SV).</td>
</tr>
<tr>
<td>.TP</td>
<td><strong>Teach pendant program files</strong> are binary files that contain instructions for teach pendant programs.</td>
</tr>
<tr>
<td>.TX</td>
<td><strong>Text files</strong> are ASCII files that can contain system-defined text or user-defined text.</td>
</tr>
<tr>
<td>.VR</td>
<td><strong>Program variable files</strong> are binary files that contain variable data for a KAREL program.</td>
</tr>
<tr>
<td>.VA</td>
<td><strong>ASCII variable files</strong> are contain the listing of a variable file with variable names, types, and contents.</td>
</tr>
<tr>
<td>.LS</td>
<td><strong>Listing files</strong> are teach pendant programs, error logs, and description histories in ASCII format.</td>
</tr>
</tbody>
</table>

### 9.3 STORAGE DEVICE ACCESS

The KAREL system can access only those storage devices that have been formatted and mounted. These procedures are performed when the devices are first installed on the KAREL system.

The following rules apply when accessing storage devices:

- Formatting a device
  - Deletes any existing data on the device. For example, if you format RD2:, you will also reformat any data existing on RD: thru RD7:.
  - Records a directory on the device
  - Records other data required by the KAREL system
— Assigns a volume name to the device

For more information on formatting a device, refer to the FORMAT_DEV Built-in in Appendix A, "KAREL Language Alphabetical Description" or the FORMAT Command in Appendix C, "KCL Command Alphabetical Description."

### 9.3.1 Overview

The following kinds of storage devices can be used to store programs and files:

- Memory Card (MC:)
- Flash File Storage disk (FR:)
- RAM Disk (RD:) (Not for SpotTool+)
- Ethernet Device (optional)
- Memory Device (MD:)
- Memory Device Binary (MDB:)
- Filtered Memory Device (FMD:)
- USB Memory Stick Device on the controller (UD1:)
- USB Memory Stick Device on the iPendant (UT1:)

This section describes how to set up storage devices for use. Depending on the storage device, this can include

- Setting up a port on the controller
- Connecting the device to the controller
- Formatting a device

**Memory Card (MC:)**

The controller supports memory cards. Memory cards support various sizes 8MB or higher. Compact Flash PC cards are also supported if used with a suitable compact adapter. The memory card requires a memory card interface which is standard on Main CPU inside the controller.

**Note** The controller supports loading software from memory cards.
Warning

Lethal voltage is present in the controller WHENEVER IT IS CONNECTED to a power source. Be extremely careful to avoid electrical shock. HIGH VOLTAGE IS PRESENT at the input side whenever the controller is connected to a power source. Turning the disconnect or circuit breaker to the OFF position removes power from the output side of the device only.

Warning

The memory card interface is located on the Main CPU on the controller cabinet. When the power disconnect circuit breaker is OFF, power is still present inside the controller. Turn off the power disconnect circuit breaker before you insert a memory card into the memory card interface; otherwise, you could injure personnel.

Caution

Do not remove the memory card when the controller is reading or writing to it. Doing so could damage the card and lose all information stored on it.

The memory card can be formatted on the controller, and can be used as a load device to install software.

Note Data on all internal file devices such as FR:, RD:, and MD: should be backed up to external file device such as ATA Flash PC card.

Note The controller formats the card with a sector size of 512 bytes.

The memory card can be formatted and used as an MS-DOS file system. It can be read from and written to the controller and an IBM PC equipped with the proper hardware and software. If the memory card is used as an MS-DOS file system, it should be formatted on the controller.

The controller can read and write memory cards that are formatted with FAT (also referred to as FAT16) or FAT32 type of formatting (File Access Tables). When a memory card is formatted on the Controller it can be formatted as FAT or FAT32 type.

The FAT32 format (32 Bit FAT) removes a few limitations that are included with FAT. One of these is the limitation that only 512 files can be created in the Root directory. Another is that FAT format type only supports memory cards up to 2 GB in size. This feature is included in the controller to increase the compatibility of the robot controller with other computer systems.
USB Memory Stick Device on the controller (UD1:), and USB Memory Stick Device on the iPendant (UT1:)

The controller supports USB Flash memory sticks up to 2GB in size if it is formatted as FAT16. FAT16 is much faster for access. USB Memory sticks with any security or encryption features are not supported. Memory sticks larger than 2GB can not be formatted FAT16, they must be formatted FAT32.

**Note** USB Memory sticks cannot be formatted when plugged into the USB connector on the iPendant (UT1:) They must be formatted on a PC or when plugged into the front panel USB connector on the controller.

**Note** USB Memory sticks with any security or encryption features are not supported.

**Note** Memory sticks larger than 2GB can not be formatted FAT16, they must be formatted FAT32.

**Note** Generally the larger the size of the device such as a USB memory stick, the slower the access speed and device performance. A USB memory stick should be formatted FAT16 if possible. Larger memory sticks formatted as FAT32 will work, but will be slower for file operations and startup of the memory stick when it is first inserted.

**Note** In order to use the UT: device on the iPendant, you must have the USB Port on iPendant option (J957) installed.

**Note** Since USB is a well established specification, FANUC Robotics America Corporation does not qualify specific USB Memory sticks for use with the robot. FANUC Robotics uses USB Flash drives manufactured by SanDisk® (CRUZER Mini and Micro) to qualify the operation of the USB interface. Therefore, FANUC Robotics recommends that you use SanDisk® USB Flash Drives, without security or encryption features. Other drives might work properly but are not specifically qualified by FANUC Robotics.

The USB Memory Stick Device requires a USB interface which is standard on the controller.

The USB memory stick device can be formatted on the controller.

Caution

Do not remove the memory stick when the controller is reading or writing to it. Doing so could damage the memory stick and lose all information stored on it.

Flash File Storage Disk (FR:)

Flash File Storage Disk is a portion of FROM memory that functions as a separate storage device. Flash file storage disk (FR:) does not require battery backup for information to be retained. You can store the following information on Flash file storage disk:

- Programs
• System variables
• Anything you can save as a file

You can format the Flash file storage disk. The size of the Flash file storage disk is set by the system at software installation. Due to the nature of FROM, each time you copy or save a file to the FR: there will be a drop in available FR: memory, even if you are working with the same file.

**RAM Disk (Not for SpotTool+)**

RAM Disk is a portion of Static RAM (SRAM) or DRAM memory that functions as a separate storage device. Any file can be stored on the RAM Disk. RAM Disk files should be copied to an external device for permanent storage.

The location and size of the RAM disk (RD:) depends on the value of the system variable $FILE_MAXSEC. The default value of $FILE_MAXSEC depends on the options and tool packages that are installed.

The value in $FILE_MAXSEC represents the memory size allocated for RD: in 512 byte sectors. For example, a value of -128 means that 64K of memory is allocated in DRAM for RD:.

- If $FILE_MAXSEC > 0, then RAM disk is defined to be in the PERM pool of SRAM. Because RAM disk is a portion of SRAM, copy all RAM disk files to an external device for permanent storage to prevent losing information due to loss of battery power or system software loading.

SRAM is battery-backed volatile memory. This means that all information in SRAM, including programs, requires battery backup for information to be retained when the controller is turned off and then on again. Teach pendant programs are automatically stored in the TPP pool of SRAM when you write a program.

<table>
<thead>
<tr>
<th>Caution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data in SRAM can be lost if the battery is removed or loses its charge, or if new system software is loaded on the controller. To prevent loss of data, back up or copy all files to permanent storage devices such as FR: or ATA Flash PC memory cards.</td>
</tr>
</tbody>
</table>

- If $FILE_MAXSEC < 0, then RAM disk is defined to be in DRAM.

DRAM is non-battery-backed volatile memory. This means that all information in DRAM disappears between power cycles. In effect, DRAM is a temporary device. Information stored in DRAM is lost when you turn off the controller.
Caution

Data in DRAM will be lost if you turn off the controller or if the controller loses power. Do not store anything you want to save beyond the next controller power cycle in DRAM, otherwise, you will lose it.

Note

Volatile means the memory is lost when power is disconnected. Non-volatile memory does not require battery power to retain.

You can store anything that is a file on the RAM Disk. The RAM disk is already formatted for you.

Information stored on RAM disk can be stored as compressed or uncompressed. By default, information is compressed. If you want information to remain uncompressed, you must use the RDU: device designation to indicate that information will be saved to that device in an uncompressed file format.

FTP Ethernet Device

FTP Ethernet devices are used to copy files from the controller to the network PC or workstation if the FTP option is installed. The client devices displayed are the client devices that have been defined and started. Refer to the Internet Options Setup and Operations Manual for more information.

Memory Device (MD:)

The memory device (MD:) treats the controller’s program memory as if it were a file device. You can access all teach pendant programs, KAREL programs, and KAREL variables loaded in the controller.

The Memory Device is a group of devices (MD:, MDB:, and optionally FMD:) that provide the following:

- MD: provides access to ASCII and binary versions of user setup and programs
- MDB: provides access to binary versions of user setup and programs (similar to "backup - all of the above" on the teach pendant file menu)
- FMD: provides access to ASCII versions of user setup and programs filtered to include only user settable information (e.g., internal timers or time system variables changed by the system are not included) making these files useful for detecting user changes.

Memory Device Binary (MDB:)

The memory device binary device (MDB:) allows you to copy the same files as provided by the Backup function on the File Menu. This allows you to back up the controller remotely such as from
SMON, FTP, or KCL. For example, you could use the MDB: device to copy all teach pendant files (including invisible files) to the memory card (KCL>copy MDB:*.* TO mc:).

**Filtered Memory Device (FMD:)**

The Filtered Memory Device option generates text versions of all backup files of user programs and variables that have been changed manually. Included are system and KAREL variables, position and data registers, teach pendant programs, and I/O configuration data.

When logging into the robot FTP server from a remote client you are defaulted into the MD: device. You can navigate to other robot file devices (FR:, RD:, MC:, MDB:, FMD:) using the change directory service in your remote FTP client. At a command line using the `cd` command where in this example fmd: is the device being used, this might look like:

```
D:\temp>ftp pderob029
Connected to pderob029.frc.com
220 FTP server ready.  [PaintTool Vx.xxp/01]
User <pderob029.frc.com:<none>>:
230 User logged in [NORM].
ftp>cd fmd:
250 CWD command successful.
ftp>
```

You can compare these files with previous versions to determine what users or operators have changed. Variables and programs that change without user input are filtered out, and will appear in filter exclusion files.

After the option is installed, it will run automatically whenever you perform an Ethernet backup of the controller from the FMD: device. After you install the Filtered Memory Device option, any of the following filter exclusion files could appear on the FR: device.

**Caution**

Do not delete these files, or filter exclusion data will be lost.

- FR:SVAREEG.DT
- FR:KVAREEG.DT
- FR:POSREEG.DT
- FR:REGEEG.DT
- FR:TPLINEEG.DT

You can view program, variable, or filter exclusion files via KCL. For example:
KCL> DIR FMD:*.*.

**Note** Computer systems that perform periodic backups could be modified to use the FMD: device instead of the MD: device for some compare operations, for example. Contact FANUC Robotics for more information.

**FRA:**

There is a special area for Automatic Backup in the controller F-ROM (FRA:). You do not need an external device to use Automatic Backup, but a memory card can also be used. Refer to your application-specific Setup and Operations Manual for more information.

### 9.3.2 Memory File Devices

The RAM and F-ROM disks allocate files using blocks. Each block is 512 bytes.

The system variable $FILE_MAXSEC specifies the number of blocks to allocate for the RAM disk. If the specified number is less than zero, the RAM disk is allocated from DRAM. If it is greater than zero, RAM disk is allocated from CMOS RAM. To change the number of blocks to allocate for the RAM disk, perform the following steps from the KCL prompt:

1. Backup all files on the RAM disk. For more information on how to back up files, refer to Chapter 8, "Program and File Manipulation" in the appropriate application-specific FANUC Robotics Setup and Operations Manual.

2. Enter DISMOUNT RD:

   KCL>DISMOUNT RD:

3. Enter SET VAR $FILE_MAXSEC

   KCL>SET VAR
   $FILE_MAXSEC = <new value>

4. Enter FORMAT RD:

   KCL>FORMAT RD:
All files will be removed from the RAM Disk when the format is performed.

5. Enter MOUNT RD:

KCL>MOUNT RD:

The RAM disk will be reformatted automatically on INIT start.

The F-ROM disk can only be formatted from the BootROM because the system software also resides on F-ROM. The number of blocks available is set by the system. The hardware supports a limited number of read and write cycles, so while the F-ROM disk will function similar to the RAM disk, it does not erase files that have been deleted or overwritten.

After some use, the F-ROM disk will have used up all blocks. At that time, a purge is required to erase the F-ROM blocks which are no longer needed. For more information on purging, refer to the PURGE_DEV Built-in in Appendix A, "KAREL Language Alphabetical Descriptions" or the PURGE Command in Appendix C, "KCL Command Alphabetical Description."

For more information on memory, refer to Section 1.4.1

### 9.3.3 Virtual Devices

KAREL Virtual Devices are similar to DOS subdirectories. For example

- In DOS, to access a file in a subdirectory, you would view FR:FR1:\>test.kl.
- In KAREL, to access the same file in a virtual device, you would view FR1:test.kl.

The controller supports 7 virtual devices. A number, which identifies the virtual device, is appended to the device name (FR 1 :). Table 9–2 shows some of the valid virtual devices available.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Actual Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD:</td>
<td>RAM disk</td>
</tr>
<tr>
<td>FR:</td>
<td>F-ROM disk - compressed and uncompressed files</td>
</tr>
<tr>
<td>MF:</td>
<td>Refers to files on both RD: and FR:</td>
</tr>
<tr>
<td>RD1: - RD7:</td>
<td>RAM disk - compressed and uncompressed files</td>
</tr>
</tbody>
</table>
Table 9–2.  Virtual Devices (Cont’d)

| FR1: - FR7: | F-ROM disk - compressed and uncompressed files |
| MF1: - MF7: | Refers to files on both the RAM disk and F-ROM disk of the respective virtual device |

Rules for Virtual Devices

The following rules apply to virtual devices.

- A file name on a virtual device is unique. A file could exist on either the RAM or F-ROM disks, but not both. For example: RD:test.kl and FR:test.kl could not both exist.

- A file name could be duplicated across virtual devices. For example: RD:test.kl, RD1:test.kl, and FR2:test.kl could all exist.

- The MF: device name could be used in any file operation to find a file on a virtual device, when the actual storage device is unknown. For example: MF:test.kl finds either RD:test.kl or FR:test.kl.

- When you use the MF: device as a storage device, the RAM disk is used by default when RD: is in CMOS and $FILE_MAXSEC > 0. The F-ROM disk is used by default when RD: is in DRAM and $FILE_MAXSEC < 0. For example: KCL>COPY FILE FLPY:test.kl to MF2: . The file will actually exist on RD2:

- When listing the MF: device directory, all files on the RAM and F-ROM disks are listed. However, only the files in the specified virtual device are displayed.

- If the RD5: directory is specified instead of MF5:, only those files on the RAM disk in virtual device 5 are listed. If the FR3: directory is specified, only those files on the F-ROM disk in virtual device 3 are listed. For example: KCL>DIR RD5:

- A file could be copied from one virtual device to another virtual device. A file could also be copied from the RAM disk to the F-ROM disk, and vice versa, if the virtual device is different. For example: KCL>COPY RD1:test.kl to FR3:

- A file could be renamed only within a virtual device and only on the same device. For example: KCL>RENAME FR2:test.kl FR2:example.kl

- A file could be moved within a virtual device from the RAM disk to the F-ROM disk and vice versa, using a special command which is different from copy. For example: KCL>MOVE MF1:test.kl moves test.kl from the F-ROM disk to the RAM disk. KCL>COPY FR1:test.kl TO RD1:test.kl will also move the file from the F-ROM Disk to the RAM Disk. This is because unique file names can only exist on one device. For more information on moving files, refer to the MOVE_FILE Built-in in Appendix A, "KAREL Language Alphabetical Descriptions" or the MOVE FILE Command in Appendix C, "KCL Command Alphabetical Description."

- Formatting the RAM disk, RD: or MF:, clears all the RAM disk files on all the virtual devices. The files on the F-ROM disk remain intact. For example: KCL>FORMAT RD1: reformats all RAM disk virtual devices (RD: through RD7:). Reformattting will cause existing data to be removed.
9. FILE SYSTEM

- Purge erases all blocks that are no longer needed for all the virtual devices. For more information on purging, refer to the PURGE_DEV Built-in in Appendix A, "KAREL Language Alphabetical Description" or the PURGE Command in Appendix C, "KCL Command Alphabetical Description."

9.3.4 File Pipes

The PIP: device allows you to access any number of pipe files. This access is to files that are in the controller’s memory. This means that the access to these files is very efficient. The size of the files and number of files are limited by available controller memory. This means that the best use of a file pipe is to buffer data or temporarily hold it.

The file resembles a water pipe where data is poured into one end by the writing task and the data flows out the other end into the reading task. This is why the term used is a pipe. This concept is very similar to pipe devices implemented on UNIX, Windows and Linux.

Files on the pipe device have a limited size but the data is arranged in a circular buffer. This is also called a circular queue. This means that a file pipe of size 8kbytes (this is the default size) will contain the last 8k of data written to the pipe. When the user writes the ninth kilobyte of data to the pipe, the first kilobyte will be overwritten.

Since a pipe is really used to transfer data from one place to another some application will be reading the data out of the pipe. In the default mode, the reader will WAIT until information has been written. Once the data is available in the pipe the read will complete. A KAREL application might use BYTES_AHEAD to query the pipe for the amount of data available to read. This is the default read mode.

A second read mode is provided which is called "snapshot." In this mode the reader will read out the current content of the pipe. Once the current content is read the reader receives an end of the file. This can be applied in an application like a "flight recorder". This allows you to record information leading up to an event (such as an error) and then retrieve the last set of debug information written to the pipe. Snapshot mode is a read attribute. It is configured using SET_FILE_ATTR builtin. By default, the read operation is not in snapshot mode.

Typical pipe applications involve one process writing data to a pipe. The data can debug information, process parameters or robot positions. The data can then be read out of the pipe by another application. The reading application can be a KAREL program which is analyzing the data coming out of the pipe or it can be KCL or the web server reading the data out and displaying it to the user in ASCII form.

KAREL Examples

The following apply to KAREL examples.

- Two KAREL tasks can share data through a pipe. One KAREL task can write data to the pipe while a second KAREL task reads from the pipe. In this case the file attribute ATR_PIPWAIT can be used for the task that is reading from the pipe. In this case the reading KAREL task will wait.
on the read function until the write task has finished writing the data. The default operation of the pipe is to return an end of file when there is no data to be read from the pipe.

- A KAREL application might be executing condition handlers at a very fast data rate. In this case it might not be feasible for the condition handler routine to write data out to the teach pendant display screen because this would interfere with the performance of the condition handler. In this case you could write the data to the PIP: device from the condition handler routine. Another KAREL task might read the data from the PIP: device and display it to the teach pendant. In this case the teach pendant display would not be strictly real time. The PIP: device acts as a buffer in this case so that the condition handler can move on to its primary function without waiting for the display to complete. You can also type the file from KCL at the same time the application is writing to it.

PIP: devices are similar to other devices in the following ways:

- The pipe device is similar in some ways to the RD: device. The RD: device also puts the file content in the system memory. The PIP device is different primarily because the pipe file can be opened simultaneously for read and write.
- Similarly to MC: and FR: devices, the PIP: device is used when you want to debug or diagnose real time software. This allows you to output debug information that you can easily view without interfering with the operation that is writing the debug data. This also allows one task to write information that another task can read.
- The function of the PIP: device is similar to all other devices on the controller. This means that all file I/O operations are supported on this device. All I/O functions are supported and work the same except the following: Chdir, Mkdir, and Rmdir.
- The PIP: device is similar to writing directly to a memory card. However, writing to a memory card will delay the writing task while the delay to the PIP: device is much smaller. This means that any code on the controller can use this device. It also has the ability to retain data through a power cycle.

**Rules for PIP: Devices**

The following rules apply to PIP: devices:

- The PIP: device can be used by any application or you can specify an associated common option such as KAREL.
- The device is configurable. You can configure how much memory it uses and whether that memory is CMOS (retained) or DRAM (not retained). You are also able to configure the format of the data in order to read out formatted ASCII type data. The device is configured via the PIPE_CONFIG built-in.

**Installation, Setup and Operation Sequence**

In general the PIP: device operates like any other device. A typical operation sequence includes:

```plaintext
OPEN myfile ('PIP:/myfile.dat', 'RW',)
Write myfile ('Data that I am logging', CR)
```
Close myfile

If you want to be able to access myfile.dat from the Web server, put a link to it on the diagnostic Web page.

The files on the PIP: device are configurable. By default the pipe configuration is specified in the $PIPE_CONFIG system variable. The fields listed in Table 9–3 have the following meanings:

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DEFAULT</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sectors</td>
<td>8</td>
<td>Number of 1024 byte sectors in the pipe.</td>
</tr>
<tr>
<td>$filedata</td>
<td></td>
<td>Pointer to the actual pipe data (not accessible).</td>
</tr>
<tr>
<td>$recordsize</td>
<td>0</td>
<td>Binary record size, zero means its not tracked.</td>
</tr>
<tr>
<td>$auxword</td>
<td>0</td>
<td>Dictionary element if dictionary format or type checksum.</td>
</tr>
<tr>
<td>$memtyp</td>
<td>0</td>
<td>If non zero use CMOS.</td>
</tr>
<tr>
<td>$format</td>
<td>Undefined</td>
<td>Formatting mode: undefined, function, format string or KAREL type.</td>
</tr>
<tr>
<td>$formatter</td>
<td></td>
<td>Function pointer, &quot;C&quot; format specifier pointer or type code depending on $format.</td>
</tr>
</tbody>
</table>

Each pipe file can be configured via the pipe_config built-in. The pipe_config built-in will be called before the pipe file is opened for writing. Refer to Section A.17, "pipe_config built-in" for more details.

**Operational Examples**

The following example writes data from one KAREL routine into a pipe and then reads it back from another routine. These routines can be called from separate tasks so that one task was writing the data and another task can read the data.

**Program**

```karel
program pipuform
  %nolockgroup
  var
    pipe, in_file, mcfile, console:file
    record: string[80]
    status: integer
    parm1, parm2: integer
    msg: string[127]
```

9–18
--initialize file attributes
routine file_init (att_file:FILE)
begin
  set_file_atr(att_file, ATR_IA) --force reads to completion
  set_file_atr(att_file, ATR_FIELD) --force write to completion
  set_file_atr(att_file, ATR_PASSALL) --ignore cr
  set_file_atr(att_file, ATR_UF) --binary
end file_init
routine write_pipe
begin
  --file is opened
  file_init (pipe)
  open file pipe (‘rw’, ‘pip:example.dat’)
  status = io_status(pipe)
  write console (‘Open pipe status:’, status, cr)
  --write extra parameters to pipe
  write pipe (msg::8)
  status = io_status(pipe)
end write_pipe
routine read_pipe
var
  record: string[128]
  status: integer
  entry: integer
  num_bytes: integer
begin
  file_init (in_file)
  open file in_file (‘ro’, ‘pip:example.dat’)
  BYTES_AHEAD(in_file, entry, status)
  status = 0
  read in_file (parm1::4)
  status = IO_STATUS(in_file)
  write console (‘parm1 read’, status, cr)
  write console (‘parm1’, parm1, cr)
  read in_file (parm2::4)
  status = IO_STATUS(in_file)
  write console (‘parm2 read’, parm2, status, cr)
end read_pipe
begin
  SET_FILE_ATR(console, atr_ia, 0) --ATR_IA is defined in flbt.ke
  OPEN FILE console (‘RW’, ‘CONS:’)
if(uninit(msg)) then
    msg = ‘Example’
endif
if(uninit(n_sectors)) then
    cmos_flag = true
    n_sectors = 16
    record_size = 128
    form_dict = ‘test’
    form_ele = 1
endif
-- [in] pipe_name: STRING; name of tag
-- [in] cmos_flag: boolean;
-- [in] n_sectors: integer;
-- [in] record_size: integer;
-- [in] form_dict: string;
-- [in] form_ele: integer;
-- [out] status: INTEGER
pipe_config(‘pip:example.dat’, cmos_flag, n_sectors,
            record_size, form_dict, form_ele, status)
write_pipe
read_pipe
close file pipe
close file in_file
end pipuform

9.4 FILE ACCESS

You can access files using the FILE and SELECT screens on the CRT/KB or teach pendant, or by using KAREL language statements. During normal operations, files will be loaded automatically into the controller. However, other functions could need to be performed.

9.5 FORMATTING XML INPUT

9.5.1 Overview

This feature allows KAREL programs to input data via an XML (eXtended Markup Language) formatted text file. The XML rather than binary format allows the file to be manipulated easily on a PC.

The XML files must follow the most basic XML syntax requirements. These requirements are:
• XML files can have ONLY ONE top level element.
• The start tag must have a matching end tag.
• Empty tags can be represented as <tag parameters/>
• Tags cannot contain special characters such as the set of *, $, and [ ]
• They must not contain unprintable characters
• Attributes must be of the form attr="value"
• Special characters are used for the following (outside of tags):
  — < is substituted with &lt;
  — > is substituted with &gt;
  — & is substituted with &amp;
  — " can be substituted with &quot;
• This feature provides an XML parser and the means for both KAREL and C programmers to easily extract binary data from the text information in an XML file. It does not require the application program to do any parsing of the XML file.

Note XML files can have only one top level element. For example,

```
<GRID>
  <TPPROG>
    </TPPROG>
  </GRID>
```

is legal. It has one top level element (GRID).

```
<GRID>
  </GRID>
<TPPROG>
  </TPPROG>
```

is not legal. The master tag can be used to distinguish a GRID file from a password configuration file, for example.

### 9.5.2 Installation Sequence

This feature consists of KAREL built-ins which provide access to this library for KAREL users. The environment file xml.ev must be on the translator path to translate KAREL programs which reference these built-ins. These built-ins are XML_ADDTAG, XML_GETDATA, XML_REMTAG, XML_SCAN, and XML_SETVAR. Refer to Appendix A for more information on these built-ins.
9.5.3 Example KAREL Program Referencing an XML File

- Parse the XML file referred to by xml_name and return the settings in that file to xmlstruct.
- The attribute name-value pairs are returned as strings in atrtnames and attrvalues. It is not required that the data in the XML file be set to a structure in some applications the name-value pairs are used directly.
- The most efficient XML implementation uses many name-value pairs and only a few tags. It takes the same amount of time to return one name-value pair from a tag as it takes for 32 pairs. Thirty-two tags will take 32 times longer.
- The maximum number of pairs supported is 32.
- There are two different types of XML files. Figure 1–1 and Figure 1–2 illustrate the two types of tag constructs:
  - For separated start and end tags (Figure 1–1) the tag processing must be done on the XML_START return code.
  - For combined start and end tags (Figure 1–2) you cannot provide any text within the tag. KAREL XML processing provides the means to extract this text when required.
  - For combined start and end tags (Figure 1–2) the tag processing must be done on the XML_END return code.
  - The XML_START return code needs to set a flag indicating that the tag has been processed.
  - The XML_END return code needs to check to see if processing was already done on the start code and reset the flag.

Figure 9–1. XML File 1 Separated start and end tags
`<?xml version="1.0" ?><!-- This is a comment -- >
<xmlstruct_t first="123456" second="7.8910" third="1" fourth="A string"> Text associated with xmlstruct_t tag </xmlstruct_t>`

Figure 9–2. XML File 2 Combined start and end tags
`<?xml version="1.0" ?><!-- This is a comment -- >
<xmlstruct_t first="78910" second="12.3456" third="0" fourth="A string"/>`

Figure 9–3. XML File 3 GRID tag not registered or processed
`<?xml version="1.0" ?>-<!--comment
<GRID rows="16" cols="24" scale="80">
    <xmlstruct_t first="123456" second="78910" third="1" fourth="A String">
        special characters < > & "
    </xmlstruct_t>
</GRID>"`
The GRID tag can be in the XML file but not processed by this example program. In general XML tags can be processed by different software. Information is only returned to the KAREL program for tags which are registered by the KAREL program.

Figure 9–4. KAREL Program

---

```
PROGRAM xmlparse
%COMMENT = 'XML Parse'

%NOPAUSE_SHFT
%NOPAUSE = ERROR + TPENABLE + COMMAND
%NOABORT = ERROR + COMMAND
%NOLOCKGROUP
%NOBUSYLAMP
%ENVIRONMENT xml
%include klerxmlf
CONST
MYXML_CONST = 3
TYPE
    xmlstruct_t = STRUCTURE
        first: INTEGER
        second: REAL
        third: BOOLEAN
        fourth: STRING[20]
    ENDSTRUCTURE
VAR
    xml_name : string[20]
    tag_name : string[32]
    text : array[32] of string[128]
    attrnames : array[32] of string[32]
    attrvalues : array[32] of string[64]
    xml_file : FILE
    status : INTEGER
    xmlstruct: xmlstruct_t
    tag_ident: integer
    func_code: integer
    text_idx: integer
    numattr: integer
    textdone: BOOLEAN
    done: BOOLEAN
    console: FILE
    startdata: BOOLEAN
```

---

--

-- There are two types of XML file constructs. In one the end tag is embedded in the start tag in the other the end tag is separate. A proper parser must handle both tag constructs

---
-- For the case that the end tag is separate from the start tag
-- (Figure 1.1) the following writes show the sequence of returns:
-- Scanned (Rev D) xmlstrct_t 3 100 129015
-- Start Tag processing...
-- Scanned (Rev D) xmlstrct_t 3 101 129015
-- End Tag
-- Processed at start tag...
-- Scanned (Rev D) xmlstrct_t 0 101 0
--
-- For the case where the end tag and start tag are together (Figure 1.2)
-- the following writes show the sequences of returns:
-- Scanned (Rev D) xmlstrct_t 3 101 129015
-- End Tag
-- End Tag processing...
-- Scanned (Rev D) xmlstrct_t 0 101 0
--
BEGIN
SET_FILE_ATR(console, ATR_IA, 0) -- ATR_IA is defined in flbt.ke
OPEN FILE console ('RW', 'CONS: ')
IF UNINIT(xml_name) THEN
    xml_name = 'mc:kl16004.xml'
ENDIF
SET_FILE_ATR(xml_file, ATR_XML) -- XML
CLR_IO_STAT(xml_file)
OPEN FILE xml_file ('RO', xml_name) -- Open does new operation
status = IO_STATUS(xml_file)
IF status <> 0 THEN
    POST_ERR(status, '', 0, 0)
    abort
ENDIF
xml_addtag(xml_file, 'xmlstrct_t', 32, FALSE, MYXML_CONST, status)
textdone = TRUE
done = FALSE
startdata=FALSE
WHILE (done = FALSE) DO
    xml_scan(xml_file, tag_name, tag_ident, func_code, status)
    if (status = 0) THEN
        done= TRUE
    ENDIF
WRITE console ('Scanned (Rev D) ', tag_name,' ', tag_ident, ' ',
    func_code, ' ',STATUS,' ', CR)
IF (status = XML_FUNCTION) THEN
    status = 0
    SELECT tag_ident OF
    CASE (MYXML_CONST) :
        SELECT func_code OF
CASE (XML_START) :
  WRITE console ('Start Tag processing...', CR)
  text_idx = 1
  xml_setvar(xml_file, 'kl16004', 'xmlstrct', status)
-- Already looked at the attributes get the text
  xml_getdata(xml_file, numattr, attrnames, attrvalues,
             text[text_idx], textdone, status)
  startdata = TRUE
CASE (XML_STEND) :
-- This tag is never returned
  WRITE console ('StEnd Tag', CR)
CASE (XML_END) :
  WRITE console ('End Tag', CR)
  if(startdata = TRUE) THEN
    startdata=FALSE
    WRITE console ('Processed at start tag...', CR)
  ELSE
    WRITE console ('End Tag processing...', CR)
    text_idx = 1
    xml_setvar(xml_file, 'kl16004', 'xmlstrct', status)
-- Already looked at the attributes get the text
    xml_getdata(xml_file, numattr, attrnames, attrvalues,
                text[text_idx], textdone, status)
  ENDIF
CASE (XML_TXCONT) :
-- Usually the user will do one or the other but not both of
-- these calls
  text_idx = text_idx + 1
  xml_getdata(xml_file, numattr, attrnames, attrvalues,
              text[text_idx], textdone, status)
ELSE:
ENDSELECT
ELSE:
ENDSELECT
ELSE:
  IF(status <> XML_SCANLIM) THEN
    POST_ERR(status, ',', 0, 0)
    done = TRUE
  ENDIF
ENDIF -- Good status from xml_parse
ENDWHILE
-- This is not required but allows the user to dynamically remove
-- and add tags
xml_remtag(xml_file, 'xmlstrct_t', status)
CLOSE FILE xml_file
status = IO_STATUS(xml_file)
IF status <> 0 THEN
Executing this program will extract the attributes first, second, third, and fourth, and their values from the XML file. These values will be set in the variable xmlstruct that has fields first, second, third, and fourth. The string variables will also be set to KAREL string variables.

### 9.5.4 Parse Errors

- `XML_TAG_SIZE "Tag too long"
- `XML_ATTR_SIZE "Attribute too long"
- `XML_NOSLASH "Invalid use of / character"
- `XML_INVTAG "Invalid character in tag"
- `XML_ATTRMATCH "No value for attribute"
- `XML_TAGMATCH "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"
- `XML_BADEXCHAR "Unknown character &xxx;"
- `XML_TAGNFND "Tag not found"
- `XML_INVEOF "Unexpected end of file"
- `XML_SCANLIM "Scan limit exceeded"

**Note** `XML_SCANLIM` means that the file is too long to be processed in one request. The remedy for this error is to just re-call the XML scan routine as illustrated in the example.

- `XML_FUNCTION "Function code return"
9.6 MEMORY DEVICE

The Memory device (MD:) treats controller memory programs and variable memory as if it were a file device. Teach pendant programs, KAREL programs, program variables, SYSTEM variables, and error logs are treated as individual files. This provides expanded functions to communication devices, as well as normal file devices. For example:

1. FTP can load a PC file by copying it to the MD: device.
2. The error log can be retrieved and analyzed remotely by copying from the MD: device.
3. An ASCII listing of teach pendant programs can be obtained by copying ***.LS from the MD: device.
4. An ASCII listing of system variables can be obtained by copying SYSVARS.VA from the MD: device.

Refer to Table 9–4 for listings and descriptions of files available on the MD device.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCNTG.DG</td>
<td>This file shows the system accounting of Operating system tasks.</td>
</tr>
<tr>
<td>ACCOFF.DG</td>
<td>This file shows the system accounting is turned off.</td>
</tr>
<tr>
<td>AXIS.DG</td>
<td>This file shows the Axis and Servo Status.</td>
</tr>
<tr>
<td>CONFIG.DG</td>
<td>This file shows a summary of system configuration</td>
</tr>
<tr>
<td>CONSLOG.DG</td>
<td>This file is an ASCII listing of the system console log.</td>
</tr>
<tr>
<td>CONSTAIL.DG</td>
<td>This file is an ASCII listing of the last lines of the system console log.</td>
</tr>
<tr>
<td>CURPOS.DG</td>
<td>This file shows the current robot position.</td>
</tr>
<tr>
<td>*.DF</td>
<td>This file contains the TP editor default setting.</td>
</tr>
<tr>
<td>DIOCFGSV.IO</td>
<td>This file contains I/O configuration information in binary form.</td>
</tr>
<tr>
<td>DIOCFGSV.VA</td>
<td>This file is an ASCII listing of DIOCFGSV.IO.</td>
</tr>
<tr>
<td>ERRACT.LS</td>
<td>This file is an ASCII listing of active errors.</td>
</tr>
<tr>
<td>ERRALL.LS</td>
<td>This file is an ASCII listing of error logs.</td>
</tr>
<tr>
<td>ERRAPP.LS</td>
<td>This file is an ASCII listing of application errors.</td>
</tr>
<tr>
<td>ERRCOMM.LS</td>
<td>This file shows communication errors.</td>
</tr>
<tr>
<td>ERRCURR.LS</td>
<td>This file is an ASCII listing of system configuration.</td>
</tr>
<tr>
<td>ERRHIST.LS</td>
<td>This file is an ASCII listing of system configuration.</td>
</tr>
<tr>
<td>ERRMOT.LS</td>
<td>This file is an ASCII listing of motion errors.</td>
</tr>
<tr>
<td>ERRPWD.LS</td>
<td>This file is an ASCII listing of password errors.</td>
</tr>
</tbody>
</table>
### Table 9-4. File Listings for the MD Device (Cont’d)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRSYS.LS</td>
<td>This file is an ASCII listing of system errors.</td>
</tr>
<tr>
<td>ETHERNET</td>
<td>This file shows the Ethernet Configuration.</td>
</tr>
<tr>
<td>FRAME.DG</td>
<td>This file shows Frame assignments.</td>
</tr>
<tr>
<td>FRAMEVAR.VR</td>
<td>This file contains system frame and tool variable information in binary form.</td>
</tr>
<tr>
<td>FRAMEVAR.VA</td>
<td>This file is an ASCII listing of FRAMEVAR.VR.</td>
</tr>
<tr>
<td>HIST.LS</td>
<td>This file shows history register dumps.</td>
</tr>
<tr>
<td>HISTE.LS</td>
<td>This file is an ASCII listing of general fault exceptions.</td>
</tr>
<tr>
<td>HISTP.LS</td>
<td>This file is an ASCII listing of powerfail exceptions.</td>
</tr>
<tr>
<td>HISTS.LS</td>
<td>This file is an ASCII listing of servo exceptions.</td>
</tr>
<tr>
<td>IOCONFIG.DG</td>
<td>This file shows IO configuration and assignments.</td>
</tr>
<tr>
<td>IOSTATE.DG</td>
<td>This file is an ASCII listing of the state of the I/O points.</td>
</tr>
<tr>
<td>IOSTATUS.CM</td>
<td>This file is a system command file used to restore I/O.</td>
</tr>
<tr>
<td>LOG CONSTAIL.DG</td>
<td>This file is the last line of Console Log.</td>
</tr>
<tr>
<td>NUMREG.VA</td>
<td>This file is an ASCII listing of NUMREG.VR.</td>
</tr>
<tr>
<td>NUMREG.VR</td>
<td>This file contains system numeric registers.</td>
</tr>
<tr>
<td>MACRO.DG</td>
<td>This file shows the Macro Assignment.</td>
</tr>
<tr>
<td>MEMORY.DG</td>
<td>This file shows current memory usage.</td>
</tr>
<tr>
<td>PORT.DG</td>
<td>This file shows the Serial Port Configuration.</td>
</tr>
<tr>
<td>POSREG.VA</td>
<td>This file is an ASCII listing of POSREG.VR.</td>
</tr>
<tr>
<td>POSREG.VR</td>
<td>This file contains system position register information.</td>
</tr>
<tr>
<td>PRGSTATE.DG</td>
<td>This file is an ASCII listing of the state of the programs.</td>
</tr>
<tr>
<td>RIPELOG.DG</td>
<td>This file contains detailed status information such as the times when robots go ON and OFFLINE, and other diagnostic data. Refer to the <em>Internet Options Manual</em> for more information.</td>
</tr>
<tr>
<td>RIPESTAT.DG</td>
<td>This file contains performance data for you to determine how well the network is performing. Refer to the <em>Internet Options Manual</em> for more information.</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SFTYSIG.DG</td>
<td>This file is an ASCII listing of the state of the safety signals.</td>
</tr>
<tr>
<td>STATUS.DG</td>
<td>This file shows a summary of system status.</td>
</tr>
<tr>
<td>SUMMARY.DG</td>
<td>This file shows diagnostic summaries.</td>
</tr>
<tr>
<td>SYCLDINT.VA</td>
<td>This file is an ASCII listing of system variables initialized at a Cold start.</td>
</tr>
<tr>
<td>SYMOTN.VA</td>
<td>This file is an ASCII listing of motion system variables.</td>
</tr>
<tr>
<td>SYNOSAVE.VA</td>
<td>This file is an ASCII listing of non-saved system variables.</td>
</tr>
<tr>
<td>SYSFRAME.SV</td>
<td>This file contains $MNUTOOL, $MNUFRAME, $MNUTOOLNUM, and $MNUFRAMENUM. These variables were in SYSVARS.SV in releases before V7.20.</td>
</tr>
<tr>
<td>SYSMACRO.SV</td>
<td>This file is a listing of system macro definitions.</td>
</tr>
<tr>
<td>SYSMACRO.VA</td>
<td>This file is an ASCII listing of SYSMACRO.SV.</td>
</tr>
<tr>
<td>SYSMAST.SV</td>
<td>This file is a listing of system mastering information.</td>
</tr>
<tr>
<td>SYSMAST.VA</td>
<td>This file is an ASCII listing of SYSMAST.SV.</td>
</tr>
<tr>
<td>SYSSERVO.SV</td>
<td>This file is a listing of system servo parameters.</td>
</tr>
<tr>
<td>SYSSERVO.VA</td>
<td>This file is an ASCII listing of SYSSERVO.SV.</td>
</tr>
<tr>
<td>SYSTEM.DG</td>
<td>This file shows a summary of system information.</td>
</tr>
<tr>
<td>SYSTEM.VA</td>
<td>This file is an ASCII listing of non motion system variables.</td>
</tr>
<tr>
<td>SYSVARS.SV</td>
<td>This file is a listing of system variables.</td>
</tr>
<tr>
<td>SYSVARS.VA</td>
<td>This file is an ASCII listing of SYSVARS.SV.</td>
</tr>
<tr>
<td>SYS****.SV</td>
<td>This file contains application specific system variables.</td>
</tr>
<tr>
<td>SYS****.VA</td>
<td>This file is an ASCII listing of SYS****.VA.</td>
</tr>
<tr>
<td>TASKLIST.DG</td>
<td>This file shows the system task information.</td>
</tr>
<tr>
<td>TESTRUN.DG</td>
<td>This file shows the Testrun Status.</td>
</tr>
<tr>
<td>TIMERS.DG</td>
<td>This file shows the System and Program Timer Status.</td>
</tr>
<tr>
<td>TPACCN.DG</td>
<td>This file shows TP Accounting Status.</td>
</tr>
<tr>
<td>VERSION.DG</td>
<td>This file shows System, Software, and Servo Version Information.</td>
</tr>
<tr>
<td>***.PC</td>
<td>This file is a KAREL binary program.</td>
</tr>
<tr>
<td>***.VA</td>
<td>This file is an ASCII listing of KAREL variables.</td>
</tr>
<tr>
<td>***.VR</td>
<td>This file contains KAREL variables in binary form.</td>
</tr>
<tr>
<td>***.LS</td>
<td>This file is an ASCII listing of a teach pendant program.</td>
</tr>
<tr>
<td>***.TP</td>
<td>This file is a teach pendant binary program.</td>
</tr>
</tbody>
</table>
### Table 9–4. File Listings for the MD Device (Cont’d)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>***.TX</td>
<td>This file is a dictionary file.</td>
</tr>
<tr>
<td>***.HTM</td>
<td>This file is an HTML web page.</td>
</tr>
<tr>
<td>***.STM</td>
<td>This file is an HTML web page using an iPendant Control or Server Side Include.</td>
</tr>
<tr>
<td>***.GIF</td>
<td>This file is a GIF image file.</td>
</tr>
<tr>
<td>***.JPG</td>
<td>This file is a JPEG image file.</td>
</tr>
</tbody>
</table>

Refer to Table 9–5 for a listing of restrictions when using the MD: device.

### Table 9–5. Testing Restrictions when Using the MD: Device

<table>
<thead>
<tr>
<th>File Name or Type</th>
<th>READ</th>
<th>WRITE</th>
<th>DELETE</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>***.DG</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>Diagnostic text file.</td>
</tr>
<tr>
<td>***.PC</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>***.VR</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>With restriction of no references.</td>
</tr>
<tr>
<td>***.LS</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>***.TP</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>***.LS</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>FFF.DF</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>SYS***.SV</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>Write only at CTRL START.</td>
</tr>
<tr>
<td>SYS***.VA</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ERR***.LS</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>HISTX.LS</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>***.REG.VR</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>***.REG.VA</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DIOCFGHSV.IO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>Write only at CTRL START.</td>
</tr>
<tr>
<td>DIOCFGHSV.VA</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 10

DICTIONARIES AND FORMS

Contents

Chapter 10 DICTIONARIES AND FORMS ................................................................. 10–1
10.1 OVERVIEW .................................................................................................. 10–3
10.2 CREATING USER DICTIONARIES .............................................................. 10–3
  10.2.1 Dictionary Syntax ..................................................................................... 10–3
  10.2.2 Dictionary Element Number ..................................................................... 10–4
  10.2.3 Dictionary Element Name ......................................................................... 10–5
  10.2.4 Dictionary Cursor Positioning ................................................................... 10–5
  10.2.5 Dictionary Element Text ........................................................................... 10–6
  10.2.6 Dictionary Reserved Word Commands ...................................................... 10–8
  10.2.7 Character Codes ....................................................................................... 10–10
  10.2.8 Nesting Dictionary Elements ..................................................................... 10–10
  10.2.9 Dictionary Comment .................................................................................. 10–11
  10.2.10 Generating a KAREL Constant File ......................................................... 10–11
  10.2.11 Compressing and Loading Dictionaries on the Controller ......................... 10–11
  10.2.12 Accessing Dictionary Elements from a KAREL Program ......................... 10–12
10.3 CREATING USER FORMS ......................................................................... 10–13
  10.3.1 Form Syntax .............................................................................................. 10–14
  10.3.2 Form Attributes ........................................................................................ 10–15
  10.3.3 Form Title and Menu Label ....................................................................... 10–16
  10.3.4 Form Menu Text ........................................................................................ 10–17
  10.3.5 Form Selectable Menu Item ....................................................................... 10–18
  10.3.6 Edit Data Item ............................................................................................ 10–19
  10.3.7 Dynamic Forms using Tree View ............................................................... 10–26
  10.3.8 Non-Selectable Text ................................................................................... 10–27
  10.3.9 Display Only Data Items .......................................................................... 10–27
  10.3.10 Cursor Position Attributes ...................................................................... 10–27
  10.3.11 Form Reserved Words and Character Codes .............................................. 10–28
  10.3.12 Form Function Key Element Name or Number ........................................... 10–30
  10.3.13 Form Function Key Using a Variable ....................................................... 10–31
  10.3.14 Form Help Element Name or Number ...................................................... 10–31
10.3.15  Teach Pendant Form Screen ................................................................. 10–32
10.3.16  CRT/KB Form Screen ............................................................................. 10–32
10.3.17  Form File Naming Convention .............................................................. 10–33
10.3.18  Compressing and Loading Forms on the Controller .............................. 10–34
10.3.19  Displaying a Form .................................................................................. 10–36
10.1 OVERVIEW

Dictionaries and forms are used to create operator interfaces on the teach pendant and CRT/KB screens with KAREL programs.

This chapter includes information about

- Creating user dictionary files, refer to Section 10.2.
- Creating and using forms, refer to Section 10.3.

In both cases, the text and format of a screen exists outside of the KAREL program. This allows easy modification of screens without altering KAREL programs.

10.2 CREATING USER DICTIONARIES

A dictionary file provides a method for customizing displayed text, including the text attributes (blinking, underline, double wide, etc.), and the text location on the screen, without having to re-translate the program.

The following are steps for using dictionaries.

1. Create a formatted ASCII dictionary text file with a .UTX file extension.
2. Compress the dictionary file using the KCL COMPRESS DICT command. This creates a loadable dictionary file with a .TX extension. Once compressed, the .UTX file can be removed from the system. **Only the compressed dictionary (.TX) file is loaded.**
3. Load the compressed dictionary file using the KCL LOAD DICT command or the KAREL ADD_DICT built-in.
4. Use the KAREL dictionary built-Ins to display the dictionary text. Refer to Section 10.2.12, "Accessing Dictionary Elements from a KAREL Program," for more information.

Dictionary files are useful for running the same program on different robots, when the text displayed on each robot is slightly different. For example, if a program runs on only one robot, using KAREL WRITE statements is acceptable. However, using dictionary files simplifies the displaying of text on many robots, by allowing the creation of multiple dictionary files which use the same program to display the text.

**Note** Dictionary files are useful in multi-lingual programs.

10.2.1 Dictionary Syntax

The syntax for a user dictionary consists of one or more dictionary elements. Dictionary elements have the following characteristics:
10. DICTIONARIES AND FORMS

• A dictionary element can contain multiple lines of information, up to a full screen of information. A user dictionary file has the following syntax:

```<*comment>
$n<,ele_name>@cursor_pos>&res_word><#chr_code>"Ele_text">&res_word
<#chr_code>+nest_ele>
<*comment>
$n+1>
```

— Items in brackets <> are optional.
— *comment is any item beginning with *. All text to the end of the line is ignored. Refer to Section 10.2.9.
— $n specifies the element number. n is a positive integer 0 or greater. Refer to Section 10.2.2.
— ,ele_name specifies a comma followed by the element name. Refer to Section 10.2.3.
— @cursor_pos specifies a cursor position (two integers separated by a comma.) Cursor positions begin at @1,1. Refer to Section 10.2.4.
— &res_word specifies a dictionary reserve word. Refer to Section 10.2.6.
— "Ele_text" specifies the element text to be displayed. Refer to Section 10.2.5.
— +nest_ele specifies the next dictionary text. Refer to Section 10.2.8.

• A dictionary element does not have to reside all on one line. Insert a carriage return at any place a space is allowed, except within quoted text. Quoted text must begin and end on the same line.

• Dictionary elements can contain text, position, and display attribute information. Table 10–2 lists the attributes of a dictionary element.

10.2.2 Dictionary Element Number

A dictionary element number identifies a dictionary element. A dictionary element begins with a “$” followed by the element number. Element numbers have the following characteristics:

• Element numbers begin at 0 and continue in sequential order.
• If element numbers are skipped, the dictionary compressor will add an extra overhead of 5 bytes for each number skipped. Therefore you should not skip a large amount of numbers.
• If you want the dictionary compressor to automatically generate the element numbers sequentially, use a “-” in place of the number. In the following example, the “-” is equated to element number 7.

```
$1

$2
```
10.2.3 Dictionary Element Name

Each dictionary element can have an optional element name. The name is separated from the element number by a comma and zero or more spaces. Element names are case sensitive. Only the first 12 characters are used to distinguish element names.

The following are examples of element names:

$1, KCMN_SH_LANG
$2, KCMN_SH_DICT

Dictionary elements can reference other elements by their name instead of by number. Additionally, element names can be generated as constants in a KAREL include file.

10.2.4 Dictionary Cursor Positioning

Dictionary elements are displayed in the specified window starting from the current position of the cursor. In most cases, move the cursor to a particular position and begin displaying the dictionary element there.

- The cursor position attribute "@" is used to move the cursor on the screen within the window.
- The "@" sign is followed by two numbers separated by a comma. The first number is the window row and the second number is the window column.

For example, on the teach pendant, the "t_fu" window begins at row 5 of the "t_sc" screen and is 10 rows high and 40 columns wide.

- Cursor position "@1,1" is the upper left position of the "t_fu" window and is located at the "t_sc" screen row 5 column 1.
- The lower right position of the "t_fu" window is "@10,40" and is located at the "t_sc" screen row 15 column 40.

Refer to Section 7.10.1 for more information on the teach pendant screens and windows.

For example, on the CRT/KB, the "c_fu" window begins at row 5 of the "c_sc" screen and is 17 rows high and 80 columns wide.
— Cursor position "@1,1" is the upper left position of the "c_fu" window and is located at the "c_sc" screen row 5 column.
— The lower right position of the window is "@17,80" and is located at the "c_sc" screen row 21, column 80.

Refer to Section 7.10.2 for more information on the CRT/KB screens and windows.

The window size defines the display limits of the dictionary elements.

### 10.2.5 Dictionary Element Text

Element text, or quoted text, is the information (text) you want to be displayed on the screen.

- The element text must be enclosed in double quote characters " ".
- To insert a back-slash within the text, use \ (double back-slash.)
- To insert a double-quote within the text, use " (back-slash, quote.)
- More than one element text string can reside in a dictionary element, separated by reserve words. Refer to Section 10.2.6 for more information.
- To include the values of KAREL variables in the element text, use the KAREL built-ins WRITE_DICT_V and READ_DICT_V, to pass the values of the variables.
- To identify the place where you want the KAREL variables to be inserted, use format specifiers in the text.
- A format specifier is the character "%" followed by some optional fields and then a conversion character. A format specifier has the following syntax:

\[ \%<-><+><width><.precision>conversion_character<^argument_number> \]

**Format Specifier**

- Items enclosed in < > are optional.
- The - sign means left justify the displayed value.
- The + sign means always display the sign if the argument is a number.
- The width field is a number that indicates the minimum number of characters the field should occupy.
- .precision is the . character followed by a number. It has a specific meaning depending upon the conversion character:
- conversion_characters identify the data type of the argument that is being passed. They are listed in Table 10–1.
- ^argument_number is the ^ (up-caret character) followed by a number.
Conversion Character

The conversion character is used to identify the data type of the KAREL variable that was passed. Table 10–1 lists the conversion characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type: Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>INTEGER; decimal number.</td>
</tr>
<tr>
<td>o</td>
<td>INTEGER; unsigned octal notation (without a leading zero).</td>
</tr>
<tr>
<td>x, X</td>
<td>INTEGER; unsigned hexadecimal notation (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>u</td>
<td>INTEGER; unsigned decimal notation.</td>
</tr>
<tr>
<td>s</td>
<td>STRING; print characters from the string until end of string or the number of characters given by the precision.</td>
</tr>
<tr>
<td>f</td>
<td>REAL; decimal notation of the form &lt;-&gt;mmm.dddddd, where the number of d's is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>e, E</td>
<td>REAL; decimal notation of the form &lt;-&gt;mmm.dddddd, where the number of d's is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>g, G</td>
<td>REAL; %e or %E is used if the exponent is less than -4 or greater than or equal to the precision; otherwise %f is used. Trailing zeros and a trailing decimal pointer are not printed.</td>
</tr>
<tr>
<td>%</td>
<td>No argument is converted; print a %.</td>
</tr>
</tbody>
</table>

- The characters d, o, x, X, and u can be used with the INTEGER, SHORT, BYTE, and BOOLEAN data types. A BOOLEAN data type is displayed as 0 for FALSE and 1 for TRUE.
- The f, e, E, g, and G characters can be used with the REAL data type.
- The character s is for a STRING data type.

Caution

Make sure you use the correct conversion character for the type of argument passed. If the conversion character and argument types do not match, unexpected results could occur.

Width and Precision

The optional width field is used to fix the minimum number of characters the displayed variable occupies. This is useful for displaying columns of numbers.

Setting a width longer than the largest number aligns the numbers.
If the displayed number has fewer characters than the width, the number will be padded on the left (or right if the "-" character is used) by spaces.

If the width number begins with "0", the field is padded with zeros instead.

The precision has the following meaning for the specified conversion character:

- **d, o, x, X, and u** - The minimum number of digits to be printed. If the displayed integer is less than the precision, leading zeros are padded. This is the same as using a leading zero on the field width.

- **s** - The maximum number of characters to be printed. If the string is longer than the precision, the remaining characters are ignored.

- **f, e, and E** - The number of digits to be printed after the decimal point.

- **g and G** - The number of significant digits.

**Argument Ordering**

An element text string can contain more than one format specifier. When a dictionary element is displayed, the first format specifier is applied against the first argument, the second specifier for the second argument, and so on. In some instances, you may need to apply a format specifier out of sequence. This can happen if you develop your program for one language, and then translate the dictionary to another language.

To re-arrange the order of the format specifiers, follow the conversion character with the "^" character and the argument number. As an example,

```
$20, file_message "File %s^2 on device %s^1 not found" &new_line
```

means use the second argument for the first specifier and the first argument for the second specifier.

**Caution**

You cannot re-arrange arguments that are SHORT or BYTE type because these argument are passed differently than other data types. Re-arranging SHORT or BYTE type arguments could cause unexpected results.

### 10.2.6 Dictionary Reserved Word Commands

Reserve words begin with the "&" character and are used to control the screen. They effect how, and in some cases where, the text is going to be displayed. They provide an easy and self-documenting way of adding control information to the dictionary. Refer to Table 10–2 for a list of the available reserved words.
### Table 10–2. Reserved Words

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;bg_black</td>
<td>Background color black</td>
</tr>
<tr>
<td>&amp;bg_blue</td>
<td>Background color blue</td>
</tr>
<tr>
<td>&amp;bg_cyan</td>
<td>Background color cyan</td>
</tr>
<tr>
<td>&amp;bg_dflt</td>
<td>Background color default</td>
</tr>
<tr>
<td>&amp;bg_green</td>
<td>Background color green</td>
</tr>
<tr>
<td>&amp;bg_magenta</td>
<td>Background color magenta</td>
</tr>
<tr>
<td>&amp;bg_red</td>
<td>Background color red</td>
</tr>
<tr>
<td>&amp;bg_white</td>
<td>Background color white</td>
</tr>
<tr>
<td>&amp;bg_yellow</td>
<td>Background color yellow</td>
</tr>
<tr>
<td>&amp;fg_black</td>
<td>Foreground color black</td>
</tr>
<tr>
<td>&amp;fg_blue</td>
<td>Foreground color blue</td>
</tr>
<tr>
<td>&amp;fg_cyan</td>
<td>Foreground color cyan</td>
</tr>
<tr>
<td>&amp;fg_dflt</td>
<td>Foreground color default</td>
</tr>
<tr>
<td>&amp;fg_green</td>
<td>Foreground color green</td>
</tr>
<tr>
<td>&amp;fg_magenta</td>
<td>Foreground color magenta</td>
</tr>
<tr>
<td>&amp;fg_red</td>
<td>Foreground color red</td>
</tr>
<tr>
<td>&amp;fg_white</td>
<td>Foreground color white</td>
</tr>
<tr>
<td>&amp;fg_yellow</td>
<td>Foreground color yellow</td>
</tr>
<tr>
<td>&amp;clear_win</td>
<td>Clear window (#128)</td>
</tr>
<tr>
<td>&amp;clear_2_eol</td>
<td>Clear to end of line (#129)</td>
</tr>
<tr>
<td>&amp;clear_2_eow</td>
<td>Clear to end of window (#130)</td>
</tr>
<tr>
<td>$cr</td>
<td>Carriage return (#132)</td>
</tr>
<tr>
<td>$lf</td>
<td>Line feed (#133)</td>
</tr>
<tr>
<td>&amp;rev_if</td>
<td>Reverse line feed (#134)</td>
</tr>
<tr>
<td>&amp;new_line</td>
<td>New line (#135)</td>
</tr>
<tr>
<td>&amp;bs</td>
<td>Back space (#136)</td>
</tr>
<tr>
<td>&amp;home</td>
<td>Home cursor in window (#137)</td>
</tr>
<tr>
<td>&amp;blink</td>
<td>Blink video attribute (#138)</td>
</tr>
<tr>
<td>&amp;reverse</td>
<td>Reverse video attribute (#139)</td>
</tr>
<tr>
<td>&amp;bold</td>
<td>Bold video attribute (#140)</td>
</tr>
</tbody>
</table>
The attributes &normal_size, &double_high, and &double_wide are used to clear data from a line on a screen. However, they are only effective for the line the cursor is currently on. To use these attributes, first position the cursor on the line you want to resize. Then write the attribute, and the text.

- **For the teach pendant**, &double_high means both double high and double wide are active, and &double_wide is the same as &normal_size.
- **For the CRT/KB**, &double_high means both double high and double wide are active, and &double_wide means double wide but normal height.

### 10.2.7 Character Codes

A character code is the “#” character followed by a number between 0 and 255. It provides a method of inserting special printable characters, that are not represented on your keyboard, into your dictionary. Refer to Appendix D, for a listing of the ASCII character codes.

### 10.2.8 Nesting Dictionary Elements

The plus “+” attribute allows a dictionary element to reference another dictionary element from the same dictionary, up to a maximum of five levels. These nested elements can be referenced by element name or element number and can be before or after the current element. When nested elements are displayed, all the elements are displayed in their nesting order as if they are one single element.
10.2.9 Dictionary Comment

The asterisk character (*) indicates that all text, to the end of the line, is a comment. All comments are ignored when the dictionary is compressed. A comment can be placed anywhere a space is allowed, except within the element text.

10.2.10 Generating a KAREL Constant File

The element numbers that are assigned an element name in the dictionary can be generated into a KAREL include file for KAREL programming. The include file will contain the CONST declarator and a constant declaration for each named element.

```
element_name = element_number
```

Your KAREL program can include this file and reference each dictionary element by name instead of number.

To generate a KAREL include file, specify “.kl”, followed by the file name, on the first line of the dictionary file. The KAREL include file is automatically generated when the dictionary is compressed.

The following would create the file kcmn.kl when the dictionary is compressed.

```
.kl kcmn
$-, move_home, "press HOME to move home"
```

The kcmn.kl file would look as follows

```
-- WARNING: This include file generated by dictionary compressor.
--
-- Include File: kcmn.kl
-- Dictionary file: apkcmneg.utx
--CONST
move_home = 0
```

**Note** If you make a change to your dictionary that causes the element numbers to be re-ordered, you must re-translate your KAREL program to insure that the proper element numbers are used.

10.2.11 Compressing and Loading Dictionaries on the Controller

The KAREL editor can be used to create and modify the user dictionary. When you have finished editing the file, you compress it from the KCL command prompt.

```
KCL> COMPRESS DICT filename
```
Do not include the .UTX file type with the file name. If the compressor detects any errors, it will point to the offending word with a brief explanation of what is wrong. Edit the user dictionary and correct the problem before continuing.

A loadable dictionary with the name filename but with a .TX file type will be created. If you used the .kl symbol, a KAREL include file will also be created. Figure 10–1 illustrates the compression process.

Figure 10–1. Dictionary Compressor and User Dictionary File

Before the KAREL program can use a dictionary, the dictionary must be loaded into the controller and given a dictionary name. The dictionary name is a one to four character word that is assigned to the dictionary when it is loaded. Use the KCL LOAD DICT command to load the dictionary.

```
KCL> LOAD DICT filename dictname <lang_name>
```

The optional lang_name allows loading multiple dictionaries with the same dictionary name. The actual dictionary that will be used by your program is determined by the current value of $LANGUAGE. This system variable is set by the KCL SET LANGUAGE command or the SET_LANG KAREL built-in. The allowed languages are ENGLISH, JAPANESE, FRENCH, GERMAN, SPANISH, or DEFAULT.

The KAREL program can also load a dictionary. The KAREL built-in ADD_DICT is used to load a dictionary into a specified language and assign a dictionary name.

### 10.2.12 Accessing Dictionary Elements from a KAREL Program

Your KAREL program uses either the dictionary name and an element number, or the element name to access a dictionary element. The following KAREL built-ins are used to access dictionary elements:

- ADD_DICT - Add a dictionary to the specified language.
• REMOVE_DICT - Removes a dictionary from the specified language and closes the file or frees the memory it resides in.
• WRITE_DICT - Write a dictionary element to a window.
• WRITE_DICT_V - Write a dictionary element that has format specifiers for a KAREL variable, to a window.
• READ_DICT - Read a dictionary element into a KAREL STRING variable.
• READ_DICT_V - Read a dictionary element that has format specifiers into a STRING variable.
• CHECK_DICT - Check if a dictionary element exists.

10.3 CREATING USER FORMS

A form is a type of dictionary file necessary for creating menu interfaces that have the same "look and feel" as the R-30iA menu interface.

The following are steps for using forms:

1. Create an ASCII form text file with the .FTX file extension.
2. Compress the form file using the KCL COMPRESS FORM command. This creates a loadable dictionary file with a .TX extension and an associated variable file (.VR).
3. Load the form.
   • From KCL, use the KCL LOAD FORM command. This will load the dictionary file (.TX) and the associated variable file (.VR).
   • From KAREL, use the ADD_DICT built-in to load the dictionary file (.TX), and the LOAD built-in to load the association variable file (.VR).
4. Use the KAREL DISCTRL_FORM built-in to display the form text. The DISCTRL_FORM built-in handles all input operations including cursor position, scrolling, paging, input validation, and choice selections. Refer to the DISCTRL_FORM built-in, Appendix A, "KAREL Language Alphabetical Description."

Forms are useful for programs which require the user to enter data. For example, once the user enters the data, the program must test this data to make sure that it is in an acceptable form. Numbers must be entered with the correct character format and within a specified range, text strings must not exceed a certain length and must be a valid selection. If an improper value is entered, the program must notify the user and prompt for a new entry. Forms provide a way to automatically validate entered data. Forms also allow the program to look as if it is integrated into the rest of the system menus, by giving the operator a familiar interface.

Forms must have the USER2 menu selected. Forms use the "t_sc" and "c_sc" screens for teach pendant and CRT/KB respectively. The windows that are predefined by the system are used for displaying the form text. For both screens, this window is 10 rows high and 40 columns wide. This means that the &double_high and &double_wide attributes are used on the CRT/KB and cannot be changed.
10. DICTIONARIES AND FORMS

10.3.1 Form Syntax

A form defines an operator interface that appears on the teach pendant or CRT/KB screens. A form is a special dictionary element. Many forms can reside in the same dictionary along with other (non-form) dictionary elements.

Note If your program requires a form dictionary file (.FTX), you do not have to create a user dictionary file (.UTX). You may place your user dictionary elements in the same file as your forms.

To distinguish a form from other elements in the dictionary, the symbol “.form” is placed before the element and the symbol “.endform” is placed after the element. The symbols must reside on their own lines. The form symbols are omitted from the compressed dictionary.

The following is the syntax for a form:

Form Syntax

```
.form <form_attributes>
$n, form_name@cursor_pos<&res_word>"Menu_title"<&res_work>&new_line
  @cursor_pos<&res_word>"Menu_label"<&res_word>&new_line
  @cursor_pos<&res_word>"-Selectable_item"<&res_word>&new_line
  @cursor_pos<&res_word>"-%Edit_item"<&res_word>&new_line
  @cursor_pos<&res_word>"Non_selectable_text"<&res_word>&new_line
  @cursor_pos<&res_word>"Display_only_item"<&res_word>&new_line
  ^function_key &new_line
  <?help_menu &new_line
.endform
$n, function_key
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "help_label" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name" &new_line
  "Key_name"
> $n, help_menu
  "Help_text" &new_line
  "Help_text" &new_line
  "Help_text">
```

Restrictions

- Items in brackets <> are optional.
• Symbols not defined here are standard user dictionary element symbols ($n, @cursor_pos, &res_word, &new_line).
• form_attributes are the key words unnumber and unclear.
• form_name specifies the element name that identifies the form.
• "$Menu_title" and "Menu_label" specify element text that fills the first two lines of the form and are always displayed.
• "- Selectable_item" specifies element text that can be cursored to and selected.
• "-%Editable_item" specifies element text that can be cursored to and edited.
• "NonSelectable_text" specifies element text that is displayed in the form and cannot be cursored to.
• "%Display_only_item" specifies element text using a format specifier. It cannot be cursored to.
• ^function_key defines the labels for the function keys using an element name.
• ?help_menu defines a page of help text that is associated with a form using an element name.
• "Key_name" specifies element text displayed over the function keys.
• "Help_label" is the special label for the function key 5. It can be any label or the special word HELP.
• "Help_text" is element text up to 40 characters long.
• Color attributes can be specified in forms. The i Pendant will display the color. The monochrome pendant will ignore the color attributes.

10.3.2 Form Attributes

Normally, a form is displayed with line numbers in front of any item the cursor can move to. To keep a form from generating and displaying line numbers, the symbol “.form unnumber” is used.

To keep a form from clearing any windows before being displayed, the symbol “.form noclear” is used. The symbols “noclear” and “unnumber” can be used in any order.

In the following example, MH_TOOLDEFN is an unnumbered form that does not clear any windows. MH_APPLIO is a numbered form.

```
.form unnumber noclear

$1, MH_TOOLDEFN
.endform

$2, MH_PORT

$3, MH_PORTFKEY
```
10. DICTIONARIES AND FORMS

10.3.3 Form Title and Menu Label

The menu title is the first element of text that follows the form name. The menu label follows the menu title. Each consists of one row of text in a non-scrolling window.

- **On the teach pendant** the first row of the "full" window is used for the menu title. The second row is used for the menu label.
- **On the CRT/KB** the first row of the "cr05" widow is used for the menu title. The second row is used for the menu label.
- The menu title is positioned at row 3, column 1-21.
- The menu label is positioned at row 4, column 1-40.

Unless the "noclear" form attribute is specified both the menu title and menu label will be cleared.

The reserved word &home must be specified before the menu title to insure that the cursor is positioned correctly. The reserved word &reverse should also be specified before the menu title and the reserved word &standard should follow directly after the menu title. These are necessary to insure the menu appears to be consistent with the R-30iA menu interface. The reserved word &new_line must be specified after both the menu title and menu label to indicate the end of the text. The following is an example menu title and menu label definition.

```
.form

$1, mis_form

&home &reverse "Menu Title" &standard &new_line

"Menu Label" &new_line

.endform
```
If no menu label text is desired, the &new_line can be specified twice after the menu title as in the following example.

.form

$1,misc_form

&home &reverse "Menu Title" &standard &new_line &new_line

.endform

If the cursor position attribute is specified, it is not necessary to specify the &new_line reserved word. The following example sets the cursor position for the menu title to row 1, column 2, and the menu label to row 2, column 5.

.form

$1,misc_form

@1,2 &reverse "Menu Title" &standard

@2,5 "Menu Label"

.endform

### 10.3.4 Form Menu Text

The form menu text follows the menu title and menu label. It consists of an unlimited number of lines that will be displayed in a 10 line scrolling window named “fscr” on the teach pendant and “ct06” on the CRT/KB. This window is positioned at rows 5-14 and columns 1-40. Unless the “noclear” option is specified, all lines will be cleared before displaying the form.

Menu text can consist of the following:

- Selectable menu items
- Edit data items of the following types:
10. DICTIONARIES AND FORMS

— INTEGER
— INTEGER port
— REAL
— SHORT (32768 to 32766)
— BYTE (0 to 255)
— BOOLEAN
— BOOLEAN port
— STRING
— Program name string
— Function key enumeration type
— Subwindow enumeration type
— Subwindow enumeration type using a variable
— Port simulation

• Non-selectable text
• Display only data items with format specifiers
• Cursor position attributes
• Reserve words or ASCII codes
• Function key element name or number
• Help element name or number

Each kind of menu text is explained in the following sections.

10.3.5 Form Selectable Menu Item

Selectable menu items have the following characteristics:

• A selectable menu item is entered in the dictionary as a string enclosed in double quotes.

• The first character in the string must be a dash, ‘-’. This character will not be printed to the screen. For example,
  "- Item 1 "

• The entire string will be highlighted when the selectable item is the default.

• If a selectable item spans multiple lines, the concatenation character ‘+’ should be used as the last character in the string. The concatenation character will not be printed to the screen. The attribute &new_line is used to signal a new line. For example,
  "- Item 1, line 1 +" &new_line
The automatic numbering uses the first three columns and does not shift the form text. Therefore, the text must allow for the three columns by either adding spaces or specifying cursor positions. For example,

"- Item 1 " \new_line
"- Item 2 " \new_line
"- Item 3 "

or

@3,4"- Item 1 "
@4,4"- Item 2 "
@5,4"- Item 3 "

The first line in the scrolling window is defined as row 3 of the form.

Pressing enter on a selectable menu item will always cause the form processor to exit with the termination character of \texttt{ky\_select}, regardless of the termination mask setting. The item number selected will be returned.

Selecting the item by pressing the ITEM hardkey on the teach pendant will only highlight the item. It does not cause an exit.

Short-cut number selections are not handled automatically, although they can be specified as a termination mask.

### 10.3.6 Edit Data Item

You can edit data items that have the following characteristics:

- Data item is entered in the dictionary as a string enclosed in double quotes.
- The first character in the string must be a dash, '‐'. This character is not printed to the screen.
- The second character in the string must be a ‘%’. This character marks the beginning of a format specifier.
- Each format specifier begins with a % and ends with a conversion character. All the characters between these two characters have the same meaning as user dictionary elements.
Note You should provide a field width with each format specifier, otherwise a default will be used. This default might cause your form to be mis-aligned.

Table 10–3 lists the conversion characters for an editable data item.

Table 10–3. Conversion Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type: Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>INTEGER; decimal number.</td>
</tr>
<tr>
<td>o</td>
<td>INTEGER; unsigned octal notation (without a leading zero).</td>
</tr>
<tr>
<td>x, X</td>
<td>INTEGER; unsigned hexadecimal notation (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>u</td>
<td>INTEGER; unsigned decimal notation.</td>
</tr>
<tr>
<td>pu</td>
<td>INTEGER port; unsigned decimal notation.</td>
</tr>
<tr>
<td>px</td>
<td>INTEGER port; unsigned hexadecimal notation (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>f</td>
<td>REAL; decimal notation of the form &lt;-&gt;mmm.dddddd, where the number of d’s is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>e, E</td>
<td>REAL; decimal notation of the form &lt;-&gt;m.dddddde+-xx or &lt;-&gt;m.ddddddE+-xx, where the number of d’s is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>g, G</td>
<td>REAL; %e or %E is used if the exponent is less than -4 or greater than or equal to the precision; otherwise %f is used. Trailing zeros and a trailing decimal pointer are not printed.</td>
</tr>
<tr>
<td>h</td>
<td>SHORT; signed short.</td>
</tr>
<tr>
<td>b</td>
<td>BYTE; unsigned byte.</td>
</tr>
<tr>
<td>B</td>
<td>BOOLEAN; print characters from boolean enumeration string.</td>
</tr>
<tr>
<td>P</td>
<td>BOOLEAN port; print characters from boolean port enumeration string.</td>
</tr>
<tr>
<td>S</td>
<td>INTEGER or BOOLEAN port; print characters from port simulation enumeration string.</td>
</tr>
<tr>
<td>k</td>
<td>STRING; print characters from KAREL string until end of string or the number of characters given by the precision.</td>
</tr>
<tr>
<td>pk</td>
<td>STRING; print program name from KAREL string until end of string or the number of characters given by the precision.</td>
</tr>
<tr>
<td>n</td>
<td>INTEGER; print characters from function key enumeration string. Uses dictionary elements to define the enumeration strings.</td>
</tr>
<tr>
<td>w</td>
<td>INTEGER; print characters from subwindow enumeration string. Uses dictionary elements to define the enumeration strings.</td>
</tr>
<tr>
<td>v</td>
<td>INTEGER; print characters from subwindow enumeration string. Uses a variable to define the enumeration strings.</td>
</tr>
</tbody>
</table>
The following is an example of a format specifier:

"-%5d" or "-%10s"

The form processor retrieves the values from the input value array and displays them sequentially. All values are dynamically updated.

**Edit Data Items: INTEGER, INTEGER Ports, REAL, SHORT, BYTE**

- You can specify a range of acceptable values by giving each format specifier a minimum and maximum value allowed "(min, max)." If you do not specify a minimum and maximum value, any integer or floating point value will be accepted. For example,

"-%3d(1,255)" or "-%10.3f(0.,100000.)"

- When an edit data item is selected, the form processor calls the appropriate input routine. The input routine reads the new value (with inverse video active) and uses the minimum and maximum values specified in the dictionary element, to determine whether the new value is within the valid range.
  - If the new value is out of range, an error message will be written to the prompt line and the current value will not be modified.
  - If the new value is in the valid range, it will overwrite the current value.

**Edit Data Item: BOOLEAN**

- The format specifier %B is used for KAREL BOOLEAN values, to display and select menu choice for the F4 and F5 function keys. The name of the dictionary element, that contains the function key labels, is enclosed in parentheses and is specified after the %B. For example,

"-%4B(enum_bool)"

The dictionary element defining the function keys should define the FALSE value first (F5 label) and the TRUE value second (F4 label). For example,

```karel
$2,enum_bool

" NO" &new_line

" YES"
```
The form processor will label the function keys when the cursor is moved to the enumerated item. The value shown in the field is the same as the function key label except all leading blanks are removed.

**Edit Data Item: BOOLEAN Port**

- The format specifier %P is used for KAREL BOOLEAN port values, to display and select menu choices from the F4 and F5 function keys. The name of the dictionary element, that contains the function key labels, is enclosed in parentheses and is specified after the %P. For example, 
  "-%3P(enum_bool)"

The dictionary element defining the function keys should define the 0 value first (F5 label) and the 1 value second (F4 label). For example,
$2, enum_bool

" OFF" &new_line

" ON"

The form processor will label the function keys when the cursor is moved to the enumerated item. The value shown in the field is the same as the function key label except all leading blanks are removed.

**Edit Data Item: Port Simulation**

- The format specifier %S is used for port simulation, to display and select menu choices from the F4 and F5 function keys. The name of the dictionary element, that contains the function key labels, is enclosed in parentheses and is specified after the %S. For example,
  "-%1S(sim_fkey)"

The dictionary element defining the function keys should define the 0 value first (F5 label) and the 1 value second (F4 label). For example,
$-, sim_fkey

" UNSIM " &new_line * F5 key label, port will be unsimulated
"SIMULATE" &new_line * F4 key label, port will be simulated
The form processor will label the function keys when the cursor is moved to the enumerated item. The value shown in the field is the same as the function key label except all leading blanks are removed and the value will be truncated to fit in the field width.

**Edit Data Item: STRING**

- You can choose to clear the contents of a string before editing it. To do this follow the STRING format specifier with the word "clear", enclosed in parentheses. If you do not specify "(clear)", the default is to modify the existing string. For example,

  `-%10k(clear)`

**Edit Data Item: Program Name String**

- You can use the `%pk` format specifier to display and select program names from the subwindow. The program types to be displayed are enclosed in parenthesis and specified after `%pk`. For example,

  `-%12pk(1)` * specifies TP programs

  `-%12pk(2)` * specifies PC programs

  `-%12pk(6)` * specifies TP, PC, VR

  `-%12pk(16)` * specifies TP & PC

All programs that match the specified type and are currently in memory, are displayed in the subwindow. When a program is selected, the string value is copied to the associated variable.

**Edit Data Item: Function Key Enumeration**

- You can use the format specifier `%n` (for enumerated integer values) to display and select choices from the function keys. The name of the dictionary element that contains the list of valid choices is enclosed in parentheses and specified after `%n`. For example,

  `-%6n(enum_fkey)`

The dictionary element defining the function keys should list one function key label per line. If function keys to the left of those specified are not active, they should be set to "". A maximum of 5 function keys can be used. For example,

 `$2,enum_fkey$

  "" &new_line *Specifies F1 is not active

  "JOINT" &new_line *Specifies F2

  "LINEAR" &new_line *Specifies F3`
"CIRC" *Specifies F4

The form processor will label the appropriate function keys when the enumerated item is selected. When a function key is selected, the value set in the integer is as follows:

User presses F1, value = 1

User presses F2, value = 2

User presses F3, value = 3

User presses F4, value = 4

User presses F5, value = 5

The value shown in the field is the same as the function key label except all leading blanks are removed.

<table>
<thead>
<tr>
<th>JOINT</th>
<th>LINEAR</th>
<th>CIRC</th>
</tr>
</thead>
</table>

**Edit Data Item: Subwindow Enumeration**

- You can use the format specifier %w (for enumerated integer values) to display and select choices from the subwindow. The name of the dictionary element, containing the list of valid choices, is enclosed in parentheses and specified after %w. For example,

"-%8w(enum_sub)"

One dictionary element is needed to define each choice in the subwindow. 35 choices can be used. If fewer than 35 choices are used, the last choice should be followed by a dictionary element that contains "\a". The choices will be displayed in 2 columns with 7 choices per page. If only 4 or less choices are used, the choices will be displayed in 1 column with a 36 character width. For example,

$2,enum_sub "Option 1"

$3 "Option 2"

$4 "Option 3"

$5 "\a"

The form processor will label F4 as “[CHOICE]” when the cursor is moved to the enumerated item. When the function key F4, [CHOICE] is selected, it will create the subwindow with the appropriate display. When a choice is selected, the value set in the integer is the number selected. The value shown in the field is the same as the dictionary label except all leading blanks are removed.
Edit Data Item: Subwindow Enumeration using a Variable

- You can also use the format specifier %v (for enumerated integer values) to display and select choices from the subwindow. However, instead of defining the choices in a dictionary they are defined in a variable. The name of the dictionary element, which contains the program and variable name, is enclosed in parentheses and specified after %v. For example, "-%v(enum_var)"

$-,enum_var

"RUNFORM" &new_line * program name of variable

"CHOICES" &new_line * variable name containing choices

[RUNFORM] CHOICES must be defined as a KAREL string array. Each element of the array should define a choice in the subwindow. This format specifier is similar to %w. However, the first element is related to the value 0 and is never used. Value 1 begins at the second element. The last value is either the end of the array or the first uninitialized value.


[1] *uninit*
[2] 'Red' <= value 1
[3] 'Blue' <= value 2
[4] 'Green' <= value 3
[5] *uninit*
[6] *uninit*

Edit Data Item: Tree View

- The format specifier %t is used to specify a Tree View item. It uses a KAREL BOOLEAN value to determine whether the tree is expanded or collapsed. For example, "-%t"

The form processor will change %t to %c. When the BOOLEAN value is FALSE, a '+' will be displayed using the format specifier and the items following the tree view will not be shown (collapsed state). When the BOOLEAN value is TRUE, a '-' will be displayed using the format specifier and the items following the tree view will be shown (expanded state). Table 10–4 displays some formatting examples.
### 10.3.7 Dynamic Forms using Tree View

The Tree View can be used to create dynamic forms. It can be used instead of concatenating multiple forms into one. In this case, the Tree View Data Item is used without the leading -. For example,

\[
\text{"%t" &new_line}
\]

#### Table 10-4. Tree View Format

<table>
<thead>
<tr>
<th>Format</th>
<th>FALSE value</th>
<th>TRUE value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%t&quot;</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>&quot;%2t&quot;</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>&quot;-%t Weld schedule:&quot;</td>
<td>+ Weld schedule:</td>
<td>– Weld schedule:</td>
</tr>
</tbody>
</table>

- The KAREL BOOLEAN value will only determine the initial state of the tree view. It is not monitored. When the user selects the tree view item and presses ENTER, the tree view state will toggle and the BOOLEAN value will be set to the resultant value. This state is maintained when the form is exited and reentered. If the KAREL BOOLEAN value is in CMOS or SHADOW, then the state is maintained between power cycles.

- There is a way to refresh the tree view. Passing ctl_w to the form will expand or collapse all tree view items based on their associated KAREL BOOLEAN values. The default item will be maintained. If the default item is no longer shown its tree view item will become the new default item.

- Nesting of tree view items is not allowed. If another %t is found, the current tree view is ended and a new one is started. When a tree view item should be ended without creating a new tree view item, .endtree can be used. For example:

\[
\text{"-%t Burnback schedule:" &new_line} \\
\text{Wire feed "%-6.2f(0.0, 9999.0)" &new_line} \\
\text{Trim "%-6.2f(0.0, 9999.0)" &new_line} \\
\text{.endtree} \\
\text{" Gas postflow time:" &new_line}
\]

- If a form is constructed by concatenating multiple forms and an item will be within a tree view, then .tree must be used. The .endtree is optional and only necessary if other items will not be within the tree view.

\[
\text{.form} \\
\text {$-,wl\_data\_cmd \&new\_line \&new\_line} \\
\text{.tree} \\
\text{" Wire Feed "%-6.2f(0.0, 9999.0)" \&new\_line} \\
\text{.endform}
\]
When the KAREL BOOLEAN value is FALSE, the items following the tree view will be invisible. When the KAREL BOOLEAN value is TRUE, the items following the tree view will be visible. The tree view line is always invisible. Any other items on its line are also invisible.

The KAREL BOOLEAN value will only determine the initial state of the tree view. It is not monitored. The user will never be able to change the state since the tree view item is invisible.

There is a way to refresh the tree view. Passing ctl_w to the form will expand or collapse all tree view items based on their associated KAREL BOOLEAN values. The default item will be maintained. If the default item is no longer shown, the previous item will become the new default item.

### 10.3.8 Non-Selectable Text

Non-selectable text can be specified in the form. These items have the following characteristics:

- Non-selectable text is entered in the dictionary as a string enclosed in double quotes.
- Non-selectable text can be defined anywhere in the form, but must not exceed the maximum number of columns in the window.

### 10.3.9 Display Only Data Items

Display only data items can be specified in the form. These items have the following characteristics:

- Display only data items are entered in the dictionary as a string enclosed in double quotes.
- The first character in the string must be a ‘%’. This character marks the beginning of a format specifier.
- The format specifiers are the same as defined in the previous section for an edit data item.

### 10.3.10 Cursor Position Attributes

Cursor positioning attributes can be used to define the row and column of any text. The row is always specified first. The dictionary compressor will generate an error if the form tries to backtrack to a previous row or column. The form title and label are on rows 1 and 2. The scrolling window starts on row 3. For example,

@3,4 "- Item 1"
Even though the scrolling window is only 10 lines, a long form can specify row positions that are greater than 12. The form processor keeps track of the current row during scrolling.

### 10.3.11 Form Reserved Words and Character Codes

Reserved words or character codes can be used. Refer to Table 10–5 for a list of all available reserved words. However, only the reserved words which do not move the cursor are allowed in a scrolling window. Refer to Table 10–6 for a list of available reserved words for a scrolling window.

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;bg_black</td>
<td>Background color black</td>
</tr>
<tr>
<td>&amp;bg_blue</td>
<td>Background color blue</td>
</tr>
<tr>
<td>&amp;bg_cyan</td>
<td>Background color cyan</td>
</tr>
<tr>
<td>&amp;bg_dflt</td>
<td>Background color default</td>
</tr>
<tr>
<td>&amp;bg_green</td>
<td>Background color green</td>
</tr>
<tr>
<td>&amp;bg_magenta</td>
<td>Background color magenta</td>
</tr>
<tr>
<td>&amp;bg_red</td>
<td>Background color red</td>
</tr>
<tr>
<td>&amp;bg_white</td>
<td>Background color white</td>
</tr>
<tr>
<td>&amp;bg_yellow</td>
<td>Background color yellow</td>
</tr>
<tr>
<td>&amp;fg_black</td>
<td>Foreground color black</td>
</tr>
<tr>
<td>&amp;fg_blue</td>
<td>Foreground color blue</td>
</tr>
<tr>
<td>&amp;fg_cyan</td>
<td>Foreground color cyan</td>
</tr>
<tr>
<td>&amp;fg_dflt</td>
<td>Foreground color default</td>
</tr>
<tr>
<td>&amp;fg_green</td>
<td>Foreground color green</td>
</tr>
<tr>
<td>&amp;fg_magenta</td>
<td>Foreground color magenta</td>
</tr>
<tr>
<td>&amp;fg_red</td>
<td>Foreground color red</td>
</tr>
<tr>
<td>&amp;fg_white</td>
<td>Foreground color white</td>
</tr>
<tr>
<td>&amp;fg_yellow</td>
<td>Foreground color yellow</td>
</tr>
<tr>
<td>&amp;clear_win</td>
<td>Clear window (#128)</td>
</tr>
</tbody>
</table>
### Table 10–5. Reserved Words (Cont’d)

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;clear_2_eol</td>
<td>Clear to end of line (#129)</td>
</tr>
<tr>
<td>&amp;clear_2_eow</td>
<td>Clear to end of window (#130)</td>
</tr>
<tr>
<td>$cr</td>
<td>Carriage return (#132)</td>
</tr>
<tr>
<td>$lf</td>
<td>Line feed (#133)</td>
</tr>
<tr>
<td>&amp;rev_lf</td>
<td>Reverse line feed (#134)</td>
</tr>
<tr>
<td>&amp;new_line</td>
<td>New line (#135)</td>
</tr>
<tr>
<td>&amp;bs</td>
<td>Back space (#136)</td>
</tr>
<tr>
<td>&amp;home</td>
<td>Home cursor in window (#137)</td>
</tr>
<tr>
<td>&amp;blink</td>
<td>Blink video attribute (#138)</td>
</tr>
<tr>
<td>&amp;reverse</td>
<td>Reverse video attribute (#139)</td>
</tr>
<tr>
<td>&amp;bold</td>
<td>Bold video attribute (#140)</td>
</tr>
<tr>
<td>&amp;under_line</td>
<td>Underline video attribute (#141)</td>
</tr>
<tr>
<td>&amp;double_wide</td>
<td>Wide video size (#142) (refer to description below for usage)</td>
</tr>
<tr>
<td>&amp;standard</td>
<td>All attributes normal (#143)</td>
</tr>
<tr>
<td>&amp;graphics_on</td>
<td>Turn on graphic characters (#146)</td>
</tr>
<tr>
<td>&amp;ascii_on</td>
<td>Turn on ASCII characters (#147)</td>
</tr>
<tr>
<td>&amp;double_high</td>
<td>High video size (#148) (refer to description below for usage)</td>
</tr>
<tr>
<td>&amp;normal_size</td>
<td>Normal video size (#153)</td>
</tr>
<tr>
<td>&amp;multi_on</td>
<td>Turn on multi-national characters (#154)</td>
</tr>
</tbody>
</table>

Table 10–6 lists the reserved words that can be used for a scrolling window.

### Table 10–6. Reserved Words for Scrolling Window

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;new_line</td>
<td>New line (#135)</td>
</tr>
<tr>
<td>&amp;blink</td>
<td>Blink video attribute (#138)</td>
</tr>
<tr>
<td>&amp;reverse</td>
<td>Reverse video attribute (#139)</td>
</tr>
<tr>
<td>&amp;bold</td>
<td>Bold video attribute (#140)</td>
</tr>
<tr>
<td>&amp;under_line</td>
<td>Underline video attribute (#141)</td>
</tr>
<tr>
<td>&amp;standard</td>
<td>All attributes normal (#143)</td>
</tr>
<tr>
<td>&amp;graphics_on</td>
<td>Turn on graphic characters (#146)</td>
</tr>
</tbody>
</table>
Table 10-6. Reserved Words for Scrolling Window (Cont’d)

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;ascii_on</td>
<td>Turn on ASCII characters (#147)</td>
</tr>
<tr>
<td>&amp;multi_on</td>
<td>Turn on multi-national characters (#154)</td>
</tr>
</tbody>
</table>

10.3.12 Form Function Key Element Name or Number

Each form can have one related function key menu. A function key menu has the following characteristics:

- The function key menu is specified in the dictionary with a caret, ^, immediately followed by the name or number of the function key dictionary element. For example, ^misc_fkey

- The dictionary element defining the function keys should list one function key label per line. If function keys to the left of those specified are not active, then they should be set to "". A maximum of 10 function keys can be used. For example,

```
$3,misc_fkey

" F1" &new_line
" F2" &new_line
" F3" &new_line
" F4" &new_line
" HELP >" &new_line
" " &new_line
" " &new_line
" F8" &new_line
" F9" &new_line
```

- The form processor will label the appropriate function keys and return from the routine if a valid key is pressed. The termination character will be set to ky_f1 through ky_f10.

- The function keys will be temporarily inactive if an enumerated data item is using the same function keys.
• If function key F5 is labeled HELP, it will automatically call the form’s help menu if one exists.

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>HELP &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F8  F9

### 10.3.13 Form Function Key Using a Variable

A function key menu can also be defined in a variable. The function key dictionary item will contain the program and variable name, prefixed with an asterisk to distinguish it from function key text. For example,

* Specify the function keys in a variable
* whose type is an ARRAY[m] of STRING[n].

\[
\text{
$3, \text{misc}_\_f\text{key} \\
\text{"*RUNFORM"} \ &\text{new\_line} \ \ast \ \text{program name of variable} \\
\text{"*FKEYS"} \ &\text{new\_line} \ \ast \ \text{variable name containing function keys}
\]

[RUNFORM] FKEYS must be defined as a KAREL string array. Each element of the array should define a function key label.

\[
[\text{RUNFORM}] \ \text{FKEYS:ARRAY[10] OF STRING[12]} = \\
[1] ' \ F1' \\
[2] ' \ F2' \\
[3] ' \ F3' \\
[4] ' \ F4' \\
[5] ' \ \text{HELP} \ >' \\
[6] '' \\
[7] '' \\
[8] ' \ F8' \\
[9] ' \ F9' \\
[10] ' \ >'
\]

### 10.3.14 Form Help Element Name or Number

Each form can have one related help menu. The help menu has the following characteristics:
• A help element name or number is specified in the dictionary with a question mark, ?, immediately followed by the name or number of the help dictionary element. For example, ?misc_help

• The dictionary element defining the help menu is limited to 48 lines of text.
• The form processor will respond to the help key by displaying the help dictionary element in a predefined window. The predefined window is 40 columns wide and occupies rows 3 through 14.
• The help menu responds to the following inputs:
  — Up or down arrows to scroll up or down 1 line.
  — Shifted up or down arrows to scroll up or down 3/4 of a page.
  — Previous, to exit help. The help menu restores the previous screen before returning.

### 10.3.15 Teach Pendant Form Screen

You can write to other active teach pendant windows while the form is displayed. The screen itself is named "tpsc." Figure 10–2 shows all the windows attached to this screen. Unless the noclear option is specified, “full,” “fscr,” “prmp,” and “ftnk” windows will be cleared before displaying the form.

#### Figure 10–2. Teach Pendant Form Screen

```
+ | err | + 
+ | stat |
+ | full |
+ | full |
= | fscr |
= | prmp |
= | ftnk |
+ | + |
```

### 10.3.16 CRT/KB Form Screen

You can write to other active CRT/KB windows while the form is displayed. The screen itself is named “ctsc.” All lines in the screen are set to double high and double wide video size. Figure 10–3
shows all the windows attached to this screen. Unless the “noclear” option is specified, “ct05,” “ct06,” “ct03,” and “ct04” windows will be cleared before displaying the form.

Figure 10–3. CRT/KB Form Screen

10.3.17 Form File Naming Convention

Uncompressed form dictionary files must use the following file name conventions:

- The first two letters in the dictionary file name can be an application prefix.
- If the file name is greater than four characters, the form processor will skip the first two letters when trying to determine the dictionary name.
- The next four letters must be the dictionary name that you use to load the .TX file, otherwise the form processor will not work.
- The last two letters are optional and should be used to identify the language:
  - “EG” for ENGLISH
  - “JP” for JAPANESE
  - “FR” for FRENCH
  - “GR” for GERMAN
  - “SP” for SPANISH
- A dictionary file containing form text must have a .FTX file type, otherwise the dictionary compressor will not work. After it is compressed, the same dictionary file will have a .TX file type instead.

The following is an example of an uncompressed form dictionary file name:
MHPALTEG.FTX
MH stands for Material Handling, PALT is the dictionary name that is used to load the dictionary on the controller, and EG stands for English.

### 10.3.18 Compressing and Loading Forms on the Controller

The form file can only be compressed on the RAM disk RD:. Compressing a form is similar to compressing a user dictionary. From the KCL command prompt, enter:

```
KCL> COMPRESS FORM filename
```

Do not include the .FTX file type. If the compressor detects any errors, it will point to the offending word with a brief explanation of what is wrong. Edit the form and correct the problem before continuing.

**Note** The form file must be an uncompressed file in order for the errors to point to the correct line.

Two files will be created by the compressor. One is a loadable dictionary file with the name filename but with a .TX file type. The other will be a variable file with a .VR file type but with the four character dictionary name as the file name. The dictionary name is extracted from filename as described previously. A third file may also be created if you used the “.kl” symbol to generate a KAREL include file. **Figure 10–4** illustrates compressing.

**Figure 10–4. Dictionary Compressor and Form Dictionary File**

![Diagram](image)

Each form will generate three kinds of variables. These variables are used by the form processor. They must be reloaded each time the form dictionary is recompressed. The variables are as follows:

1. **Item array variable** - The variable name will be the four-character dictionary name, concatenated with the element number, concatenated with _IT.

2. **Line array variable** - The variable name will be the four-character dictionary name, concatenated with the element number, concatenated with _LN.
3. Miscellaneous variable - The variable name will be the four-character dictionary name, concatenated with the element number, concatenated with _MS.

The data defining the form is generated into KAREL variables. These variables are saved into the variable file and loaded onto the controller. The name of the program is the dictionary name preceded by an asterisk. For example, Dictionary MHPALTEG.FTX contains:

```karel
.form unnumber
$1, MH_TOOLDEFN
.endform

$2, MH_PORT

$3, MH_PORTFKEY

.form

$6, MH_APPLIO
.endform
```

As explained in the file naming conventions section, the dictionary name extracted from the file name is “PALT”. Dictionary elements 1 and 6 are forms. A variable file named PALT.VR is generated with the program name “*PALT.” It contains the following variables:

- PALT1_IT, PALT1_LN, and PALT1_MS
- PALT6_IT, PALT6_LN, and PALT6_MS

**Note** KCL CLEAR ALL will not clear these variables. To show or clear them, you can SET VAR $CRT_DEFPROG = '*PALT’ and use SHOW VARS and CLEAR VARS.

The form is loaded using the KCL LOAD FORM command.

```
KCL> LOAD FORM filename
```

The name filename is the name of the loadable dictionary file. After this file is loaded, the dictionary name is extracted from filename and is used to load the variable file. This KCL command is equivalent to

```
KCL> LOAD DICT filename dict_name DRAM

KCL> LOAD VARS dict_name
```
10.3.19 Displaying a Form

The DISCTRL_FORM built-in is used to display and control a form on the teach pendant or CRT/KB screens. All input keys are handled within DISCTRL_FORM. This means that execution of your KAREL program will be suspended until an input key causes DISCTRL_FORM to exit the form. Any condition handlers will remain active while your KAREL program is suspended.

**Note** DISCTRL_FORM will only display the form if the USER2 menu is the selected menu. Therefore, use FORCE_SPMENU(device_stat, SPI_TPUSER2, 1) before calling DISCTRL_FORM to force the USER2 menu.

The following screen shows the first template in FORM.FTX as displayed on the teach pendant. This example contains four selectable menu items.

**Figure 10–5. Example of Selectable Menu Items**

<table>
<thead>
<tr>
<th>RUNFORM</th>
<th>LINE 22</th>
<th>RUNNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title here</td>
<td>JOINT 10%</td>
<td></td>
</tr>
<tr>
<td>label here</td>
<td>1/5</td>
<td></td>
</tr>
<tr>
<td>1 Menu item 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Menu item 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Menu item 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Menu item 4 line 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu item 4 line 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Menu item 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The dictionary elements in FORM.FTX, shown in *Example Form Dictionary for Selectable Menu Items*, were used to create the form shown in Figure 10–5.

**Example Form Dictionary for Selectable Menu Items**

```
* Dictionary Form File: form.ftx
*
* Generate form.kl which should be included in your KAREL program
.kl form
.form
$-,forml
&home &reverse "Title here" &standard $new_line
" label here " $new_line
@3,10   " - Menu item 1 "
@4,10   " - Menu item 2 "
@5,10   " - Menu item 3 "
@6,10   " - Menu item 4 line 1 +"
@7,10   " Menu item 4 line 2 "
@8,10   " - Menu item 5 "
* Add as many items as you wish.
```
* The form manager will scroll them.

`^form1_fkey` * specifies element which contains
* function key labels

`?form1_help` * element which contains help

```
.endform
```

```
$-,form1_fkey
   * function key labels
   " F1" &new_line
   " F2" &new_line
   " F3" &new_line
   " F4" &new_line
   " HELP >" &new_line * help must be on F5
   " F6" &new_line
   " F7" &new_line
   " F8" &new_line
   " F9" &new_line
   " F10 >"

* you can have a maximum of 10 function keys labeled
```

```
$-, form1_help
   * help text
   "Help Line 1" &new_line
   "Help Line 2" &new_line
   "Help Line3" &new_line

* You can have a maximum of 48 help lines
```

The program shown in Example Program for Selectable Menu Items was used to display the form shown in Figure 10–5.

Example Program for Selectable Menu Items

```assembly
PROGRAM runform
%NOLOCKGROUP
%INCLUDE form    -- allows you to access form element numbers
%INCLUDE klevccdf
%INCLUDE klevkeys
%INCLUDE klevkmsk
VAR
   device_stat: INTEGER    --tp_panel or crt_panel
   inact_array: ARRAY [1] OF BOOLEAN   --not used
   change_array: ARRAY [1] OF BOOLEAN   --not used
   def_item:   INTEGER
   term_char:   INTEGER
   status:   INTEGER
BEGIN
   device_stat = tp_panel
   FORCE_SPMENU (device_stat, SPI_TPUSER2, 1)--forces the TP USER2 menu
```
def_item = 1 -- start with menu item 1
-- Displays form named FORM1
DISCTRL_FORM ("FORM", form1, value_array, inact_array,
    change_array, kc_func_key, def_item, term_char, status)
WRITE TPERROR (CHR(cc_clear_win)) -- clear the TP error window
IF term_char = ky_select THEN
    WRITE TPERROR ("Menu item", def_item: :1, 'was selected.' )
ELSE
    WRITE TPERROR ('Func key', term_char: :1, ' was selected.' )
ENDIF
END runform

Figure 10–6 shows the second template in FORM.FTX as displayed on the CRT/KB (only 10 numbered lines are shown at one time). This example contains all the edit data types.

Figure 10–6. Example of Edit Data Items

<table>
<thead>
<tr>
<th>RUNFORM</th>
<th>LINE 81</th>
<th>RUNNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title here</td>
<td>JOINT 10%</td>
<td></td>
</tr>
<tr>
<td>label here</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Integer:</td>
<td>12345</td>
</tr>
<tr>
<td>2</td>
<td>Integer:</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Real:</td>
<td>0.000000</td>
</tr>
<tr>
<td>4</td>
<td>Boolean:</td>
<td>TRUE</td>
</tr>
<tr>
<td>5</td>
<td>String:</td>
<td>This is a test</td>
</tr>
<tr>
<td>6</td>
<td>String:</td>
<td>**************</td>
</tr>
<tr>
<td>7</td>
<td>Byte:</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Short:</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>DIN[1]:</td>
<td>OFF</td>
</tr>
<tr>
<td>10</td>
<td>AIN[1]:</td>
<td>0 S</td>
</tr>
<tr>
<td>11</td>
<td>AOUT[2]:</td>
<td>0 U</td>
</tr>
<tr>
<td>12</td>
<td>Enum Type:</td>
<td>FINE</td>
</tr>
<tr>
<td>13</td>
<td>Enum Type:</td>
<td>Green</td>
</tr>
<tr>
<td>14</td>
<td>Enum Type:</td>
<td>Red</td>
</tr>
<tr>
<td>15</td>
<td>Prog Type:</td>
<td>MAINTEST</td>
</tr>
<tr>
<td>16</td>
<td>Prog Type:</td>
<td>RUNFORM</td>
</tr>
<tr>
<td>17</td>
<td>Prog Type:</td>
<td>PRG1</td>
</tr>
<tr>
<td>18</td>
<td>Prog Type:</td>
<td>MAINTEST</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>ITEM</td>
<td>PAGE</td>
<td>PAGE+</td>
</tr>
<tr>
<td>F6</td>
<td>F7</td>
<td>F8</td>
</tr>
</tbody>
</table>

The dictionary elements in FORM.FTX, shown in Example Dictionary for Edit Data Items, were used to create the form shown in Figure 10–6.
Example Dictionary for Edit Data Items

* Dictionary Form File: form.ftx
*
* Generate form.kl which should be included in your KAREL program
 .kl form
 .form
$-,form2
&home &reverse " Title here" &standard &new_line
 " label here " &new_line
 " Integer: " "%-10d" &new_line
 " Integer: " "%-10d(1,32767)" &new_line
 " Real: " "%-12f" &new_line
 " Bolean: " "%-10B(bool_fkey)" &new_line
 " String: " "%-12k(clear)" &new_line
 " String: " "%-20k" &new_line
 " Byte: " "%-10b" &new_line
 " Short: " "%-10h" &new_line
 " DIN[1]: " "%-10P(dout_fkey)" &new_line
 " AIN[1]: " "%-10pu" " "%-1S(sim_fkey)" &new_line
 " AOUT[2]: " "%-10px" " "%-1S(sim_fkey)" &new_line
 " Enum Type: " "%-8n(enum_fkey)" &new_line
 " Enum Type: " "%-6w(enum_subwin)" &new_line
 " Enum Type: " "%-6V(ENUM_VAR)" &new_line
 " Prog Type: " "%-12pk(1)" &new_line
 " Prog Type: " "%-12pk(2)" &new_line
 " Prog Type: " "%-12pk(6)" &new_line
 " Prog Type: " "%-12pk(16)" &new_line
 "form2_fkey
 .'endform
$-,form2_fkey
EXIT" &new_line

*Allows you to specify the labels for F4 and F5 function keys
$-,bool_fkey
"FALSE" &new_line * F5 key label, value will be set FALSE
"TRUE" &new_line * F4 key label, value will be set TRUE
* Allows you to specify the labels for F4 and F5 function keys
$-, dout_fkey
"OFF" &new_line * F5 key label, value will be set OFF
"ON" &new_line * F4 key label, value will be set ON

*Allows you to specify the labels for F4 and F5 function keys
$-, sim_fkey
" UNSIM " &new_line * F5 key label, port will be unsimulated
"SIMULATE" &new_line * F4 key label, port will be simulated

*Allows you to specify the labels for 5 function keys
$-, enum_fkey
"FINE" &new_line * F1 key label, value will be set to 1
"COARSE" &new_line * F2 key label, value will be set to 2
"NOSETTL" &new_line * F3 key label, value will be set to 3
"NODECEL" &new_line * F4 key label, value will be set to 4
"VARDECEL" &new_line * F5 key label, value will be set to 5
* Allows you to specify a maximum of 35 choices in a subwindow
$-,enum_subwin
"Red" * value will be set to 1
$-
"Blue" * value will be set to 2
$-
"Green"
$-
"Yellow"
$-
"\a" * specifies end of subwindow list
* Allows you to specify the choices for the subwindow in a
* variable whose type is an ARRAY[m] of STRING[n].
$-,enum_var
"RUNFORM" &new_line * program name of variable
"CHOICES" &new_line * Variable name containing choices

The program shown in Example Program for Edit Data Items was used to display the form in Figure 10–6.

Example Program for Edit Data Items

PROGRAM runform
%NOLOCKGROUP
%INCLUDE form -- allows you to access form element numbers
%INCLUDE klevccdf
%INCLUDE klevkeys
%INCLUDE klevkmsk
TYPE
   mystruc = STRUCTURE
       byte_var1: BYTE
       byte_var2: BYTE
       short_var: SHORT
ENDSTRUCTURE
VAR
   device_stat: INTEGER -- tp_panel or crt_panel
   value_array: ARRAY [20] OF STRING [40]
   inact_array: ARRAY [1] OF BOOLEAN
   change_array: ARRAY[1] OF BOOLEAN
   def_item: INTEGER
   term_char: INTEGER
status: INTEGER
int_var1: INTEGER
int_var2: INTEGER
real_var: REAL
bool_var: BOOLEAN
str_var1: STRING[20]
str_var2: STRING[12]
struc_var: mystruc
color_sel1: INTEGER
color_sel2: INTEGER
prog_name1: INTEGER[12]
prog_name2: STRING[12]
Prog_name3: STRING[12]
prog_name4: STRING[12]
BEGIN

value_array [1] = 'int_var1'
value_array [2] = 'int_var2'
value_array [3] = 'real_var'
value_array [4] = 'bool_var'
value_array [5] = 'str_var1'
value_array [6] = 'str_var2'
value_array [7] = 'struc_var.byte_var1'
value_array [8] = 'struc_var.short_var'
value_array [9] = 'din[1]'
value_array [10] = 'ain[1]'
value_array [12] = 'aout[2]'
value_array [13] = 'aout[2]'
value_array [14] = '[[*system*]$group[1].$termtype'
value_array [15] = 'color_sel1'
value_array [16] = 'color_sel2'
value_array [17] = 'prog_name1'
value_array [18] = 'prog_name2'
value_array [19] = 'prog_name3'
value_array [20] = 'prog_name4'
choices [1] = '' --not used
choices [2] = 'Red' --corresponds to color_sel12 = 1
choices [3] = 'Blue' --corresponds to color_sel12 = 2
choices [4] = 'Green' --corresponds to color_sel12 = 3
choices [5] = 'Yellow' --corresponds to color_sel12 = 4

-- Initialize variables
int_var1 = 12345
-- int_var2 is purposely left uninitialized
real_var = 0
bool_var = TRUE
str_var1 = 'This is a test'
-- str_var = is purposely left uninitialized
struc_var.byte_var1 = 10
struc_var.short_var = 30
color_sel1 = 3 --corresponds to third item of enum_subwin
color_sel2 = 1
device_stat = crt_panel --specify the CRT/KB for displaying

FORCE_SPMENU(device_stat, SPI_TPUSER2,1)
def_item = 1 -- start with menu item 1
DISCTRL_FORM('FORM', form2, value_array, inact_array,
           change_array, kc_func_key, def_item, term_char, status);
END runform

Figure 10–7 shows the third template in FORM.FTX as displayed on the teach pendant. This example contains display only items. It shows how to automatically load the form dictionary file and the variable data file, from a KAREL program.

Figure 10–7. Example of Display Only Data Items

<table>
<thead>
<tr>
<th>RUNFORM</th>
<th>LINE 53</th>
<th>RUNNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title here</td>
<td>JOINT 10%</td>
<td></td>
</tr>
<tr>
<td>label here</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int: 12345</td>
<td>Bool: TRUE</td>
<td></td>
</tr>
<tr>
<td>Real: 0.000000</td>
<td>Enum: FINE</td>
<td></td>
</tr>
<tr>
<td>DIN[1]: OFF</td>
<td>UNSIMULATED</td>
<td></td>
</tr>
</tbody>
</table>

The dictionary elements in FORM.FTX, shown in Example Dictionary for Display Only Data Items, were used to create the form shown in Figure 10–7.

Example Dictionary for Display Only Data Items

* Dictionary Form File: form.ftx
* Generate form.kl which should be included in your KAREL program
.kl form
.form
$-,form3
   &home &reverse "Title here" &standard &new_line
   "label here" &new_line &new_line
   "Int: " "%-10d" " Bool: " "%-10B(bool_fkey)" &new_line
   "Real: " "%-10f" " Enum: " "%-10n(enum_fkey)" &new_line
   "DIN[""%-1d"]": " "%-10P(dout_fkey)" "%-12S(sim2_fkey)"
*You can have as many columns as you wish without exceeding * 40 columns.
*You can specify blank lines too.
.endform
$-,sim2_fkey
"UNSIMULATED" &new_line * F5 key label, port will be unsimulated
"SIMULATED" &new_line * F4 key label, port will be simulated

The program shown in Example Program for Display Only Data Items was used to display the form shown in Figure 10–7.

Example Program for Display Only Data Items

PROGRAM runform
%NOLOCKGROUP
%INCLUDE form -- allows you to access form element numbers
%INCLUDE klevccdf
%INCLUDE klevkeys
%INCLUDE klevkmsk
    device_stat: INTEGER -- tp_panel or crt_panel
    value_array: ARRAY [20] OF STRING [40]
    inact_array: ARRAY [1] OF BOOLEAN -- not used
    change_array: ARRAY[1] OF BOOLEAN -- not used
    def_item: INTEGER
    term_char: INTEGER
    status: INTEGER
    loaded: BOOLEAN
    initialized: BOOLEAN
    int_var1: INTEGER
    int_var2: INTEGER
    real_var: REAL
    bool_var: BOOLEAN
BEGIN
    -- Make sure 'FORM' dictionary is loaded.
    CHECK_DICT('FORM', form3, status)
    IF status <> 0 THEN
        WRITE TPPROMPT (CR, 'Loading form.....')
        KCL ('CD MF2:', status) --Use the KCL CD command to
            --change directory to MF2:
        KCL ( 'LOAD FORM', status) --Use the KCL load for command
            --to load in the form
        IF status <> 0 THEN
            WRITE TPPROMPT(CR, 'loading from failed, STATUS=',status)
            ABORT --Without the dictionary this program cannot continue.
        ENDIF
    ELSE
        WRITE TPPROMPT (CR, 'FORM already loaded.')
    ENDIF
value_array [1] = 'int_var1'
value_array [2] = 'bool_var'
value_array [3] = 'real_var'
value_array [4] = '[$system*].$group[1].$termtype'
value_array [5] = 'int_var2'
value_array [6] = 'din[1]'
value_array [7] = 'din[1]'

int_var1 = 12345
bool_var = TRUE
real_var = 0
int_var2 = 1
device_stat = tp_panel
FORCE_SPMENU(device_stat, SPI_TPUSER2,1)
def_item = 1 -- start with menu item 1
DISCTRL_FORM('FORM', form3, value_array, inact_array,
             change_array, kc_func_key, def_item, term_char, status);
END runform
Chapter 11

SOCKET MESSAGING

Contents

Chapter 11  SOCKET MESSAGING ................................................................. 11–1
11.1  OVERVIEW ............................................................................................ 11–2
11.2  SYSTEM REQUIREMENTS ........................................................................ 11–2
11.2.1  Overview ............................................................................................ 11–2
11.2.2  Software Requirements ....................................................................... 11–2
11.2.3  Hardware Requirements ..................................................................... 11–2
11.3  CONFIGURING THE SOCKET MESSAGING OPTION ............................. 11–3
11.3.1  Overview ............................................................................................ 11–3
11.3.2  Setting up a Server Tag ..................................................................... 11–3
11.3.3  Setting up a Client Tag ..................................................................... 11–6
11.4  SOCKET MESSAGING AND KAREL .................................................. 11–9
11.4.1  Overview ............................................................................................ 11–9
11.4.2  MSG_CONN( string, integer ) .............................................................. 11–9
11.4.3  MSG_DISCO( string, integer ) .............................................................. 11–9
11.4.4  MSG_PING( string, integer ) ............................................................... 11–10
11.4.5  Exchanging Data during a Socket Messaging Connection ................. 11–10
11.5  NETWORK PERFORMANCE .................................................................. 11–10
11.5.1  Overview ............................................................................................ 11–10
11.5.2  Guidelines for a Good Implementation ............................................. 11–11
11.6  PROGRAMMING EXAMPLES ............................................................. 11–11
11.6.1  ........................................................................................................ 11–11
11.6.2  A KAREL Client Application ............................................................ 11–11
11.6.3  A KAREL Server Application ............................................................ 11–14
11.6.4  ANSI C Loopback Client Example ...................................................... 11–16
11.1 **OVERVIEW**

The User Socket Messaging Option gives you the benefit of using TCP/IP socket messaging from KAREL.

Socket Messaging enables data exchange between networked robots and a remote PC with LINUX, or a UNIX workstation. A typical application of Socket Messaging might be a robot running a KAREL program that sends process information to a monitoring program on the remote PC. The combination of PC-Interface option on the robot and PC-Developers Kit on the PC is recommended for data exchange between the robot and a Windows-based PC.

Socket Messaging uses the TCP/IP protocol to transfer raw data, or data that is in its original, unformatted form across the network. Commands and methods that Socket Messaging uses to transfer data are part of the TCP/IP protocol. Since Socket Messaging supports client and server tags, applications requiring timeouts, heartbeats, or data formatting commands can provide these additional semantics at both the client and server (application) sides of the socket messaging connection.

11.2 **SYSTEM REQUIREMENTS**

11.2.1 Overview

This section contains information about the compatibility of socket messaging with some typical network software, transmission protocols, and interface hardware.

11.2.2 Software Requirements

Socket Messaging is compatible with all other Internet Options including DNS, FTP, Web Server, and Telnet.

*Note* Client and Server tags are shared between Socket Messaging and other supported protocols, such as FTP. For example, a tag can be set for either FTP operation, or for SM (Socket Messaging) operation.

11.2.3 Hardware Requirements

Socket Messaging is compatible with all network hardware configurations that use the TCP/IP network protocol. Some of these network hardware configurations include Ethernet, serial PPP connections and PPP modem connections.
### 11.3 CONFIGURING THE SOCKET MESSAGING OPTION

#### 11.3.1 Overview

In order to use Socket Messaging, you need to configure the following network hardware and software parameters:

- On the server,
  - The port you want to use for socket messaging
- On the client,
  - The IP address or name of your server
  - The port on the server that you want to use for socket messaging.

Use Procedure 11-1 to set up a Socket Messaging Server Tag. Use Procedure 11-2 to set up a Socket Messaging Client Tag.

**Note** The server port at which the server listens on should match the port the client tries to connect on.

#### 11.3.2 Setting up a Server Tag

You need configure the server tags you want to use for socket messaging. Use Procedure 11-1 to set up your server tags.

**Note** If the server tags you want to use are being used by a network protocol other than TCP/IP, you need to undefine the tags before they can be used for socket messaging. After making sure the tag you want to use is not critical to another component of your network, you must undefine the tag.

**Procedure 11-1 Setting up a Server Tag**

**Conditions**

- The tag you want to set up is not configured to be used by another device on your network.

**Steps**

1. Cold start the controller.
   a. **On the teach pendant**, press and hold the SHIFT and RESET keys. Or, **on the operator panel**, press and hold RESET.
   b. While still pressing SHIFT and RESET on the teach pendant (or RESET on the operator panel), turn on the power disconnect circuit breaker.
   c. Release all of the keys.
2. On the teach pendant, press MENUS.
3. Select SETUP.
4. Press F1, [TYPE].
5. Select Host Comm.
6. Press F4, [SHOW].
7. Choose Servers.
8. Move the cursor to the tag you want set up for Socket Messaging, and press F3, DETAIL. You will see screen similar to the following.

```
SETUP Tags

Tag S3:

Comment: ****************
Protocol Name: ********
Current State: UNDEFINED
Startup State:
Server IP/Hostname:*****************
Remote Path/Share: *****************
Port: *****
Inactivity Timeout: 15 min
Username: anonymous
Password **********
```

9. Move the cursor to Protocol name, and press F4, [CHOICE].
10. Select SM.
11. Move the cursor to Startup State, and press F4, [CHOICE].
12. Select START.
13. Press F2, [ACTION].
14. Select DEFINE.
15. Press F2, [ACTION].
16. Select START.

17. **Set the system variable:**
   a. Press MENUS.
   b. Select NEXT.
   c. Select SYSTEM, and press F1, [TYPE].
   d. Select Variables.
e. Move the cursor to $HOSTS_CFG, and Press ENTER.

f. Move the cursor to the structure corresponding to the tag selected in Step 8. For example, if you are setting up tag S3, move the cursor structure element [3], as shown in the following screen.

```
SYSTEM Variables
$HOSTS_CFG
1 [1] HOST_CFG_T
2 [2] HOST_CFG_T
3 [3] HOST_CFG_T
4 [4] HOST_CFG_T
5 [5] HOST_CFG_T
6 [6] HOST_CFG_T
7 [7] HOST_CFG_T
8 [8] HOST_CFG_T
```

g. Press ENTER. You will see a screen similar to the following.

```
SYSTEM Variables
$HOSTS_CFG[3]
1 $COMMENT *uninit*
2 $PROTOCOL 'SM'
3 $PORT *uninit*
4 $OPER 3
5 $STATE 3
6 $MODE *uninit*
7 $REMOTE *uninit*
8 $REPERRS FALSE
9 $TIMEOUT 15
10 $PATH *uninit*
11 $STRT_PATH *uninit*
12 $STRT_REMOTE *uninit*
13 $USERNAME *uninit*
14 $PWRD_TIMOUT 0
15 $SERVER_PORT 0
```

h. Move the cursor to $SERVER_PORT. Type in the name of the TCP/IP port you want to use for socket messaging. The server tag is now ready to use from a KAREL program.
11.3.3 Setting up a Client Tag

You need configure the client tags you want to use for socket messaging. Use Procedure 11-2 to set up your server tags. You can also use Procedure 11-2 to undefine tags.

**Note** If the client tags you want to use are being used by a network protocol other than TCP/IP, you need to undefine the tags before they can be used for socket messaging.

**Procedure 11-2 Setting up a Client Tag**

**Conditions**

- The tag you want to set up is not configured to be used by another device on your network.

**Steps**

1. Cold start the controller.
   a. **On the teach pendant**, press and hold the SHIFT and RESET keys. Or, **on the operator panel**, press and hold RESET.
   b. While still pressing SHIFT and RESET on the teach pendant (or RESET on the operator panel), turn on the power disconnect circuit breaker.
   c. Release all of the keys.
2. On the teach pendant, press MENUS.
3. Select SETUP.
4. Press F1, [TYPE].
5. Select Host Comm.
6. Press F4, [SHOW].
7. Choose Clients.
8. Move the cursor to the tag you want set up for Socket Messaging, and press F3, DETAIL. You will see screen similar to the following.
SETUP Tags

Tag C3:

Comment: ****************
Protocol Name: ********
Current State: UNDEFINED
Startup State:
Server IP/Hostname: ****************
Remote Path/Share: ****************
Port: *****
Inactivity Timeout: 15 min
Username: anonymous
Password **********

9. Move the cursor to the Protocol Name item, and press F4, [CHOICE].
10. Select SM.
11. Move the cursor to the Startup State item, press F4, [CHOICE], and choose DEFINE.
12. Move the cursor to the Server IP/Hostname item, and press ENTER.
13. Type in hostname or IP address the of the remote host server you want to use for socket messaging.

Note If you are not using DNS, you must add the remote host and its IP address into the host entry table.

14. Press F2, [ACTION], and select DEFINE.
15. Set the system variable:
   a. Press MENUS.
   b. Select NEXT.
   c. Select SYSTEM, and press F1, [TYPE].
   d. Select Variables.
   e. Move the cursor to $HOSTC_CFG, and press ENTER.
   f. Move the cursor to the structure corresponding to the tag selected in Step 8. For example, if you are setting up tag C3, move the cursor structure element [3], as shown in the following screen.
### SYSTEM Variables

<table>
<thead>
<tr>
<th>$HOSTC_CFG</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>2</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>3</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>4</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>5</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>6</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>7</td>
<td>HOST_CFG_T</td>
</tr>
<tr>
<td>8</td>
<td>HOST_CFG_T</td>
</tr>
</tbody>
</table>

- **g.** Press ENTER. You will see a screen similar to the following.

<table>
<thead>
<tr>
<th>$HOSTC_CFG[3]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  $COMMENT</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>2  $PROTOCOL</td>
<td>'SM'</td>
</tr>
<tr>
<td>3  $PORT</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>4  $OPER</td>
<td>3</td>
</tr>
<tr>
<td>5  $STATE</td>
<td>3</td>
</tr>
<tr>
<td>6  $MODE</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>7  $REMOTE</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>8  $REPERRS</td>
<td>FALSE</td>
</tr>
<tr>
<td>9  $TIMEOUT</td>
<td>15</td>
</tr>
<tr>
<td>10 $PATH</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>11 $STRT_PATH</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>12 $STRT_REMOTE</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>13 $USERNAME</td>
<td><em>uninit</em></td>
</tr>
<tr>
<td>14 $PWRD_TIMEOUT</td>
<td>0</td>
</tr>
<tr>
<td>15 $SERVER_PORT</td>
<td>0</td>
</tr>
</tbody>
</table>

- **h.** Move the cursor to $SERVER_PORT. Type in the name of the TCP/IP server port you want to use for socket messaging. The client tag is now ready to use from a KAREL program.
11.4 SOCKET MESSAGING AND KAREL

11.4.1 Overview

Socket messaging is an integrated component of KAREL. When you use socket messaging functions and utilities from a KAREL program, the syntax is similar to other file read and write operations, except that you need to establish a network connection when you use socket messaging functions and utilities.

The following KAREL socket messaging functions and utilities enable the server to establish a connection with a remote host on your network. There are several KAREL program samples in this section that provide examples of how these functions and utilities can be used with KAREL file read and write functions and utilities to write a complete Socket Messaging KAREL client or a server program or application. The Environment flbt statement is required to use any of the listed builtins (%ENVIRONMENT flbt).

11.4.2 MSG_CONN( string, integer )

MSG_CONN needs to be called before any tag can be used for socket messaging.

The first parameter of this command contains the tag name ("S1:" for example) and the second parameter is an integer that will contain the status of the operation. If you are using this command to connect to a server tag, this command will return a status value only after a remote client device has established a connection with this server tag.

If you are using this command to connect to a client tag, this command will return a status value only if the remote server is attempting to accept the connection. If the connection was successful, the command will return a value indicating a successful connection was made. If the connection was not successful, the command will return a value indicating that a connection error has occurred.

During a socket messaging session, you must use MSG_DISCO to close the socket connection with a client or server tag before any subsequent attempts to connect to the same client or server tag can be made using MSG_CONN.

11.4.3 MSG_DISCO( string, integer )

MSG_DISCO is used to close socket messaging connections. If a connection is lost, perhaps because a READ or WRITE error occurred when the remote server terminated a socket messaging connection, you will need to use MSG_DISCO to close the connection to the remote server. In this case, MSG_DISCO must be used to close the connection at the client side before MSG_CONN can be used to establish another connection to the remote server.
The first parameter of this command contains the tag name (e.g. "S1:") and the second parameter is an integer value that indicates the status of closing the connection on the client side.

11.4.4 MSG_PING( string, integer )

MSG_PING is a utility command used to check network connections with a remote host, so that you can determine if it is currently connected to the network. The MSG_PING command sends ping packets to the remote host and waits for a reply. (Ping packets are chunks of data that are transferred between hosts on a network.) If there is no reply from the remote host, this usually means that you will not be able to use other network protocols like FTP, TELNET, or Socket Messaging to connect to that host. If you have attempted without success to use Socket Messaging to connect to a remote host, the MSG-PING utility is a good place to start in trying to diagnose the problem.

The first parameter of this command contains the name of the remote host to ping. If you are not using DNS on your network, the host name and IP address of the remote host will have to exist in the Host Entry table.

11.4.5 Exchanging Data during a Socket Messaging Connection

After you have successfully established a socket messaging connection, you can use KAREL commands to exchange data between connected devices. KAREL has several commands that can be used for data exchange operations:

- OPEN FILE
- WRITE
- READ
- BYTES_AHEAD

Refer to the KAREL Reference Manual for details.

11.5 NETWORK PERFORMANCE

11.5.1 Overview

Performance of socket messaging on your network will vary depending upon the number of devices connected to the network, the number of applications being run on the controller, the network cabling configuration, and number of hops that the message will have to make to reach its destination device.
Note Hops is a term that indicates the number of routers between the source host and destination host. In general, the fewer the number of hops the data makes from router to router, the faster the data is transmitted between the source and destination hosts. Data transfer is fastest between hosts on the same network.

11.5.2 Guidelines for a Good Implementation

Use the following guidelines when implementing a solution for any application using socket messaging.

- You must not transfer small data separately, instead gather the data and transfer as a larger packet. This applies to both read and write.
- You must understand that the rate of data does not flood the remote side.
- You must understand that there are other nodes on the Ethernet network so performance cannot be guaranteed.
- You must understand that other applications on robot also use TCP/IP and performance can be affected.

11.6 PROGRAMMING EXAMPLES

11.6.1

This section contains programming examples for a KAREL socket messaging client, and a KAREL socket messaging server. There is also a UNIX-based ANSI C example for a loopback client application, which assumes that you have access to a UNIX-compatible ANSI C compiler, and a basic knowledge of programming in the ANSI C language.

Note The KAREL examples assume the appropriate tags (C2 for client and S3 for Server) have been setup for socket messaging using Procedure 11-1 and Procedure 11-2.

11.6.2 A KAREL Client Application

A KAREL Client Application provides code for a basic KAREL client application that can be used to establish a socket messaging connection to a remote host, which could be the KAREL server socket messaging application shown in KAREL Server Application.

A KAREL Client Application

-- This material is the joint property of Fanuc Robotics Corporation and
-- FANUC LTD Japan, and must be returned to either Fanuc Robotics
-- Corporation or FANUC LTD Japan immediately upon request. This material
-- and the information illustrated or contained herein may not be
-- reproduced, copied, used, or transmitted in whole or in part in any way
-- without the prior written consent of both Fanuc Robotics and FANUC
--
-- All Rights Reserved
-- Copyright (C) 2000
-- Fanuc Robotics Corporation
-- FANUC LTD Japan
--
-- Karel is a registered trademark of
-- Fanuc Robotics Corporation
--
-- Program: loopcl.kl - Program for TCP Messaging
--
-- Description:
--
-- This program serves as an example on how to use TCP messaging
-- and write a client Karel program.
--
-- Authors: Fanuc Robotics Corporation
-- 3900 West Hamlin
-- Rochester Hills, MI 48309
--
-- Modification history:
--
---------------------------------------------------------------------
PROGRAM loopcl
%STACKSIZE = 4000
%NOLOCKGROUP
%NOPAUSE=ERROR+COMMAND+TPENABLE
%ENVIRONMENT uif
%ENVIRONMENT sysdef
%ENVIRONMENT memo
%ENVIRONMENT kclop
%ENVIRONMENT bynam
%ENVIRONMENT fdev
%ENVIRONMENT flbt
%INCLUDE klevccdf
%INCLUDE klevkeys
%INCLUDE klevkmsk
---------------------------------------------------------------------
VAR
  file_var : FILE
  tmp_int : INTEGER
  tmp_str : STRING[128]
BEGIN
SET_FILE_ATR(file_var, ATR_IA)
SET_VAR(entry, '*SYSTEM*', $HOSTC_CFG[2].$SERVER_PORT', 59002, status)
-- Connect the tag
WRITE('Connecting..', cr)
MSG_CONNECT('C2:', status)
WRITE(' Connect Status = ', status, cr)
loop1 = TRUE
IF status = 0 THEN
WHILE loop1 = TRUE DO
   WRITE('Opening File..', cr)
   OPEN FILE file_var('rw', 'C2:')
   status = IO_STATUS(file_var)
   IF status = 0 THEN
      FOR tmp_int = 1 TO 100 DO
         tmp_str = '0123456789012345'
         WRITE file_var(tmp_str::10)
         WRITE('Wrote 126 Bytes', cr)
         IF status <> 0 THEN
            WRITE('Loop Test Fails', cr)
            loop1 = FALSE
            tmp_int = 100
         ELSE
            WRITE('Read 126 Bytes', cr)
            READ file_var(tmp_str::10)
            status = IO_STATUS(file_var)
            WRITE('Read Status ', status, cr)
         ENDIF
      ENDFOR
      WRITE('Closed File', cr)
      CLOSE FILE file_var
   ELSE
      WRITE('Error Opening File', cr)
      loop1 = FALSE
   ENDIF
ENDWHILE
WRITE('Disconnecting..', cr)
MSG_DISCO('C2:', status)
WRITE('Done.', cr)
11.6.3 A KAREL Server Application

KAREL Server Application provide code for a basic KAREL server application that can be used to host a socket messaging connection made by a remote client, which could be the KAREL client socket messaging application shown in A KAREL Client Application.

KAREL Server Application

--- This material is the joint property of Fanuc Robotics Corporation and FANUC LTD Japan, and must be returned to either Fanuc Robotics Corporation or FANUC LTD Japan immediately upon request. This material and the information illustrated or contained herein may not be reproduced, copied, used, or transmitted in whole or in part in any way without the prior written consent of both Fanuc Robotics and FANUC.
---
--- All Rights Reserved
--- Copyright (C) 2000
--- Fanuc Robotics Corporation
--- FANUC LTD Japan
--- Karel is a registered trademark of
--- Fanuc Robotics Corporation
--- +
--- Program: tcpserv3.kl - Program for TCP Messaging
---
--- Description:
---
--- This program serves as an example on how to use TCP messaging and write a server Karel program.
---
--- Authors: Fanuc Robotics Corporation
--- 3900 West Hamlin
--- Rochester Hills, MI 48309
---
--- Modification history:
---
--- PROGRAM tcpserv3
%STACKSIZE = 4000
%NOLOCKGROUP
%NOPAUSE=ERROR+COMMAND+TPENABLE
%ENVIRONMENT uif
%ENVIRONMENT sysdef
%ENVIRONMENT memo
%ENVIRONMENT kclop
%ENVIRONMENT bynam
%ENVIRONMENT fdev
%ENVIRONMENT flbt
%INCLUDE klevccdf
%INCLUDE klevkeys
%INCLUDE klevkmsk

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 the server port before doing a connect
    SET_VAR(entry, '**SYSTEM**', '$HOSTS_CFG[3].$SERVER_PORT', 59002, status)
    WRITE('Connecting..', cr)
    MSG_CONNECT('S3:', status)
    WRITE(' Connect Status = ', status, cr)
    IF status = 0 THEN
        -- Open S3:
        WRITE ('Opening', cr)
        FOR tmp_int1 = 1 TO 20 DO
            OPEN FILE file_var ('rw', 'S3:')
            status = IO_STATUS(file_var)
            WRITE (status, cr)
            IF status = 0 THEN
                -- write an integer
                FOR tmp_int = 1 TO 1000 DO
                    WRITE('Reading', cr)
                    -- Read 10 bytes
                    BYTES_AHEAD(file_var, entry, status)
                    WRITE(entry, status, cr)
                    READ file_var (tmp_str::10)
                    status = IO_STATUS(file_var)
                    WRITE (status, cr)
            END IF
        END FOR
    END IF
END
-- Write 10 bytes
WRITE (tmp_str::10, cr)
status = IO_STATUS(file_var)
WRITE (status, cr)
ENDFOR
CLOSE FILE file_var
ENDDIF
ENDFOR
WRITE(‘Disconnecting..’,cr)
MSG_DISCO(‘S3:’,status)
WRITE(‘Done.’,cr)
ENDIF
END tcpserv3

11.6.4 ANSI C Loopback Client Example

ANSI C UNIX-Based Loopback Client Example provides an example of a UNIX-based loopback client that can be used to establish a connection with a remote host.

ANSI C UNIX-Based Loopback Client Example

/* BSD Standard Socket Programming Example - UNIX */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define SERV_TCP_PORT 59002
#define SERV_HOST_ADDR "199.5.148.56"
#define MAXLINE 512
int written(int fd, char *ptr, int nbytes);
int readline(int fd, char *ptr, int maxlen);
void str_cli(int sockfd);
char *pname;
int main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
    pname = argv[0];

    bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
serv_addr.sin_port = htons(SERV_TCP_PORT);

if((sockfd = socket(AF_INET, SOCK_STREAM,0)) < 0){
    printf("Client: Can't Open Stream Socket\n");
}

printf("Client: Connecting...\n");

if(connect(sockfd,(struct sockaddr *)&serv_addr, sizeof(serv_addr))<0){
    printf("Client: Can't Connect to the server\n");
} else{
    str_cli(sockfd);
}
exit(0);

void str_cli (int sockfd)
{
    int n, i;
    char sendline[MAXLINE], recvline[MAXLINE + 1];
    while(1)
    {
        memset (sendline, 2, 128);
        if(written(sockfd, sendline, 126)!=126){
            printf("strcli:written error on sock\n");
        }
        i = readline(sockfd, recvline, 126);
    }
}

int readline(int fd, char *ptr, int maxlen)
{
    int n, rc;
    char c;
    for(n = 0; n < maxlen; n++){
        if((rc = read(fd, &c, 1)) == 1){
            *ptr++ = c;
            if(c==\n'){
                break;
            }
        } else if(rc== 0) {
            if(n== 0) {
                return (0);
            }
else{
    break;
}
}
}
else{
    return (-1);
}
}
*ptr = 0;
return (n);
}
int written(int fd, char *ptr, int nbytes)
{
    int nleft, nwritten;
    nleft = nbytes;
    while(nleft > 0) {
        nwritten = write(fd, ptr, nleft);
        if(nwritten <= 0) {
            return(nwritten);
        }
        nleft -= nwritten;
        ptr += nwritten;
    }
    return(nbytes - nleft);
}
Chapter 12

SYSTEM VARIABLES

Contents

Chapter 12  SYSTEM VARIABLES ...................................................................................... 12–1
12.1  ACCESS RIGHTS ........................................................................................................ 12–2
12.2  STORAGE .................................................................................................................... 12–3
System variables are variables that are declared as part of the KAREL system software. They have permanently defined variable names that begin with a dollar sign ($). Many system variables are structure variables, in which case each field also begins with a dollar sign ($). Many are robot specific, meaning their values depend on the type of robot that is attached to the system.

System variables have the following characteristics:

- They have predefined data types that can be any one of the valid KAREL data types.
- The initial values of the system variables are either internal default values or variables stored in the default system variable file, SYSDEF.SV.
- When loading and saving system variables from the FILE screen or KCL, the system variable file name defaults to SYSVARS.SV.
- Access rights govern whether or not you can examine or change system variables.
- Modified system variables can be saved to reflect the current status of the system.

See Also: Chapter 2 LANGUAGE ELEMENTS for more information on the data types available in KAREL.

### 12.1 ACCESS RIGHTS

The following rules apply to system variables:

- If a system variable allows a KAREL program to read its value, you can use that value in the same context as you use program variable values or constant values in KAREL programs.

  For example, these system variables can be used on the right hand side of an assignment statement or as a test condition in a control statement.

- If a system variable allows a KAREL program to write its value, you can use that system variable in any context where you assign values to variables in KAREL programs.

The symbols for the program access rights are listed in Table 12-1. These symbols are given for each of the system variables in the FANUC Robotics SYSTEM Software Reference Manual.

<table>
<thead>
<tr>
<th>Access</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>No access</td>
</tr>
<tr>
<td>RO</td>
<td>Read only</td>
</tr>
<tr>
<td>RW</td>
<td>Read and write</td>
</tr>
<tr>
<td>FP</td>
<td>Field protection; if it is a structure variable, one of the first three access rights will apply.</td>
</tr>
</tbody>
</table>

Table 12-1. Access Rights for System Variables
See Also: *FANUC Robotics Software Reference Manual* for system variables

### 12.2 STORAGE

System variables are assigned an initial value upon power up based on

- Internal default values
- Values stored in the default system variable file, SYSDEF.SV
Chapter 13

KAREL COMMAND LANGUAGE (KCL)

Contents

Chapter 13  KAREL COMMAND LANGUAGE (KCL) .................................................. 13–1
13.1  COMMAND FORMAT ................................................................. 13–2
13.1.1  Default Program ................................................................. 13–2
13.1.2  Variables and Data Types .................................................. 13–3
13.2  PROGRAM CONTROL COMMANDS .......................................... 13–3
13.3  ENTERING COMMANDS ............................................................ 13–3
13.3.1  Abbreviations ........................................................................ 13–4
13.3.2  Error Messages ....................................................................... 13–4
13.3.3  Subdirectories ......................................................................... 13–4
13.4  COMMAND PROCEDURES ....................................................... 13–4
13.4.1  Command Procedure Format ............................................... 13–5
13.4.2  Creating Command Procedures ........................................... 13–6
13.4.3  Error Processing .................................................................... 13–6
13.4.4  Executing Command Procedures ........................................... 13–6
The KAREL command language (KCL) environment contains a group of commands that can be used to direct the KAREL system. KCL commands allow you to develop and execute programs, work with files, get information about the system, and perform many other daily operations.

The KCL environment can be displayed on the CRT/KB by pressing MENUS (F10) and selecting KCL from the menu.

In addition to entering commands directly at the KCL prompt, KCL commands can be executed from command files.

### 13.1 COMMAND FORMAT

A command entry consists of the command keyword and any arguments or parameters that are associated with that command. Some commands also require identifiers specifying the object of the command.

- KCL command keywords are action words such as LOAD, EDIT, and RUN. Command arguments, or parameters, help to define on what object the keyword is supposed to act.
- Many KCL commands have default arguments associated with them. For these commands, you need to enter only the keyword and the system will supply the default arguments.
- KCL supports the use of an asterisk (*) as a wildcard, which allows you to specify a group of objects as a command argument for the following KCL commands:
  - COPY
  - DELETE FILE
  - DIRECTORY
- KCL identifiers follow the same rules as the identifiers in the KAREL programming language.
- All of the data types supported by the KAREL programming language are supported in KCL. Therefore, you can create and set variables in KCL.

See Also: Chapter 2 LANGUAGE ELEMENTS, and Chapter 9 FILE SYSTEM.

### 13.1.1 Default Program

Setting a program name as a default for program name arguments and file name arguments allows you to issue a KCL command without typing the name.

The KCL default program can be set by doing one of the following:

- Using the SET DEFAULT KCL command
- Selecting a program name at the SELECT menu on the CRT/KB
13.1.2 Variables and Data Types

The KCL> CREATE VARIABLE command allows you to declare variables. The KCL> SET VARIABLE command permits you to assign values to declared variables. Assigned values can be INTEGER, REAL, BOOLEAN, and STRING data types. Values can be assigned to particular ARRAY elements or specified PATH nodes. VECTOR variables are assigned as three REAL values, and POSITION variables are assigned as six REAL values.

See Also: CREATE VARIABLE and SET VARIABLE KCL commands in Appendix C, “KCL Command Alphabetical Description”

13.2 PROGRAM CONTROL COMMANDS

KCL commands can be used to run programs. In some cases, these programs may cause motion such as when a teach pendant program is run or when a KAREL program that calls a teach pendant program is run. The device from which the KCL command is issued must have motion control in order to do this. Refer to the $RMT_MASTER description in the FANUC Robotics Software Reference Manual for more information about assigning motion control to a remote device.

Program control commands:

- Can immediately cause robot and/or auxiliary axis motion, or have the potential to cause motion
- Can be executed only if a number of conditions are met

The following commands are program control commands:

- CONTINUE
- RUN

Warning

Be sure that the robot work envelope is clear of personnel before issuing a program control command or starting a robot that automatically executes a program at power up. Otherwise, you could injure personnel or damage equipment.

13.3 ENTERING COMMANDS

You can enter KCL commands only from the CRT/KB.

To enter KCL commands:
1. Press MENUS (F10) at the CRT/KB.
2. Select KCL.
3. Enter commands at the KCL prompt.

By entering the first keyword of a KCL command that requires more than one keyword, and by pressing ENTER, a list of all additional KCL keywords will be displayed.

For example, entering **DELETE** at the KCL prompt will display the following list of possible commands: “FILE, NODE, or VARIABLE.”

**Note** The up arrow key can be used to recall any of the last ten commands entered.

### 13.3.1 Abbreviations

Any KCL command can be abbreviated as long as the abbreviations are unique in KCL. For example, **TRAN** is unique to TRANSLATE and **ED**, to EDIT.

### 13.3.2 Error Messages

If you enter a KCL command incorrectly, KCL displays the appropriate error message and returns the KCL> prompt, allowing you to reenter the command. An up arrow (^) indicates the offending character or the beginning of the offending word.

### 13.3.3 Subdirectories

Subdirectories are available on the memory card device. Subdirectories allow both memory cards and Flash disk cards to be formatted on any MS-DOS file system. You can perform all KCL file related commands on subdirectories. You can nest subdirectories up to many levels. However, FANUC Robotics does not recommend nesting subdirectories greater than eight levels.

### 13.4 COMMAND PROCEDURES

Command procedures are a sequence of KCL commands that are stored in a command file (.CF file type) and can be executed automatically in sequence.

- Command procedures allow you to use a sequence of KCL commands without typing them over and over.
- Command procedures are executed using the RUNCF command.
13.4.1 Command Procedure Format

All KCL commands except RUNCF can be used inside a command procedure. For commands that require confirmation, you can enter either the command and confirmation on one line or KCL will prompt for the confirmation on the input line. Confirmation in a Command Procedure displays CLEAR ALL as the KCL command and YES as the confirmation.

Confirmation in a Command Procedure
Enter command and confirmation on one line:
CLEAR ALL YES

Nesting Command Procedures

Use the following guidelines when nesting command procedures:

- Command procedures can be nested by using %INCLUDE filename inside a command procedure.
- Nesting of command procedures is restricted to four levels.

If nesting of more than four command procedures is attempted, KCL will detect the error and take the appropriate action based on the system variable $STOP_ON_ERR. Refer to Section 13.4.3 for more information on $STOP_ON_ERR.

See Also: Section 13.4.3, “Error Processing”

Continuation Character

The KCL continuation character, ampersand (&), allows you to continue a command entry across more than one line in a command procedure.

You can break up KCL commands between keywords or between special characters.

For example, use the ampersand (&) to continue a command across two lines:
CREATE VAR [TESTING_PROG.]PICK_UP_PNT &:POSITION

Comments

Comment lines can be used to document command procedures. The following rules apply to using comments in command procedures:

- Precede comments with two consecutive hyphens (--).
- Comments can be placed on a line by themselves or at the end of a command line.
13.4.2 Creating Command Procedures

A command procedure can be created by typing in the list of commands into a command file and saving the file. This can be done using the full screen editor.

See Also: EDIT KCL commands, Appendix C, “KCL Command Alphabetical Description”

13.4.3 Error Processing

If the system detects a KCL error while a command procedure is being executed, the system handles the error in one of two ways, depending on the value of the system variable $STOP_ON_ERR:

- If $STOP_ON_ERR is TRUE when a KCL error is detected, the command procedure terminates and the KCL> prompt returns.
- If $STOP_ON_ERR is FALSE, the system ignores KCL errors and the command procedure runs to completion.

13.4.4 Executing Command Procedures

Each command in a command procedure is displayed as it is executed unless the SET VERIFY OFF command is used. Each command is preceded with the line number from the command file. However, if the file is not on the RD: device, the entire command file is read into memory before execution and line numbers will be omitted from the display.

Command procedures can be executed using the KCL RUNCF command.
Chapter 14

INPUT/OUTPUT SYSTEM

Contents

Chapter 14  INPUT/OUTPUT SYSTEM ................................................................. 14–1
  14.1 USER-DEFINED SIGNALS ................................................................. 14–2
     14.1.1 DIN and DOUT Signals ............................................................ 14–2
     14.1.2 GIN and GOUT Signals ......................................................... 14–3
     14.1.3 AIN and AOUT Signals ......................................................... 14–3
     14.1.4 Hand Signals ........................................................................ 14–5
  14.2 SYSTEM-DEFINED SIGNALS .......................................................... 14–5
     14.2.1 Robot Digital Input and Output Signals (RDI/RDO) ................ 14–6
     14.2.2 Operator Panel Input and Output Signals (OPIN/OPOUT) ...... 14–6
     14.2.3 Teach Pendant Input and Output Signals (TPIN/TPOUT) ...... 14–17
  14.3 Serial Input/Output ........................................................................ 14–22
     14.3.1 Serial Input/Output ................................................................. 14–22
The Input/Output (I/O) system provides user access with KAREL to user-defined I/O signals, system-defined I/O signals and communication ports. The user-defined I/O signals are controlled in a KAREL program and allow you to communicate with peripheral devices and the robot end-of-arm tooling. System-defined I/O signals are those that are designated by the KAREL system for specific purposes. Standard and optional communications port configurations also exist.

The number of user-defined I/O signals is dependent on the controller hardware and on the types and number of modules selected.

14.1 USER-DEFINED SIGNALS

User-defined signals are those input and output signals whose meaning is defined by a KAREL program. You have access to user-defined signals through the following predefined port arrays:

- DIN (digital input) and DOUT (digital output)
- GIN (group input) and GOUT (group output)
- AIN (analog input) and AOUT (analog output)

In addition to the port arrays, you have access to robot hand control signals through KAREL OPEN and CLOSE HAND statements.

14.1.1 DIN and DOUT Signals

The DIN and DOUT signals provide access to data on a single input or output line in a KAREL program.

The program treats the data as a BOOLEAN data type. The value is either ON (active) or OFF (inactive). You can define the polarity of the signal as either active-high (ON when voltage is applied) or active-low (ON when voltage is not applied).

Input signals are accessed in a KAREL program by the name DIN[n], where “n” is the signal number.

Evaluating DIN signals causes the system to perform read operations of the input port. Assigning a value to a DIN signal is an invalid operation unless the DIN signal has been simulated. These can never be set in a KAREL program, unless the DIN signal has been simulated.

Evaluating DOUT signals causes the system to return the currently output value from the specified output signal. Assigning a value to a DOUT signal causes the system to set the output signal to ON or OFF.

To turn on a DOUT:

DOUT[n] = TRUE or
DOUT\[n\] = ON

**To turn off a DOUT:**

DOUT\[n\] = FALSE or  
DOUT\[n\] = OFF

You assign digital signals to the ports on I/O devices using teach pendant I/O menus or the KAREL built-in routine SET_PORT_ASG.

### 14.1.2 GIN and GOUT Signals

The GIN and GOUT signals provide access to DINs and DOUTs as a group of input or output signals in a KAREL program. A group can have a size of 1 to 16 bits, with each bit corresponding to an input or output signal. You define the group size and the DINs or DOUTs associated with a specific group. The first (lowest numbered) port is the least significant bit of the group value.

The program treats the data as an INTEGER data type. The unused bits are interpreted as zeros.

Input signals are accessed in KAREL programs by the name GIN\[n\], where “n” is the group number.

Evaluating GIN signals causes the system to perform read operations of the input ports. Assigning a value to a GIN signal is an invalid operation unless the GIN signal has been simulated. These can never be set in a KAREL program, unless the GIN signal has been simulated.

Setting GOUT signals causes the system to return the currently output value from the specified output port. Assigning a value to a GOUT signal causes the system to perform an output operation.

To control a group output, the integer value equivalent to the desired binary output is used. For example, the command GOUT\[n\] = 25 will have the following binary result “000000011001” where 1 = output on and 0 = output off, least significant bit (LSB) being the first bit on the right.

You assign group signals using teach pendant I/O menus or the KAREL built-in routine SET_PORT_ASG.

### 14.1.3 AIN and AOUT Signals

The AIN and AOUT signals provide access to analog electrical signals in a KAREL program. For input signals, the analog data is digitized by the system and passed to the KAREL program as a 16 bit binary number, of which 14 bits, 12 bits, or 8 bits are significant depending on the analog module. The program treats the data as an INTEGER data type. For output signals, an analog voltage corresponding to a programmed INTEGER value is output.
Input signals are accessed in KAREL programs by the name AIN[n], where “n” is the signal number.

Evaluating AIN signals causes the system to perform read operations of the input port. Setting an AIN signal at the Teach Pendant is an invalid operation unless the AIN signal has been simulated. These can never be set in a KAREL program, unless the AIN signal has been simulated.

The value displayed on the TP or read by a program from an analog input port are dependent on the voltage supplied to the port and the number of bits of significant data supplied by the analog-to-digital conversion. For positive input voltages, the values read will be in the range from 0 to \(2^{**}(N-1)-1\), where N is the number of bits of significant data. For 12 bit devices (most FANUC modules), this is \(2^{11}-1\), or 2047.

For negative input voltages, the value will be in the range \(2^{N} - 1\) to \(2^{(N-1)}\) as the voltage varies from the smallest detectable negative voltage to the largest negative voltage handled by the device. For 12 bit devices, this is from 4095 to 2048.

An example of the KAREL logic for converting this input to a real value representing the voltage, where the device is a 12 bit device which handles a range from +10v to -10v would be as follows:

### Figure 14–1. KAREL Logic for Converting Input to a Real Value Representing the Voltage

```karel
V: REAL
AINP: INTEGER
AINP = AIN[1]
IF (AINP <= 2047) THEN
  V = AINP * 10.0 / 2047.0
ELSE
  V = (AINP - 4096) * 10.0 / 2047
ENDIF
```

In TPP, the following logic would be used:

```karel
IF (R[1] > 2047) JMP LBL[1]
JMP LBL[2]
LBL[1]:
LBL[2]
```

R[2] has the desired voltage.
Evaluating AOUT signals causes the system to return the currently output value from the specified output signal. Assigning a value to an AOUT signal causes the system to perform an output operation.

An AOUT can be turned on in a KAREL program with AOUT[n] = (an integer value). The result will be the output voltage on the AOUT signal line[n] of the integer value specified. For example, AOUT[1] = 1000 will output a +5 V signal on Analog Output line 1 (using an output module with 12 significant bits).

You assign analog signals using teach pendant I/O menus or the KAREL built-in routine SET_PORT_ASG.

### 14.1.4 Hand Signals

You have access to a special set of robot hand control signals used to control end-of-arm tooling through the KAREL language HAND statements, rather than through port arrays. HAND signals provide a KAREL program with access to two output signals that work in a coordinated manner to control the tool. The signals are designated as the open line and the close line. The system can support up to two HAND signals.

HAND[1] uses the same physical outputs as RDO[1] and RDO[2].

HAND[2] uses the same physical outputs as RDO[3] and RDO[4].

The following KAREL language statements are provided for controlling the signal, where “n” is the signal number.

- OPEN HAND n activates open line, and deactivates close line
- CLOSE HAND n deactivates open line, and activates close line
- RELAX HAND n deactivates both lines

### 14.2 SYSTEM-DEFINED SIGNALS

System-defined I/O signals are signals designated by the controller software for a specific purpose. Except for certain UOP signals, system-defined I/O cannot be reassigned.

You have access to system-defined I/O signals through the following port arrays:

- Robot digital input (RDI) and robot digital output (RDO)
- Operator panel input (OPIN) and operator panel output (OPOUT)
- Teach pendant input (TPIN) and teach pendant output (TPOUT)
14. INPUT/OUTPUT SYSTEM

14.2.1 Robot Digital Input and Output Signals (RDI/RDO)

Robot I/O is the input and output signals between the controller and the robot. These signals are sent to the EE (End Effector) connector located on the robot. The number of robot input and output signals (RDI and RDO) varies depending on the number of axes in the system. For more information on configuring Robot I/O, refer to the appropriate application-specific FANUC Robotics Setup and Operations Manual.

RDI[1] through RDI[8] are available for tool inputs. All or some of these signals can be used, depending on the robot model. Refer to the Maintenance Manual specific to your robot model, for more information.

RDO[1] through RDO[8] are available for tool control. All or some of these signals can be used, depending on the robot model. Refer to the Maintenance Manual specific to your robot model, for more information.


14.2.2 Operator Panel Input and Output Signals (OPIN/OPOUT)

Operator panel input and output signals are the input and output signals for the standard operator panel (SOP) and for the user operator panel (UOP).

Operator panel input signals are assigned as follows:

- The first 16 signals, OPIN[0] - OPIN[15], are assigned to the standard operator panel.
- The next 18 signals, OPIN[16] - OPIN[33], are assigned to the user operator panel (UOP). If you have a process I/O board, these 18 UOP signals are mapped to the first 18 input ports on the process I/O board.

Operator panel output signals are assigned as follows:

- The first 16 signals, OPOUT[0] - OPOUT[15], are assigned to the standard operator panel.
- The next 20 signals, OPOUT[16] - OPOUT[35], are assigned to the user operator panel (UOP). If you have a process I/O board, these 20 UOP signals are mapped to the first 20 output ports on the process I/O board.

Standard Operator Panel Input and Output Signals

Standard operator panel input and output signals are recognized by the KAREL system as OPIN[0] - OPIN[15] and OPOUT[0] - OPOUT[15] and by the screens on the teach pendant as SI[0] - SI[15] and SO[0] - SO[15]. Table 14–1 lists each standard operator panel input signal. Table 14–2 lists each standard operator panel output signal.
### Table 14–1. Standard Operator Panel Input Signals

<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>SI[n]</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPIN[0]</td>
<td>SI[0]</td>
<td>NOT USED</td>
<td>-</td>
</tr>
<tr>
<td>OPIN[1]</td>
<td>SI[1]</td>
<td>FAULT RESET</td>
<td>This signal is normally turned OFF, indicating that the FAULT RESET button is not being pressed.</td>
</tr>
<tr>
<td>OPIN[2]</td>
<td>SI[2]</td>
<td>REMOTE</td>
<td>This signal is normally turned OFF, indicating that the controller is not set to remote.</td>
</tr>
<tr>
<td>OPIN[3]</td>
<td>SI[3]</td>
<td>HOLD</td>
<td>This signal is normally turned ON, indicating that the HOLD button is not being pressed.</td>
</tr>
<tr>
<td>OPIN[6]</td>
<td>SI[6]</td>
<td>CYCLE START</td>
<td>This signal is normally turned OFF, indicating that the CYCLE START button is not being pressed.</td>
</tr>
<tr>
<td>SI[8], SI[9]</td>
<td>CE/CR Select b0, CE/CR Select b1</td>
<td></td>
<td>This signal is two bits and indicates the status of the mode select switch.</td>
</tr>
</tbody>
</table>

### Table 14–2. Standard Operator Panel Output Signals

<table>
<thead>
<tr>
<th>OPUT[n]</th>
<th>SO[n]</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPUT[0]</td>
<td>SO[0]</td>
<td>REMOTE LED</td>
<td>This signal indicates that the controller is set to remote.</td>
</tr>
<tr>
<td>OPUT[1]</td>
<td>SO[1]</td>
<td>CYCLE START</td>
<td>This signal indicates that the CYCLE START button has been pressed or that a program is running.</td>
</tr>
<tr>
<td>OPUT[2]</td>
<td>SO[2]</td>
<td>HOLD</td>
<td>This signal indicates that the HOLD button has been pressed or that a hold condition exists.</td>
</tr>
<tr>
<td>OPUT[3]</td>
<td>SO[3]</td>
<td>FAULT LED</td>
<td>This signal indicates that a fault has occurred.</td>
</tr>
<tr>
<td>OPUT[4]</td>
<td>SO[4]</td>
<td>BATTERY ALARM</td>
<td>This signal indicates that the CMOS battery voltage is low.</td>
</tr>
<tr>
<td>OPUT[7]</td>
<td>SO[7]</td>
<td>TEACH PENDANT ENABLED</td>
<td>This signal indicates that the teach pendant is enabled.</td>
</tr>
</tbody>
</table>
User Operator Panel Input and Output Signals

User operator panel input and output signals are recognized by the KAREL system as OPIN[16]-OPIN[33] and OPOUT[16]-OPOUT[35] and by the screens on the teach pendant as UI[1]-UI[18] and UO[1]-UO[20]. On the process I/O board, UOP input signals are mapped to the first 18 digital input signals and UOP output signals are mapped to the first 20 digital output signals. Table 14–3 lists and describes each user operator panel input signal. Table 14–4 lists each user operator panel output signal. Figure 14–2 and Figure 14–3 illustrate the timing of the UOP signals.
<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>UI[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| OPIN[16] | UI[1] | 1 | *IMSTP Always active | *IMSTP is the immediate stop software signal. *IMSTP is a normally OFF signal held ON that when set to OFF will  
  • Pause a program if running.  
  • Shut off power to the servos.  
  • Immediately stop the robot and applies robot brakes.  
  Error code SRVO-037 *IMSTP Input (Group:i) will be displayed when this signal is lost. This signal is always active. |
| OPIN[17] | UI[2] | 2 | *HOLD Always active | *HOLD is the external hold signal. *HOLD is a normally OFF signal held ON that when set to OFF will  
  • Pause program execution.  
  • Slow motion to a controlled stop and hold.  
  • Optional Brake on Hold shuts off servo power after the robot stops. |
## Table 14-3. User Operator Panel Input Signals (Cont’d)

<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>UI[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| OPIN[18]| UI[3] | 3                  | *SFSPD Always active | *SFSPD is the safety speed input signal. This signal is usually connected to the safety fence.*SFSPD is a normally OFF signal held ON that when set OFF will:  
  • Pause program execution.  
  • Reduce the speed override value to that defined in a system variable. This value cannot be increased while *SFSPD is OFF.  
  • Display error code message MF-0004 Fence Open.  
  • Not allow a REMOTE start condition. Start inputs from UOP or SOP are disabled when *SFSPD is set to OFF. |
Table 14–3. User Operator Panel Input Signals (Cont’d)

<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>UI[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>

CSTOPI is the cycle stop input. When CSTOPI becomes TRUE, the system variable $CSTOP is set to TRUE. In addition, if the system variable $SHELL_CONFIG.$shell_name is not TRUE or is uninitialized at power up, CSTOPI functions as follows, depending on the system variable $SHELL_CFG.$USE_ABORT. If the system variable $SHELL_CFG.$USE_ABORT is set to FALSE, the CSTOPI input

- Clears the queue of programs to be executed that were sent by RSR signals.

**Warning**

When $SHELL_CFG.USE_ABORT is FALSE, CSTOPI does not immediately stop automatic program execution.

- Automatic execution will be stopped after the current program has finished executing.

If the system variable $SHELL_CFG.$USE_ABORT is set to TRUE, the CSTOPI input

- Clears the queue of programs to be executed that were sent by RSR signals.
- Immediately aborts the currently executing program for programs that were sent to be executed by either RSR or PNS.
## Table 14–3. User Operator Panel Input Signals (Cont’d)

<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>UI[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FAULT_RESET is the external fault reset signal. When this signal is received</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Error status is cleared.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Servo power is turned ON.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• A paused program will not be resumed.</td>
</tr>
<tr>
<td>OPIN[21]</td>
<td>UI[6]</td>
<td>6</td>
<td>START</td>
<td>Active when the robot is in a remote condition (CMDENBL = ON)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>START is the remote start input. How this signal functions depends on the system variable $SHELL_CFG.$CONT_ONLY. If the system variable $SHELL_CFG.$CONT_ONLY is set to FALSE, the START input signal will</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Resume a paused program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If a program is not paused, the currently selected program starts from the position of the cursor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the system variable $SHELL_CFG.$CONT_ONLY is set to TRUE, the START input signal will</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Resume a paused program only. The PROD_START input must be used to start a program from the beginning.</td>
</tr>
<tr>
<td>OPIN[22]</td>
<td>UI[7]</td>
<td>7</td>
<td>HOME</td>
<td>Active when the robot is in a remote condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HOME is the home input. When this signal is received the robot moves to the defined home position.</td>
</tr>
<tr>
<td>OPIN[23]</td>
<td>UI[8]</td>
<td>8</td>
<td>ENBL</td>
<td>Always active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ENBL is the enable input. This signal must be ON to have motion control ability. When this signal is OFF, robot motion can be done. When ENBL is ON and the REMOTE switch on the operator panel is in the REMOTE position, the robot is in a remote operating condition.</td>
</tr>
</tbody>
</table>
Table 14–3. User Operator Panel Input Signals (Cont’d)

<table>
<thead>
<tr>
<th>OPIN[n]</th>
<th>UI[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPIN[32]</td>
<td>UI[17]</td>
<td>17</td>
<td>PNSTROBE Active when the robot is in a remote condition (CMDENBL = ON)</td>
<td>The PNSTROBE input is the program number select strobe input signal. See Figure 14–3.</td>
</tr>
<tr>
<td>OPIN[33]</td>
<td>UI[18]</td>
<td>18</td>
<td>PROD_START Active when the robot is in a remote condition (CMDENBL = ON)</td>
<td>The PROD_START input, when used with PNS, will initiate execution of the selected program from the PNS lines. When used without PNS, PROD_START executes the selected program from the current cursor position. See Figure 14–3.</td>
</tr>
</tbody>
</table>

RSR1-8 are the robot service request input signals. When one of these signals is received, the corresponding RSR program is executing, or a program is running currently, stored in a queue for later execution. RSR signals are used for production operation and can be received while an ACK output is being pulsed. See Figure 14–2.

PNS 1-8 are program number select input signals. PNS selects programs for execution, but does not execute programs. Programs that are selected by PNS are executed using the START input or the PROD_START input depending on the value of the system variable $SHELL_CFG.$CONT_ONLY. The PNS number is output by pulsing the SNO signal (selected number output) and the SNACK signal (selected number acknowledge). See Figure 14–3.
<table>
<thead>
<tr>
<th>OPOUT[n]</th>
<th>UO[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPOUT[16]</td>
<td>UO[1]</td>
<td>1</td>
<td>CMDENBL</td>
<td>CMDENBL is the command enable output. This output indicates that the robot is in a remote condition. This signal goes ON when the REMOTE switch is turned to ON or when the ENBL input is received. This output only stays on when the robot is not in a fault condition. See Figure 14–2 and Figure 14–3.</td>
</tr>
<tr>
<td>OPOUT[17]</td>
<td>UO[2]</td>
<td>2</td>
<td>SYSRDY</td>
<td>SYSRDY is the system ready output. This output indicates that servos are turned ON.</td>
</tr>
<tr>
<td>OPOUT[18]</td>
<td>UO[3]</td>
<td>3</td>
<td>PROGRUN</td>
<td>PROGRUN is the program run output. This output turns on when a program is running. See Figure 14–3.</td>
</tr>
<tr>
<td>OPOUT[19]</td>
<td>UO[4]</td>
<td>4</td>
<td>PAUSED</td>
<td>PAUSED is the paused program output. This output turns on when a program is paused.</td>
</tr>
<tr>
<td>OPOUT[20]</td>
<td>UO[5]</td>
<td>5</td>
<td>HELD</td>
<td>HELD is the hold output. This output turns on when the SOP HOLD button has been pressed, or the UOP *HOLD input is OFF.</td>
</tr>
<tr>
<td>OPOUT[21]</td>
<td>UO[6]</td>
<td>6</td>
<td>FAULT</td>
<td>FAULT is the error output. This output turns on when a program is in an error condition.</td>
</tr>
<tr>
<td>OPOUT[23]</td>
<td>UO[8]</td>
<td>8</td>
<td>TPENBL</td>
<td>TPENBL is the teach pendant enable output. This output turns on when the teach pendant is on.</td>
</tr>
<tr>
<td>OPOUT[24]</td>
<td>UO[9]</td>
<td>9</td>
<td>BATALM</td>
<td>BATALM is the battery alarm output. This output turns on when the CMOS RAM battery voltage goes below 3.6 volts.</td>
</tr>
<tr>
<td>OPOUT[25]</td>
<td>UO[10]</td>
<td>10</td>
<td>BUSY</td>
<td>BUSY is the processor busy output. This signal turns on when the robot is executing a program or when the processor is busy.</td>
</tr>
</tbody>
</table>
**Table 14-4. User Operator Panel Output Signals (Cont’d)**

<table>
<thead>
<tr>
<th>OPOUT[n]</th>
<th>UO[n]</th>
<th>Process I/O Number</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPOUT[26]</td>
<td>UO[11]- UO[18]</td>
<td>11- 18</td>
<td>ACK1/SNO1, ACK2/SNO2, ACK3/SNO3, ACK4/SNO4, ACK5/SNO5, ACK6/SNO6, ACK7/SNO7, ACK8/SNO8</td>
<td>ACK 1-8 are the acknowledge signals output 1 through 4. These signals turn on when the corresponding RSR signal is received. See Figure 14–2. SNACK1-8 are the signal number outputs. These signals carry the 8-bit representation of the corresponding PNS selected program number. If the program cannot be represented by an 8-bit number, the signal is set to all zeroes or off. See Figure 14–3.</td>
</tr>
<tr>
<td>OPOUT[34]</td>
<td>UO[19]</td>
<td>19</td>
<td>SNACK</td>
<td>SNACK is the signal number acknowledge output. This output is pulsed if the program is selected by PNS input. See Figure 14–3.</td>
</tr>
<tr>
<td>OPOUT[35]</td>
<td>UO[20]</td>
<td>20</td>
<td>RESERVED</td>
<td>-</td>
</tr>
<tr>
<td>OPOUT[36]</td>
<td>UO[21]</td>
<td>21</td>
<td>UNCAL (option)</td>
<td>UNCAL is the uncalibrated output. This output turns on when the robot is not calibrated. The robot is uncalibrated when the controller loses the feedback signals from one or all of the motors. Set $OPWORK.$OPT_OUT = 1 to use this signal.</td>
</tr>
<tr>
<td>OPOUT[37]</td>
<td>UO[22]</td>
<td>22</td>
<td>UPENBL (option)</td>
<td>UPENBL is the user panel enable output. This output indicates that the robot is in a remote condition. This signal goes on when the remote switch is turned to ON or when the ENBL input is received. <strong>This output will stay on even if the robot is in a fault condition.</strong> Set $OPWORK.$OPT_OUT = 1 to use this signal.</td>
</tr>
<tr>
<td>OPOUT[38]</td>
<td>UO[23]</td>
<td>23</td>
<td>LOCKED (option)</td>
<td>-</td>
</tr>
<tr>
<td>OPOUT[39]</td>
<td>UO[24]</td>
<td>24</td>
<td>CSTOPO (option)</td>
<td>CSTOPO is the cycle stop output. This output turns on when the CSTOPI input has been received. Set $OPWORK.$OPT_OUT = 1 to use this signal.</td>
</tr>
</tbody>
</table>
Figure 14–2. RSR Timing Diagram

- CMDENBL OUTPUT (Remote Condition)
- RSR1 INPUT
  - $SCR.$cond_time milliseconds maximum delay
- ACK1 OUTPUT
  - Pulse width is specified in RSR Setup screen
- RSR2 INPUT
- ACK2 OUTPUT
- RSR3 INPUT
- ACK3 OUTPUT
- RSR4 INPUT
- ACK4 OUTPUT

Another RSR signal can be received while an ACK is being pulsed.
14.2.3 Teach Pendant Input and Output Signals (TPIN/TPOUT)

The teach pendant input signals (TPIN) provide read access to input signals generated by the teach pendant keys. Teach pendant inputs can be accessed through the TPIN port arrays. A KAREL program treats teach pendant input data as a BOOLEAN data type. The value is either ON (active--the key is pressed) or OFF (inactive--the key is not pressed). TPIN signals are accessed in KAREL programs by the name TPIN[n], where “n” is the signal number, which is assigned internally. Refer to Table 14–5 for teach pendant input signal assignments.

<table>
<thead>
<tr>
<th>TPIN[n]</th>
<th>Teach Pendant Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[250]</td>
<td>EMERGENCY STOP</td>
</tr>
<tr>
<td>TPIN[249]</td>
<td>ON/OFF switch</td>
</tr>
<tr>
<td>TPIN[247]</td>
<td>Right DEADMAN switch</td>
</tr>
<tr>
<td>TPIN[248]</td>
<td>Left DEADMAN switch</td>
</tr>
<tr>
<td>Arrow Keys</td>
<td></td>
</tr>
</tbody>
</table>
Table 14–5. Teach Pendant Input Signal Assignments (Cont’d)

<table>
<thead>
<tr>
<th>TPIN[n]</th>
<th>Teach Pendant Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[212]</td>
<td>Up arrow</td>
</tr>
<tr>
<td>TPIN[213]</td>
<td>Down arrow</td>
</tr>
<tr>
<td>TPIN[208]</td>
<td>Right arrow</td>
</tr>
<tr>
<td>TPIN[209]</td>
<td>Left arrow</td>
</tr>
<tr>
<td>TPIN[0]</td>
<td>Left and/or right shift</td>
</tr>
<tr>
<td>TPIN[204]</td>
<td>Shifted Up arrow</td>
</tr>
<tr>
<td>TPIN[205]</td>
<td>Shifted Down arrow</td>
</tr>
<tr>
<td>TPIN[206]</td>
<td>Shifted Right arrow</td>
</tr>
<tr>
<td>TPIN[207]</td>
<td>Shifted Left arrow</td>
</tr>
<tr>
<td>Keypad Keys (shifted or unshifted)</td>
<td></td>
</tr>
<tr>
<td>TPIN[13]</td>
<td>ENTER</td>
</tr>
<tr>
<td>TPIN[8]</td>
<td>BACK SPACE</td>
</tr>
<tr>
<td>TPIN[48]</td>
<td>0</td>
</tr>
<tr>
<td>TPIN[49]</td>
<td>1</td>
</tr>
<tr>
<td>TPIN[50]</td>
<td>2</td>
</tr>
<tr>
<td>TPIN[51]</td>
<td>3</td>
</tr>
<tr>
<td>TPIN[52]</td>
<td>4</td>
</tr>
<tr>
<td>TPIN[53]</td>
<td>5</td>
</tr>
<tr>
<td>TPIN[54]</td>
<td>6</td>
</tr>
<tr>
<td>TPIN[55]</td>
<td>7</td>
</tr>
<tr>
<td>TPIN[56]</td>
<td>8</td>
</tr>
<tr>
<td>TPIN[57]</td>
<td>9</td>
</tr>
<tr>
<td>Function Keys</td>
<td></td>
</tr>
<tr>
<td>TPIN[128]</td>
<td>PREV</td>
</tr>
<tr>
<td>TPIN[129]</td>
<td>F1</td>
</tr>
<tr>
<td>TPIN[131]</td>
<td>F2</td>
</tr>
<tr>
<td>TPIN[132]</td>
<td>F3</td>
</tr>
<tr>
<td>TPIN[133]</td>
<td>F4</td>
</tr>
<tr>
<td>TPIN[134]</td>
<td>F5</td>
</tr>
<tr>
<td>TPIN[135]</td>
<td>NEXT</td>
</tr>
<tr>
<td>TPIN[136]</td>
<td>Shifted PREV</td>
</tr>
<tr>
<td>TPIN[137]</td>
<td>Shifted F1</td>
</tr>
<tr>
<td>TPIN[138]</td>
<td>Shifted F2</td>
</tr>
<tr>
<td>TPIN[139]</td>
<td>Shifted F3</td>
</tr>
<tr>
<td>TPIN[140]</td>
<td>Shifted F4</td>
</tr>
<tr>
<td>TPIN[141]</td>
<td>Shifted F5</td>
</tr>
<tr>
<td>TPIN[142]</td>
<td>Shifted NEXT</td>
</tr>
<tr>
<td>Menu Keys</td>
<td></td>
</tr>
</tbody>
</table>
Table 14-5. Teach Pendant Input Signal Assignments (Cont’d)

<table>
<thead>
<tr>
<th>TPIN[n]</th>
<th>Teach Pendant Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[143]</td>
<td>SELECT</td>
</tr>
<tr>
<td>TPIN[144]</td>
<td>MENU</td>
</tr>
<tr>
<td>TPIN[145]</td>
<td>EDIT</td>
</tr>
<tr>
<td>TPIN[146]</td>
<td>DATA</td>
</tr>
<tr>
<td>TPIN[147]</td>
<td>FCTN</td>
</tr>
<tr>
<td>TPIN[148]</td>
<td>ITEM</td>
</tr>
<tr>
<td>TPIN[149]</td>
<td>+%</td>
</tr>
<tr>
<td>TPIN[150]</td>
<td>-%</td>
</tr>
<tr>
<td>TPIN[151]</td>
<td>HOLD</td>
</tr>
<tr>
<td>TPIN[152]</td>
<td>STEP</td>
</tr>
<tr>
<td>TPIN[153]</td>
<td>RESET</td>
</tr>
<tr>
<td>TPIN[240]</td>
<td>DISP</td>
</tr>
<tr>
<td>TPIN[203]</td>
<td>HELP</td>
</tr>
<tr>
<td>TPIN[154]</td>
<td>Shifted ITEM</td>
</tr>
<tr>
<td>TPIN[155]</td>
<td>Shifted +%</td>
</tr>
<tr>
<td>TPIN[156]</td>
<td>Shifted -%</td>
</tr>
<tr>
<td>TPIN[157]</td>
<td>Shifted STEP</td>
</tr>
<tr>
<td>TPIN[158]</td>
<td>Shifted HOLD</td>
</tr>
<tr>
<td>TPIN[159]</td>
<td>Shifted RESET</td>
</tr>
<tr>
<td>TPIN[227]</td>
<td>Shifted DISP</td>
</tr>
<tr>
<td>TPIN[239]</td>
<td>Shifted HELP</td>
</tr>
</tbody>
</table>

User Function Keys

<table>
<thead>
<tr>
<th>TPIN[173]</th>
<th>USER KEY 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[174]</td>
<td>USER KEY 2</td>
</tr>
<tr>
<td>TPIN[175]</td>
<td>USER KEY 3</td>
</tr>
<tr>
<td>TPIN[176]</td>
<td>USER KEY 4</td>
</tr>
<tr>
<td>TPIN[177]</td>
<td>USER KEY 5</td>
</tr>
<tr>
<td>TPIN[178]</td>
<td>USER KEY 6</td>
</tr>
<tr>
<td>TPIN[210]</td>
<td>USER KEY 7</td>
</tr>
<tr>
<td>TPIN[179]</td>
<td>Shifted USER KEY 1</td>
</tr>
<tr>
<td>TPIN[180]</td>
<td>Shifted USER KEY 2</td>
</tr>
<tr>
<td>TPIN[181]</td>
<td>Shifted USER KEY 3</td>
</tr>
<tr>
<td>TPIN[182]</td>
<td>Shifted USER KEY 4</td>
</tr>
<tr>
<td>TPIN[183]</td>
<td>Shifted USER KEY 5</td>
</tr>
<tr>
<td>TPIN[184]</td>
<td>Shifted USER KEY 6</td>
</tr>
<tr>
<td>TPIN[211]</td>
<td>Shifted USER KEY 7</td>
</tr>
</tbody>
</table>

Motion Keys
### Table 14-5. Teach Pendant Input Signal Assignments (Cont’d)

<table>
<thead>
<tr>
<th>TPIN[n]</th>
<th>Teach Pendant Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[185]</td>
<td>FWD</td>
</tr>
<tr>
<td>TPIN[186]</td>
<td>BWD</td>
</tr>
<tr>
<td>TPIN[187]</td>
<td>COORD</td>
</tr>
<tr>
<td>TPIN[28]</td>
<td>GROUP</td>
</tr>
<tr>
<td>TPIN[188]</td>
<td>+X</td>
</tr>
<tr>
<td>TPIN[189]</td>
<td>+Y</td>
</tr>
<tr>
<td>TPIN[190]</td>
<td>+Z</td>
</tr>
<tr>
<td>TPIN[191]</td>
<td>+X rotation</td>
</tr>
<tr>
<td>TPIN[192]</td>
<td>+Y rotation</td>
</tr>
<tr>
<td>TPIN[193]</td>
<td>+Z rotation</td>
</tr>
<tr>
<td>TPIN[12290]</td>
<td>+J7</td>
</tr>
<tr>
<td>TPIN[12294]</td>
<td>+J8</td>
</tr>
<tr>
<td>TPIN[194]</td>
<td>-X</td>
</tr>
<tr>
<td>TPIN[195]</td>
<td>-Y</td>
</tr>
<tr>
<td>TPIN[196]</td>
<td>-Z</td>
</tr>
<tr>
<td>TPIN[197]</td>
<td>-X rotation</td>
</tr>
<tr>
<td>TPIN[198]</td>
<td>-Y rotation</td>
</tr>
<tr>
<td>TPIN[199]</td>
<td>-Z rotation</td>
</tr>
<tr>
<td>TPIN[12292]</td>
<td>-J7</td>
</tr>
<tr>
<td>TPIN[12296]</td>
<td>-J8</td>
</tr>
<tr>
<td>TPIN[226]</td>
<td>Shifted FWD</td>
</tr>
<tr>
<td>TPIN[207]</td>
<td>Shifted BWD</td>
</tr>
<tr>
<td>TPIN[202]</td>
<td>Shifted COORD</td>
</tr>
</tbody>
</table>

#### Motion Keys Cont’d

| TPIN[214]  | Shifted +X             |
| TPIN[215]  | Shifted +Y             |
| TPIN[216]  | Shifted +Z             |
| TPIN[217]  | Shifted +X rotation    |
| TPIN[218]  | Shifted +Y rotation    |
| TPIN[219]  | Shifted +Z rotation    |
| TPIN[12291]| Shifted +J7           |
| TPIN[12295]| Shifted +J8           |
| TPIN[220]  | Shifted -X             |
| TPIN[221]  | Shifted -Y             |
| TPIN[222]  | Shifted -Z             |
| TPIN[223]  | Shifted -X rotation    |
| TPIN[224]  | Shifted -Y rotation    |
| TPIN[225]  | Shifted -Z rotation    |
| TPIN[12293]| Shifted -J7           |
| TPIN[12297]| Shifted -J8           |

#### i Keys
### Teach Pendant Input Signal Assignments (Cont’d)

<table>
<thead>
<tr>
<th>TPIN[n]</th>
<th>Teach Pendant Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIN[12288]</td>
<td>key</td>
</tr>
<tr>
<td>TPIN[12298]</td>
<td>MENU, Top Menu</td>
</tr>
<tr>
<td>TPIN[12299]</td>
<td>SELECT, 4D Select Node Map</td>
</tr>
<tr>
<td>TPIN[12300]</td>
<td>DATA, 4D Position Register</td>
</tr>
<tr>
<td>TPIN[12301]</td>
<td>FCTN, Related Views</td>
</tr>
<tr>
<td>TPIN[12302]</td>
<td>STEP</td>
</tr>
<tr>
<td>TPIN[12303]</td>
<td>COORD</td>
</tr>
<tr>
<td>TPIN[12304]</td>
<td>GROUP</td>
</tr>
<tr>
<td>TPIN[12305]</td>
<td>+%</td>
</tr>
<tr>
<td>TPIN[12306]</td>
<td>-%</td>
</tr>
<tr>
<td>TPIN[12307]</td>
<td>Up arrow</td>
</tr>
<tr>
<td>TPIN[12308]</td>
<td>Down arrow</td>
</tr>
<tr>
<td>TPIN[12309]</td>
<td>Right arrow</td>
</tr>
<tr>
<td>TPIN[12310]</td>
<td>Left arrow</td>
</tr>
<tr>
<td>TPIN[12311]</td>
<td>DISP, Focus window</td>
</tr>
<tr>
<td>TPIN[12312]</td>
<td>HELP</td>
</tr>
<tr>
<td>TPIN[12313]</td>
<td>USER KEY 7, 4D Display</td>
</tr>
<tr>
<td>TPIN[12338]</td>
<td>USER KEY 6</td>
</tr>
<tr>
<td>TPIN[12339]</td>
<td>USER KEY 5</td>
</tr>
<tr>
<td>TPIN[12340]</td>
<td>ENTER, space</td>
</tr>
<tr>
<td>TPIN[12341]</td>
<td>Keys Cont’d</td>
</tr>
<tr>
<td>TPIN[12314]</td>
<td>+X</td>
</tr>
<tr>
<td>TPIN[12315]</td>
<td>+Y</td>
</tr>
<tr>
<td>TPIN[12316]</td>
<td>+Z</td>
</tr>
<tr>
<td>TPIN[12317]</td>
<td>+X rotation</td>
</tr>
<tr>
<td>TPIN[12318]</td>
<td>+Y rotation</td>
</tr>
<tr>
<td>TPIN[12319]</td>
<td>+Z rotation</td>
</tr>
<tr>
<td>TPIN[12320]</td>
<td>+J7 rotation</td>
</tr>
<tr>
<td>TPIN[12321]</td>
<td>+J8 rotation</td>
</tr>
<tr>
<td>TPIN[12322]</td>
<td>-X</td>
</tr>
<tr>
<td>TPIN[12323]</td>
<td>-Y</td>
</tr>
<tr>
<td>TPIN[12324]</td>
<td>-Z</td>
</tr>
<tr>
<td>TPIN[12325]</td>
<td>-X rotation</td>
</tr>
<tr>
<td>TPIN[12326]</td>
<td>-Y rotation</td>
</tr>
<tr>
<td>TPIN[12327]</td>
<td>-Z rotation</td>
</tr>
<tr>
<td>TPIN[12328]</td>
<td>-J7 rotation</td>
</tr>
<tr>
<td>TPIN[12329]</td>
<td>-J8 rotation</td>
</tr>
</tbody>
</table>

Three teach pendant output signals are available for use:

- TPOUT[6] - controls teach pendant USER LED #1
• TPOUT[7] - controls teach pendant USER LED #2
• TPOUT[8] - controls teach pendant USER LED #3

14.3 Serial Input/Output

14.3.1 Serial Input/Output

The serial I/O system allows you to communicate with peripheral serial devices connected to the KAREL system. For example, you could use serial I/O to write messages from one of the communications ports to a remote terminal across a cable that connects to the controller.

To use serial I/O you must provide a serial device and the appropriate cable. Refer to the Maintenance Manual, specific to your robot model, for electrical specifications.

The communications ports that you use to read and write serial data are defined in the system software. Each software port is associated with physical connectors on the controller to which you attach the communications cable.

Figure 14–4 shows the location of the ports on the controller.
Ports

Setting up a port means initializing controller serial ports to use specific devices, such as the CRT/KB. Initializing ports involves setting up specific information for a port based on the kind of device that will connect to the port. This is done on the teach pendant PORT INIT screen.

The controller supports up to four serial ports. Several different kinds of devices can be connected to these ports.
Up to four ports are available, P1-P4. Table 14–6 lists the ports. You can set up ports P2 through P4 if you have them, but you cannot set up the teach pendant port, P1.

Table 14–6. Ports P1 - P4

<table>
<thead>
<tr>
<th>Port</th>
<th>Item Name on Screen</th>
<th>Kind of Port</th>
<th>Use</th>
<th>Default Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>JRS16 RS-232–C</td>
<td>RS-232-C</td>
<td>Any device</td>
<td>Maintenance Console</td>
</tr>
<tr>
<td>P3</td>
<td>JD17 RS-232–C on Main CPU card</td>
<td>RS-232-C</td>
<td>KCL</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>JD17 on Main CPU card. This port is displayed on the teach pendant if $RS232_NPORT=4.</td>
<td>RS-422</td>
<td>No use</td>
<td></td>
</tr>
</tbody>
</table>

**Devices**

You can modify the default communications settings for each port except port 1, which is dedicated to the teach pendant (TP). Table 14–7 lists the default settings for each kind of device you can connect to a port.

Table 14–7. Default Communications Settings for Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Speed (baud)</th>
<th>Parity Bit</th>
<th>Stop Bit</th>
<th>Timeout Value (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor*</td>
<td>4800</td>
<td>Odd</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>Host Comm.*</td>
<td>4800</td>
<td>Odd</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>KCL/CRT</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>Maintenance Console</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>Factory Terminal</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>TP Demo Device</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>No Use</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>Current Position (for use with the Current Position option)</td>
<td>9600</td>
<td>None</td>
<td>1 bit</td>
<td>0</td>
</tr>
<tr>
<td>PMC Programmer</td>
<td>9600</td>
<td>None</td>
<td>2 bit</td>
<td>0</td>
</tr>
<tr>
<td>Modem/PPP</td>
<td>Refer to the <em>FANUC Robotics Internet Options Setup and Operations Manual</em> for information on the supported modems.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI Device</td>
<td>19200</td>
<td>Odd</td>
<td>1 bit</td>
<td>0</td>
</tr>
</tbody>
</table>

*You can adjust these settings; however, if you do, they might not function as intended because they are connected to an external device.*
After the hardware has been connected and the appropriate port is configured and the external port is connected, you can use KAREL language OPEN FILE, READ, and WRITE statements to communicate with the peripheral device.

Higher levels of communication protocol are supported as an optional feature.

See Also: Appendix A for more information on the statements and built-ins available in KAREL.

Refer to the application-specific FANUC Robotics Setup and Operations Manual for more information about setting up ports.
Chapter 15

MULTI-TASKING

Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>MULTI-TASKING</td>
<td>15-1</td>
</tr>
<tr>
<td>15.1</td>
<td>MULTI-TASKING TERMINOLOGY</td>
<td>15-2</td>
</tr>
<tr>
<td>15.2</td>
<td>INTERPRETER ASSIGNMENT</td>
<td>15-3</td>
</tr>
<tr>
<td>15.3</td>
<td>MOTION CONTROL</td>
<td>15-3</td>
</tr>
<tr>
<td>15.4</td>
<td>TASK SCHEDULING</td>
<td>15-4</td>
</tr>
<tr>
<td>15.4.1</td>
<td>Priority Scheduling</td>
<td>15-5</td>
</tr>
<tr>
<td>15.4.2</td>
<td>Time Slicing</td>
<td>15-6</td>
</tr>
<tr>
<td>15.5</td>
<td>STARTING TASKS</td>
<td>15-6</td>
</tr>
<tr>
<td>15.5.1</td>
<td>Running Programs from the User Operator Panel (UOP) PNS Signal</td>
<td>15-7</td>
</tr>
<tr>
<td>15.5.2</td>
<td>Child Tasks</td>
<td>15-7</td>
</tr>
<tr>
<td>15.6</td>
<td>TASK CONTROL AND MONITORING</td>
<td>15-7</td>
</tr>
<tr>
<td>15.6.1</td>
<td>From TPP Programs</td>
<td>15-8</td>
</tr>
<tr>
<td>15.6.2</td>
<td>From KAREL Programs</td>
<td>15-8</td>
</tr>
<tr>
<td>15.6.3</td>
<td>From KCL</td>
<td>15-8</td>
</tr>
<tr>
<td>15.7</td>
<td>USING SEMAPHORES AND TASK SYNCHRONIZATION</td>
<td>15-9</td>
</tr>
<tr>
<td>15.8</td>
<td>USING QUEUES FOR TASK COMMUNICATIONS</td>
<td>15-14</td>
</tr>
</tbody>
</table>
Multi-tasking allows more than one program to run on the controller on a time-sharing basis, so that multiple programs appear to run simultaneously.

Multi-tasking is especially useful when you are executing several sequences of operations which can generally operate independently of one another, even though there is some interaction between them. For example:

- A process of monitoring input signals and setting output signals.
- A process of generating and transmitting log information to a cell controller and receiving commands or other input data from a cell controller.

It is important to be aware that although multiple tasks seem to operate at the same time, they are sharing use of the same processor, so that at any instant only one task is really being executed. With the exception of interruptible statements, once execution of a statement is started, it must complete before statements from another task can be executed. The following statements are interruptible:

- READ
- DELAY
- WAIT
- WAIT FOR

Refer to Section 15.4, “Task Scheduling” for information on how the system decides which task to execute first.

## 15.1 MULTI-TASKING TERMINOLOGY

The following terminology and expressions are used in this chapter.

- **Task or User task**
  
  A task, or user task, is a user program that is running or paused. A task is executed by an "interpreter." A task is created when the program is started and eliminated when the interpreter it is assigned to, becomes assigned to another task.

- **Interpreter**
  
  An interpreter is a system component that executes user programs. At a cold or controlled start, ($\text{MAXNUMTASKS} + 2$) interpreters are created. These interpreters are capable of concurrently executing tasks.

- **Task name**
  
  Task name is the program name specified when the task is created. When you create a task, specify the name of the program to be executed as the task name.
The task name does not change once the task is created. Therefore, when an external routine is executing, the current executing program name is not the same as the task name. When you send any requests to the task, use the task name, not the current program name.

Motion control

Motion control is defined by a bit mask that specifies the motion groups of which a task has control. Only one task at a time can control a motion group. However, different tasks can control different motion groups simultaneously. Refer to Section 15.3, “Motion Control,” for more information.

15.2 INTERPRETER ASSIGNMENT

When a task is started, it is assigned to an interpreter. The interpreter it is assigned to (1, 2, 3, ...) determines its task number. The task number is used in PAUSE PROGRAM, ABORT PROGRAM and CONTINUE PROGRAM condition handler actions. The task number for a task can be determined using the GET_TSK_INFO built-in.

The following are rules for assigning a task to an interpreter:

- If the task is already assigned to an interpreter, it uses the same interpreter.
- A task is assigned to the first available interpreter that currently has no tasks assigned to it.
- If all interpreters are assigned to tasks, a new task will be assigned to the first interpreter that has an aborted task.
- If none of the above can be done, the task cannot be started.

15.3 MOTION CONTROL

An important restriction in multi-tasking is in the control of the various motion groups. Only one task can have control, or use of, a group of axes. A task requires control of the group(s) in the following situations:

- When the task starts, if the controller directive %NOLOCKGROUP is not used. If the %LOCKGROUP directive is not used, the task requires control of all groups by default. If %LOCKGROUP is used, control of the specified groups is required.

For teach pendant programs, motion control is required when the program starts, unless the DETAIL page from the SELECT screen is used to set the Group Mask to [*,*,*,*,*].

- When a task executes the LOCK_GROUP built-in, it requires the groups specified by the group mask.
• When a task calls a ROUTINE or teach pendant program, it requires control of those group(s). The group(s) required by a ROUTINE or TPP+ program are those specified, or implied, by controller directives or in the teach pendant DETAIL setup.

A task will be given control of the required group(s), assuming:

• No other task has control of the group.
• The teach pendant is not enabled, with the exception that motion control can be given to a program when it is started using shift-FWD at the teach pendant or if it has the %TPMOTION directive.
• There are no emergency stops active.
• The servos are ready.
• The UOP signal IMSTP is not asserted.

A task will be paused if it is not able to get control of the required group(s).

After a task gets control of a group, it keeps it until one of the following:

• The task ends (aborts).
• The task executes the UNLOCK_GROUP built-in.
• The task passes control of the group(s) in a RUN_TASK built-in.
• The ROUTINE or teach pendant program returns, and groups were required by a ROUTINE or teach pendant program, but not by the calling program.

### 15.4 TASK SCHEDULING

A task that is currently running (not aborted or paused) will execute statements until one of the following:

• A hold condition occurs.
• A higher priority program becomes ready to run.
• The task time slice expires.
• The program aborts or pauses.

The following are examples of hold conditions:

• Waiting for a read operation to complete.
• Waiting for a motion to complete.
• Waiting for a WAIT, WAIT FOR, or DELAY statement to complete.
A task is ready to run when it is in running state and has no hold conditions. Only one task is actually executed at a time. There are two rules for determining which task will be executed when more than one task is ready to run:

- Priority - If two or more tasks of different priority are ready to run, the task with higher priority is executed first. Refer to Section 15.4.1, “Priority Scheduling,” for more information.
- Time-slicing - If two tasks of the same priority are ready to run, execution of the tasks is time-sliced. Refer to Section 15.4.2, “Time Slicing,” for more information.

### 15.4.1 Priority Scheduling

If two or more tasks with different priorities are ready to run, the task with the highest priority will run first. The priority of a task is determined by its priority number. Priority numbers must be in the range from -8 to 143. The lower the priority number, the higher the task priority.

For example: if TASK_A has a priority number of 50 and TASK_B has a priority number of 60, and both are ready to run, TASK_A will execute first, as long as it is ready to run.

A task priority can be set in one of the following ways:

- By default, each user task is assigned a priority of 50.
- KAREL programs may contain the %PRIORITY translator directive.
- The SET_TSK_ATTR built-in can be used to set the current priority of any task.

In addition to affecting other user tasks, task priority also affects the priority of the interpreter executing it, relative to that of other system functions. If the user task has a higher priority (lower priority number) than the system function, as long as the user task is ready to run, the system function will be not be executed. The range of user task priorities is restricted at the high priority end. This is done so that the user program cannot interfere with motion interpolation. Motion interpolation refers to the updates required to cause a motion to complete.

The following table indicates the priority of some other system functions.

<table>
<thead>
<tr>
<th>Priority</th>
<th>System Function</th>
<th>Effect of Delaying Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>Maximum priority</td>
<td>New motions delayed.</td>
</tr>
<tr>
<td>-1</td>
<td>Motion Planner</td>
<td>New motions delayed.</td>
</tr>
<tr>
<td>4</td>
<td>TP Jog</td>
<td>Jogging from the Teach Pendant delayed.</td>
</tr>
<tr>
<td>54</td>
<td>Error Logger</td>
<td>Update of system error log delayed.</td>
</tr>
<tr>
<td>73</td>
<td>KCL</td>
<td>Execution of KCL commands delayed.</td>
</tr>
</tbody>
</table>
15. MULTI-TASKING

Table 15-1. System Function Priority Table (Cont’d)

<table>
<thead>
<tr>
<th>Priority</th>
<th>System Function</th>
<th>Effect of Delaying Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>CRT manager</td>
<td>Processing of CRT soft-keys delayed.</td>
</tr>
<tr>
<td>88</td>
<td>TP manager</td>
<td>General teach pendant activity delayed.</td>
</tr>
<tr>
<td>143</td>
<td>Lowest priority</td>
<td>Does not delay any of the above.</td>
</tr>
</tbody>
</table>

15.4.2 Time Slicing

If two or more tasks of the same priority are ready to run, they will share the system resources by time-slicing, or alternating use of the system.

A time-slice permits other tasks of the same priority to execute, but not lower priority tasks.

The default time-slice for a task is 256 msec. Other values can be set using the %TIMESLICE directive or the SET_TSK_ATTR built-in.

15.5 STARTING TASKS

There are a number ways to start a task.

- KCL RUN command. Refer to Appendix C, “KCL Command Alphabetic Descriptions.”
- Operator Panel start key. Refer to the appropriate application-specific FANUC Robotics Setup and Operations Manual.
- User operator panel start signal. Refer to the appropriate application-specific FANUC Robotics Setup and Operations Manual.
- User operator panel PNS signal. Refer to Section 15.5.1, “Running Programs from the User Operator Panel (UOP) PNS Signal,” for more information.
- Teach pendant shift-FWD key. Refer to the appropriate application-specific FANUC Robotics Setup and Operations Manual, Chapter on “Testing a Program and Running Production,” for more information.
- Teach pendant program executes a RUN instruction. Refer to Section 15.5.2, “Child Tasks,” for more information.
- KAREL program executes the RUN_TASK built-in. Refer to Section 15.5.2, “Child Tasks,” for more information.

In each case, the task will not start running if it requires motion control that is not available.
15.5.1 Running Programs from the User Operator Panel (UOP) PNS Signal

A program is executed:

- If the binary value of the UOP PNS signals is non-zero and the UOP PROGSTART signal is asserted
- If there is currently a program with the name “PNSnnnn,” where nnnn is the decimal value of the PNS signals plus the current value of $SHELLCFG.$jobbase.

A program is not executed:

- If the binary value of the PNS signals is zero.

Multiple programs can be started in this way, as long as there is no motion group overlap.

If the task name determined from the PNS is in a paused state, the PROGSTART signal is interpreted as a CONTINUE signal. If $SHELLCFG.$contonly is TRUE, this is the only function of the PNS/PROGSTART signals.

If $SHELLCFG.$useabort is TRUE, the PNS signals can be used to abort a running task. The name of the task to be aborted is the same as that used with the PROGSTART signal. In this case, abort is triggered by the UOP CSTOPI signal.

15.5.2 Child Tasks

A running task can create new tasks. This new task is called a child task. The task requesting creation of the child task is called the parent task. In teach pendant programs, a new task is created by executing a RUN instruction. In KAREL programs a new task can be created using the RUN_TASK built-in.

The parent and child task may not require the same motion group. In the case of RUN_TASK, however, it is possible to release control of motion groups for use by the child task.

Once a child task is created, it runs independently of its parent task, with the following exception:

- If a parent task is continued and its child task is paused, the child task is also continued.
- If a parent task is put in STEP mode, the child task is also put in STEP mode.

If you want the child task to be completely independent of the parent, a KAREL program can initiate another task using the KCL or KCL_NOWAIT built-ins to issue a KCL>RUN command.

15.6 TASK CONTROL AND MONITORING

There are three environments from which you can control and monitor tasks:
15.6.1 From TPP Programs

The TPP instruction RESUME_PROG can be used to continue a PAUSEd task.

15.6.2 From KAREL Programs

There are a number of built-ins used to control and monitor other tasks. See the description of these built-ins in Appendix A.

- RUN_TASK executes a task.
- CONT_TASK resumes execution of a PAUSEd task.
- PAUSE_TASK pauses a task.
- ABORT_TASK aborts a task.
- CONTINUE condition handler action causes execution of a task.
- ABORT condition handler action causes a task to be aborted.
- PAUSE condition handler action causes a task to be paused.
- GET_TSK_INFO determines whether a specified task is running, paused, or aborted. Also determines what program and line number is being executed, and what, if anything, the task is waiting for.

15.6.3 From KCL

The following KCL commands can be used to control and monitor the status of tasks. Refer to Appendix C, "KCL Command Alphabetic Descriptions," for more information.

- RUN <task_name> starts or continues a task.
- CONT <task_name> continues a task.
- PAUSE <task_name> pauses a task.
- ABORT <task_name> aborts a task.
- SHOW TASK <task_name> displays the status of a task.
- SHOW TASKS displays the status of all tasks.
15.7 USING SEMAPHORES AND TASK SYNCHRONIZATION

Good design dictates that separate tasks be able to operate somewhat independently. However, they should also be able to interact.

The KAREL controller supports counting semaphores. The following operations are permitted on semaphores:

• **Clear a semaphore** (KAREL: CLEAR_SEMA built-in): sets the semaphore count to zero.

  All semaphores are cleared at cold start. It is good practice to clear a semaphore prior to using it. Before several tasks begin sharing a semaphore, one and only one of these task, should clear the semaphore.

• **Post to a semaphore** (KAREL: POST_SEMA built-in): adds one to the semaphore count.

  If the semaphore count is zero or greater, when the post semaphore is issued, the semaphore count will be incremented by one. The next task waiting on the semaphore will decrement the semaphore count and continue execution. Refer to Figure 15–1.

  If the semaphore count is negative, when the post semaphore is issued, the semaphore count will be incremented by one. The task which has been waiting on the semaphore the longest will then continue execution. Refer to Figure 15–1.

• **Read a semaphore** (KAREL: SEMA_COUNT built-in): returns the current semaphore count.

• **Wait for a semaphore** (KAREL: PEND_SEMA built-in, SIGNAL SEMAPHORE Action):

  If the semaphore count is greater than zero when the wait semaphore is issued, the semaphore count will be decremented and the task will continue execution. Refer to Figure 15–1.

  If the semaphore count is less than or equal to zero (negative), the wait semaphore will decrement the semaphore count and the task will wait to be released by a post semaphore. Tasks are released on a first-in/first-out basis. For example, if task A waits on semaphore 1, then task B waits on semaphore 1. When task D posts semaphore 1, only task A will be released. Refer to Figure 15–1.
Figure 15–1. Task Synchronization Using a Semaphore

Example: Semaphores can be used to implement a task that acts as a request server. In the following example, the main task waits for the server to complete its operation. Semaphore[4] is used to control access to rqst_param or R[5]. Semaphore[5] is used to signal the server task that service is being requested; semaphore[6] is used by the server to signal that the operation is complete.

The main task would contain the following KAREL:

Main Task

```
--KAREL
CLEAR_SEMA(4)
CLEAR_SEMA(5)
CLEAR_SEMA(6)
RUN TASK('server',0,TRUE,TRUE,1,STATUS)
PEND_SEMA(4,max_time,time_out)
rqst_param=10
POST_SEMA(5)
PEND_SEMA(6,max_time,time_out)
```

The server task would contain the following KAREL code:
Server Task

--KAREL
POST_SEMA (4)
WHILE TRUE DO
  PEND_SEMA(5, max_time, time_out)
  IF rqst_param=10 THEN
    do_something
  ENDIF
  POST_SEMA(4)
  POST_SEMA(6)
ENDWHILE

Example: The program example in Semaphore and Task Synchronization Program Example - MAIN TASK thru Semaphore and Task Synchronization Program Example - TASK B shows how semaphores and tasks can be used together for synchronization. MAIN_TASK.KL is used to initialize the semaphore (MOTION_CTRL) and then runs both TASK_A.KL and TASK_B.KL. MAIN_TASK.KL then waits for TASK_A and TASK_B to abort before completing. TASK_A waits until you press F1 and then moves the robot to the HOME position. TASK_B waits until you press F2 and then moves the robot along a path.

Semaphore and Task Synchronization Program Example - MAIN TASK

PROGRAM main_task
%unlockgroup
VAR
  motion_ctrl: INTEGER
  tsk_a_done : BOOLEAN
  tsk_b_done : BOOLEAN
  tmr : INTEGER
  status : INTEGER
  -----------------------------------------------
  --
  -- INIT_LOCK: Initialize the semaphore
  -- to make sure its count is at
  -- zero before using it. Then
  -- post this semaphore which will--
  -- allow the first pend to the
  -- semaphore to continue
  -- execution.
  --
  -----------------------------------------------
ROUTINE init_lock
BEGIN
  CLEAR_SEMA (motion_ctrl) -- makes sure semaphore is zero before using it.
  POST_SEMA (motion_ctrl) -- makes motion_ctrl available immediately
END init_lock
  -----------------------------------------------
MULTI-TASKING

-- IS_TSK_DONE : Find out if the specified --
-- task is running or not. --
-- If the task is aborted then --
-- return TRUE otherwise FALSE.--

ROUTINE is_tsk_done (task_name:STRING): BOOLEAN

VAR
status : INTEGER -- The status of the operation of GET_TSK_INFO
task_no : INTEGER -- Receives the current task number for task_name
attr_out: INTEGER -- Receives the TSK_STATUS output
dummy : STRING[2] -- Does not receive any information

BEGIN
GET_TSK_INFO (task_name, task_no, TSK_STATUS, attr_out, dummy, status)
IF (attr_out = PG_ABORTED) THEN
RETURN (TRUE) -- If task is aborted then return TRUE
ENDIF
RETURN(FALSE) -- otherwise task is not aborted and return FALSE
END is_tsk_done

BEGIN
motion_ctrl = 1 -- Semaphore to allow motion control
init_lock -- Make sure this is done just once
FORCE_SPMENU ( tp_panel, spi_tpuser, 1) -- Force the Teach Pendant
-- user screen to be seen
RUN_TASK('task_a', 1, FALSE, FALSE, 1, status) -- Run task_a
RUN_TASK('task_b', 1, FALSE, FALSE, 1, status) -- Run task_b
REPEAT
    tsk_a_done = is_tsk_done ('task_a')
    tsk_b_done = is_tsk_done ('task_b')
    delay (100)
UNTIL (tsk_a_done and tsk_b_done) -- Repeat until both task_a
END main_task -- and task_b are aborted

Semaphore and Task Synchronization Program Example - TASK A

PROGRAM task_a
%nolockgroup

VAR
    motion_ctrl FROM main_task: INTEGER
    home_pos : XZWP
    status : INTEGER

RUN_HOME : Lock the robot motion --
    control. This task is --
-- moving the robot and must --
-- have control. --
--

------------------------------------------------
ROUTINE run_home
VAR
  time_out: BOOLEAN
BEGIN
  PEND_SEMA(motion_ctrl,-1,time_out)-- lock motion_ctrl from other tasks
  -- keep other tasks from moving robot
  LOCK_GROUP (1, status)
  SET_POS_REG(1, home_pos, status)
  do_move -- call TP program to move to the home position
  UNLOCK_GROUP (1, status)
  POST_SEMA(motion_ctrl) -- unlock motion_ctrl
  -- allow other task to move robot
END run_home
BEGIN
  set_cursor (tpfunc, 1, 4, status)
  write tpfunc ('HOME',CR)
  wait for TPIN[129]+ -- wait for F1 to be pressed
  run_home
END task_a

Semaphore and Task Synchronization Program Example - TASK B

PROGRAM task_b
%nolockgroup
VAR
  motion_ctrl FROM main_task : INTEGER
  work_path : PATH
  move_pos : XYZWPR
  status : INTEGER

  -----------------------------------------------
  --
  -- do_work : Lock the robot from other --
  -- tasks and do work. This --
  -- task is doing motion and --
  -- must lock motion control so --
  -- that another task does not --
  -- try to do motion at the --
  -- same time.
  --
  -----------------------------------------------
ROUTINE do_work
VAR
  time_out: BOOLEAN
15. MULTI-TASKING

```plaintext
BEGIN
    PEND_SEMA (motion_ctrl,-1,time_out) -- lock motion_ctrl from other
    -- tasks keep other tasks from
    -- moving robot
    LOCK_GROUP (1, status)
    FOR i=1 to PATH_LEN(work_path) DO
        move_pos = work_path[i]
        SET_POS_REG(1, move_pos, status)
        do_move -- call TP program to move to path node
    ENDFOR
    UNLOCK_GROUP (1, status)
    POST_SEMA(motion_ctrl) -- unlock motion_ctrl allow
    -- other task to move robot
END do_work
BEGIN
    set_cursor(tpfunc, 1, 10, status)
    write tpfunc('WORK',CR)
    wait for TPIN[131]+ -- wait until F2 is pressed
    do_work
END task_b
```

15.8 USING QUEUES FOR TASK COMMUNICATIONS

Queues are supported only in KAREL. A queue is a first-in/first-out list of integers. They are used to pass information to another task sequentially. A queue consists of a user variable of type QUEUE_TYPE and an ARRAY OF INTEGER. The maximum number of entries in the queue is determined by the size of the array.

The following operations are supported on queues:

- INIT_QUEUE initializes a queue and sets it to empty.
- APPEND_QUEUE adds an integer to the list of entries in the queue.
- GET_QUEUE: reads the oldest (top) entry from the queue and deletes it.

These, and other built-ins related to queues (DELETE_QUEUE, INSERT_QUEUE, COPY_QUEUE) are described in Appendix A.

A QUEUE_TYPE Data Type has one user accessible element, n_entries. This is the number of entries that have been added to the queue and not read out. The array of integer used with a queue, is used by the queue built-ins and should not be referenced by the KAREL program.

**Example:** The following example illustrates a more powerful request server, in which more than one task is posting requests and the requester does not wait for completion of the request.
The requester would contain the following code:

**Requester**

```plaintext
--declarations
VAR
rqst_queue FROM server: QUEUE_TYPE
rqst_data FROM server: ARRAY[100] OF INTEGER
status: INTEGER
seq_no: INTEGER
-- posting to the queue --
APPEND_QUEUE (req_code, rqst_queue, rqst_data, seq_no, status)
```

The server task would contain the following code:

**Server**

```plaintext
PROGRAM server
VAR
rqst_queue: QUEUE_TYPE
rqst_data : ARRAY[100] OF INTEGER
status : INTEGER
seq_no : INTEGER
rqst_code : INTEGER
BEGIN
INIT_QUEUE(rqst_queue) --initialization
WHILE TRUE DO --serving loop
    WAIT FOR rqst_code.n_entries > 0
    GET_QUEUE (rqst_queue, rqst_data, rqst_code, seq_no, status)
    SELECT rqst_code OF
    CASE (1): do_something
    ENDSELECT
ENDWHILE
END server
```
## Appendix A

### KAREL LANGUAGE ALPHABETICAL DESCRIPTION

## Contents

<table>
<thead>
<tr>
<th>Appendix A</th>
<th>KAREL LANGUAGE ALPHABETICAL DESCRIPTION</th>
<th>A–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>OVERVIEW</td>
<td>A–9</td>
</tr>
<tr>
<td>A.2</td>
<td>- A - KAREL LANGUAGE DESCRIPTION</td>
<td>A–18</td>
</tr>
<tr>
<td>A.2.1</td>
<td>ABORT Action</td>
<td>A–18</td>
</tr>
<tr>
<td>A.2.2</td>
<td>ABORT Condition</td>
<td>A–18</td>
</tr>
<tr>
<td>A.2.3</td>
<td>ABORT Statement</td>
<td>A–19</td>
</tr>
<tr>
<td>A.2.4</td>
<td>ABORT_TASK Built-In Procedure</td>
<td>A–19</td>
</tr>
<tr>
<td>A.2.5</td>
<td>ABS Built-In Function</td>
<td>A–20</td>
</tr>
<tr>
<td>A.2.6</td>
<td>ACOS Built-In Function</td>
<td>A–20</td>
</tr>
<tr>
<td>A.2.7</td>
<td>ACT SCREEN Built-In Procedure</td>
<td>A–21</td>
</tr>
<tr>
<td>A.2.8</td>
<td>ACT_TBL Built-In Procedure</td>
<td>A–22</td>
</tr>
<tr>
<td>A.2.9</td>
<td>ADD_BYNAMEPC Built-In Procedure</td>
<td>A–24</td>
</tr>
<tr>
<td>A.2.10</td>
<td>ADD_DICT Built-In Procedure</td>
<td>A–25</td>
</tr>
<tr>
<td>A.2.11</td>
<td>ADD_INTPC Built-In Procedure</td>
<td>A–26</td>
</tr>
<tr>
<td>A.2.12</td>
<td>ADD_REALPC Built-In Procedure</td>
<td>A–27</td>
</tr>
<tr>
<td>A.2.13</td>
<td>ADD_STRINGPC Built-In Procedure</td>
<td>A–29</td>
</tr>
<tr>
<td>A.2.14</td>
<td>%ALPHABETIZE Translator Directive</td>
<td>A–30</td>
</tr>
<tr>
<td>A.2.15</td>
<td>APPEND_NODE Built-In Procedure</td>
<td>A–30</td>
</tr>
<tr>
<td>A.2.16</td>
<td>APPEND_QUEUE Built-In Procedure</td>
<td>A–31</td>
</tr>
<tr>
<td>A.2.17</td>
<td>APPROACH Built-In Function</td>
<td>A–32</td>
</tr>
<tr>
<td>A.2.18</td>
<td>ARRAY Data Type</td>
<td>A–33</td>
</tr>
<tr>
<td>A.2.19</td>
<td>ARRAY_LEN Built-In Function</td>
<td>A–34</td>
</tr>
<tr>
<td>A.2.20</td>
<td>ASIN Built-In Function</td>
<td>A–34</td>
</tr>
<tr>
<td>A.2.21</td>
<td>Assignment Action</td>
<td>A–35</td>
</tr>
<tr>
<td>A.2.22</td>
<td>Assignment Statement</td>
<td>A–36</td>
</tr>
<tr>
<td>A.2.23</td>
<td>ATAN2 Built-In Function</td>
<td>A–38</td>
</tr>
<tr>
<td>A.2.24</td>
<td>ATACH Statement</td>
<td>A–39</td>
</tr>
<tr>
<td>A.2.25</td>
<td>ATT_WINDOW_D Built-In Procedure</td>
<td>A–39</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>A.2.26</td>
<td>ATT_WINDOW_S Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.2.27</td>
<td>AVL_POS_NUM Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.3.2</td>
<td>BYNAME Built-In Function</td>
<td></td>
</tr>
<tr>
<td>A.3.3</td>
<td>BYTE Data Type</td>
<td></td>
</tr>
<tr>
<td>A.3.4</td>
<td>BYTES_AHEAD Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.3.5</td>
<td>BYTES_LEFT Built-In Function</td>
<td></td>
</tr>
<tr>
<td>A.4.4</td>
<td>CALL_PROG Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.5</td>
<td>CALL_PROGLIN Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.6</td>
<td>CLOSE FILE Statement</td>
<td></td>
</tr>
<tr>
<td>A.4.7</td>
<td>CLOSE HAND Statement</td>
<td></td>
</tr>
<tr>
<td>A.4.8</td>
<td>CLEAR Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.9</td>
<td>CLR_IO_STAT Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.10</td>
<td>CLR_POS_REG Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.11</td>
<td>%CMOSVARS Translator Directive</td>
<td></td>
</tr>
<tr>
<td>A.4.12</td>
<td>%CMOS2SHADOW Translator Directive</td>
<td></td>
</tr>
<tr>
<td>A.4.13</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.14</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.15</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.16</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.17</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.18</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.19</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.20</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.21</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.22</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.23</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.24</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.25</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.26</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.27</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.28</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.29</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.30</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.31</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.32</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.33</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.34</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.35</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.36</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.37</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.38</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.39</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.40</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.41</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.42</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.43</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.44</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.45</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.46</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.47</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.48</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.49</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.50</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.51</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.52</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.53</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.54</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.55</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.56</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.57</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.58</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.59</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.60</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.61</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.62</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.63</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.64</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.65</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.66</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.67</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.68</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.69</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.70</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.71</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.72</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.73</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.74</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.75</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.76</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.77</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.78</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.79</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.80</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
<tr>
<td>A.4.81</td>
<td>COMPARE_FILE Built-In Procedure</td>
<td></td>
</tr>
</tbody>
</table>
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION  MARRC75KR07091E Rev D

A.7.5 FORMAT_DEV Built-In Procedure ................................................................. A-141
A.7.6 FRAME Built-In Function ........................................................................... A-142
A.7.7 FROM Clause ............................................................................................. A-144

A.8 - G - KAREL LANGUAGE DESCRIPTION ..................................................... A-145
A.8.1 GET_ATTR_PRG Built-In Procedure ........................................................... A-145
A.8.2 GET_FILE_POS Built-In Function .............................................................. A-147
A.8.3 GET_JPOS_REG Built-In Function .............................................................. A-148
A.8.4 GET_JPOS_TPE Built-In Function .............................................................. A-148
A.8.5 GET_PORT_ASG Built-in Procedure ........................................................... A-149
A.8.6 GET_PORT_ATR Built-In Function ............................................................ A-151
A.8.7 GET_PORT_CMT Built-In Procedure ......................................................... A-153
A.8.8 GET_PORT_MOD Built-In Procedure ......................................................... A-154
A.8.9 GET_PORT_SIM Built-In Procedure ......................................................... A-155
A.8.10 GET_PORT_VAL Built-In Procedure ......................................................... A-156
A.8.11 GET_POS_FRM Built-In Procedure ........................................................... A-157
A.8.12 GET_POS_REG Built-In Function ............................................................ A-158
A.8.13 GET_POS_TPE Built-In Function ............................................................ A-158
A.8.14 GET_POS_TYP Built-In Procedure ........................................................... A-159
A.8.15 GET_PREG_CMT Built-In-Procedure ....................................................... A-160
A.8.16 GET_QUEUE Built-In Procedure ............................................................. A-161
A.8.17 GET_REG Built-In Procedure ................................................................. A-162
A.8.18 GET_REG_CMT ....................................................................................... A-163
A.8.19 GET_SREG_CMT Builtin Procedure ....................................................... A-163
A.8.20 GET_STR_REG Built-In Procedure ........................................................... A-164
A.8.21 GET_TIME Built-In Procedure ............................................................... A-164
A.8.22 GET_TPE_CMT Built-in Procedure ......................................................... A-165
A.8.23 GET_TPE_PRM Built-in Procedure ......................................................... A-166
A.8.24 GET_TSK_INFO Built-In Procedure ........................................................ A-168
A.8.25 GET_USEC_SUB Built-In Procedure ...................................................... A-170
A.8.26 GET_USEC_TIM Built-In Function ........................................................ A-171
A.8.27 GET_VAR Built-In Procedure ................................................................ A-171
A.8.28 GO TO Statement ................................................................................. A-175

A.9 - H - KAREL LANGUAGE DESCRIPTION .................................................... A-176
A.9.1 HOLD Action .......................................................................................... A-176
A.9.2 HOLD Statement ................................................................................... A-177

A.10 - I - KAREL LANGUAGE DESCRIPTION ................................................... A-178
A.10.1 IF ... ENIF Statement ......................................................................... A-178
A.10.2 IN Clause ............................................................................................ A-179
A.10.3 %INCLUDE Translator Directive ........................................................... A-180
A.10.4 INDEX Built-In Function ..................................................................... A-181
A.10.5 INI_DYN_DISB Built-In Procedure ...................................................... A-181
A.10.6 INI_DYN_DISI Built-In Procedure ...................................................... A-183
A.10.7 INI_DYN_DISR Built-In Procedure ...................................................... A-184
A.10.8 INI_DYN_DISP Built-In Procedure ...................................................... A-186
A.10.9 INI_DYN_DISR Built-In Procedure ...................................................... A-187
A.10.10 INI_DYN_DISS Built-In Procedure .................................................... A-188
A.10.11 INIT_QUEUE Built-In Procedure ....................................................... A-189
A.10.12 INIT_TBL Built-In Procedure ............................................................. A-190
A.10.13 IN_RANGE Built-In Function ............................................................... A-201
A.10.14 INSERT_NODE Built-In Procedure ..................................................... A-202
A.10.15 INSERT_QUEUE Built-In Procedure ................................................... A-203
A.10.16 INTEGER Data Type .......................................................................... A-204
A.10.17 INV Built-In Function ...................................................................... A-205
A.10.18 IO_MOD_TYPE Built-In Procedure ..................................................... A-206
A.10.19 IO_STATUS Built-In Function .......................................................... A–207
A.11 - J - KAREL LANGUAGE DESCRIPTION ........................................... A–208
A.11.1 J_IN_RANGE Built-In Function ......................................................... A–208
A.11.2 JOINTPOS Data Type ................................................................. A–209
A.11.3 JOINT2POS Built-In Function ...................................................... A–209
A.12 - K - KAREL LANGUAGE DESCRIPTION ........................................... A–211
A.12.1 KCL Built-In Procedure .............................................................. A–211
A.12.2 KCL_NO_WAIT Built-In Procedure ................................................ A–212
A.12.3 KCL_STATUS Built-In Procedure .................................................. A–213
A.13 - L - KAREL LANGUAGE DESCRIPTION ........................................... A–213
A.13.1 LN Built-In Function ................................................................. A–213
A.13.2 LOAD Built-In Procedure ............................................................. A–214
A.13.3 LOAD_STATUS Built-In Procedure ................................................ A–215
A.13.4 LOCK_GROUP Built-In Procedure ................................................ A–216
A.13.5 %LOCKGROUP Translator Directive ............................................. A–217
A.14 - M - KAREL LANGUAGE DESCRIPTION ........................................... A–218
A.14.1 MIRROR Built-In Function .......................................................... A–218
A.14.2 MODIFY_QUEUE Built-In Procedure .............................................. A–219
A.14.3 MOTION_CTL Built-In Function .................................................... A–221
A.14.4 MOUNT_DEV Built-In Procedure .................................................. A–221
A.14.5 MOVE_FILE Built-In Procedure .................................................... A–222
A.14.6 MSG_CONNECT Built-In Procedure ............................................... A–223
A.14.7 MSG_DISCO Built-In Procedure .................................................. A–225
A.14.8 MSG_PING ................................................................................. A–225
A.15 - N - KAREL LANGUAGE DESCRIPTION ........................................... A–226
A.15.1 NOABORT Action ................................................................. A–226
A.15.2 %NOABORT Translator Directive ............................................... A–227
A.15.3 %NOBUSYLAMP Translator Directive .......................................... A–227
A.15.4 NODE_SIZE Built-In Function ..................................................... A–227
A.15.5 %NOLOCKGROUP Translator Directive ....................................... A–229
A.15.6 NOMESSAGE Action ............................................................... A–230
A.15.7 NOPAUSE Action ................................................................. A–230
A.15.8 %NOPAUSE Translator Directive ............................................... A–231
A.15.9 %NOPAUSESHFT Translator Directive ....................................... A–231
A.16 - O - KAREL LANGUAGE DESCRIPTION ........................................... A–232
A.16.1 OPEN FILE Statement ............................................................ A–232
A.16.2 OPEN HAND Statement ........................................................... A–233
A.16.3 OPEN_TPE Built-In Procedure .................................................... A–233
A.16.4 ORD Built-In Function ............................................................ A–234
A.16.5 ORIENT Built-In Function ........................................................ A–235
A.17 - P - KAREL LANGUAGE DESCRIPTION ........................................... A–236
A.17.1 PATH Data Type ................................................................. A–236
A.17.2 PATH_LEN Built-In Function ....................................................... A–238
A.17.3 PAUSE Action ................................................................. A–238
A.17.4 PAUSE Condition ............................................................... A–239
A.17.5 PAUSE Statement ............................................................... A–240
A.17.6 PAUSE_TASK Built-In Procedure ............................................... A–241
A.17.7 PEND_SEMA Built-In Procedure ................................................ A–241
A.17.8 PIPE_CONFIG Built-In Procedure ................................................ A–242
A.17.9 POP_KEY_RD Built-In Procedure ................................................ A–243
A.17.10 Port_Id Action ............................................................... A–244
A.17.11 Port_Id Condition ............................................................... A–245
A.17.12 POS Built-In Function .......................................................... A–246
A.17.13  POS2JOINT Built-In Function ................................................................. A–246
A.17.14  POS_REG_TYPE Built-In Procedure ......................................................... A–248
A.17.15  POSITION Data Type ........................................................................... A–249
A.17.16  POST_ERR Built-In Procedure ............................................................... A–250
A.17.17  POST_ERR_L Built-In Procedure ............................................................ A–251
A.17.18  POST_SEMA Built-In Procedure ............................................................ A–253
A.17.19  PRINT_FILE Built-In Procedure ............................................................. A–253
A.17.20  %PRIORITY Translator Directive .......................................................... A–254
A.17.21  PROG_BACKUP Built-In Procedure ......................................................... A–255
A.17.22  PROG_CLEAR Built-In Procedure ............................................................ A–258
A.17.23  PROG_RESTORE Built-In Procedure ....................................................... A–260
A.17.24  PROG_LIST Built-In Procedure ............................................................. A–262
A.17.25  PROGRAM Statement ........................................................................ A–263
A.17.26  PULSE Action ..................................................................................... A–264
A.17.27  PULSE Statement ............................................................................... A–265
A.17.28  PURGE CONDITION Statement ......................................................... A–266
A.17.29  PURGE_DEV Built-In Procedure ......................................................... A–267
A.17.30  PUSH_KEY_RD Built-In Procedure ....................................................... A–268
A.18 - Q - KAREL LANGUAGE DESCRIPTION .................................................. A–269
A.18.1  QUEUE_TYPE Data Type ....................................................................... A–269
A.19 - R - KAREL LANGUAGE DESCRIPTION ................................................ A–269
A.19.1  READ Statement ................................................................................. A–269
A.19.2  READ_DICT Built-In Procedure .............................................................. A–271
A.19.3  READ_DICT_V Built-In Procedure ......................................................... A–272
A.19.4  READ_KB Built-In Procedure ................................................................. A–273
A.19.5  REAL Data Type ................................................................................... A–277
A.19.6  Relational Condition ............................................................................ A–279
A.19.7  RELAX HAND Statement .................................................................. A–280
A.19.8  RELEASE Statement .......................................................................... A–281
A.19.9  REMOVE_DICT Built-In Procedure ...................................................... A–281
A.19.10  RENAME_FILE Built-In Procedure ..................................................... A–282
A.19.11  RENAME_VAR Built-In Procedure ...................................................... A–283
A.19.12  RENAME_VARS Built-In Procedure .................................................... A–283
A.19.13  REPEAT ... UNTIL Statement ............................................................ A–284
A.19.15  RESUME Action ............................................................................... A–286
A.19.16  RESUME Statement ......................................................................... A–286
A.19.17  RETURN Statement ......................................................................... A–287
A.19.18  ROUND Built-In Function ................................................................ A–288
A.19.19  ROUTINE Statement ....................................................................... A–289
A.19.20  RUN_TASK Built-In Procedure ............................................................. A–290
A.20 - S - KAREL LANGUAGE DESCRIPTION .................................................. A–291
A.20.1  SAVE Built-In Procedure ..................................................................... A–291
A.20.2  SAVE_DRAM Built-In Procedure ............................................................ A–291
A.20.3  SELECT ... ENDBLOCK Statement .................................................... A–292
A.20.4  SELECT_TPE Built-In Procedure ......................................................... A–293
A.20.5  SEMA_COUNT Built-In Function .......................................................... A–294
A.20.6  SEMAPHORE Condition ........................................................................ A–295
A.20.7  SEND_DATAPC Built-In Procedure ....................................................... A–295
A.20.8  SEND_EVENTPC Built-In Procedure ..................................................... A–296
A.20.9  SET_ATTR_PRG Built-In Procedure ....................................................... A–297
A.20.10  SET_CURSOR Built-In Procedure ......................................................... A–299
A.20.11  SET_EPOS_REG Built-In Procedure ..................................................... A–300
A.20.12  SET_EPOS_TPE Built-In Procedure ..................................................... A–301
A.20.13 SET_FILE_ATR Built-In Procedure ................................................................. A-302
A.20.14 SET_FILE_POS Built-In Procedure ............................................................ A-303
A.20.15 SET_INT_REG Built-In Procedure ............................................................... A-304
A.20.16 SET_JPOS_REG Built-In Procedure ............................................................ A-304
A.20.17 SET_JPOS_TPE Built-In Procedure ............................................................ A-305
A.20.18 SET_LANG Built-In Procedure ................................................................. A-306
A.20.19 SET_PERCH Built-In Procedure .............................................................. A-307
A.20.20 SET_PORT_ASG Built-In Procedure ......................................................... A-308
A.20.21 SET_PORT_ATR Built-In Function .......................................................... A-309
A.20.22 SET_PORT_CMT Built-In Procedure ....................................................... A-311
A.20.23 SET_PORT_MOD Built-In Procedure ......................................................... A-312
A.20.24 SET_PORT_SIM Built-In Procedure ......................................................... A-313
A.20.25 SET_PORT_VAL Built-In Procedure ......................................................... A-314
A.20.26 SET_POS_REG Built-In Procedure ............................................................ A-315
A.20.27 SET_POS_TPE Built-In Procedure ........................................................... A-316
A.20.28 SET_PREG_CMT Built-In Procedure ....................................................... A-317
A.20.29 SET_REAL_REG Built-In Procedure ......................................................... A-317
A.20.30 SET_REG_CMT Built-In Procedure ........................................................ A-317
A.20.31 SET_SREG_CMT Built-In Procedure ....................................................... A-318
A.20.32 SET_STR_REG Built-In Procedure .......................................................... A-319
A.20.33 SET_TIME Built-In Procedure ................................................................. A-319
A.20.34 SET_TPE_CMT Built-In Procedure ........................................................ A-321
A.20.35 SET_TRNS_TPE Built-In Procedure ........................................................ A-321
A.20.36 SET_TSK_ATTR Built-In Procedure ........................................................ A-322
A.20.37 SET_TSK_NAME Built-In Procedure ....................................................... A-323
A.20.38 SET_VAR Built-In Procedure ................................................................ A-324
A.20.39 %SHADOWVARS Translator Directive .................................................. A-327
A.20.40 SHORT Data Type ............................................................................... A-327
A.20.41 SIGNAL EVENT Action ....................................................................... A-328
A.20.42 SIGNAL EVENT Statement ................................................................ A-328
A.20.43 SIGNAL SEMAPHORE Action ................................................................. A-329
A.20.44 SIN Built-In Function ............................................................................ A-329
A.20.45 SQRT Built-In Function ........................................................................ A-330
A.20.46 %STACKSIZE Translator Directive ...................................................... A-330
A.20.47 STD_PTH_NODE Data Type ................................................................. A-331
A.20.48 STOP Action ......................................................................................... A-331
A.20.49 STOP Statement .................................................................................. A-332
A.20.50 STRING Data Type ............................................................................... A-333
A.20.51 STR_LEN Built-In Function ................................................................ A-334
A.20.52 STRUCTURE Data Type ..................................................................... A-334
A.20.53 SUB_STR Built-In Function ................................................................ A-335

A.21 - T - KAREL LANGUAGE DESCRIPTION .................................................. A-336
A.21.1 TAN Built-In Function .......................................................................... A-336
A.21.2 %TIMESLICE Translator Directive ........................................................ A-337
A.21.3 %TMOTION Translator Directive ............................................................ A-337
A.21.4 TRANSLATE Built-In Procedure ............................................................. A-337
A.21.5 TRUNC Built-In Function .................................................................... A-339

A.22 - U - KAREL LANGUAGE DESCRIPTION .................................................. A-340
A.22.1 UNHOLD Action .................................................................................. A-340
A.22.2 UNHOLD Statement ........................................................................... A-340
A.22.3 UNINIT Built-In Function .................................................................... A-341
A.22.4 %UNINITVARS Translator Directive .................................................. A-341
A.22.5 UNLOCK_GROUP Built-In Procedure ..................................................... A-342
A.22.6 UNPAUSE Action ................................................................................. A-343
A.22.7 UNPOS Built-In Procedure ................................................................ A-344
A.22.8 USING ... ENDUSING Statement ................................................................. A–344
A.23 - V - KAREL LANGUAGE DESCRIPTION .................................................. A–345
A.23.1 V_CAM_CALIB /RVision Built-In Procedure ............................................. A–345
A.23.2 V_GET_OFFSET /RVision Built-In Procedure ........................................... A–347
A.23.3 V_GET_PASSFL /RVision Built-In Procedure ........................................... A–348
A.23.4 V_GET_QUEUE /RVision Built-in Procedure ........................................... A–350
A.23.5 V_INIT_QUEUE /RVision Built-in Procedure .......................................... A–350
A.23.6 V_RALC_QUEUE /RVision Built-in Procedure ........................................ A–351
A.23.7 V_RUN_FIND /RVision Built-In Procedure .............................................. A–351
A.23.8 V_SET_REF /RVision Built-In Procedure ............................................... A–353
A.23.9 V_START_VTRK /RVision Built-in Procedure ......................................... A–354
A.23.10 V_STOP_VTRK /RVision Built-in Procedure .......................................... A–354
A.23.11 VAR_INFO Built-In Procedure .............................................................. A–355
A.23.12 VAR_LIST Built-In Procedure .............................................................. A–357
A.23.13 VECTOR Data Type .............................................................................. A–360
A.23.14 VOL_SPACE Built-In Procedure ............................................................ A–361
A.23.15 VREG_FND_POS /RVision Built-in Procedure ....................................... A–362
A.23.16 VREG_OFFSET /RVision Built-in Procedure ........................................ A–363
A.24 - W - KAREL LANGUAGE DESCRIPTION ............................................. A–364
A.24.1 WAIT FOR Statement ....................................................................... A–364
A.24.2 WHEN Clause ...................................................................................... A–364
A.24.3 WHILE...ENDWHILE Statement .......................................................... A–365
A.24.4 WITH Clause ...................................................................................... A–365
A.24.5 WRITE Statement ............................................................................... A–366
A.24.6 WRITE_DICT Built-In Procedure .......................................................... A–367
A.24.7 WRITE_DICT_V Built-In Procedure ...................................................... A–368
A.25 - X - KAREL LANGUAGE DESCRIPTION ............................................. A–369
A.25.1 XML_ADDTAG Built-In Procedure .......................................................... A–369
A.25.2 XML_GETDATA Built-In Procedure ........................................................ A–370
A.25.3 XML_REMTAG Built-In Procedure ........................................................ A–371
A.25.4 XML_SCAN Built-In Procedure .............................................................. A–371
A.25.5 XML_SETVAR Built-In Procedure ........................................................ A–373
A.25.6 XYZWPR Data Type ............................................................................ A–374
A.25.7 XYZWPREXT Data Type ...................................................................... A–375
A.26 - Y - KAREL LANGUAGE DESCRIPTION ............................................. A–375
A.27 - Z - KAREL LANGUAGE DESCRIPTION ............................................. A–375
A.1 OVERVIEW

This appendix describes, in alphabetical order, each standard KAREL language element, including:

- Data types
- Executable statements and clauses
- Condition handler conditions and actions
- Built-in routines
- Translator directives

A brief example of a typical use of each element is included in each description.

Note If, during program execution, any uninitialized variables are encountered as arguments for built-in routines, the program pauses and an error is displayed. Either initialize the variable, KCL> SET VARIABLE command, or abort the program, using the KCL> ABORT command.

Conventions

This section describes each standard element of the KAREL language in alphabetical order. Each description includes the following information:

- **Purpose:** Indicates the specific purpose the element serves in the language
- **Syntax:** Describes the proper syntax needed to access the element in KAREL. Table A-1 describes the syntax notation that is used.

Table A–1. Syntax Notation

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt;</td>
<td>Enclosed words are optional</td>
<td>AAA &lt;BBB&gt;</td>
<td>AAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAA BBB</td>
<td></td>
</tr>
<tr>
<td>{}</td>
<td>Enclosed words are optional and can be repeated</td>
<td>AAA {BBB}</td>
<td>AAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAA BBB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAA BBB BBB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAA BBB BBB BBB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates alternatives</td>
<td>AAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BBB</td>
</tr>
<tr>
<td>&lt;</td>
<td>&gt;</td>
<td>Separates an alternative if only one or none can be used</td>
<td>AAA &lt;BBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AAA BBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AAA CCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exactly one alternative must be used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AAA CCC</td>
</tr>
</tbody>
</table>
Table A–1. Syntax Notation (Cont’d)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| { | } | Any combination of alternatives can be used | AAA \{BBB | CCC\} | AAA  
AAA BBB  
AAA CCC  
AAA BBB CCC  
AAA CCC BBB  
AAA BBB CCC BBB BBB |
| < | <> | Nesting of symbols is allowed. Look at the innermost notation first to see what it describes, then look at the next innermost layer to see what it describes, and so forth. | AAA <BBB <CCC | DDD> | AAA  
AAA BBB  
AAA BBB CCC  
AAA BBB DDD |

If the built-in is a function, the following notation is used to identify the data type of the value returned by the function:

**Function Return Type:** data_typ

Input and output parameter data types for functions and procedures are identified as:

[in] param_name: data_type

[out] param_name: data_type

where:

[in] specifies the data type of parameters which are passed into the routine

[out] specifies the data type of parameters which are passed back into the program from the routine

%ENVIRONMENT Group specifies the %ENVIRONMENT group for built-in functions and procedures, which is used by the off-line translator. Valid values are: BYNAM, CTDEF, ERRS, FDEV, FLBT, IOSETUP, KCL, MEMO, MIR, MOTN, MULTI, PATHOP, PBQMGR, REGOPE, STRNG, SYSDEF, TIM, TPE, TRANS, UIF, VECTR. The SYSTEM group is automatically used by the off-line translator.

- **Details:** Lists specific rules that apply to the language element. An italics-type font is used to denote keywords input by the user within the syntax of the element.
- **See Also:** Refers the reader to places in the document where more information can be found.
• **Example:** Displays a brief example and explanation of the element. Table A–2 through Table A–8 list the KAREL language elements, described in this appendix, by the type of element. Table A–9 lists these elements in alphabetical order.

<table>
<thead>
<tr>
<th>Table A–2.</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT Action</td>
<td></td>
</tr>
<tr>
<td>Assignment Action</td>
<td></td>
</tr>
<tr>
<td>CANCEL Action</td>
<td></td>
</tr>
<tr>
<td>CONTINUE Action</td>
<td></td>
</tr>
<tr>
<td>DISABLE CONDITION Action</td>
<td></td>
</tr>
<tr>
<td>ENABLE CONDITION Action</td>
<td></td>
</tr>
<tr>
<td>HOLD Action</td>
<td></td>
</tr>
<tr>
<td>NOABORT Action</td>
<td></td>
</tr>
<tr>
<td>NOMESSAGE Action</td>
<td></td>
</tr>
<tr>
<td>NOPAUSE Action</td>
<td></td>
</tr>
<tr>
<td>PAUSE Action</td>
<td></td>
</tr>
<tr>
<td>Port_Id Action</td>
<td></td>
</tr>
<tr>
<td>PULSE Action</td>
<td></td>
</tr>
<tr>
<td>RESUME Action</td>
<td></td>
</tr>
<tr>
<td>SIGNAL EVENT Action</td>
<td></td>
</tr>
<tr>
<td>SIGNAL SEMAPHORE Action</td>
<td></td>
</tr>
<tr>
<td>STOP Action</td>
<td></td>
</tr>
<tr>
<td>UNHOLD Action</td>
<td></td>
</tr>
<tr>
<td>UNPAUSE Action</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table A–3.</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL Clause</td>
<td></td>
</tr>
<tr>
<td>FROM Clause</td>
<td></td>
</tr>
<tr>
<td>IN Clause</td>
<td></td>
</tr>
<tr>
<td>WHEN Clause</td>
<td></td>
</tr>
<tr>
<td>WITH Clause</td>
<td></td>
</tr>
</tbody>
</table>
Table A–4. Conditions

<table>
<thead>
<tr>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT Condition</td>
</tr>
<tr>
<td>CONTINUE Condition</td>
</tr>
<tr>
<td>ERROR Condition</td>
</tr>
<tr>
<td>EVENT Condition</td>
</tr>
<tr>
<td>PAUSE Condition</td>
</tr>
<tr>
<td>Port_Id Condition</td>
</tr>
<tr>
<td>Relational Condition</td>
</tr>
<tr>
<td>SEMAPHORE Condition</td>
</tr>
</tbody>
</table>

Table A–5. Data Types

<table>
<thead>
<tr>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY Data Type</td>
</tr>
<tr>
<td>BOOLEAN Data Type</td>
</tr>
<tr>
<td>BYTE Data Type</td>
</tr>
<tr>
<td>CONFIG Data Type</td>
</tr>
<tr>
<td>DISP_DAT_T Data Type</td>
</tr>
<tr>
<td>FILE Data Type</td>
</tr>
<tr>
<td>INTEGER Data Type</td>
</tr>
<tr>
<td>JOINTPOS Data Type</td>
</tr>
<tr>
<td>PATH Data Type</td>
</tr>
<tr>
<td>POSITION Data Type</td>
</tr>
<tr>
<td>QUEUE_TYPE Data Type</td>
</tr>
<tr>
<td>REAL Data Type</td>
</tr>
<tr>
<td>SHORT Data Type</td>
</tr>
<tr>
<td>STD_PTH_NODE Data Type</td>
</tr>
<tr>
<td>STRING Data Type</td>
</tr>
<tr>
<td>STRUCTURE Data Type</td>
</tr>
<tr>
<td>VECTOR Data Type</td>
</tr>
<tr>
<td>XYZWPR Data Type</td>
</tr>
<tr>
<td>XYZWPREXT Data Type</td>
</tr>
</tbody>
</table>
Table A–6. Directives

<table>
<thead>
<tr>
<th>Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ALPHABETIZE</td>
</tr>
<tr>
<td>%CMOSVARS</td>
</tr>
<tr>
<td>%CMOS2SHADOW</td>
</tr>
<tr>
<td>%COMMENT</td>
</tr>
<tr>
<td>%CRTDEVICE</td>
</tr>
<tr>
<td>%DEFGROUP</td>
</tr>
<tr>
<td>%DELAY</td>
</tr>
<tr>
<td>%ENVIRONMENT</td>
</tr>
<tr>
<td>%INCLUDE</td>
</tr>
<tr>
<td>%LOCKGROUP</td>
</tr>
<tr>
<td>%NOABORT</td>
</tr>
<tr>
<td>%NOBUSYLAMP</td>
</tr>
<tr>
<td>%NOLOCKGROUP</td>
</tr>
<tr>
<td>%NOPAUSE</td>
</tr>
<tr>
<td>%NOPAUSESHFT</td>
</tr>
<tr>
<td>%PRIORITY</td>
</tr>
<tr>
<td>%SHADOWVARS</td>
</tr>
<tr>
<td>%STACKSIZE</td>
</tr>
<tr>
<td>%TIMESLICE</td>
</tr>
<tr>
<td>%TPMOTION</td>
</tr>
<tr>
<td>%UNINITVARS</td>
</tr>
</tbody>
</table>

Table A–7. KAREL Built-In Routine Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Identifier</th>
<th>CURR_PROG</th>
<th>VAR_INFO</th>
<th>PROG_LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byname</strong></td>
<td></td>
<td>CALL_PROG</td>
<td>FILE_LIST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL_PROGLIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Acquisition</strong></td>
<td></td>
<td>DAQ_CHECKP</td>
<td>DAQ_START</td>
<td>DAQ_UNREG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAQ_REGPIPE</td>
<td>DAQ_STOP</td>
<td>DAQ_WRITE</td>
</tr>
<tr>
<td><strong>Error Code Handling</strong></td>
<td></td>
<td>ERR_DATA</td>
<td>POST_ERR</td>
<td>POST_ERR_L</td>
</tr>
<tr>
<td><strong>File and Device Operation</strong></td>
<td></td>
<td>CHECK_NAME</td>
<td>MOUNT_DEV</td>
<td>XML_ADDTAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMPARE_FILE</td>
<td>MOVE_FILE</td>
<td>XML_GETDATA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY_FILE</td>
<td>PRINT_FILE</td>
<td>XML_REMTAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE_FILE</td>
<td>PURGE_DEV</td>
<td>XML_SCAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISMOUNT_DEV</td>
<td>RENAME_FILE</td>
<td>XML_SETVAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DOSFILE_INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORMAT_DEV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Identifier</td>
<td>Identifier</td>
<td>Identifier</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>Serial I/O, File Usage</td>
<td>BYTES_AHEAD</td>
<td>IO_STATUS</td>
<td>SET_FILE_ATR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BYTES_LEFT</td>
<td>MSG_CONNECT</td>
<td>SET_FILE_POS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLR_IO_STAT</td>
<td>MSG_DISCO</td>
<td>SET_PORT_ATR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_FILE_POS</td>
<td>MSG_PING</td>
<td>VOL_SPACE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_PORT_POS</td>
<td>PIPE_CONFIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_PORT_ATR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process I/O Setup</td>
<td>CLR_PORT_SIM</td>
<td>GET_PORT_SIM</td>
<td>SET_PORT_CMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_PORT_ASG</td>
<td>GET_PORT_VAL</td>
<td>SET_PORT_MODE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_PORT_CMT</td>
<td>IO_MOD_TYPE</td>
<td>SET_PORT_SIM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_PORT_MOD</td>
<td>SET_PORT_ASG</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KCL Operation</td>
<td>KCL</td>
<td>KCL_NO_WAIT</td>
<td>KCL_STATUS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Operation</td>
<td>CLEAR</td>
<td>PROG_BACKUP</td>
<td>RENAME_VARS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CREATE_VAR</td>
<td>PROG_CLEAR</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD</td>
<td>PROG_RESTORE</td>
<td>SAVE_DRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD_STATUS</td>
<td>RENAME_VAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mirror</td>
<td>MIRROR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motion and Program Control</td>
<td>CNCL_STP_MTN</td>
<td>MOTION_CTL</td>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-programming</td>
<td>ABORT_TASK</td>
<td>PAUSE_TASK</td>
<td>SEMA_COUNT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLEAR_SEMA</td>
<td>PEND_SEMA</td>
<td>SET_TSK_ATTR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONT_TASK</td>
<td>POST_SEMA</td>
<td>SET_TSK_NAME</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_TSK_INFO</td>
<td>RUN_TASK</td>
<td>UNLOCK_GROUP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCK_GROUP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path Operation</td>
<td>APPEND_NODE</td>
<td>DELETE_NODE</td>
<td>NODE_SIZE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COPY_PATH</td>
<td>INSERT_NODE</td>
<td>PATH_LEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Computer Communications</td>
<td>ADD_BYNAMEPC</td>
<td>ADD_REALPC</td>
<td>SEND_DATAPC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD_INTPC</td>
<td>ADD_STRINGPC</td>
<td>SEND_EVENTPC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>CHECK_EPOS</td>
<td>FRAME</td>
<td>POS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CNV_JPOS_REL</td>
<td>IN_RANGE</td>
<td>POS2JOINT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CNV_REL_JPOS</td>
<td>J_IN_RANGE</td>
<td>SET_PERCH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CURPOS</td>
<td>JOINT2POS</td>
<td>UNPOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CURJPOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queue Manager</td>
<td>APPEND_QUEUE</td>
<td>GET_QUEUE</td>
<td>INSERT_QUEUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COPY_QUEUE</td>
<td>INIT_QUEUE</td>
<td>MODIFY_QUEUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETE_QUEUE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A–7. KAREL Built—In Routine Summary (Cont’d)

<table>
<thead>
<tr>
<th>Category</th>
<th>Identifier</th>
<th>Identifier</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register Operation</strong></td>
<td>CLR_POS_REG</td>
<td>GET_SREG_CMT</td>
<td>SET_POS_REG</td>
</tr>
<tr>
<td></td>
<td>GET_JPOS_REG</td>
<td>GET_STR_REG</td>
<td>SET_PREG_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_POS_REG</td>
<td>POS_REG_TYPE</td>
<td>SET_REAL_REG</td>
</tr>
<tr>
<td></td>
<td>GET_PREG_REG</td>
<td>SET_EPOS_REG</td>
<td>SET_REG_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_REG</td>
<td>SET_INT_REG</td>
<td>SET_SREG_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_REG_CMT</td>
<td>SET_JPOS_REG</td>
<td>SET_STR_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_SREG_CMT</td>
<td>SET_REAL_REG</td>
<td></td>
</tr>
<tr>
<td><strong>String Operation</strong></td>
<td>CNV_CNF_STRG</td>
<td>CNV_CONF_STR</td>
<td>STR_LEN</td>
</tr>
<tr>
<td></td>
<td>CNV_CONF_STR</td>
<td>CNV_INT_STR</td>
<td>SUB_STR</td>
</tr>
<tr>
<td></td>
<td>CNV_REAL_STR</td>
<td>CNV_REAL_STR</td>
<td></td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>ABS</td>
<td>COS</td>
<td>ROUND</td>
</tr>
<tr>
<td></td>
<td>ACOS</td>
<td>EXP</td>
<td>SET_VAR</td>
</tr>
<tr>
<td></td>
<td>ARRAY_LEN</td>
<td>GET_VAR</td>
<td>SIN</td>
</tr>
<tr>
<td></td>
<td>ASIN</td>
<td>INDEX</td>
<td>SQRT</td>
</tr>
<tr>
<td></td>
<td>ATAN2</td>
<td>INV</td>
<td>TAN</td>
</tr>
<tr>
<td></td>
<td>BYNAME</td>
<td>LN</td>
<td>TRUNC</td>
</tr>
<tr>
<td></td>
<td>CHR</td>
<td>ORD</td>
<td>UNINIT</td>
</tr>
<tr>
<td><strong>Time-of-Day Operation</strong></td>
<td>CNV_STR_TIME</td>
<td>GET_TIME</td>
<td>GET_USEC_TIM</td>
</tr>
<tr>
<td></td>
<td>CNV_TIME_STR</td>
<td>GET_USEC_SUB</td>
<td>SET_TIME</td>
</tr>
<tr>
<td><strong>TPE Program</strong></td>
<td>AVL_POS_NUM</td>
<td>GET_POS_FRM</td>
<td>SET_ATTR_PRG</td>
</tr>
<tr>
<td></td>
<td>CLOSE_TPE</td>
<td>GET_POS_TPE</td>
<td>SET_EPOS_TPE</td>
</tr>
<tr>
<td></td>
<td>COPY_TPE</td>
<td>GET_POS_TYP</td>
<td>SET_JPOS_TPE</td>
</tr>
<tr>
<td></td>
<td>CREATE_TPE</td>
<td>GET_TPE_CMT</td>
<td>SET_POS_TPE</td>
</tr>
<tr>
<td></td>
<td>DEL_INST_TPE</td>
<td>GET_TPE_PRM</td>
<td>SET_TPE_CMT</td>
</tr>
<tr>
<td></td>
<td>GET_ATTR_PRG</td>
<td>OPEN_TPE</td>
<td>SET_TRNS_TPE</td>
</tr>
<tr>
<td></td>
<td>GET_JPOS_TPE</td>
<td>SELECT_TPE</td>
<td></td>
</tr>
<tr>
<td><strong>Translate</strong></td>
<td>TRANSLATE</td>
<td>DET_WINDOW</td>
<td>INI_DYN_DISS</td>
</tr>
<tr>
<td><strong>User Interface</strong></td>
<td>ACT_SCREEN</td>
<td>DISCTRL_ALPH</td>
<td>INIT_TBL</td>
</tr>
<tr>
<td></td>
<td>ADD_DICT</td>
<td>DISCTRL_FORM</td>
<td>POP_KEY_RD</td>
</tr>
<tr>
<td></td>
<td>ATT_WINDOW_D</td>
<td>DISCTRL_LIST</td>
<td>PUSH_KEY_RD</td>
</tr>
<tr>
<td></td>
<td>ATT_WINDOW_S</td>
<td>DISCTRL_PLMN</td>
<td>READ_DICT</td>
</tr>
<tr>
<td></td>
<td>CHECK_DICT</td>
<td>DISCTRL_SBMN</td>
<td>READ_DICT_V</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISP</td>
<td>DISCTRL_TBL</td>
<td>READ_KB</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISE</td>
<td>FORCE_SPMENU</td>
<td>REMOVE_DICT</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISI</td>
<td>INI_DYN_DISB</td>
<td>SET_CURSOR</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISR</td>
<td>INI_DYN_DISE</td>
<td>SET_LANG</td>
</tr>
<tr>
<td></td>
<td>CNC_DYN_DISS</td>
<td>INI_DYN_DISI</td>
<td>WRITE_DICT</td>
</tr>
<tr>
<td></td>
<td>DEF_SCREEN</td>
<td>INI_DYN_DISP</td>
<td>WRITE_DICT_V</td>
</tr>
<tr>
<td></td>
<td>DEF_WINDOW</td>
<td>INI_DYN_DISR</td>
<td></td>
</tr>
</tbody>
</table>
### Table A-7. KAREL Built-In Routine Summary (Cont’d)

<table>
<thead>
<tr>
<th>Category</th>
<th>Identifier</th>
<th>Identifier</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision Operation</td>
<td>V_CAM_CALIB</td>
<td>V_INIT_QUEUE</td>
<td>V_START_VTRK</td>
</tr>
<tr>
<td></td>
<td>V_GET_OFFSET</td>
<td>V_RALC_QUEUE</td>
<td>V_STOP_VTRK</td>
</tr>
<tr>
<td></td>
<td>V_GET_PASSFL</td>
<td>V_RUN_FIND</td>
<td>VREG_FND_POS</td>
</tr>
<tr>
<td></td>
<td>V_GET_QUEUE</td>
<td>V_SET_REF</td>
<td>VREG_OFFSET</td>
</tr>
</tbody>
</table>

### Table A-8. Items

CR Input/Output Item
<table>
<thead>
<tr>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT Statement</td>
</tr>
<tr>
<td>Assignment Statement</td>
</tr>
<tr>
<td>ATTACH Statement</td>
</tr>
<tr>
<td>CANCEL Statement</td>
</tr>
<tr>
<td>CANCEL FILE Statement</td>
</tr>
<tr>
<td>CLOSE FILE Statement</td>
</tr>
<tr>
<td>CLOSE HAND Statement</td>
</tr>
<tr>
<td>CONDITION..ENDCONDITION Statement</td>
</tr>
<tr>
<td>CONNECT TIMER Statement</td>
</tr>
<tr>
<td>DELAY Statement</td>
</tr>
<tr>
<td>DISABLE CONDITION Statement</td>
</tr>
<tr>
<td>DISCONNECT TIMER Statement</td>
</tr>
<tr>
<td>ENABLE CONDITION Statement</td>
</tr>
<tr>
<td>FOR..ENDFOR Statement</td>
</tr>
<tr>
<td>GO TO Statement</td>
</tr>
<tr>
<td>HOLD Statement</td>
</tr>
<tr>
<td>IF..ENDIF Statement</td>
</tr>
<tr>
<td>OPEN FILE Statement</td>
</tr>
<tr>
<td>OPEN HAND Statement</td>
</tr>
<tr>
<td>PAUSE Statement</td>
</tr>
<tr>
<td>PROGRAM Statement</td>
</tr>
<tr>
<td>PULSE Statement</td>
</tr>
<tr>
<td>PURGE CONDITION Statement</td>
</tr>
<tr>
<td>READ Statement</td>
</tr>
<tr>
<td>RELAX HAND Statement</td>
</tr>
<tr>
<td>RELEASE Statement</td>
</tr>
<tr>
<td>REPEAT...UNTIL Statement</td>
</tr>
<tr>
<td>RESUME Statement</td>
</tr>
<tr>
<td>RETURN Statement</td>
</tr>
<tr>
<td>ROUTINE Statement</td>
</tr>
<tr>
<td>SELECT..ENDSELECT Statement</td>
</tr>
<tr>
<td>SIGNAL EVENT Statement</td>
</tr>
<tr>
<td>STOP Statement</td>
</tr>
<tr>
<td>UNHOLD Statement</td>
</tr>
<tr>
<td>USING..ENDUSING Statement</td>
</tr>
<tr>
<td>WAIT FOR Statement</td>
</tr>
<tr>
<td>WHILE..ENDWHILE Statement</td>
</tr>
<tr>
<td>WRITE Statement</td>
</tr>
</tbody>
</table>
A.2 - A - KAREL LANGUAGE DESCRIPTION

A.2.1 ABORT Action

**Purpose:** Aborts execution of a running or paused task

**Syntax:** ABORT <PROGRAM[n]>

**Details:**
- If task execution is running or paused, the ABORT action will abort task execution.
- The ABORT action can be followed by the clause PROGRAM[n], where n is the task number to be aborted. Use GET_TASK_INFO to get a task number.
- If PROGRAM[n] is not specified, the current task execution is aborted.

**See Also:** GET_TSK_INFO Built-in

*Chapter 6 CONDITION HANDLERS*

**Example:** Refer to Section B.6, "Path Variables and Condition Handlers Program (PTHMOVE.KL)," for a detailed program example.

A.2.2 ABORT Condition

**Purpose:** Monitors the aborting of task execution

**Syntax:** ABORT <PROGRAM[n]>

- The ABORT condition is satisfied when the task is aborted. The actions specified by the condition handler will be performed.
- If PROGRAM [n] is not specified, the current task number is used.
- Actions that are routine calls will not be executed if task execution is aborted.
- The ABORT condition can be followed by the clause PROGRAM[n], where n is the task number to be monitored. Use GET_TSK_INFO to get a task number.

**See Also:** CONDITION ... ENDCONDITION Statement, GET_TSK_INFO Built-in, Chapter 6 CONDITION HANDLERS, Appendix E, “Syntax Diagrams,” for additional syntax information

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

Section B.6, "Path Variables and Condition Handlers Program" (PTHMOVE.KL)
A.2.3 ABORT Statement

**Purpose:** Terminates task execution and cancels any motion in progress (or pending)

**Syntax:** ABORT <PROGRAM[n]>

- After an ABORT, the program cannot be resumed. It must be restarted.
- The statement can be followed by the clause PROGRAM[n], where n is the task number to be aborted.

**See Also:** “Syntax Diagrams,” for additional syntax information

**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.12, "Displaying a List From a Dictionary File" (DCLST_EX.KL)
- Section B.1, "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.2.4 ABORT_TASK Built-In Procedure

**Purpose:** Aborts the specified running or paused task

**Syntax:** ABORT_TASK(task_name, force_sw, cancel_mtn_sw, status)

**Input/Output Parameters:**

- [in] task_name :STRING
- [in] force_sw :BOOLEAN
- [in] cancel_mtn_sw :BOOLEAN
- [out] status :INTEGER

**Details:**

%ENVIRONMENT Group :MULTI
• **task_name** is the name of the task to be aborted. If task name is '*ALL*’, all executing or paused tasks are aborted except the tasks that have the “ignore abort request” attribute set.

• **force_sw**, if true, specifies to abort a task even if the task has the “ignore abort request” set. **force_sw** is ignored if **task_name** is '*ALL*'.

• **cancel_mtn_sw** specifies whether motion is canceled for all groups belonging to the specified task.

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

**See Also:** CONT_TASK, RUN_TASK, PAUSE_TASK Built-In Procedures, NO_ABORT Action, %NO_ABORT Translator Directive, Chapter 15 MULTI-TASKING

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

### A.2.5 ABS Built-In Function

**Purpose:** Returns the absolute value of the argument x, which can be an INTEGER or REAL expression

**Syntax:** ABS(x)

Function Return Type :INTEGER or REAL

Input/Output Parameters :

[in] x :INTEGER or REAL expression

%ENVIRONMENT Group :SYSTEM

**Details:**

• Returns the absolute value of x, with the same data type as x.

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

### A.2.6 ACOS Built-In Function

**Purpose:** Returns the arc cosine (cos-1) in degrees of the specified argument

**Syntax:** ACOS(x)

Function Return Type :REAL
Input/Output Parameters:

[in] x :REAL

%ENVIRONMENT Group :SYSTEM

Details:

- x must be between -1.0 and 1.0; otherwise the program will abort with an error.
- Returns the arccosine of x.

Example: The following example sets ans_r to the arccosine of -1 and writes this value to the screen. The output for the following example is 180 degrees.

ACOS Built-In Function

```
routine take_acos
var
   ans_r: real
begin
   ans_r = acos (-1)
   WRITE (‘acos -1 ’, ans_r, CR)
END take_acos
```

The second example causes the program to abort since the input value is less than -1 and not within the valid range.

ACOS Built-In Function

```
routine take_acos
var
   ans_r: real
begin
   ans_r = acos (-1.5) -- causes program to abort
   WRITE (‘acos -1.5 ’, ans_r, CR)
END take_acos
```

A.2.7 ACT_SCREEN Built-In Procedure

Purpose: Activates a screen

Syntax: ACT_SCREEN

Input/Output Parameters:

[in] screen_name :STRING
Details:

- Causes the display device associated with the screen to be cleared and all windows attached to the screen to be displayed.
- `screen_name` must be a string containing the name of a previously defined screen, see DEF_SCREEN Built-in.
- The name of the screen that this replaces is returned in `old_screen_n`.
- Requires the USER or USER2 menu to be selected before activating the new screen, otherwise the status will be set to 9093.
  - To force the selection of the teach pendant user menu before activating the screen, use FORCE_SPMENU (tp_panel, SPI_TPUSER, 1).
  - To force the selection of the CRT/KB user menu before activating the screen, use FORCE_SPMENU (crt_panel, SPI_TPUSER, 1).
- If the USER menu is exited and re-entered, your screen will be reactivated as long as the KAREL task which called ACT_SCREEN continues to run. When the KAREL task is aborted, the system’s user screen will be re-activated. Refer to Section 7.10 for details on the system’s user screen.
- `status` explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: DEF_SCREEN Built-In Procedure

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

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

### A.2.8 ACT_TBL Built-In Procedure

**Purpose:** Acts on a key while displaying a table on the teach pendant

**Syntax:** ACT_TBL(action, def_item, table_data, term_char, attach_wind, status)

**Input/Output Parameters:**

- `[in,out]` action :INTEGER
- `[in,out]` def_item :INTEGER
- `[in,out]` table_data :XWORK_T
[out] term_char :INTEGER

[out] attach_wind :BOOLEAN

[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

• The INIT_TBL and ACT_TBL built-in routines should only be used instead of DISCTRL_TBL if special processing needs to be done with each keystroke or if function key processing needs to be done without exiting the table menu.

• The built-in INIT_TBL must be called before using this built-in. Do not continue if an error occurs in INIT_TBL.

• action must be one of the constants defined in the include file KLEVKEYS.KL. ACT_TBL will act on the key and return. The following keys have special meanings:
  — ky_disp_updt (Initial Display) - Table title is displayed, function key labels are displayed. Default item is displayed and highlighted in first line. Remaining lines are displayed. Dynamic display is initiated for all values. This should be the first key passed into ACT_TBL.
  — ky_reissue (Read Key) - ACT_TBL reads a key, acts on it, and returns it in action.
  — ky_cancel (Cancel Table) - All dynamic display is cancelled. This should be the last key passed into ACT_TBL if term_char does not equal ky_new_menu. If a new menu was selected, ACT_TBL will have already cancelled the dynamic display and you should not use ky_cancel.

• def_item is the row containing the item you want to be highlighted when the table is entered. On return, def_item is the row containing the item that was currently highlighted when the termination character was pressed.

• table_data is used to display and control the table. **Do not change this data; it is used internally.**

• term_char receives a code indicating the character or other condition that terminated the table. The codes for key terminating conditions are defined in the include file KLEVKEYS.KL. Keys normally returned are pre-defined constants as follows:
  — ky_undef - No termination character was pressed
  — ky_select - A selectable item was selected
  — ky_new_menu - A new menu was selected
  — ky_f1 through ky_f10 - A function key was selected

• attach_wind should be set TRUE if the table manager needs to be attached to the display devices when action is ky_disp_updt and detached from the display devices when action is ky_cancel. If it is already attached, this parameter can be set to FALSE. Typically this should be TRUE.

• status explains the status of the attempted operation. If not equal to 0, then an error occurred.
Example: Refer to the example for the INIT_TBL built-in, Section A.10.12.

A.2.9 ADD_BYNAMEPC Built-In Procedure

**Purpose:** To add an integer, real, or string value into a KAREL byte given a data buffer.

**Syntax:** 
`ADD_BYNAMEPC(dat_buffer, dat_index, prog_name, var_name, status)`

**Input/Output Parameters:**

- `[in] dat_buffer : ARRAY OF BYTE`
- `[in,out] dat_index : INTEGER`
- `[in] prog_name : STRING`
- `[in] var_name : STRING`
- `[out] status : INTEGER`

%ENVIRONMENT Group : PC

**Details:**

- `dat_buffer` - an array of up to 244 bytes.
- `dat_index` - the starting byte number to place the string value.
- `prog_name` - specifies the name of the program that contains the specified variable.
- `var_name` - refers to a static program variable. This is only supported by an integer, real, or string variable (arrays and structures are not supported).
- `status` - the status of the attempted operation. If not 0, then an error occurred and data was not placed into the buffer.

The ADD_BYNAMEPC built-in adds integer, real, and string values to the data buffer in the same manner as the KAREL built-ins ADD_INTPC, ADD_REALPC, and ADD_STRINGPC.

**See Also:** ADD_BYNAMEPC, ADD_INTPC, ADD_REALPC, ADD_STRINGPC

**Example:** See the following for an example of the ADD_BYNAMEPC built-in.

**ADD_BYNAMEPC Built-In Procedure**

```plaintext
PROGRAM TESTBYNM
%ENVIRONMENT PC
CONST
    er_abort = 2
VAR
```
dat_buffer: ARRAY[100] OF BYTE
index: INTEGER
status: INTEGER

BEGIN
    index = 1
    ADD_BYNAMEPC(dat_buffer,index, 'TESTDATA', 'INDEX', status)
    IF status<>0 THEN
        POST_ERR(status,'',0,er_abort)
    ENDIF
END testbym

A.2.10 ADD_DICT Built-In Procedure

Purpose: Adds the specified dictionary to the specified language.

Syntax: ADD_DICT(file_name, dict_name, lang_name, add_option, status)

Input/Output Parameters:

[in] file_name :STRING
[in] dict_name :STRING
[in] lang_name :STRING
[in] add_option :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- file_name specifies the device, path, and file name of the dictionary file to add. The file type is assumed to be '.TX' (text file).
- dict_name specifies the name of the dictionary to use when reading and writing dictionary elements. Only 4 characters are used.
- lang_name specifies to which language the dictionary will be added. One of the following pre-defined constants should be used:
  dp_default
  dp_english
• The default language should be used unless more than one language is required.
• add_option should be the following:
  dp_dram Dictionary will be loaded to DRAM memory and retained until the next INIT START.
• status explains the status of the attempted operation. If not equal to 0, then an error occurred
  adding the dictionary file.

See Also: READ_DICT, WRITE_DICT, REMOVE_DICT Built-In Procedures, Chapter 10
DICTIONARIES AND FORMS

Example: Refer to the following sections for detailed program examples:
  Section B.12, "Displaying a List From a Dictionary File" (DCLST_EX.KL)
  Section B.12, "Dictionary Files" (DCALPHEG.UTX)

A.2.11 ADD_INTPC Built-In Procedure

Purpose: To add an INTEGER value (type 16 - 10 HEX) into a KAREL byte data buffer.

Syntax : ADD_INTPC(dat_buffer, dat_index, number, status)

Input/Output Parameters :

[in] dat_buffer :ARRAY OF BYTE
[in,out] dat_index :INTEGER
[in] number :INTEGER
[ out] status :INTEGER

%ENVIRONMENT Group :PC

Details:

• dat_buffer - an array of up to 244 bytes.
• dat_index - the starting byte number to place the integer value.
• *number* - the integer value to place into the buffer.
• *status* - the status of the attempted operation. If not 0, then an error occurred and data was not put into the buffer.

The KAREL built-ins ADD_BYPNAMEPC, ADD_INTPC, ADD_REALPC, and ADD_STRINGPC can be used to format a KAREL byte buffer in the following way: INTEGER data is added to the buffer as follows (buffer bytes are displayed in HEX):

beginning index = dat_index

2 bytes - variable type

4 bytes - the number

2 bytes of zero (0) - end of buffer marker

The following is an example of an INTEGER placed into a KAREL array of bytes starting at index = 1:

```
0 10 0 0 0 5 0 0
```

where:

```
0 10 = INTEGER variable type
0 0 0 5 = integer number 5
0 0 = end of data in the buffer
```

On return from the built-in, index = 7.

See Also: ADD_BYPNAMEPC, ADD_INTPC, ADD_REALPC, ADD_STRINGPC

Example: Refer to the TESTDATA example in the built-in function SEND_DATAPC.

### A.2.12 ADD_REALPC Built-In Procedure

**Purpose:** To add a REAL value (type 17 - 11 HEX) into a KAREL byte data buffer.

**Syntax:** ADD_REALPC(dat_buffer, dat_index, number, status)

Input/Output Parameters:

[in] dat_buffer : ARRAY OF BYTE

[in, out] dat_index : INTEGER
Details:

- `dat_buffer` - an array of up to 244 bytes.
- `dat_index` - the starting byte number to place the real value.
- `number` - the real value to place into the buffer.
- `status` - the status of the attempted operation. If not 0, then an error occurred and data was not placed into the buffer.

The KAREL built-ins `ADD_BYNAMEPC`, `ADD_INTPC`, `ADD_REALPC`, and `ADD_STRINGPC` can be used to format a KAREL byte buffer in the following way:

REAL data is added to the buffer as follows (buffer bytes are displayed in HEX):

beginning index = `dat_index`

2 bytes - variable type

4 bytes - the number

2 bytes of zero (0) - end of buffer marker

The following is an example of an REAL placed into a KAREL array of bytes starting at index = 1:

01 14 3A CC CC D00

where:

01 = REAL variable type

43 AC CC CD = real number 345.600006

00 = end of data in the buffer

On return from the built-in, index = 7.

See Also: `ADD_BYNAMEPC`, `ADD_INTPC`, `ADD_REALPC`, `ADD_STRINGPC`

Example: Refer to the TESTDATA example in the built-in function `SEND_DATAPC`. 
A.2.13 ADD_STRINGPC Built-In Procedure

**Purpose:** To add a string value (type 209 - D1 HEX) into a KAREL byte data buffer.

**Syntax:** ADD_STRINGPC(dat_buffer, dat_index, item, status)

**Input/Output Parameters:**
- [in] dat_buffer :ARRAY OF BYTE
- [in,out] dat_index :INTEGER
- [in] item :string
- [out] status :INTEGER

%ENVIRONMENT Group :PC

**Details:**
- *dat_buffer* - an array of up to 244 bytes.
- *dat_index* - the starting byte number to place the string value.
- *item* - the string value to place into the buffer.
- *status* - the status of the attempted operation. If not 0, then an error occurred and data was not placed into the buffer.

The KAREL built-ins ADD_BYNAMEPC, ADD_INTPC, ADD_REALPC, and ADD_STRINGPC can be used to format a KAREL byte buffer in the following way:

STRING data is added to the buffer as follows:

beginning index = dat_index

2 bytes - variable type

1 byte - length of text string

text bytes

2 bytes of zero (0) - end of buffer marker

The following is an example of an STRING placed into a KAREL array of bytes starting at index = 1:

0 D1 7 4D 48 53 48 45 4C 4C 0 0 0

where:
0 D1 = STRING variable type

7 = there are 7 characters in string 'MHSHELL'

4D 48 53 48 45 4C 4C 0 = 'MHSHELL' with end of string 0

0 0 = end of data in the buffer

On return from the built-in, index = 12.

See Also: ADD_BYNAMEPC, ADD_INTPC, ADD_REALPC, ADD_STRINGPC

Example: Refer to the TESTDATA example in the built-in function SEND_DATAPC.

### A.2.14 %ALPHABETIZE Translator Directive

**Purpose:** Specifies that static variables will be created in alphabetical order when p-code is loaded.

**Syntax:** %ALPHABETIZE

**Details:**

- Static variables can be declared in any order in a KAREL program and %ALPHABETIZE will cause them to be displayed in alphabetical order in the DATA menu or KCL> SHOW V ARS listing.

**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.10**, "Using Dynamic Display Built-ins" (DYN_DISP.KL)
- **Section B.12**, "Displaying a List From a Dictionary File" (DCLST_EX.KL)

### A.2.15 APPEND_NODE Built-In Procedure

**Purpose:** Adds an uninitialized node to the end of the PATH argument

**Syntax:** APPEND_NODE(path_var, status)

Input/Output Parameters:
[in] path_var :PATH
[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- `path_var` is the path variable to which the node is appended.
- The appended PATH node is uninitialized. The node can be assigned values by directly referencing its NODEDATA structure.
- `status` explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: DELETE_NODE, INSERT_NODE Built-In Procedures

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

### A.2.16 APPEND_QUEUE Built-In Procedure

**Purpose:** Appends an entry to a queue if the queue is not full

**Syntax:** APPEND_QUEUE(value, queue, queue_data, sequence_no, status)

**Input/Output Parameters:**

- [in] value :INTEGER
- [in,out] queue :QUEUE_TYPE
- [in,out] queue_data :ARRAY OF INTEGER
- [out] sequence_no :INTEGER
- [out] status :INTEGER

%ENVIRONMENT Group :PBQMGR

**Details:**

- `value` specifies the value to be appended to the queue.
- `queue` 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.
- sequence_no is returned with the sequence number of the entry just appended.
- status is returned with the zero if an entry can be appended to the queue. Otherwise it is returned with 61001, “Queue is full.”

See Also: DELETE_QUEUE, INSERT_QUEUE Built-In Procedures. Refer to Section 15.8, "Using Queues for Task Communication," for more information and an example.

A.2.17 APPROACH Built-In Function

Purpose: Returns a unit VECTOR representing the z-axis of a POSITION argument

Syntax : APPROACH(posn)

Function Return Type :VECTOR

Input/Output Parameters :

[in] posn :POSITION

%ENVIRONMENT Group :VECTR

Details:

- Returns a VECTOR consisting of the approach vector (positive z-axis) of the argument posn.

Example: This program allows you to create a position that is 500 mm away from another position along the z-axis.

**APPROACH Function**

PROGRAM p_approach
VAR
    start_pos : POSITION
    app_vector : VECTOR
BEGIN
    app_vector = APPROACH (start_pos) --sets app_vector equal to the --z-axis of start_pos
    start_pos.location = start_pos.location + app_vector *500 --creates start_pos + 500 mm --in z direction
END p_approach

Note Approach has been left in for older versions of KAREL. You should now directly access the vectors of a POSITION (i.e., posn. approach.)
A.2.18 ARRAY Data Type

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

**Syntax:** `ARRAY[size{,size}] OF data_type`

where:

- `size`: an INTEGER literal or constant
- `data_type`: any type except PATH

**Details:**

- `size` indicates the number of elements in an ARRAY variable.
- `size` must be in the range 1 through 32767 and must be specified in a normal ARRAY variable declaration. The amount of available memory in your controller might restrict the maximum `size` of an ARRAY.
- Individual elements are referenced by the ARRAY name and the subscript `size`. For example, `table[1]` refers to the first element in the ARRAY `table`.
- An entire ARRAY can be used only in assignment statements or as an argument in routine calls. In an assignment statement, both ARRAY variables must be of the same `size` and `data_type`. If `size` is different, the program will be translated successfully but will be aborted during execution, with error 12304, "Array Length Mismatch."
- `size` is not specified when declaring ARRAY routine parameters; an ARRAY of any size can be passed as an ARRAY parameter to a routine.
- `size` is not used when declaring an ARRAY return type for a function. However, the returned ARRAY must be of the same `size` as the ARRAY to which it is assigned in the function call.
- Each element is of the same type designated by `data_type`.
- Valid ARRAY operators correspond to the valid operators of the individual elements in the ARRAY.
- Individual elements of an array can be read or written only in the format that corresponds to the data type of the ARRAY.
- Arrays of multiple dimensions can be defined. Refer to Chapter 2 for more information.
- Variable-sized arrays can be defined. Refer to Chapter 2 for more information.

**See Also:** ARRAY_LEN Built-In Function, Chapter 5 ROUTINES, for information on passing ARRAY variables as arguments in routine calls Chapter 7 FILE INPUT/OUTPUT OPERATIONS

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

- Section B.2, "Copying Path Variables" (CPY_PTH.KL)
Section B.8, "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)

Section B.9, "Using the File and Device Built-ins" (FILE_EX.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.2.19 ARRAY_LEN Built-In Function

**Purpose:** Returns the number of elements contained in the specified array argument

**Syntax:** ARRAY_LEN(ary_var)

Function Return Type : INTEGER

Input/Output Parameters:

[in] ary_var : ARRAY

%ENVIRONMENT Group : SYSTEM

- The returned value is the number of elements declared for ary_var, not the number of elements that have been initialized in ary_var.

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

### A.2.20 ASIN Built-In Function

**Purpose:** Returns arcsine (sin-1) in degrees of the specified argument

**Syntax:** ASIN(x)

Function Return Type : REAL

Input/Output Parameters:

[in] x : REAL

%ENVIRONMENT Group : SYSTEM

**Details:**

- Returns the arcsine of x.
• x must be between -1 and 1, otherwise the program will abort with an error.

Example: The following example sets ans_r to the arcsine of -1 and writes this value to the screen. The output for the following example is -90 degrees.

**ASIN Built-In Function**

ROUTINE take_asin
VAR
  ans_r: REAL
BEGIN
  ans_r = ASIN (-1)
  WRITE ('asin -1 ', ans_r, CR)
END take_asin

The second example causes the program to abort since the input value is less than -1 and not within the valid range.

**ASIN Built-In Function**

ROUTINE take_asin
VAR
  ans_r: REAL
BEGIN
  ans_r = ASIN (-1.5) -- causes program to abort
  WRITE ('asin -1.5 ', ans_r, CR)
END take_asin

**A.2.21 Assignment Action**

**Purpose:** Sets the value of a variable to the result of an evaluated expression

**Syntax:** variable [[subscript{,subscript}]. field] = expn

where:

variable : any KAREL variable

subscript : an INTEGER expression

expn : a valid KAREL expression

field : any field from a structured variable

**Details:**
• *variable* can be any user-defined variable, system variable with write access, or output port array with write access.

• *subscript* is used to access elements of an array.

• *field* is used to access fields in a structure.

• *expn* must be of the same type as the variable or element of *variable*.

• An exception is that an INTEGER expression can be assigned to a REAL. Any positional types can be assigned to each other.

• Only system variables with write access (listed as RW in Table 11-3, “System Variables Summary”) can be used on the left side of an assignment statement. System variables with read only (RO) or read write (RW) access can be used on the right side.

• Input port arrays cannot be used on the left side of an assignment statement.

**See Also:** Chapter 3 *USE OF OPERATORS*, for detailed information about expressions and their evaluation Chapter 6 *CONDITION HANDLERS*, for more information about using assignment actions.

**Example:** The following example uses the assignment action to turn DOUT[1] off and set *port_var* equal to DOUT[2] when EVENT[1] turns on.

**Assignment Action**

```plaintext
CONDITI0N[1]:
  WHEN EVENT[1] DO
    DOUT[1] = OFF
    port_var = DOUT[2]
ENDCONDITION
```

**A.2.22 Assignment Statement**

**Purpose:** Sets the value of a variable to the result of an evaluated expression

**Syntax:** `variable {[subscript{,subscript}] . field} = expn`

**where:**

*variable*: any KAREL variable

*subscript*: an INTEGER expression

*expn*: a valid KAREL expression

*field*: any field from a structured variable
Details:

- **variable** can be any user-defined variable, system variable with write access, or output port array with write access.
- **subscript** is used to access elements of an array.
- **field** is used to access fields in a structure.
- **expn** must be of the same type as the variable or element of **variable**.
- An exception is that an INTEGER expression can be assigned to a REAL. Any positional types can be assigned to each other. INTEGER, SHORT, and BYTE can be assigned to each other.
- If **variable** is of type ARRAY, and no subscript is supplied, the expression must be an ARRAY of the same type and size. A type mismatch will be detected during translation. A size mismatch will be detected during execution and causes the program to abort with error 12304, "Array Length Mismatch."
- If **variable** is a user-defined structure, and no field is supplied, the expression must be a structure of the same type.
- Only system variables with write access (listed as RW in Table 11-3, “System Variables Summary”) can be used on the left side of an assignment statement. System variables with read only (RO) or read write (RW) access can be used on the right side.

If read only system variables are passed as parameters to a routine, they are passed by value, so any attempt to modify them (with an assignment statement) through the parameter in the routine has no effect.

- Input port arrays cannot be used on the left side of an assignment statement.

**See Also:** Chapter 3 *USE OF OPERATORS*, for detailed information about expressions and their evaluation, Chapter 2 *LANGUAGE ELEMENTS*. Refer to Appendix B, "KAREL Example Programs," for more detailed program examples.

**Example:** The following example assigns an INTEGER literal to an INTEGER variable and then increments that variable by a literal and value.

**Assignment Statement**

```
int_var = 5
int_var = int_var + 5
```

**Example:** The next example multiplies the system variable $SPEED by a REAL value. It is then used to assign the ARRAY variable `array_1`, element `loop_count` to the new value of the system variable $SPEED.

**Assignment Statement**

```
$SPEED = $SPEED * .25
array_1[loop_count] = $SPEED
```
Example: The last example assigns all the elements of the ARRAY array_1 to those of ARRAY array_2, and all the fields of structure struc_var_1 to those of struc_var_2.

Assignment Statement
array_2 = array_1
struc_var_2 = struc_var_1

A.2.23 ATAN2 Built-In Function

Purpose: Returns a REAL angle, measured counterclockwise in degrees, from the positive x-axis to a line connecting the origin and a point whose x- and y- coordinates are specified as the x- and y-arguments

Syntax: ATAN2(x1, y1)

Function Return Type: REAL

Input/Output Parameters:
[in] x1 : REAL
[in] y1 : REAL

%ENVIRONMENT Group : SYSTEM

Details:
• x1 and y1 specify the x and y coordinates of the point.
• If x1 and y1 are both zero, the interpreter will abort the program.

Example: The following example uses the values 100, 200, and 300 respectively for x, y, and z to compute the orientation component direction. The position, p1 is then defined to be a position with direction as its orientation component.

ATAN2 Built-In Function
PROGRAM p_atan2
VAR
    p1 : POSITION
    x, y, z, direction : REAL
BEGIN
    x = 100 -- use appropriate values
    y = 200 -- for x,y,z on
    z = 300 -- your robot
    direction = ATAN2(x, y)
    p1 = POS(x, y, z, 0, 0, direction, 'n') -- r orientation component
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

--of POS equals angle

END p_atan2

--returned by ATAN2(100,200)

A.2.24 ATTACH Statement

**Purpose:** Gives the KAREL program control of motion for the robot arm and auxiliary and extended axes

**Syntax:** ATTACH

**Details:**

- Used with the RELEASE statement. If motion control is not currently released from program control, the ATTACH statement has no effect.
- If the teach pendant is still enabled, execution of the KAREL program is delayed until the teach pendant is disabled. The task status will show a hold of "attach done."
- Stopped motions can only be resumed following execution of the ATTACH statement.

**See Also:** RELEASE Statement, Appendix E, “Syntax Diagrams,” for additional syntax information.

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

A.2.25 ATT_WINDOW_D Built-In Procedure

**Purpose:** Attach a window to the screen on a display device

**Syntax:** ATT_WINDOW_D(window_name, disp_dev_nam, row, col, screen_name, status)

**Input/Output Parameters:**

- [in] window_name :STRING
- [in] disp_dev_nam :STRING
- [in] row :INTEGER
- [in] col :INTEGER
- [out] screen_name :STRING
- [out] status :INTEGER
%ENVIRONMENT Group :PBCORE

Details:

- Causes data in the specified window to be displayed or attached to the screen currently active on the specified display device.
- `window_name` must be a previously defined window.
- `disp_dev_nam` must be one of the display devices already defined:
  
  'CRT' CRT Device
  
  'TP' Teach Pendant Device

- `row` and `col` indicate the position in the screen. Row 1 indicates the top row; col 1 indicates the left-most column. The entire window must be visible in the screen where positioned. For example, if the screen is 24 rows by 80 columns (as defined by its associated display device) and the window is 2 rows by 80 columns, `row` must be in the range 1-23; `col` must be 1.

- The name of the active screen is returned in `screen_name`. This can be used to detach the window later.

- It is an error if the window is already attached to the screen.

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

A.2.26 ATT_WINDOW_S Built-In Procedure

**Purpose:** Attach a window to a screen

**Syntax:** ATT_WINDOW_S(window_name, screen_name, row, col, status)

Input/Output Parameters:

- [in] window_name :STRING
- [in] screen_name :STRING
- [in] row :INTEGER
- [in] col :INTEGER
- [out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:
• Causes data in the specified window to be displayed or attached to the specified screen at a specified row and column.

• *window_name* and *screen_name* must be previously defined window and screen names.

• *row* and *col* indicate the position in the screen. Row 1 indicates the top row; col 1 indicates the left-most column. The entire window must be visible in the screen as positioned. For example, if the screen is 24 rows by 80 columns (as defined by its associated display device) and the window is 2 rows by 80 columns, row must be in the range 1-23; col must be 1.

• If the screen is currently active, the data will immediately be displayed on the device. Otherwise, there is no change in the displayed data.

• It is an error if the window is already attached to the screen.

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

**See Also:** Section 7.10, "User Interface Tips," DET_WINDOW Built-In

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

### A.2.27 AVL_POS_NUM Built-In Procedure

**Purpose:** Returns the first available position number in a teach pendant program

**Syntax :** AVL_POS_NUM(open_id, pos_num, status)

**Input/Output Parameters :**

[in] open_id :INTEGER

[out] pos_num : INTEGER

[out] status : INTEGER

%ENVIRONMENT Group :TPE

**Details:**

• *open_id* specifies the opened teach pendant program. A program must be opened before calling this built-in.

• *pos_num* is set to the first available position number.

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

**Example:** Refer to Section B.14, "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL), for a detailed program example.
A.3 - B - KAREL LANGUAGE DESCRIPTION

A.3.1 BOOLEAN Data Type

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

**Syntax:** BOOLEAN

**Details:**

- The BOOLEAN data type represents the BOOLEAN predefined constants TRUE, FALSE, ON, and OFF.

  Table A–10 lists some examples of valid and invalid BOOLEAN values used to represent the Boolean predefined constants.

<table>
<thead>
<tr>
<th>VALID</th>
<th>INVALID</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>T</td>
<td>Must use entire word</td>
</tr>
<tr>
<td>ON</td>
<td>1</td>
<td>Cannot use INTEGER values</td>
</tr>
</tbody>
</table>

- TRUE and FALSE typically represent logical flags, and ON and OFF typically represent signal states. TRUE and ON are equivalent, as are FALSE and OFF.

- Valid BOOLEAN operators are
  - AND, OR, and NOT
  - Relational operators (>, >=, =, <=>, <, and <=)

- The following have BOOLEAN values:
  - BOOLEAN constants, whether predefined or user-defined (for example, ON is a predefined constant)
  - BOOLEAN variables and BOOLEAN fields in a structure
  - ARRAY OF BOOLEAN elements
  - Values returned by BOOLEAN functions, whether user-defined or built-in (for example, IN_RANGE(pos_var))
  - Values resulting from expressions that use relational or BOOLEAN operators (for example, x > 5.0)
  - Values of digital ports (for example, DIN[2])

- Only BOOLEAN expressions can be assigned to BOOLEAN variables, returned from BOOLEAN function routines, or passed as arguments to BOOLEAN parameters.
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.5, "Using Register Built-Ins" (REG_EX.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.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.3.2 BYNAME Built-In Function

Purpose: Allows a KAREL program to pass a variable, whose name is contained in a STRING, as a parameter to a KAREL routine. This means the programmer does not have to determine the variable name during program creation and translation.

Syntax: BYNAME (prog_name, var_name, entry)

Input/Output Parameters:

[in] prog_name : STRING
[in] var_name : STRING
[in,out] entry : INTEGER

%ENVIRONMENT Group : system

Details:

- This built-in can be used only to pass a parameter to a KAREL routine.
- \textit{entry} returns the entry number in the variable data table where \textit{var_name} is located. This variable does not need to be initialized and should not be modified.
- \textit{prog_name} specifies the name of the program that contains the specified variable. If \textit{prog_name} is equal to "" (double quotes), then the routine defaults to the task name being executed.
• `var_name` must refer to a static, program variable.
• If `var_name` does not contain a valid variable name or if the variable is not of the type expected as a routine parameter, the program is aborted.
• System variables cannot be passed using BYNAME.
• The PATH data type cannot be passed using BYNAME. However, a user-defined type that is a PATH can be used instead.

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

### A.3.3 BYTE Data Type

**Purpose:** Defines a variable as a BYTE data type

**Syntax:** `BYTE`

**Details:**

• BYTE has a range of \(0 \leq n \leq 255\). No uninitialized checking is done on bytes.
• BYTES are allowed only within an array or within a structure.
• BYTES can be assigned to SHORTs and INTEGERS, and SHORTs and INTEGERS can be assigned to BYTES. An assigned value outside the BYTE range will be detected during execution and cause the program to abort.

Example: The following example defines an array of BYTE and a structure containing BYTES.

**BYTE Data Type**

```plaintext
PROGRAM byte_ex
%NOLOCKGROUP
TYPE
  mystruct = STRUCTURE
    param1: BYTE
    param2: BYTE
    param3: SHORT
ENDSTRUCTURE
VAR
  array_byte: ARRAY[10] OF BYTE
  myvar: mystruct
BEGIN
  array_byte[1] = 254
  myvar.param1 = array_byte[1]
END byte_ex
```
A.3.4 BYTES_AHEAD Built-In Procedure

**Purpose:** Returns the number of bytes of input data presently in the read-ahead buffer for a KAREL file. Allows KAREL programs to check instantly if data has been received from a serial port and is available to be read by the program. BYTES_AHEAD is also supported on socket messaging and pipes.

**Syntax:** BYTES_AHEAD(file_id, n_bytes, status)

Input/Output Parameters:

- [in] file_id :FILE
- [out] n_bytes :INTEGER
- [out] status :INTEGER

%ENVIRONMENT Group :FLBT

**Details:**

- *file_id* specifies the file that was opened.
- The *file_id* must be opened with the ATR_READAHD attribute set greater than zero.
- *n_bytes* is the number of bytes in the read_ahead buffer.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
- A non-zero status will be returned for non-serial devices such as files.

**See Also:** Section 7.3.1, “File Attributes”

**Example:** The following example will clear Port 2 (FLPY:) from any bytes still remaining to be read.

BYTES_AHEAD Built-In Procedure

```
ROUTINE purge_port
VAR
  s1       : STRING[1]
  n_try    : INTEGER
  n_bytes  : INTEGER
  stat     : INTEGER
BEGIN
  stat=SET_PORT_ATR (port_2, ATR_READAHD, 1) -- sets FLPY: to have a read
  -- ahead buffer of 128 bytes
  OPEN FILE fi('RO', 'rdahd.tst')
  REPEAT
    BYTES_AHEAD (fi, n_bytes, stat)
      --Get number of bytes ready
      --to be read
```
if (n_bytes = 0) then --if there are no bytes then set stat
  stat = 282
endif
if (n_bytes >= 1) then --there are bytes to be read
  read fi(s1::1) --read in one byte at a time
  stat = io_status (fi) --get the status of the read operation
endif
UNTIL stat <> 0 --continue until no more bytes are left
END purge_port
BEGIN
  -- main program text here
END bytes_ahd

A.3.5 BYTES_LEFT Built-In Function

Purpose: Returns the number of bytes remaining in the current input data record

Syntax : BYTES_LEFT(file_id)

Function Return Type :INTEGER

Input/Output Parameters :

[in] file_id :FILE

%ENVIRONMENT Group :FLBT

Details:

- file_id specifies the file that was opened.
- If no read or write operations have been done or the last operation was a READ file_id (CR), a zero is returned.
- 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.

Note An infeed character (LF) is created when the ENTER key is pressed, and is counted by BYTES_LEFT.

- This function will return a non-zero value only when data is input from a keyboard (teach pendant or CRT/KB), not from files or ports.
Warning

This function is used exclusively for reading from a window to determine if more data has been entered. Do not use this function with any other file device. Otherwise, you could injure personnel or damage equipment.

See Also: Section 7.10.1, "User Menu on the Teach Pendant," Section 7.10.2, "User Menu on the CRT/KB"

Example: The following example reads the first number, \texttt{qd\_field}, and then uses BYTES\_LEFT to determine if the user entered any additional numbers. If so, these numbers are then read.

\textbf{BYTES\_LEFT Built-In Function}

\begin{verbatim}
PROGRAM p_bytesleft
%NOLOCKGROUP
%ENVIRONMENT flbt
CONST
  default_1 = 0
  default_2 = -1
VAR rgd_field, opt_field_1, opt_field_2: INTEGER
BEGIN
  WRITE('Enter integer field(s): ')
  READ(rgd_field)
  IF BYTES_LEFT(TPDISPLAY) > 0 THEN
    READ(opt_field_1)
  ELSE
    opt_field_1 = default_1
  ENDIF
  IF BYTES_LEFT(TPDISPLAY) > 0 THEN
    READ(opt_field_2)
  ELSE
    opt_field_2 = default_2
  ENDIF
END p_bytesleft
\end{verbatim}

\textbf{A.4 - C - KAREL LANGUAGE DESCRIPTION}

\textbf{A.4.1 CALL\_PROG Built-In Procedure}

\textbf{Purpose:} Allows a KAREL program to call an external KAREL or teach pendant program. This means that the programmer does not have to determine the program to be called until run time.

\textbf{Syntax:} CALL\_PROG(prog\_name, prog\_index)
Input/Output Parameters:

[in] prog_name : STRING

[in, out] prog_index : INTEGER

%ENVIRONMENT Group : PBCORE

Details:

• prog_name is the name of the program to be executed, in the current calling task.
• prog_index returns the entry number in the program table where prog_name is located. This variable does not need to be initialized and should not be modified.
• CALL_PROG cannot be used to run internal or external routines.

See Also: CURR_PROG and CALL_PROGLIN Built-In Functions

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

A.4.2 CALL_PROGLIN Built-In Procedure

Purpose: Allows a KAREL program to call an external KAREL or teach pendant program, beginning at a specified line. This means that the programmer does not need to know, at creation and translation, what program will be called. The programmer can decide this at run time.

Syntax: CALL_PROGLIN(prog_name, prog_line, prog_index, pause_entry)

Input/Output Parameters:

[in] prog_name : STRING

[in] prog_line : INTEGER

[in, out] prog_index : INTEGER

[in] pause_entry : BOOLEAN

%ENVIRONMENT Group : BYNAM

Details:

• prog_name is the name of the program to be executed, in the current calling task.
• prog_line specifies the line at which to begin execution for a teach pendant program. 0 or 1 is used for the beginning of the program.
• KAREL programs always execute at the beginning of the program.

• prog_index returns the entry number in the program table where prog_name is located. This variable does not need to be initialized and should not be modified.

• pause_entry specifies whether to pause program execution upon entry of the program.

• CALL_PROGLIN cannot be used to run internal or external routines.

**See Also:** CURR_PROG and CALL_PROG Built-In Function

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

### A.4.3 CANCEL Action

**Purpose:** Terminates any motion in progress

**Syntax:** CANCEL <GROUP[n\{,n\}]>

**Details:**

• Cancels a motion currently in progress or pending (but not stopped) for one or more groups.

• CANCEL does not cancel motions that are already stopped. To cancel a motion that is already stopped, use the CNCL_STP_MTN built-in routine.

• If the group clause is not present, all groups for which the task has control (when the condition is defined) will be canceled. In particular, if the program containing the condition handler definition contains the %NOLOCKGROUP directive, the CANCEL action will not cancel motion in any group.

• If a motion that is canceled and is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are canceled.

• The robot and auxiliary or extended axes decelerate smoothly to a stop. The remainder of the motion is canceled.

• Canceled motions are treated as completed and cannot be resumed.

• The CANCEL action in a global condition handler cancels any pending motions.

### A.4.4 CANCEL Statement

**Purpose:** Terminates any motion in progress.

**Syntax:** CANCEL <GROUP[n\{,n\}]>

**Details:**
Cancels a motion currently in progress or pending (but not stopped) for one or more groups.

CANCEL does not cancel motions that are already stopped. To cancel a motion that is already stopped, use the CNCL_STP_MTN built-in routine.

If the group clause is not present, all groups for which the task has control will be canceled. In particular, if the program using the CANCEL statement contains the %NOLOCKGROUP directive, the CANCEL statement will not cancel motion in any group.

If a motion that is canceled is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are canceled.

The robot and auxiliary axes decelerate smoothly to a stop. The remainder of the motion is canceled.

Canceled motions are treated as completed and cannot be resumed.

CANCEL does not affect stopped motions. Stopped motions can be resumed.

If an interrupt routine executes a CANCEL statement and the interrupted statement was a motion statement, when the interrupted program resumes, execution normally resumes with the statement following the motion statement.

CANCEL might not work as expected if it is used in a routine called by a condition handler. The motion might already be put on the stopped motion queue before the routine is called. Use a CANCEL action directly in the condition handler to be sure the motion is canceled.

Motion cannot be cancelled for a different task.

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

A.4.5 CANCEL FILE Statement

Purpose: Cancels a READ or WRITE statement that is in progress.

Syntax: CANCEL FILE [file_var]

where:

file_var :a FILE variable

Details:

• Used to cancel input or output on a specified file

• The built-in function IO_STATUS can be used to determine if a CANCEL FILE operation was successful or, if it failed to determine the reason for the failure.

See Also: IO_STATUS Built-In Function, Chapter 7 FILE INPUT/OUTPUT OPERATIONS, Appendix E, “Syntax Diagrams,” for additional syntax information
Example: The following example reads an integer, but cancels the read if the F1 key is pressed.

CANCEL FILE Statement

PROGRAM can_file_ex
%ENVIRONMENT FLBT
%ENVIRONMENT UIF
%NOLOCKGROUP
VAR
   int_var: INTEGER
ROUTINE cancel_read
BEGIN
   CANCEL FILE TPDISPLAY
END cancel_read
BEGIN
   CONDITION[1]:
      WHEN TPIN[ky_f1]+ DO
         cancel_read
         ENABLE CONDITION[1]
ENDCONDITION

   ENABLE CONDITION[1]
REPEAT
   -- Read an integer, but cancel if F1 pressed
   CLR_IO_STAT(TPDISPLAY)
   WRITE(CR, 'Enter an integer: ')
   READ(int_var)
   UNTIL FALSE
end can_file_ex

A.4.6 CHECK_DICT Built-In Procedure

Purpose: Checks the specified dictionary for a specified element

Syntax : CHECK_DICT(dict_name, element_no, status)

Input/Output Parameters :

[in] dict_name :STRING
[in] element_no :STRING
[out] status :INTEGER

%ENVIRONMENT Group :UIF
Details:

- `dict_name` is the name of the dictionary to check.
- `element_no` is the element number within the dictionary.
- `status` explains the status of the attempted operation. If not equal to 0, then the element could not be found.

See Also: ADD_DICT, READ_DICT, WRITE_DICT, REMOVE_DICT Built-In Procedures. Refer to the program example for the DISCTRL_LIST Built-In Procedure and 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.4.7 CHECK_EPOS Built-In Procedure

Purpose: Checks that the specified position is valid and that no motion errors will be generated when moving to this position.

Syntax: CHECK_EPOS (eposn, uframe, utool, status <, group_no>)

Input/Output Parameters:

- [in] `eposn`: XYZWPREXT
- [in] `uframe`: POSITION
- [in] `utool`: POSITION
- [out] `status`: INTEGER
- [in] `group_no`: INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- `eposn` is the XYZWPREXT position to be checked.
- `uframe` specifies the uframe position to use with `eposn`.
- `utool` specifies the utool position to use with `eposn`.
- `status` explains the status of the check. If the position is reachable, the status will be 0.
- `group_no` is optional, but if specified will be the group number for `eposn`. If not specified the default group of the program is used.
See Also: GET_POS_FRM

Example: Refer to Section B.8 (GEN_HEX.KL)

A.4.8 CHECK_NAME Built-In Procedure

**Purpose:** Checks a specified file or program name for illegal characters.

**Syntax:** CHECK_NAME (name_spec, status)

Input/Output Parameters:

[in] name_spec :STRING

[out] status :INTEGER

%ENVIRONMENT Group :FDEV

**Details:**

- Name_spec specifies the string to check for illegal characters. The string can be the file name or program name. It should not include the extension of the file or the program. This built-in does not handle special system names such as *SYSTEM*.

A.4.9 CHR Built-In Function

**Purpose:** Returns the character that corresponds to a numeric code

**Syntax:** CHR (code)

Function Return Type :STRING

Input/Output Parameters:

[in] code :INTEGER

%ENVIRONMENT Group :SYSTEM

**Details:**

- *code* represents the numeric code of the character for either the ASCII, Graphic, or Multinational character set.
- Returns a single character string that is assigned the value of *code*.
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

See Also: Appendix D, “ASCII Character Codes”

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

Section B.4, "Standard Routines" (ROUT_EX.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.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.4.10 CLEAR Built-In Procedure

Purpose: Clears the specified program and/or variables from memory

Syntax: CLEAR(file_spec, status)

Input/Output Parameters:

[in] file_spec : STRING
[out] status : INTEGER

%ENVIRONMENT Group : PBCORE

Details:

• file_spec specifies the program name and type of data to clear. The following types are valid:

  no ext : KAREL or Teach Pendant program and variables.TP : Teach Pendant program.PC : KAREL program.VR : KAREL variables

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

Example: The following example clears a KAREL program, clears the variables for a program, and clears a teach pendant program.

CLEAR Built-In Procedure

-- Clear KAREL program
CLEAR(‘test1.pc’, status)
-- Clear KAREL variables
CLEAR(‘testvars.vr’, status)
A.4.11  CLEAR_SEMA Built-In Procedure

**Purpose:** Clear the indicated semaphore by setting the count to zero

**Syntax:** CLEAR_SEMA(semaphore_no)

Input/Output Parameters:

[in] semaphore_no : INTEGER

%ENVIRONMENT Group : MULTI

**Details:**

• The semaphore indicated by `semaphore_no` is cleared.
• `semaphore_no` must be in the range of 1 to the number of semaphores defined on the controller.
• All semaphores are cleared at COLD start. It is good practice to clear a semaphore prior to using it. Before several tasks begin sharing a semaphore, one and only one of these tasks should clear the semaphore.

**See Also:** POST_SEMA, PEND_SEMA Built-In Procedures, SEMA_COUNT Built-In Function, examples in Chapter 14, "Multi-Tasking"

A.4.12  CLOSE FILE Statement

**Purpose:** Breaks the association between a FILE variable and a data file or communication port

**Syntax:** CLOSE FILE file_var

where:

file_var :a FILE variable

**Details:**

• `file_var` must be a static variable that was used in the OPEN FILE statement.
• Any buffered data associated with the `file_var` is written to the file or port.
• The built-in function IO_STATUS will always return zero.
See Also: IO_STATUS Built-In Function, Chapter 7 FILE INPUT/OUTPUT OPERATIONS, Appendix E, “Syntax Diagrams,” for additional syntax information

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

**A.4.13 CLOSE HAND Statement**

**Purpose:** Causes the specified hand to close

**Syntax:** CLOSE 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 in I/O system.
- The valid range of values for hand_num is 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.
- The program is aborted with an error if the value of hand_num is in range but the HAND signal represented by that value has not been assigned.

See Also: Chapter 14 INPUT/OUTPUT SYSTEM, for more information on hand signals, Appendix E, “Syntax Diagrams,” for additional syntax information

**Example:** The following example moves the robot to the first position and closes the hand specified by hand_num.

```
CLOSE HAND Statement
SET_POS_REG(1, p1, status)
move_pr – Call TP program to do move
CLOSE HAND hand_num
```

**A.4.14 CLOSE_TPE Built-In Procedure**

**Purpose:** Closes the specified teach pendant program

**Syntax:** CLOSE_TPE(open_id, status)

Input/Output Parameters:
[in] open_id :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- **open_id** indicates the teach pendant program to close. All teach pendant programs that are opened must be closed before they can be executed. Any unclosed programs remain opened until the KAREL program which opened it is aborted or runs to completion.
- **status** explains the status of the attempted operation. If 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.4.15 CLR_IO_STAT Built-In Procedure

**Purpose:** Clear the results of the last operation on the file argument

**Syntax:** CLR_IO_STAT(file_id)

**Input/Output Parameters:**

- [in] file_id :FILE

%ENVIRONMENT Group :PBCORE

**Details:**

- Causes the last operation result on file_id, which is returned by IO_STATUS, to be cleared to zero.

See Also: I/O-STATUS Built-In Function

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

### A.4.16 CLR_PORT_SIM Built-In Procedure

**Purpose:** Sets the specified port to be unsimulated

**Syntax:** CLR_PORT_SIM(port_type, port_no, status)
Input/Output Parameters :

[in] port_type :INTEGER
[in] port_no :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :iosetup

Details:

• port_type specifies the code for the type of port to unsimulate. Codes are defined in FR:KLIOTYS.KL.
• port_no specifies the port number to unsimulate.
• status is returned with zero if parameters are valid and the simulation of the specified port is cleared.

See Also: GET_PORT_SIM, SET_PORT_SIM Built-In Procedures

A.4.17 CLR_POS_REG Built-In Procedure

Purpose: Removes all data for the specified group in the specified position register

Syntax : CLR_POS_REG(register_no, group_no, status)

Input/Output Parameters :

[in] register_no :INTEGER
[in] group_no :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

• register_no specifies the register number whose data should be cleared.
• If group_no is zero, data for all groups is cleared.
• status explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: SET_POS_REG Built-In Procedure, GET_POS_REG Built-In Function

Example: The following example clears the first 100 position registers.
CLR_POS_REG Built-In Procedure

FOR register_no = 1 to 100 DO
    CLR_POS_REG(register_no, 0, status)
ENDFOR

A.4.18 %CMOSVARS Translator Directive

Purpose: Specifies the default storage for KAREL variables is permanent memory

Syntax: %CMOSVARS

Details:

- If %CMOSVARS is specified in the program, then all static variables by default will be created in permanent memory.
- If %CMOSVARS is not specified, then all static variables by default will be created in temporary memory.
- If a program specifies %CMOSVARS, but not all static variables need to be created in permanent memory, the IN DRAM clause can be used on selected variables.

See Also: Section A.10.2 IN DRAM Clause

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

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

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

A.4.19 %CMOS2SHADOW Translator Directive

Purpose: Instructs the translator to put all CMOS variables in Shadow memory

Syntax: %CMOS2SHADOW

A.4.20 CNC_DYN_DISB Built-In Procedure

Purpose: Cancels the dynamic display based on the value of a BOOLEAN variable in a specified window.

Syntax: CNC_DYN_DISB (b_var, window_name, status)
Input/Output Parameters:

[in] `b_var` : BOOLEAN
[in] `window_name` : STRING
[out] `status` : INTEGER

%ENVIRONMENT Group : UIF

Details:

- `b_var` is the boolean variable whose dynamic display is to be canceled.
- `window_name` must be a previously defined window name. See Section 7.10.1 and Section 7.10.2 for predefined window names.
- If there is more than one display active for this variable in this window, all the displays are canceled.
- `status` returns an error if there is no dynamic display active specifying this variable and window. If not equal to 0, then an error occurred.

See Also: INI_DYN_DISB Built-In Procedure

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

Section B.10, "Using Dynamic Display Built-ins" (DYN_DISP.KL)
Section B.11, "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)

### A.4.21 CNC_DYN_DISE Built-In Procedure

**Purpose:** Cancels the dynamic display based on the value of an INTEGER variable in a specified window.

**Syntax:** `CNC_DYN_DISE (e_var, window_name, status)`

Input/Output Parameters:

[in] `e_var` : INTEGER
[in] `window_name` : STRING
[out] `status` : INTEGER

%ENVIRONMENT Group : UIF
Details:

- $e\_var$ is the integer variable whose dynamic display is to be canceled.
- Refer to the CNC\_DY\_DISB built-in procedure for a description of the other parameters listed above.

See Also: INI\_DY\_DISE Built-In Procedure

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

Section B.10, "Using Dynamic Display Built-ins" (DYN\_DISP.KL)

Section B.11, "Manipulating Values of Dynamically Displayed Variables" (CHG\_DATA.KL)

### \textbf{A.4.22} CNC\_DY\_DISI Built-In Procedure

**Purpose:** Cancels the dynamic display of an INTEGR\_E variable in a specified window.

**Syntax:**
CNC\_DY\_DISI(int\_var, window\_name, status)

**Input/Output Parameters:**
- [in] int\_var :INTEGER
- [in] window\_name :STRING
- [out] status :INTEGER

**Details:**

- $int\_var$ is the integer variable whose dynamic display is to be canceled.
- Refer to the CNC\_DY\_DISB built-in procedure for a description of the other parameters listed above.

See Also: INI\_DY\_DISI Built-In Procedure

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

Section B.10, "Using Dynamic Display Built-ins" (DYN\_DISP.KL)

Section B.11, "Manipulating Values of Dynamically Displayed Variables" (CHG\_DATA.KL)
A.4.23 CNC_DYN_DISP Built-In Procedure

**Purpose:** Cancels the dynamic display based on the value of a port in a specified window.

**Syntax:** CNC_DYN_DISP(port_type, port_no, window_name, status)

Input/Output Parameters:
- [in] port_type : INTEGER
- [in] port_no : INTEGER
- [in] window_name : STRING
- [out] status : INTEGER

**Details:**
- `port_type` and `port_no` are integer values specifying the port whose dynamic display is to be canceled.
- Refer to the CNC_DYN_DISB built-in procedure for a description of the other parameters listed above.

**See Also:** INI_DYN_DISP Built-In Procedure for information on `port_type` codes.

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

- Section B.10, "Using Dynamic Display Built-ins" (DYN_DISP.KL)
- Section B.11, "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)

A.4.24 CNC_DYN_DISR Built-In Procedure

**Purpose:** Cancels the dynamic display of a REAL number variable in a specified window.

**Syntax:** CNC_DYN_DISR(real_var, window_name, status)

Input/Output Parameters:
- [in] real_var : REAL
- [in] window_name : STRING
- [out] status : INTEGER
%ENVIRONMENT Group :UIF

Details:

- *real_var* is the REAL variable whose dynamic display is to be canceled.
- Refer to the CNC_DYN_DISB built-in procedure for a description of the other parameters listed above.

See Also: INI_DYN_DISR Built-In Procedure

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

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

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

A.4.25 CNC_DYN_DISS Built-In Procedure

Purpose: Cancels the dynamic display of a STRING variable in a specified window.

Syntax : CNC_DYN_DISS(str_var, window_name, status)

Input/Output Parameters :

[in] str_var :STRING

[in] window_name :STRING

[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

- *str_var* is the STRING variable whose dynamic display is to be canceled.
- Refer to the CNC_DYN_DISB built-in procedure for a description of the other parameters listed above.

See Also: INI_DYN_DISS Built-In Procedure

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

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

Section B.11 , "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)
A.4.26 CNCL_STP_MTN Built-In Procedure

Purpose: Cancels all stopped motions

Syntax: CNCL_STP_MTN

%ENVIRONMENT Group :motn

- All stopped motions will be canceled for all groups that the program controls.
- The statements following the motion statements will be executed.
- CNCL_STP_MTN will have no effect if no motions are currently stopped.
- Motion cannot be cancelled for a different task.

Example: The following example will cancel all stopped motions for all groups that the program controls after an emergency stop has occurred.

CNCL_STP_MTN Built-In Procedure

ROUTINE e_stop_hndlr
BEGIN
CNCL_STP_MTN
END e_stop_hndlr
CONDITION[100]:
WHEN ERROR[estop] DO
UNPAUSE
ENABLE CONDITION[100]
e_stop_hndlr
END CONDITION
ENABLE CONDITION[100]

A.4.27 CNV_CNF_STRG Built-In Procedure

Purpose: Converts the specified CONFIG into a STRING using an optional group_no

Syntax: CNV_CNF_STRG(source, target <,group_no>)

Input/Output Parameters:

[in] source :CONFIG
[out] target :STRING
[in] group_no :INTEGER
%ENVIRONMENT Group :STRNG

Details:

- \textit{target} receives the STRING form of the configuration specified by \textit{source}.
- \textit{target} must be long enough to accept a valid configuration string for the robot arm attached to the controller. Otherwise, the program will be aborted with an error.

Using a length of 25 is generally adequate because the longest configuration string of any robot is 25 characters long.

- The config mask from \texttt{group_no} is used during conversion. If \texttt{group_no} is omitted, then default group for the program is assumed.

If \texttt{group_no} is specified, it must be in the range 1 to the total number of groups defined on the controller.

See Also: CNV_STR_CONF Built-In Procedure

\textbf{Example:} The following example converts the configuration from position \texttt{posn} into a STRING and puts it into \texttt{config_string} using the configuration mask from group 2. The string is then displayed on the screen.

\begin{verbatim}
CNV_CNFSTRG Built-In Procedure

CNV_CNFSTRG(posn.pos_config, config_string, 2)
WRITE(’Configuration of posn in group 2: ’, config_string, cr)
\end{verbatim}

\textbf{A.4.28} \textit{CNV_CONF_STR} Built-In Procedure

\textbf{Purpose:} Converts the specified \texttt{CONFIG} into a \texttt{STRING}

\textbf{Syntax:} \texttt{CNV_CONF_STR(source, target)}

\textbf{Input/Output Parameters:}

\begin{itemize}
  \item [in] source :CONFIG
  \item [out] target :STRING
\end{itemize}

%ENVIRONMENT Group :STRNG

\textbf{Details:}

- \textit{target} receives the STRING form of the configuration specified by \textit{source}.
• target must be long enough to accept a valid configuration string for the robot arm attached to the controller. Otherwise, the program will be aborted with an error.

Using a length of 25 is generally adequate because the longest configuration string of any robot is 25 characters long.

See Also: CNV_STR_CONF Built-In Procedure

Example: The following example converts the configuration from position posn into a STRING and puts it into config_string. The string is then displayed on the screen.

**CNV_CONF_STR Built-In Procedure**

```
CNV_CONF_STR(posn.pos_config, config_string)
WRITE('Configuration of posn: ', config_string, cr)
```

---

**A.4.29 CNV_INT_STR Built-In Procedure**

**Purpose**: Formats the specified INTEGER into a STRING

**Syntax**: CNV_INT_STR(source, length, base, target)

**Input/Output Parameters**:
- [in] source :INTEGER expression
- [in] length :INTEGER expression
- [in] base :INTEGER expression
- [out] target :STRING expression

**%ENVIRONMENT Group :PBCORE**

**Details**:

- **source** is the INTEGER to be formatted into a STRING.
- **length** specifies the minimum length of the target. The actual length of target may be greater if required to contain the contents of source and at least one leading blank.
- **base** indicates the number system in which the number is to be represented. base must be in the range 2-16 or 0 (zero) indicating base 10.
- If the values of length or base are invalid, target is returned uninitialized.
- If target is not declared long enough to contain source and at least one leading blank, it is returned with one blank and the rest of its declared length filled with “*”.
A.4.30 CNV_JPOS_REL Built-In Procedure

Purpose: Allows a KAREL program to examine individual joint angles as REAL values

Syntax: CNV_JPOS_REL(jointpos, real_array, status)

Input/Output Parameters:
[in] joint_pos :JOINTPOS
[out] real_array :ARRAY [num_joints] OF REAL
[out] status :INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- joint_pos is one of the KAREL joint position data types: JOINTPOS, or JOINTPOS1 through JOINTPOS9.
- num_joints can be smaller than the number of joints in the system. A value of nine can be used if the actual number of joints is unknown. Joint number one will be stored in real_array element number one, etc. Excess array elements will be ignored.
- The measurement of the real_array elements is in degrees.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CNV_REL_JPOS 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.4.31 CNV_REAL_STR Built-In Procedure

Purpose: Formats the specified REAL value into a STRING

Syntax: CNV_REAL_STR(source, length, num_digits, target)
Input/Output Parameters:

[in] source :REAL expression
[in] length :INTEGER expression
[in] num_digits :INTEGER expression
[out] target :STRING

%ENVIRONMENT Group :STRNG

Details:

• source is the REAL value to be formatted.
• length specifies the minimum length of the target. The actual length of target may be greater if required to contain the contents of source and at least one leading blank.
• num_digits specifies the number of digits displayed to the right of the decimal point. If num_digits is a negative number, source will be formatted in scientific notation (where the ABS(num_digits) represents the number of digits to the right of the decimal point.) If num_digits is 0, the decimal point is suppressed.
• If length or num_digits are invalid, target is returned uninitialized.
• If the declared length of target is not large enough to contain source with one leading blank, target is returned with one leading blank and the rest of its declared length filled with “*”s (asterisks).

See Also: CNV_STR_REAL Built-In Procedure

Example: The following example converts the REAL number in cur_volts into a STRING and puts it into volt_string. The minimum length of cur_volts is specified to be seven characters with two characters after the decimal point. The contents of volt_string is then displayed on the screen.

```
CNV_REAL_STR Built-In Procedure
cur_volts = AIN[2]
CNV_REAL_STR(cur_volts, 7, 2, volt_string)
WRITE('Voltage=',volt_string,CR)
```

A.4.32 CNV_REL_JPOS Built-In Procedure

Purpose: Allows a KAREL program to manipulate individual angles of a joint position

Syntax: CNV_REL_JPOS(real_array, joint_pos, status)
[in] real_array :ARRAY [num_joints] OF REAL

[out] joint_pos :JOINTPOS

[out] status :INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- *real_array* must have a declared size, equal to or greater than, the number of joints in the system. A value of nine can be used for *num_joints*, if the actual number of joints is unknown. Array element number one will be stored in joint number one, and so forth. Excess array elements will be ignored. If the array is not large enough the program will abort with an invalid argument error.

- If any of the elements of *real_array* that correspond to a joint angle are uninitialized, the program will be paused with an uninitialized variable error.

- The measurement of the *real_array* elements is degrees.

- *joint_pos* is one of the KAREL joint position types: JOINTPOS, or JOINTPOS1 through JOINTPOS9.

- *joint_pos* receives the joint position form of *real_array*.

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

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

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.4.33 CNV_STR_CONF Built-In Procedure

**Purpose:** Converts the specified configuration string into a CONFIG data type

**Syntax:** CNV_STR_CONF(source, target, status)

Input/Output Parameters :

[in] source :STRING

[out] target :CONFIG

[out] status :INTEGER
%ENVIRONMENT Group :STRNG

Details:

- target receives the CONFIG form of the configuration string specified by source.
- source must be a valid configuration string for the robot arm attached to the controller.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

See Also: CNV_CONF_STR Built-In Procedure

Example: The following example sets the configuration of position xyz_pos to the configuration specified by config_string and then moves the TCP to that position.

CNV_STR_CONF Built-In Procedure

CNV_STR_CONF(config_string, xyz_pos.config_data, status)

SET_POS_REG(1, xyz_pos, status) — Put xyz_pos in PR[1]
move_to_pr — Call TP program to move to PR[1]

A.4.34 CNV_STR_INT Built-In Procedure

Purpose: Converts the specified STRING into an INTEGER

Syntax : CNV_STR_INT(source, target)

Input/Output Parameters :

[in] source :STRING
[out] target :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

- source is converted into an INTEGER and stored in target.
- If source does not contain a valid representation of an INTEGER, target is set uninitialized.

See Also: CNV_INT_STR Built-In Procedure

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)
A.4.35  CNV_STR_REAL Built-In Procedure

**Purpose:** Converts the specified STRING into a REAL

**Syntax:** CNV_STR_REAL(source, target)

<table>
<thead>
<tr>
<th>Input/Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] source :STRING</td>
</tr>
<tr>
<td>[out] target :REAL</td>
</tr>
</tbody>
</table>

%ENVIRONMENT Group :PBCORE

**Details:**

- Converts *source* to a REAL number and stores the result in *target*.
- If *source* is not a valid decimal representation of a REAL number, *target* will be set uninitialized. *source* may contain scientific notation of the form `nn.nnEsnn` where `s` is a + or - sign.

**See Also:** CNV_REAL_STR Built-In Procedure

**Example:** The following example converts the STRING *str* into a REAL and puts it into *rate*.

```karel
CNV_STR_REAL Built-In Procedure

REPEAT
    WRITE(‘Enter rate:’)  
    READ(str)
    CNV_STR_REAL(str, rate)
UNTIL NOT UNINIT(rate)
```

A.4.36  CNV_STR_TIME Built-In Procedure

**Purpose:** Converts a string representation of time to an integer representation of time.

**Syntax:** CNV_STR_TIME(source, target)

<table>
<thead>
<tr>
<th>Input/Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] source :STRING</td>
</tr>
<tr>
<td>[out] target :INTEGER</td>
</tr>
</tbody>
</table>

%ENVIRONMENT Group :TIM
Details:

- The size of the string parameter, source, is STRING[20].
- source must be entered using “DD-MMM-YYY HH:MM:SS” format. The seconds specifier, “SS,” is optional. A value of zero (0) is used if seconds is not specified. If source is invalid, target will be set to 0.
- target can be used with the SET_TIME Built-In Procedure to reset the time on the system. If target is 0, the time on the system will not be changed.

See Also: SET_TIME Built-In Procedure

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 procedure. SET_TIME is then used to set the time within the KAREL system to the time specified by int_time.

CNV_STR_TIME Built-In Procedure

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

A.4.37 CNV_TIME_STR Built-In Procedure

Purpose: Converts an INTEGER representation of time to a STRING

Syntax: CNV_TIME_STR(source, target)

Input/Output Parameters:

[in] source :INTEGER
[out] target :STRING

%ENVIRONMENT Group :TIM

Details:

- The GET_TIME Built-In Procedure is used to determine the INTEGER representation of time. CNV_TIME_STR is used to convert source to target, which will be displayed in “DD-MMM-YYY HH:MM:” format.

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

A.4.38 %COMMENT Translator Directive

**Purpose:** Specifies a comment of up to 16 characters

**Syntax:**

%COMMENT = 'ssssssssssssssss'

where sssssssssssssss = space

**Details:**

- The comment can be up to 16 characters long.
- During load time, the comment will be stored as a program attribute and can be displayed on the teach pendant or CRT/KB.
- %COMMENT must be used after the PROGRAM statement, but before any CONST, TYPE, or VAR sections.

**See Also:** SET_ATTR_PRG and GET_ATTR_PRG Built-In Procedures

**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.5, "Using Register Built-ins" (REG_EX.KL)
- Section B.6, "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)
- 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.1, "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.4.39 COMPARE_FILE Built-in Procedure

**Purpose:** Compares the contents of one file with another file
Syntax: COMPARE_FILE(filea, fileb, result_file, ascii_flag, diff_count, status)

Input/Output Parameters:

[in] filea : STRING
[in] fileb : STRING
[out] result_file : FILE
[in] ascii_flag: BOOLEAN
[out] diff_count : INTEGER
[out] status : INTEGER

%ENVIRONMENT Group : FDEV

Details:

• filea specifies the device, name, and type of the file to compare. If no device is specified, the default device is used. You must specify both a name and type.

• fileb specifies the device, name, and type of the file to compare. If no device is specified, the default device is used. You must specify both a name and type.

• result_file this must be an open FILE variable open for write. The comparison result will be written to this FILE.

• ascii_flag specifies that the file(s) should be compared based on ASCII (text) content.

• diff_count how many ASCII (text) lines differed.

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

See Also: COPY_FILE, RENAME_FILE, DELETE_FILE Built-In Procedures

The following example illustrates how this built-in can be used to detect if any TP programs have changed since a reference copy was saved to MC:ref. To try this out copy md:*ls to mc:ref (or another file storage area of your choosing) and then run this program.

Figure A-1. Example of COMPARE_FILE built-in Procedure

-----------------------------------------------------------
PROGRAM compfiles
-----------------------------------------------------------
VAR
 status: integer
 prog_name: string[128]
 logfile: file
 file_ary: array [1638] of string[MAX_PROG_NAM]
tempstore in CMOS: string[40]
index: integer
findex: integer
n_progs: integer
diff_count: integer

BEGIN
    prog_name = '***'
    index = 0
    n_progs = 1
    diff_count = 1
    IF uninit(tempstore) THEN
        tempstore = 'MC:\ref'
    ENDIF

    OPEN FILE logfile ('RW', 'CMPFILES.LOG')
    PROG_LIST(prog_name, 2, 0, 1, file_ary, n_progs, index)
    FOR index = 1 TO n_progs DO

        -- COMPARE_FILE( filename1: string; (IN) First file name for comparison
        -- filename2: string; (IN) Second file name for comparison
        -- out_file: file; (OUT) Open KAREL file for output
        -- ascii_flag: boolean; (IN) If TRUE the comprison is ASCII
        -- count: integer; (OUT) ASCII- Number of differences/
        -- Binary first different byte
        -- status: integer) (OUT) Result such as cannot open file

        COMPARE_FILE('MD:\' + file_ary[index] + '.LS',
                     tempstore + '\ ' + file_ary[index] + '.LS', logfile, TRUE,
                     diff_count, status)
        write logfile ('Compared ' + file_ary[index], ' ', status, ' ',
                       diff_count, ' ', index, CR)

    ENDFOR
    CLOSE FILE logfile
END compfiles

A.4.40 CONDITION...ENDCONDITION Statement

Purpose: Defines a global condition handler

Syntax: CONDITION[cond_hand_no]: [with_list]

WHEN cond_list DO action_list
{WHEN cond_list DO action_list}

ENDCONDITION

Details:

• cond_hand_no specifies the number associated with the condition handler and must be in the range of 1-1000. The program is aborted with an error if it is outside this range.

• If a condition handler with the specified number already exists, the old one is replaced with the new one.

• The optional [with_list] can be used to specify condition handler qualifiers. See the WITH clause for more information.

• All of the conditions listed in a single WHEN clause must be satisfied simultaneously for the condition handler to be triggered.

• Multiple conditions must all be separated by the AND operator or the OR operator. Mixing of AND and OR is not allowed.

• The actions listed after DO are to be taken when the corresponding conditions of a WHEN clause are satisfied simultaneously.

• Multiple actions are separated by a comma or on a new line.

• Calls to function routines are not allowed in a CONDITION statement.

• The condition handler is initially disabled and is disabled again whenever it is triggered. Use the ENABLE statement or action, specifying the condition handler number, to enable it.

• Use the DISABLE statement or action to deactivate a condition handler.

• The condition handler remains defined and can subsequently be reactivated by the ENABLE statement or action.

• The PURGE statement can be used to delete the definition of a condition handler.

• Condition handlers are known only to the task which defines them. Two different tasks can use the same cond_hand_no even though they specify different conditions.

See Also: Chapter 6 CONDITION HANDLERS, Appendix E, “Syntax Diagrams,” for additional 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.4.41 CONFIG Data Type

Purpose: Defines a variable or structure field as a CONFIG data type

Syntax: CONFIG

Details:

• CONFIG defines a variable or structure field as a highly compact structure consisting of fields defining a robot configuration.

• CONFIG contains the following predefined fields:
  — CFG_TURN_NO1 :INTEGER
  — CFG_TURN_NO2 :INTEGER
  — CFG_TURN_NO3 :INTEGER
  — CFG_FLIP :BOOLEAN
  — CFG_LEFT :BOOLEAN
  — CFG_UP :BOOLEAN
  — CFG_FRONT :BOOLEAN

• Variables and fields of structures can be declared as CONFIG.

• Subfields of CONFIG data type can be accessed and set using the usual structure field notation.

• Variables and fields declared as CONFIG can be
  — Assigned to one another.
  — Passed as parameters.
  — Written to and read from unformatted files.

• Each subfield of a CONFIG variable or structure field can be passed as a parameter to a routine, but is always passed by value.

• A CONFIG field is part of every POSITION and XYZWPR variable and field.

• An attempt to assign a value to a CONFIG subfield that is too large for the field results in an abort error.

Example: The following example shows how subfields of the CONFIG structure can be accessed and set using the usual structure field notation.

CONFIG Data Type

VAR
config_var1, config_var2: CONFIG
pos_var: POSITION
seam_path: PATH
A.4.42 CONNECT TIMER Statement

**Purpose:** Causes an INTEGER variable to start being updated as a millisecond clock

**Syntax:** CONNECT TIMER TO clock_var

where:

- clock_var : a static, user-defined INTEGER variable

**Details:**

- *clock_var* is presently incremented by the value of the system variable $SCR.$COND_TIME every $SCR.$COND_TIME milliseconds as long as the program is running or paused and continues until the program disconnects the timer, ends, or aborts. For example, if $SCR.$COND_TIME = 32 then *clock_var* will be incremented by 32 every 32 milliseconds.

- You should initialize *clock_var* before using the CONNECT TIMER statement to ensure a proper starting value.

- If the variable is uninitialized, it will remain so for a short period of time (up to 32 milliseconds) and then it will be set to a very large negative value (-2.0E31 + 32 milliseconds) and incremented from that value.

- The program can reset the *clock_var* to any value while it is connected.

- A *clock_var* initialized at zero wraps around from approximately two billion to approximately minus two billion after about 23 days.

- If *clock_var* is a system variable or a local variable in a routine, the program cannot be translated.

**Note** If two CONNECT TIMER statements using the same variable, are executed in two different tasks, the timer will advance twice as fast. For example, if the timer will be incremented by 2 * $SCR.$COND_TIME every $SCR.$COND_TIME ms. However, this does not occur if two or more CONNECT TIMER statements using the same variable, are executed in the same task.

**See Also:** Appendix E for additional syntax information, DISCONNECT TIMER Statement
Example: Refer to the following sections for detailed program examples:

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)

A.4.43 CONTINUE Action

Purpose: Continues execution of a paused task

Syntax : CONTINUE <PROGRAM[n]>

Details:

• The CONTINUE action will not resume stopped motions.
• If program execution is paused, the CONTINUE action will continue program execution.
• The CONTINUE action can be followed by the clause PROGRAM[n], where n is the task number to be continued. Use GET_TSK_INFO to get a task number for a specified task name.
• A task can be in an interrupt routine when CONTINUE is executed. However, you should be aware of the following circumstances because CONTINUE only affects the current interrupt level, and interrupt levels of a task might be independently paused or running.
  — If the interrupt routine and the task are both paused, CONTINUE will continue the interrupt routine but the task will remain paused.
  — If the interrupt routine is running and the task is paused, CONTINUE will appear to have no effect because it will try to continue the running interrupt routine.
  — If the interrupt routine is paused and the task is running, CONTINUE will continue the interrupt routine.

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

A.4.44 CONTINUE Condition

Purpose: Condition that is satisfied when program execution is continued

Syntax : CONTINUE <PROGRAM[n]>

Details:

• The CONTINUE condition monitors program execution.

  If program execution is paused, the CONTINUE action, issuing CONTINUE from the CRT/KB
or a CYCLE START from the operator panel, will continue program execution and satisfy the CONTINUE condition.

- The CONTINUE condition can be followed by the clause PROGRAM[n], where n is the task number to be continued. Use GET_TSK_INFO to get the task number of a specified task name.

**Example:** In the following example, program execution is being monitored. When the program is continued, a digital output will be turned on.

**CONTINUE Condition**

```plaintext
CONDITION[1]:
  WHEN CONTINUE DO DOUT[1] = ON
ENDCONDITION
```

**A.4.45 CONT_TASK Built-In Procedure**

**Purpose:** Continues the specified task

**Syntax:** CONT_TASK(task_name, status)

**Input/Output Parameters:**

- **[in] task_name :STRING**
- **[out] status :INTEGER**

**Details:**

- **task_name** is the name of the task to be continued. If the task was not paused, an error is returned in status.
- **status** explains the status of the attempted operation. If not equal to 0, then an error occurred.
- A task can be in an interrupt routine when CONT_TASK is executed. However, you should be aware of the following circumstances because CONT_TASK only affects the current interrupt level, and interrupt levels of a task might be independently paused or running.
  - If the interrupt routine and the task are both paused, CONT_TASK will continue the interrupt routine but the task will remain paused.
  - If the interrupt routine is running and the task is paused, CONT_TASK will appear to have no effect because it will try to continue the running interrupt routine.
  - If the interrupt routine is paused and the task is running, CONT_TASK will continue the interrupt routine.
See Also: RUN_TASK, ABORT_TASK, PAUSE_TASK Built-In Procedures, Chapter 15 MULTI-TASKING

Example: The following example prompts the user for the task name and continues the task execution. Refer to Chapter 15 MULTI-TASKING, for more examples.

**CONT_TASK Built-In Procedure**

```karel
PROGRAM cont_task_ex
  %ENVIRONMENT MULTI
  VAR
    task_str: STRING[12]
    status: INTEGER
  BEGIN
    WRITE('Enter task name to continue:')
    READ(task_str)
    CONT_TASK(task_str, status)
  END cont_task_ex
```

**A.4.46 COPY_FILE Built-In Procedure**

**Purpose:** Copies the contents of one file to another with the overwrite option

**Syntax:** COPY_FILE(from_file, to_file, overwrite_sw, nowait_sw, status)

Input/Output Parameters:

- [in] from_file :STRING
- [in] to_file :STRING
- [in] overwrite_sw :BOOLEAN
- [in] nowait_sw :BOOLEAN
- [out] status :INTEGER

**Details:**

- `from_file` specifies the device, name, and type of the file from which to copy. `from_file` can be specified using the wildcard (*) character. If no device is specified, the default device is used. You must specify both a name and type. However, these can be a wildcard (*) character.
- `to_file` specifies the device, name, and type of the file to which to copy. `to_file` can be specified using the wildcard (*) character. If no device is specified, the default device is used.
• *overwrite_sw* specifies that the file(s) should be overwritten if they exist.

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

• If the program is aborted during the copy, the copy will completed before aborting.

• If the device you are copying to becomes full during the copy, an error will be returned.

  **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: RENAME_FILE, DELETE_FILE Built-In Procedures

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

### A.4.47 COPY_PATH Built-In Procedure

**Purpose:** Copies a complete path, part of a path, or a path in reverse node order (including associated data), to another identical type path variable.

**Syntax:**

```plaintext
COPY_PATH (source_path, start_node, end_node, dest_path, status)
```

**Input/Output Parameters:**

[**in**] `source_path` : PATH

[**in**] `start_node` : INTEGER

[**in**] `end_node` : INTEGER

[**in**] `dest_path` : PATH

[**out**] `status` : INTEGER

%ENVIRONMENT Group : pathop

**Details:**

• *source_path* specifies the source path to copy from. This path can be a standard path or a user defined path.

• *start_node* specifies the number of the first node to copy. A value of 0 will copy the complete path, including header information. The *start_node* number must be between 0 and the highest node number in the source path. Otherwise, error status will be returned.
• *end_node* specifies the number of the last node to copy. A value of 0 will copy the complete path, including header information. The *end_node* number must be between 0 and the highest node number of the source path. Otherwise, error status will be returned.

• *dest_path* specifies the destination path to copy to. This path can be a standard path or a user defined path. However, the *dest_path* type must be identical to the *source_path* type. If they are not identical, an error status will be returned.

• *status* of 0 is returned if the parameters are valid and the COPY_PATH operation was successful. Non-zero status indicates the COPY_PATH operation was unsuccessful.

**Note** To copy a complete path from one path variable to another identical path variable, set the *start_node* and *end_node* parameters to 0 (zero).

An example of a *partial path* copy to a destination path.

Executing the COPY_PATH(P1, 2, 5, P2) command will copy node 2 through node 5 (inclusive) of path P1 to node 1 through 4 of Path P2, provided the path length of P1 is greater than or equal to 5. The destination path P2 will become a 4 node path. The original destination path is completely overwritten.

An example of a *source path copy in reverse order* to a destination path.

Executing the COPY_PATH(P1, 5, 2, P2) command will copy node 5 through node 2 (inclusive) of path P1 to node 1 through 4 of Path P2, provided the path length of P1 is greater than or equal to 5. The destination path P2 will become a 4 node path. The original destination path is completely overwritten.

Specifically, the above command will copy node 5 of P1 to node 1 of P2, node 4 of P1 to node 2 of P2, and so forth.

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

### A.4.48 COPY_QUEUE Built-In Procedure

**Purpose:** Copies one or more consecutive entries from a queue into an array of integers. The entries are not removed but are copied, starting with the oldest and proceeding to the newest, or until the output array, or integers, are full. A parameter specifies the number of entries at the head of the list (oldest entries) to be skipped.

**Syntax:** COPY_QUEUE(queue, queue_data, sequence_no, n_skip, out_data, n_got, status)

Input/Output Parameters:

[in] queue_t :QUEUE_TYPE

[in] queue_data :ARRAY OF INTEGER
[in] n_skip :INTEGER
[in] sequence_no :integer
[out] out_data :ARRAY OF INTEGER
[out] n_got :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :PBQMGR

Details:

• queue_t specifies the queue variable for the queue from which the values are to be read.
• queue_data specifies the array variable for the queue from which the values are to be read.
• sequence_no specifies the sequence number of the oldest entry to be copied. If the sequence_no is zero, the starting point for the copy is determined by the n_skip parameter.
• n_skip specifies the number of oldest entries to be skipped. A value of zero indicates to return the oldest entries.
• out_data is an integer array into which the values are to be copied; the size of the array is the maximum number of values returned.
• n_got is returned with the number of entries returned. This will be one of the following:
  — Zero if there are n_skip or fewer entries in the queue.
  — (queue_to n_entries_skip) if this is less than ARRAY_LEN(out_data)
  — ARRAY_LEN(out_data) if this is less than or equal to queue.n_entries - n_skip
• status is returned with zero

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

Example: The following example gets one “page” of a job queue and calls a routine, disp_queue, to display this. If there are no entries for the page, the routine returns FALSE; otherwise the routine returns TRUE.

COPY_QUEUE Built-In Procedure
PROGRAM copy_queue_x
%environment PBQMGR
VAR
  job_queue FROM global_vars: QUEUE_TYPE
  job_data FROM global_vars: ARRAY[100] OF INTEGER
ROUTINE disp_queue(data: ARRAY OF INTEGER;
  n_disp: INTEGER) FROM disp_prog
ROUTINE disp_page(data_array: ARRAY OF INTEGER;
VAR
  status: INTEGER
  n_got: INTEGER
BEGIN
  COPY_QUEUE(job_queue, job_data,
  (page_no - 1) * ARRAY_LEN(data_array), 0,
  data_array, n_got, status)
  IF (n_got = 0) THEN
    RETURN (FALSE)
  ELSE
    disp_queue(data_array, n_got)
    RETURN (TRUE)
  ENDIF
END disp_page
BEGIN
END copy_queue_x

A.4.49 COPY_TPE Built-In Procedure

**Purpose:** Copies one teach pendant program to another teach pendant program.

**Syntax:** COPY_TPE(from_prog, to_prog, overwrite_sw, status)

**Input/Output Parameters:**

[in] from_prog :STRING

[in] to_prog :STRING

[in] overwrite_sw :BOOLEAN

[out] status :INTEGER

%ENVIRONMENT Group :TPE

**Details:**

- *from_prog* specifies the teach pendant program name, without the .tp extension, to be copied.
- *to_prog* specifies the new teach pendant program name, without the .tp extension, that *from_prog* will be copied to.
- *overwrite_sw*, if set to TRUE, will automatically overwrite the *to_prog* if it already exists and it is not currently selected. If set to FALSE, the *to_prog* will not be overwritten if it already exists.
- *status* explains the status of the attempted operation. If not equal to 0, the copy did not occur.
See Also: CREATE_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.4.50 COS Built-In Function

**Purpose:** Returns the REAL cosine of the REAL angle argument, specified in degrees.

**Syntax:**
```
COS(angle) Function Return Type :REAL
```

**Input/Output Parameters:**
- [in] angle :REAL expression

**Details:**
- `angle` is an angle specified 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.4.51 CR Input/Output Item

**Purpose:** Can be used as a data item in a READ or WRITE statement to specify a carriage return.

**Syntax:**
```
CR
```

**Details:**
- When CR is used as a data item in a READ statement, it 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.
- When CR is used as a data item in a WRITE statement, it specifies that subsequent output to the same file will appear on a new line.

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

Example: Refer to the following sections for detailed program examples:
Section B.3 , "Saving Data to the Default Device" (SAVE_VR.KL)
Section B.4 , "Standard Routines" (ROUT_EX.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.7 , "Listing Files and Programs and Manipulating Strings" (LIST_EX.KL)
Section B.8 , "Generating and Moving Along a Hexagon Path" (GEN_HEX.KL)
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.4.52 CREATE_TPE Built-In Procedure

**Purpose:** Creates a teach pendant program of the specified name

**Syntax:** CREATE_TPE(prog_name, prog_type, status)

Input/Output Parameters:

[in] prog_name :STRING

[in] prog_type :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group :TPE

**Details:**

- *prog_name* specifies the name of the program to be created.
- *prog_type* specifies the type of the program to be created. The following constants are valid for program type: PT_MNE_UNDEF :TPE program of undefined sub type PT_MNE_JOB :TPE job PT_MNE_PROC :TPE process PT_MNE_MACRO :TPE macro
• *status* explains the status of the attempted operation. If it is not equal to 0, then an error occurred. Some of the possible errors are as follows:

7015 Specified program exist

9030 Program name is NULL

9031 Remove num from top of Program name

9032 Remove space from Program name

9036 Memory is not enough

9038 Invalid character in program name

• The program is created to reference all motion groups on the system. The program is created without any comment or any other program attributes. Once the program is created, SET_ATTR_PRG can be used to specify program attributes.

See Also: SET_ATTR_PRG Built-In Procedure

**A.4.53 CREATE_VAR Built-In Procedure**

**Purpose:** Creates the specified KAREL variable

**Syntax:** CREATE_VAR(var_prog_nam, var_nam, typ_prog_nam, type_nam, group_num, inner_dim, mid_dim, outer_dim, status, <mem_pool>)

**Input/Output Parameters :**

[in] var_prog_nam :STRING

[in] var_nam :STRING

[in] typ_prog_nam :STRING

[in] type_nam :STRING

[in] group_num :INTEGER

[in] inner_dim :INTEGER

[in] mid_dim :INTEGER

[in] outer_dim :INTEGER

[output] status :INTEGER
A–89

[en] mem_pool :INTEGER

%ENVIRONMENT Group :MEMO

Details:

• var_prog_nam specifies the program name that the variable should be created in. If var_prog_nam is ‘’, the default, which is the name of the program currently executing, is used.

• var_nam specifies the variable name that will be created.

• If a variable is to be created as a user-defined type, the user-defined type must already be created in the system. typ_prog_nam specifies the program name of the user-defined type. If typ_prog_nam is ‘’, the default, which is the name of the program currently executing, is used.

• type_nam specifies the type name of the variable to be created. The following type names are valid:

  'ARRAY OF BYTE'

  'ARRAY OF SHORT'

  'BOOLEAN'

  'CAM_SETUP'

  'CONFIG'

  'FILE'

  'INTEGER'

  'JOINTPOS'

  'JOINTPOS1'

  'JOINTPOS2'

  'JOINTPOS3'

  'JOINTPOS4'

  'JOINTPOS5'

  'JOINTPOS6'

  'JOINTPOS7'

  'JOINTPOS8'
'JOINTPOS9'

'MODEL'

'POSITION'

'REAL'

'STRING[n]', where n is the string length; the default is 12 if not specified.

'VECTOR'

'VIS_PROCESS'

'XYZWPR'

'XYZWPREXT'

Any other type names are considered user-defined types.

- **group_num** specifies the group number to be used for positional data types.
- **inner_dim** specifies the dimensions of the innermost array. For example, **inner_dim** = 30 for ARRAY[10,20,30] OF INTEGER. **inner_dim** should be set to 0 if the variable is not an array.
- **mid_dim** specifies the dimensions of the middle array. For example, **mid_dim** = 20 for ARRAY[10,20,30] OF INTEGER. **mid_dim** should be set to 0 if the variable is not a 2-D array.
- **outer_dim** specifies the dimensions of the outermost array. For example, **outer_dim** = 10 for ARRAY[10,20,30] OF INTEGER. **outer_dim** should be set to 0 if the variable is not a 3-D array.
- **status** explains the status of the attempted operation. If not equal to 0, then an error occurred.
- **mem_pool** is an optional parameter that specifies the memory pool from which the variable is created. If not specified, then the variable is created in DRAM which is temporary memory. The DRAM variable must be recreated at every power up and the value is always reset to uninitialized.
  - If **mem_pool** = -1, then the variable is created in CMOS RAM which is permanent memory.

**See Also:** CLEAR, RENAME_VAR Built-In Procedures

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

### A.4.54  %CRTDEVICE

**Purpose:** Specifies that the CRT/KB device is the default device

**Syntax:**  %CRTDEVICE
Details:

• Specifies that the INPUT/OUTPUT window will be the default in the READ and WRITE statements instead of the TPDISPLAY window.

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

A.4.55 CURJPOS Built-In Function

**Purpose:** Returns the current joint position of the tool center point (TCP) for the specified group of axes, even if one of the axes is in an overtravel

**Syntax:** CURJPOS(axs_lim_mask, ovr_trv_mask <,group_no>)

Function Return Type :JOINTPOS

Input/Output Parameters :

- [out] axs_lim_mask :INTEGER
- [out] ovr_trv_mask :INTEGER
- [in] group_no :INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

- 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.
- axs_lim_mask specifies which axes are outside the axis limits.
- ovr_trv_mask specifies which axes are in overtravel.

**Note** axis_limit_mask and ovr_trv_mask are not available in this release and can be set to 0.

See Also: CURPOS Built-In Function, Chapter 8 POSITION DATA

Example: The following example gets the current joint position of the robot.

**CURJPOS Built-In Function**

PROGRAM getpos
VAR
  jnt: JOINTPOS
A.4.56 CURPOS Built-In Function

**Purpose:** Returns the current Cartesian position of the tool center point (TCP) for the specified group of axes even if one of the axes is in an overtravel

**Syntax:** CURPOS(axis_limit_mask, ovr_trv_mask <,group_no>)

Function Return Type :XYZWPREXT

**Input/Output Parameters:**

[out] axis_limit_mask :INTEGER

[out] ovr_trv_mask :INTEGER

[in] group_no :INTEGER

%ENVIRONMENT Group :SYSTEM

**Details:**

- 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.
- The group must be kinematic.
- Returns the current position of the tool center point (TCP) relative to the current value of the system variable $UFRAME for the specified group.
- `axis_limit_mask` specifies which axes are outside the axis limits.
- `ovr_trv_mask` specifies which axes are in overtravel.

**Note** `axis_limit_mask` and `ovr_trv_mask` are not available in this release and will be ignored if set.

**See Also:** Chapter 8 *POSITION DATA*

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

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

Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)
A.4.57 CURR_PROG Built-In Function

**Purpose:** Returns the name of the program currently being executed

**Syntax:** CURR_PROG

Function Return Type :STRING[12]

%ENVIRONMENT Group :BYNAM

**Details:**
- The variable assigned to CURR_PROG must be declared with a string variable length ≥12

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

A.5 - D - KAREL LANGUAGE DESCRIPTION

A.5.1 DAQ_CHECKP Built-In Procedure

**Purpose:** To check the status of a pipe and the number of bytes available to be read from the pipe.

**Syntax:** DAQ_CHECKP(pipe_num, pipe_stat, bytes_avail)

**Input/Output Parameters :**

[in] pipe_num :INTEGER

[out] pipe_stat :INTEGER

[out] bytesavail :INTEGER

**Details:**
- *pipe_num* is the number of the pipe (1 - 5) to check.
- *pipe_stat* is the status of the pipe returned. The status is a combination of the following flags:
  - DAQ_PIPREG is when the pipe is registered (value = 1).
  - DAQ_ACTIVE is when the pipe is active, i.e., has been started (value = 2).
  - DAQ_CREATD is when the pipe is created (value = 4).
  - DAQ_SNAPSH is when the pipe is in snapshot mode (value = 8).
  - DAQ_1STRD is when the pipe has been read for the first time (value = 16).
— DAQ_OVFLOW is when the pipe is overflowed (value = 32).
— DAQ_FLUSH is when the pipe is being flushed (value = 64).

• bytes_avail is the number of bytes that are available to be read from the pipe.

**DAQ_CHECKP Built-In Procedure**

The pipe_stat returned parameter can be AND’ed with the above flag constants to determine whether the pipe is registered, is active, and so forth. For example, you must check to see if the pipe is active before writing to it.

**DAQ_CHECKP Built-In Procedure**

The DAQ_OVFLOW flag will never be set for the task that writes to the pipe when it calls DAQ_CHECKP. This flag applies only to tasks that read from the pipe.

**See Also:** DAQ_WRITE Built-In.

**Example:** Refer to the DAQ_WRITE example in the built-in function DAQ_WRITE.

**Note** This built-in is only available when DAQ or data monitor options are loaded.

### A.5.2 DAQ_REGPIPE Built-In Procedure

**Purpose:** To register a pipe for use in KAREL.

**Syntax:** DAQ_REGPIPE(pipe_num, mem_type, pipe_size, prog_name, var_name, pipe_name, stream_size, and status)

**Input/Output Parameters:**

- [in] pipe_num :INTEGER
- [in] mem_type :INTEGER
- [in] pipe_size :INTEGER
- [in] prog_name :STRING
- [in] var_name :STRING
[in] pipe_name :STRING

[in] stream_size :INTEGER

[out] status :INTEGER

Details:

- **pipe_num** is the number of the pipe (1-5) to be registered.
- **mem_type** allows you to allocate the memory to be used for the pipe. The following constants can be used:
  - DAQ_DRAM allows you to allocate DRAM memory.
  - DAQ_CMOS allows you to allocate CMOS memory.
- **pipe_size** is the size of the pipe, is expressed as the number of data records that it can hold. The data record size itself is determined by the data type of var_name.
- **prog_name** is the name of the program containing the variable to be used for writing to the pipe. If passed as an empty string, the name of the current program is used.
- **var_name** is the name of the variable that defines the data type to be used for writing to the pipe. Once registered, you can write any variable of this data type to the pipe.
- **pipe_name** is the name of the pipe file. For example, if the pipe name is passed as 'foo.dat', the pipe will be accessible using the file string 'PIP:FOO.DAT'. A unique file name with an extension is required even if the pipe is being used only for sending to the PC.
- **stream_size** is the number of records to automatically stream to an output file, if the pipe is started as a streamed pipe. A single write of the specified variable constitutes a single record in the pipe. If stream size is set to zero, the pipe will not automatically stream records to a file device; all data will be kept in the pipe until the pipe is read. Use stream_size to help optimize network loading when the pipe is used to send data to the PC. If it is zero or one, the monitoring task will send each data record as soon as it is seen in the pipe. If the number is two or more, the monitor will wait until there are that many data records in the pipe before sending them all to the PC. In this manner, the overhead of sending network packets can be minimized. Data will not stay in the pipe longer than the time specified by the FlushTime argument supplied with the FRCPipe.StartMonitor Method.
- **status** is the status of the attempted operation. If not 0, then an error occurred and the pipe was not registered.

See Also: DAQ_UNREG Built-In.

**DAQ_REGPIPE Built-In Procedure**

Pipes must be registered before they can be started and to which data is written. The registration operation tells the system how to configure the pipe when it is to be used. After it is registered, a pipe is configured to accept the writing of a certain amount of data per record, as governed by the
size of the specified variable. In order to change the configuration of a pipe, the pipe must first be unregistered using DAQ_UNREG, and then re-registered.

Example: The following example registers KAREL pipe 1 to write a variable in the program.

**DAQ_REGPIPE Built-In Procedure**

```
PROGRAM DAQREG
%ENVIRONMENT DAQ
CONST
   er_abort = 2
VAR
   status: INTEGER
   datavar: INTEGER
BEGIN
   -- Register pipe 1 DRAM as kldaq.dat
   -- It can hold 100 copies of the datavar variable
   -- before the pipe overflows
   DAQ_REGPIPE(1, DAQ_DRAM, 100, '', 'datavar', &
               'kldaq.dat', 0, status)
   IF status<>0 THEN
      POST_ERR(status,' ',0,er_abort)
   ENDIF
END DAQREG
```

**Note** This built-in is only available when DAQ or data monitor options are loaded.

### A.5.3 DAQ_START Built-In Procedure

**Purpose:** To activate a KAREL pipe for writing.

**Syntax:** DAQ_START(pipe_num, pipe_mode, stream_dev, status)

Input/Output Parameters :

- [in] pipe_num :INTEGER
- [in] pipe_mode :INTEGER
- [in] stream_dev :STRING
- [out] status :INTEGER
Details:

- `pipe_num` is the number of the pipe (1 - 5) to be started. The pipe must have been previously registered.

- `pipe_mode` is the output mode to be used for the pipe. The following constants are used:
  - `DAQ_SNAPSHOT` is the snapshot mode (each read of the pipe will result in all of the pipe’s contents).
  - `DAQ_STREAM` is the stream mode (each read from the same pipe file will result in data written since the previous read).

- `stream_dev` is the device to which records will be automatically streamed. This parameter is ignored if the stream size was set to 0 during registration.

- `status` is the status of the attempted operation. If not 0, then an error occurred and the pipe was not unregistered.

**See Also:** DAQ_REGPIPE Built-In and DAQ_STOP Built-In,

**DAQ_START Built-In Procedure**

This built-in call can be made either from the same task/program as the writing task, or from a separate activate/deactivate task. The writing task can lie dormant until the pipe is started, at which point it begins to write data.

**Example:** The following example starts KAREL pipe 1 in streaming mode.

```karel
PROGRAM PIPONOFF
  %ENVIRONMENT DAQ
  CONST
    er_abort = 2
  VAR
```
status: INTEGER
tpinput: STRING[1]
BEGIN
    -- prompt to turn on pipe
    WRITE('Press 1 to start pipe')
    READ (tpinput)
    IF tpinput = '1' THEN
        -- start pipe 1
        DAQ_START(1, DAQ_STREAM, 'RD:', status)
        IF status<>0 THEN
            POST_ERR(status,' ',0,er_abort)
        ELSE
            -- prompt to turn off pipe
            WRITE('Press any key to stop pipe')
            READ (tpinput)
            -- stop pipe 1
            DAQ_STOP(1, FALSE, status)
            IF status<>0 THEN
                POST_ERR(status,' ',0,er_abort)
            ENDIF
        ENDIF
    ENDIF
END PIPONOFF

Note This built-in is only available when DAQ or data monitor options are loaded.

A.5.4 DAQ_STOP Built-In Procedure

Purpose: To stop a KAREL pipe for writing.

Syntax : DAQ_STOP(pipe_num, force_off, status)

Input/Output Parameters :

[in] pipe_num :INTEGER
[in] force_off :BOOLEAN
[out] status :INTEGER

Details:

• pipe_num is the number of the pipe (1 - 5) to be stopped.
force_off occurs if TRUE force the pipe to be turned off, even if another application made a start request on the pipe. If set FALSE, if all start requests have been accounted for with stop requests, the pipe is turned off, else it remains on.

status is the status of the attempted operation. If not 0, then an error occurred and the pipe was not stopped.

See Also: DAQ_START Built-In.

**DAQ_STOP Built-In Procedure**

The start/stop mechanism on each pipe works on a reference count. The pipe is started on the first start request, and each subsequent start request is counted. If a stop request is received for the pipe, the count is decremented.

**DAQ_STOP Built-In Procedure**

If the pipe is not forced off, and the count is not zero, the pipe stays on. By setting the force_off flag to TRUE, the pipe is turned off regardless of the count. The count is reset.

**DAQ_STOP Built-In Procedure**

FRCPipe.StopMonitor method issued by a PC application is equivalent to a call to DAQ_STOP.

Example: Refer to the PIPONOFF example in the built-in function DAQ_START.

Note: This built-in is only available when DAQ or data monitor options are loaded.

### A.5.5 DAQ_UNREG Built-In Procedure

**Purpose:** To unregister a previously-registered KAREL pipe, so that it may be used for other data.

**Syntax:** DAQ_UNREG(pipe_num, status)

**Input/Output Parameters:**

[in] pipe_num :INTEGER

[out] status :INTEGER

**Details:**

- pipe_num is the number of the pipe (1 - 5) to be unregistered.
status is the status of the attempted operation. If not 0, then an error occurred and the pipe was not unregistered.

See Also: DAQ_REGPIPE Built-In.

**DAQ_UNREG Built-In Procedure**
Unregistering a pipe allows the pipe to be re-configured for a different data size, pipe size, pipe name, and so forth. You must un-register the pipe before re-registering using DAQ_REGPIPE.

Example: The following example unregisters KAREL pipe 1.

**DAQ_UNREG Built-In Procedure**

PROGRAM DAQUNREG
%ENVIRONMENT DAQ
CONST
  er_abort = 2
VAR
  status: INTEGER
BEGIN
  -- unregister pipe 1
  DAQ_UNREG(1, status)
  IF status<>0 THEN
    POST_ERR(status,' ',0,er_abort)
  ENDIF
END DAQUNREG

Note This built-in is only available when DAQ or data monitor options are loaded.

### A.5.6 DAQ_WRITE Built-In Procedure

**Purpose:** To write data to a KAREL pipe.

**Syntax:** DAQ_WRITE(pipe_num, prog_name, var_name, status)

Input/Output Parameters:

[in] pipe_num :INTEGER  
[in] prog_name :STRING  
[in] var_name :STRING
Details:

- `pipe_num` is the number of the pipe (1 - 5) to which data is written.
- `prog_name` is the name of the program containing the variable to be written. If passed as an empty string, the name of the current program is used.
- `var_name` is the name of the variable to be written.
- `status` is the status of the attempted operation. If not 0, then an error occurred and the data was not written.

See Also: DAQ_REGPIPE and DAQ_CHECKP.

**DAQ_WRITE Built-In Procedure**

You do not have to use the same variable for writing data to the pipe that was used to register the pipe. The only requirement is that the data type of the variable written matches the type of the variable used to register the pipe.

**Example:** The following example registers KAREL pipe 2 and writes to it when the pipe is active.

```karel
PROGRAM DAQWRITE
%ENVIRONMENT DAQ
%ENVIRONMENT SYSDEF
CONST
  er_abort = 2
TYPE
daq_data_t = STRUCTURE
  count: INTEGER
  dataval: INTEGER
ENDSTRUCTURE
VAR
  status: INTEGER
  pipestat: INTEGER
  numbytes: INTEGER
  datavar: daq_data_t
BEGIN
  -- register 10KB pipe 2 in DRAM as kldaq.dat
  DAQ_REGPIPE(2, DAQ_DRAM, 100, '', 'datavar', &
'kldaq.dat’, 1, status)
IF status<>0 THEN
    POST_ERR(status,’ ’,0,er_abort)
ENDIF
-- use DAQ_CHECKP to monitor status of pipe
DAQ_CHECKP(2, pipestat, numbytes)
datavar.count = 0
WHILE (pipestat AND DAQ_PIPREG) > 0 DO -- do while registered
    -- update data variable
    datavar.count = datavar.count + 1
    datavar.dataval = $FAST_CLOCK
    -- check if pipe is active
    IF (pipestat AND DAQ_ACTIVE) > 0 THEN
        -- write to pipe
        DAQ_WRITE(2, ‘’, datavar, status)
        IF status<>0 THEN
            POST_ERR(status,’ ’,0,er_abort)
        ENDIF
    ENDIF
    -- put in delay to reduce loading
    DELAY(200)
ENDWHILE
END DAQWRITE

Note This built-in is only available when DAQ or data monitor options are loaded.

A.5.7 %DEFGROUP Translator Directive

Purpose: Specifies the default motion group to be used by the translator

Syntax: %DEFGROUP = n

Details:

- n is the number of the motion group.
- The range is 1 to the number of groups on the controller.
- If %DEFGROUP is not specified, group 1 is used.

A.5.8 DEF_SCREEN Built-In Procedure

Purpose: Defines a screen
Syntax: DEF_SCREEN(screen_name, disp_dev_name, status)

Input/Output Parameters:

[in] screen_name : STRING
[in] disp_dev_name : STRING
[out] status : INTEGER

%ENVIRONMENT Group : UIF

Details:

• Define a screen, associated with a specified display device, to which windows could be attached and be activated (displayed).

• screen_name must be a unique, valid name (string), one to four characters long.

• disp_dev_name must be one of the display devices already defined, otherwise an error is returned. The following are the predefined display devices:

  'TP' Teach Pendant Device 'CRT' CRT/KB Device

• status explains the status of the attempted operation. (If not equal to 0, then an error occurred.)

See Also: ACT_SCREEN Built-In Procedure

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

A.5.9 DEF_WINDOW Built-In Procedure

Purpose: Define a window

Syntax: DEF_WINDOW(window_name, n_rows, n_cols, options, status)

Input/Output Parameters:

[in] window_name : STRING
[in] n_rows : INTEGER
[in] n_cols : INTEGER
[in] options : INTEGER
[out] status : INTEGER
%ENVIRONMENT Group :PBCORE

Details:

- Define a window that can be attached subsequently to a screen, have files opened to it, be written or have input echoed to it, and have information dynamically displayed in it.

- `window_name` must be a valid name string, one to four characters long, and must not duplicate a window with the same name.

- `n_rows` and `n_cols` specify the size of the window in standard-sized characters. Any line containing double-wide or double-wide-double-high characters will contain only half this many characters. The first row and column begin at 1.

- `options` must be one of the following:
  
  0 :No option

  `wd_com_cursr` :Common cursor

  `wd_scrolled` :Vertical scrolling

  `wd_com_cursr + wd_scrolled` :Common cursor + Vertical scrolling

- If common cursor is specified, wherever a write leaves the cursor is where the next write will go, regardless of the file variable used. Also, any display attributes set for any file variable associated with this window will apply to all file variables associated with the window. If this is not specified, the cursor position and display attributes (except character size attributes, which always apply to the current line of a window) are maintained separately for each file variable open to the window. The common-cursor attribute is useful for windows that can be written to by more than one task and where these writes are to appear end-to-end. An example might be a log display.

- If vertical scrolling is specified and a line-feed, new-line, or index-down character is received and the cursor is in the bottom line of the window, all lines except the top line are moved up and the bottom line is cleared. If an index-up character is written, all lines except the bottom line are moved down and the top line is cleared. If this is not specified, the bottom or top line is cleared, but the rest of the window is unaffected.

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

See Also: ATT_WINDOW_D, ATT_WINDOW_S Built-In Procedures

A.5.10 %DELAY Translator Directive

Purpose: Sets the amount of time program execution will be delayed every 250 milliseconds. Each program is delayed 8ms every 250ms by default. This allows the CPU to perform other functions such as servicing the CRT/KB and Teach Pendant user interfaces. %DELAY provides a way to change from the default and allow more CPU for system tasks such as user interface.
Syntax: \%DELAY = n

Details:

- \( n \) is the delay time in milliseconds.
- The default delay time is 8 ms, if no \%DELAY is specified.
- If \( n \) is set to 0, the program will attempt to use 100% of the available CPU time. This could result in the teach pendant and CRT/KB becoming inoperative since their priority is lower. A delay of 0 is acceptable if the program will be waiting for motion or I/O.
- While one program is being displayed, other programs are prohibited from executing. Interrupt routines (routines called from condition handlers) will also be delayed.
- Very large delay values will severely inhibit the running of all programs.
- To delay one program in favor of another, use the \%DELAY Statement instead of \%DELAY.

### A.5.11 DELAY Statement

**Purpose:** Causes execution of the program to be suspended for a specified number of milliseconds.

**Syntax:** DELAY time_in_ms

where:

time_in_ms : an INTEGER expression

**Details:**

- If motion is active at the time of the delay, the motion continues.
- \( time_{in\_ms} \) is the time in milliseconds. The actual delay will be from zero to \$SCR.\$cond_time milliseconds less than the rounded time.
- A time specification of zero has no effect.
- If a program is paused while a delay is in progress, the delay will continue to be timed.
- If the delay time in a paused program expires while the program is still paused, the program, upon resuming and with no further delay, will continue execution with the statement following the delay. Otherwise, upon resumption, the program will finish the delay time before continuing execution.
- Aborting a program, or issuing RUN from the CRT/KB when a program is paused, terminates any delays in progress.
- While a program is awaiting expiration of a delay, the KCL> SHOW TASK command will show a hold of DELAY.
- A time value greater than one day or less than zero will cause the program to be aborted with an error.
A.5.12 DELETE_FILE Built-In Procedure

**Purpose:** Deletes the specified file

**Syntax:**

```
DELETE_FILE(file_spec, nowait_sw, status)
```

**Input/Output Parameters:**

- [in] file_spec :STRING
- [in] nowait_sw :BOOLEAN
- [out] status :INTEGER

**Details:**

- `file_spec` specifies the device, name, and type of the file to delete. `file_spec` can be specified using the wildcard (*) character. If no device name is specified, the default device is used.
- 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 is complete. If you have time critical condition handlers in the 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, RENAME_FILE Built-In Procedures

**Example:** Refer to Section B.3, "Saving Data to the Default Device" (SAVE_VRS.KL), for a detailed program example.
A.5.13 DELETE_NODE Built-In Procedure

**Purpose:** Deletes a path node from a PATH

**Syntax:** DELETE_NODE(path_var, node_num, status)

**Input/Output Parameters:**
- [in] path_var :PATH
- [in] node_num :INTEGER
- [out] status :INTEGER

**Details:**
- `node_num` specifies the node to be deleted from the PATH specified by `path_var`.
- All nodes past the deleted node will be renumbered.
- `node_num` must be in the range from one to PATH_LEN(path_var). If it is outside this range, the status is returned with an error.
- `status` explains the status of the attempted operation. If not equal to 0, then an error occurred.

**See Also:** APPEND_NODE, INSERT_NODE Built-In Procedures

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

A.5.14 DELETE_QUEUE Built-In Procedure

**Purpose:** Deletes an entry from a queue

**Syntax:** DELETE_QUEUE(sequence_no, queue, queue_data, status)

**Input/Output Parameters:**
- [in] sequence_no :INTEGER
- [in,out] queue_t :QUEUE_TYPE
- [in,out] queue_data :ARRAY OF INTEGER
- [out] status :INTEGER
Details:

- Use COPY_QUEUE to get a list of the sequence numbers.
- `sequence_no` specifies the sequence number of the entry to be deleted. Use COPY_QUEUE to get a list of the sequence numbers.
- `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` returns 61003, “Bad sequence no,” if the specified sequence number is not in the queue.

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

### A.5.15 DEL_INST_TPE Built-In Procedure

**Purpose:** Deletes the specified instruction in the specified teach pendant program

**Syntax:** DEL_INST_TPE(open_id, lin_num, status)

**Input/Output Parameters:**

[in] `open_id` :INTEGER
[in] `lin_num` :INTEGER
[out] `status` :INTEGER

%ENVIRONMENT Group :TPE

**Details:**

- `open_id` specifies the opened teach pendant program. A program must be opened with read/write access, using the OPEN_TPE built-in, before calling the DEL_INST_TPE built-in.
- `lin_num` specifies the line number of the instruction to be deleted.
- `status` explains the status of the attempted operation. If not equal to 0, then an error has occurred.

See Also: CREATE_TPE, CLOSE_TPE, COPY_TPE, OPEN_TPE, SELECT_TPE Built-In Procedures
A.5.16 DET_WINDOW Built-In Procedure

**Purpose:** Detach a window from a screen

**Syntax:** DET_WINDOW(window_name, screen_name, status)

**Input/Output Parameters:**
- [in] window_name : STRING
- [in] screen_name : STRING
- [out] status : INTEGER

**Details:**
- Removes the specified window from the specified screen.
- `window_name` and `screen_name` must be valid and already defined.
- The areas of other window(s) hidden by this window are redisplayed. Any area occupied by this window and not by any other window is cleared.
- An error occurs if the window is not attached to the screen.
- `status` explains the status of the attempted operation. If not equal to 0, then an error occurred.

**See Also:** DEF_WINDOW, ATT_WINDOW_S, ATT_WINDOW_D Built-In Procedures

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

A.5.17 DISABLE CONDITION Action

**Purpose:** Used within a condition handler to disable the specified condition handler

**Syntax:** DISABLE CONDITION [cond_hand_no]

**Details:**
- If the condition handler is not defined, DISABLE CONDITION has no effect.
• If the condition handler is defined but not currently enabled, DISABLE CONDITION has no effect.

• When a condition handler is disabled, its conditions are not tested. Thus, if it is activated again, the conditions must be satisfied after the activation.

• Use the ENABLE CONDITION statement or action to reactivate a condition handler that has been disabled.

• $\text{cond\_hand\_no}$ must be in the range of 1-1000. Otherwise, the program will be aborted with an error.

See Also: Chapter 6, “Condition Handlers,” for more information on using DISABLE CONDITION in condition handlers

Example: The following example disables condition handler number 2 when condition number 1 is triggered.

DISABLE CONDITION Action

CONDITION[1] :
    WHEN EVENT[1] DO
    DISABLE CONDITION[2]
ENDCONDITION

A.5.18 DISABLE CONDITION Statement

Purpose: Disables the specified condition handler

Syntax : DISABLE CONDITION [cond\_hand\_no]

where:

cond\_hand\_no :an INTEGER expression

Details:

• If the condition handler is not defined, DISABLE CONDITION has no effect.

• If the condition handler is defined but not currently enabled, DISABLE CONDITION has no effect.

• When a condition handler is disabled, its conditions are not tested. Thus, if it is activated again, the conditions must be satisfied after the activation.

• Use the ENABLE CONDITION statement or action to reactivate a condition handler that has been disabled.

• $\text{cond\_hand\_no}$ must be in the range of 1-1000. Otherwise, the program will be aborted with an error.
See Also: Chapter 6 CONDITION HANDLERS, for more information on using DISABLE CONDITION in condition handlers, Appendix E, “Syntax Diagrams,” for additional syntax information.

Example: The following example allows the operator to choose whether or not to see count.

DISABLE CONDITION Statement

PROGRAM p_disable
VAR
  count : INTEGER
  answer : STRING[1]
ROUTINE showcount
BEGIN
  WRITE ('count = ',count::10,CR)
END showcount
BEGIN
  CONDITION[1]:
      showcount
    ENABLE CONDITION[1]
ENDCONDITION
ENABLE CONDITION[1]
count = 0
WRITE ('do you want to see count?')
READ (answer,CR)
IF answer = 'n'
  THEN DISABLE CONDITION[1] -- Disables condition[1]
  ENDIF -- Count will not be shown
FOR count = 1 TO 13 DO
  SIGNAL EVENT[1]
ENDFOR
END p_disable

A.5.19 DISCONNECT TIMER Statement

Purpose: Stops updating a clock variable previously connected as a timer

Syntax: DISCONNECT TIMER timer_var

where:

timer_var : a static, user-defined INTEGER variable

Details:
• If `timer_var` is not currently connected as a timer, the DISCONNECT TIMER statement has no effect.

• If `timer_var` is a system or local variable, the program will not be translated.

**See Also:** Appendix E, “Syntax Diagrams,” for additional syntax information, CONNECT TIMER Statement

**Example:** The following example moves the TCP to the initial position in PR[1], sets the INTEGER variable `timevar` to 0 and connects the timer. After moving to the destination position in PR[2], the timer is disconnected.

**DISCONNECT TIMER Statement**

```plaintext
move_to_pr1 – Call TP program to move to PR[1]
timevar = 0
CONNECT TIMER TO timevar

move_to_pr2 – Call TP program to move to PR[2]
DISCONNECT TIMER timevar
```

### A.5.20 DISCTRL_ALPH Built_In Procedure

**Purpose:** Displays and controls alphanumeric string entry in a specified window.

**Syntax:** DISCTRL_ALPH(window_name, row, col, str, dict_name, dict_ele, term_char, status)

**Input/Output Parameters:**

- `[in] window_name :STRING`
- `[in] row :INTEGER`
- `[in] col :INTEGER`
- `[in,out] str :STRING`
- `[in] dict_name :STRING`
- `[in] dict_ele :INTEGER`
- `[out] term_char :INTEGER`
- `[out] status :INTEGER`

**%ENVIRONMENT Group :** UIF
Details:

- `window_name` identifies the window where the `str` is currently displayed. See also Section 7.10.1 or Section 7.10.2 for a listing of windows that may be used for `window_name`.
- `row` specifies the row number where the `str` is displayed.
- `col` specifies the column number where the `str` is displayed.
- `str` specifies the KAREL string to be modified, which is currently displayed on the `window_name` at position `row` and `col`.
- `dict_name` specifies the dictionary that contains the words that can be entered. `dict_name` can also be set to one of the following predefined values.
  - 'PROG' : program name entry
  - 'COMM' : comment entry
- `dict_ele` specifies the dictionary element number for the words. `dict_ele` can contain a maximum of 5 lines with no "&new_line" accepted on the last line. See the example below.
- If a predefined value for `dict_name` is used, then `dict_ele` is ignored.
- `term_char` receives a code indicating the character that terminated the menu. The code for key terminating conditions are defined in the include file FR:KLEVKEYS.KL. The following predefined constants are keys that are normally returned:
  - `ky_enter`
  - `ky_prev`
  - `ky_new_menu`
- DISCTRL_ALPH will display and control string entry from the teach pendant device. To display and control string entry from the CRT/KB device, you must create an INTEGER variable, `device_stat`, and set it to `crt_panel`. To set control to the teach pendant device, set `device_stat` to `tp_panel`. Refer to the example below.
- `status` explains the status of an attempted operation. If not equal to 0, then an error occurred.

**Note** DISCTRL_ALPH will only display and control string entry if the USER or USER2 menu is the selected menu. Therefore, use `FORCE_SPMENU(device_stat, SPI_TPUUSER, 1)` before calling DISCTRL_ALPH to force the USER menu.

**See Also:** ACT_SCREEN, DISCTRL_LIST Built-In Procedures

**Example:** Refer to Section B.13, "Using the DISCTRL_ALPHA Built-in" (DCLAP_EX.KL), for a detailed program example.
### A.5.21 DISCTRL_FORM Built_In Procedure

**Purpose:** Displays and controls a form on the teach pendant or CRT/KB screen

**Syntax:** DISCTRL_FORM(dict_name, ele_number, value_array, inactive_array, change_array, term_mask, def_item, term_char, status)

**Input/Output Parameters:**

- **[in] dict_name:** STRING
- **[in] ele_number:** INTEGER
- **[in] value_array:** ARRAY OF STRING
- **[in] inactive_array:** ARRAY OF BOOLEAN
- **[out] change_array:** ARRAY OF BOOLEAN
- **[in] term_mask:** INTEGER
- **[in,out] def_item:** INTEGER
- **[out] term_char:** INTEGER
- **[out] status:** INTEGER

**Details:**

- **dict_name** is the four-character name of the dictionary containing the form.
- **ele_number** is the element number of the form.
- **value_array** is an array of variable names that corresponds to each edit or display only data item in the form. Each variable name can be specified as a '[@prog_name]var_name'.
  - [@prog_name] is the name of the program that contains the specified variable. If [@prog_name] is not specified, the current program being executed is used. '[@*SYSTEM*]' should be used for system variables.
  - var_name must refer to a static, global program variable.
  - var_name can contain node numbers, field names, and/or subscripts.
  - var_name can also specify a port variable with index. For example, 'DIN[1]'.
- **inactive_array** is an array of booleans that corresponds to each item in the form.
  - Each boolean defaults to FALSE, indicating it is active.
  - You can set any boolean to TRUE which will make that item inactive and non-selectable.
— The array size can be greater than or less than the number of items in the form.
— If an inactive_array is not used, then an array size of 1 can be used. The array does not need to be initialized.

• change_array is an array of booleans that corresponds to each edit or display only data item in the form.
  — If the corresponding value is set, then the boolean will be set to TRUE, otherwise it is set to FALSE. You do not need to initialize the array.
  — The array size can be greater than or less than the number of data items in the form.
  — If change_array is not used, an array size of 1 can be used.

• term_mask is a bit-wise mask indicating conditions that will terminate the form. This should be an OR of the constants defined in the include file klevkmsk.kl.

  kc_func_key — Function keys
  kc_enter_key — Enter and Return keys
  kc_prev_key — PREV key

If either a selectable item or a new menu is selected, the form will always terminate, regardless of term_mask.

• For version 6.20 and 6.21, def_item receives the item you want to be highlighted when the form is entered. def_item returns the item that was currently highlighted when the termination character was pressed.

• For version 6.22 and later, def_item receives the item you want to be highlighted when the form is entered. def_item is continuously updated while the form is displayed and contains the number of the item that is currently highlighted.

• 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_undef — No termination character was pressed
  ky_select — A selectable item was selected
  ky_new_menu — A new menu was selected
  ky_f1 — Function key 1 was selected
  ky_f2 — Function key 2 was selected
  ky_f3 — Function key 3 was selected
  ky_f4 — Function key 4 was selected
ky_f5 — Function key 5 was selected
ky_f6 — Function key 6 was selected
ky_f7 — Function key 7 was selected
ky_f8 — Function key 8 was selected
ky_f9 — Function key 9 was selected
ky_f10 — Function key 10 was selected

• DISCTRL_FORM will display the form on the teach pendant device. To display the form on the CRT/KB device, you must create an INTEGER variable, device_stat, and set it to crt_panel. To set control to the teach pendant device, set device_stat to tp_panel.

• status explains the status of the attempted operation. If status returns a value other than 0, an error has occurred.

Note DISCTRL_FORM will only display the form if the USER2 menu is the selected menu. Therefore, use FORCE_SPMENU(device_stat, SPI_TPUSER2, 1) before calling DISCTRL_FORM to force the USER2 menu.

See Also: Chapter 10 DICTIONARIES AND FORMS, for more details and examples.

A.5.22 DISCTRL_LIST Built-In Procedure

Purpose: Displays and controls cursor movement and selection in a list in a specified window

Syntax: DISCTRL_LIST(file_var, display_data, list_data, action, status)

Input/Output Parameters:

[in] file_var :FILE

[in,out] display_data :DISP_DAT_T

[in] list_data :ARRAY OF STRING

[in] action :INTEGER

[ out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

• file_var must be opened to the window where the list data is to appear.
• *display_data* is used to display the list. Refer to the DISP_DAT_T data type for details.
• *list_data* contains the list of data to display.
• *action* must be one of the following:
  
  dc_disp : Positions cursor as defined in display_data  
  
  dc_up : Moves cursor up one row  
  
  dc_dn : Moves cursor down one row  
  
  dc_lf : Moves cursor left one field  
  
  dc_rt : Moves cursor right one field  
  
  • *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.
• Using DISCTRL_FORM is the preferred method for displaying and controlling information in a window.

**See Also:** DISCTRL_FORM Built-In Procedure, Section 7.10.1, "User Menu on the Teach Pendant," Section 7.10.2, "User Menu on the CRT/KB," 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.

---

**Caution**

The input parameters are not checked for validity. You must make sure the input parameters are valid; otherwise, the built-in might not work properly.

---

**A.5.23 DISCTRL_PLMN Built-In Procedure**

**Purpose:** Creates and controls cursor movement and selection in a pull-up menu

**Syntax:** DISCTRL_PLMN(dict_name, element_no, ftn_key_no, def_item, term_char, status)

*Input/Output Parameters :*

[in] dict_name :STRING

[in] element_no :INTEGER

[in] ftn_key_no :INTEGER

[in,out] def_item :INTEGER

---
[out] term_char :INTEGER

[out] status :INTEGER

%ENVIRONMENT Group : UIF

Details:

- The menu data in the dictionary consists of a list of enumerated values that are displayed and selected from a pull-up menu on the teach pendant device. A maximum of 9 values should be used. Each value is a string of up to 12 characters.

  A sequence of consecutive dictionary elements, starting with element_no, define the values. Each value must be put in a separate element, and must not end with &new_line. The characters are assigned the numeric values 1..9 in sequence. The last dictionary element must be "".

- dict_name specifies the name of the dictionary that contains the menu data.

- element_no is the element number of the first menu item within the dictionary.

- ftn_key_num is the function key where the pull-up menu should be displayed.

- def_item is the item that should be highlighted when the menu is entered. 1 specifies the first item. On return, def_item is the item that was currently highlighted when the termination character was pressed.

- term_char receives a code indicating the character that terminated the menu. The codes for key terminating conditions are defined in the include file FROM:KLEVKEYS.KL. Keys normally returned are pre-defined constants as follows:

  ky_enter — A menu item was selected

  ky_prev — A menu item was not selected

  ky_new_menu — A new menu was selected

  ky_f1

  ky_f2

  ky_f3

  ky_f4

  ky_f5

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

Example: In this example, dictionary file TPEXAMEG.TX is loaded as 'EXAM' on the controller. TPPLMN.KL calls DISCTRL_PLMN to display and process the pull-up menu above function key 3.
DISCTRL_PLMN Built-In Procedure

---------
TPEXAMEG.TX
---------

$subwin_menu
"Option 1"
$
"Option 2"
$
"Option 3"
$
"Option 4"
$
"Option 5"
$
"......"
---------
TPPLMN.KL
---------

PROGRAM tpplmn
%ENVIRONMENT uif
VAR
  def_item: INTEGER
  term_char: INTEGER
  status: INTEGER
BEGIN
  def_item = 1
  DISCTRL_PLMN('EXAM', 0, 3, def_item, term_char, status)
  IF term_char = ky_enter THEN
    WRITE (CR, def_item, ' was selected')
  ENDIF
END tpplmn

A.5.24 DISCTRL_SBMN Built-In Procedure

Purpose: Creates and controls cursor movement and selection in a sub-window menu

Syntax : DISCTRL_SBMN(dict_name, element_no, def_item, term_char, status)

Input/Output Parameters :

[in] dict_name :STRING

[in] element_no :INTEGER
[in,out] def_item :INTEGER
[out] term_char :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group : UIF

**Details:**

- The menu data in the dictionary consists of a list of enumerated values that are displayed and selected from the 'subm' subwindow on the Teach Pendant device. There can be up to 5 subwindow pages, for a maximum of 35 values. Each value is a string of up to 16 characters. If 4 or less enumerated values are used, then each string can be up to 40 characters.

A sequence of consecutive dictionary elements, starting with *element_no*, define the values. Each value must be put in a separate element, and must not end with &new_line. The characters are assigned the numeric values 1..35 in sequence. The last dictionary element must be "".

- *dict_name* specifies the name of the dictionary that contains the menu data.
- *element_no* is the element number of the first menu item within the dictionary.
- *def_item* is the item that should be highlighted when the menu is entered. 1 specifies the first item. On return, *def_item* is the item that was currently highlighted when the termination character was pressed.
- *term_char* receives a code indicating the character that terminated the menu. The codes for key terminating conditions are defined in the include file FROM:KLEVKEYS.KL. Keys normally returned are pre-defined constants as follows:

  - ky_enter — A menu item was selected
  - ky_prev — A menu item was not selected
  - ky_new_menu — A new menu was selected
  - ky_f1
  - ky_f2
  - ky_f3
  - ky_f4
  - ky_f5

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

**Example:** In this example, dictionary file TPEXAMEG.TX is loaded as 'EXAM' on the controller. TPSBMN.KL calls DISCTRL_SBMN to display and process the subwindow menu.
DISCTRL_SBMN Built-In Procedure

-------------------------------
TPEXAMEG.TX
-------------------------------
$subwin_menu
"Red"
$
"Blue"
$
"Green"
$
"Yellow"
$
"Brown"
$
"Pink"
$
"Mauve"
$
"Black"
$
"Lime"
$
"Lemon"
$
"Beige"
$
"Blue"
$
"Green"
$
"Yellow"
$
"Brown"
$
"\a"
-------------------------------
TPSBMN.KL
-------------------------------
PROGRAM tpsbmn
%ENVIRONMENT uif
VAR
  def_item: INTEGER
  term_char: INTEGER
  status: INTEGER
BEGIN
  def_item = 1
A.5.25 DISCTRL_TBL Built-In Procedure

**Purpose**: Displays and controls a table on the teach pendant

**Syntax**: DISCTRL_TBL(dict_name, ele_number, num_rows, num_columns, col_data, inact_array, change_array, def_item, term_char, term_mask, value_array, attach_wind, status)

Input/Output Parameters:

- [in] dict_name :STRING
- [in] ele_number :INTEGER
- [in] num_rows :INTEGER
- [in] num_columns :INTEGER
- [in] col_data :ARRAY OF COL_DESC_T
- [in] inact_array :ARRAY OF BOOLEAN
- [out] change_array :ARRAY OF BOOLEAN
- [in,out] def_item :INTEGER
- [out] term_char :INTEGER
- [in] term_mask :INTEGER
- [in] value_array :ARRAY OF STRING
- [in] attach_wind :BOOLEAN
- [out] status :INTEGER

%ENVIRONMENT Group : UIF

**Details**: 

DISCTRL_SBMN(‘EXAM’, 0, def_item, term_char, status)

IF term_char = ky_enter THEN
   WRITE (CR, def_item, ‘ was selected’)
ENDIF
END tpsbmnn
- **DISCTRL_TBL** is similar to the INIT_TBL and ACT_TBL Built-In routines and should be used if no special processing needs to be done with each keystroke.

- **dict_name** is the four-character name of the dictionary containing the table header.

- **ele_number** is the element number of the table header.

- **num_rows** is the number of rows in the table.

- **num_columns** is the number of columns in the table.

- **col_data** is an array of column descriptor structures, one for each column in the table. For a complete description, refer to the INIT_TBL Built-In routine in this appendix.

- **inact_array** is an array of booleans that corresponds to each column in the table.
  - You can set each boolean to TRUE which will make that column inactive. This means you cannot move the cursor to this column.
  - The array size can be less than or greater than the number of items in the table.
  - If inact_array is not used, then an array size of 1 can be used, and the array does not need to be initialized.

- **change_array** is a two-dimensional array of BOOLEANs that corresponds to formatted data items in the table.
  - If the corresponding value is set, then the boolean will be set to TRUE, otherwise it is set to FALSE. You do not need to initialize the array.
  - The array size can be less than or greater than the number of data items in the table.
  - If change_array is not used, then an array size of 1 can be used.

- **def_item** is the row containing the item you want to be highlighted when the table is entered. On return, def_item is the row containing the item that was currently highlighted when the termination character was pressed.

- **term_char** receives a code indicating the character or other condition that terminated the table.
  The codes for key terminating conditions are defined in the include file FROM:KLEVKEYS.KL.
  Keys normally returned are pre-defined constants as follows:

  ky_undef — No termination character was pressed

  ky_select — A selectable item as selected

  ky_new_menu — A new menu was selected

  ky_f1 — Function key 1 was selected

  ky_f2 — Function key 2 was selected

  ky_f3 — Function key 3 was selected

  ky_f4 — Function key 4 was selected
ky_f5 — Function key 5 was selected
ky_f6 — Function key 6 was selected
ky_f7 — Function key 7 was selected
ky_f8 — Function key 8 was selected
ky_f9 — Function key 9 was selected
ky_f10 — Function key 10 was selected

• term_mask is a bit-wise mask indicating conditions that will terminate the request. This should be an OR of the constants defined in the include file FROM: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)

• value_array is an array of variable names that corresponds to each column of data item in the table. 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 \[prog_name\] is not specified, then the current program being executed is used.
  — \var_name\ must refer to a static, global program variable.
  — \var_name\ can contain node numbers, field names, and/or subscripts.

• attach_wind should be set to 1 if the table manager window needs to be attached to the display device. If it is already attached, this parameter can be set to 0.

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

Example: Refer to the INIT_TBL Built-In routine for an example of setting up the dictionary text and initializing the parameters.
**A.5.26 DISMOUNT_DEV Built-In Procedure**

**Purpose:** Dismounts the specified device.

**Syntax:** `DISMOUNT_DEV (device, status)`

**Input/Output Parameters:**

- `[in] device : STRING`
- `[out] status : INTEGER`

**Details:**

- `device` specifies the device to be dismounted.
- `status` explains the status of the attempted operation. If not equal to 0, then an error occurred.

**See Also:** MOUNT_DEV, FORMAT_DEV Built-In Procedures

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

**A.5.27 DISP_DAT_T Data Type**

**Purpose:** Defines data type for use in DISCTRL_LIST Built-In

**Syntax:**

```krel
disp_dat_t = STRUCTURE
  win_end : ARRAY [4] OF SHORT
  curr_win : SHORT
  cursor_row : SHORT
  lins_per_pg : SHORT
  curs_st_col : ARRAY [10] OF SHORT
  curs_en_col : ARRAY [10] OF SHORT
```

---

A-125
curr_field : SHORT
last_field : SHORT
curr_it_num : SHORT
sob_it_num : SHORT
eob_it_num : SHORT
last_it_num : SHORT
menu_id : SHORT
ENDSTRUCTURE

Details:

- *disp_dat_t* can be used to display a list in four different windows. The list can contain up to 10 fields. Left and right arrows move between fields. Up and down arrows move within a field.
- *win_start* is the starting row for each window.
- *win_end* is the ending row for each window.
- *curr_win* defines the window to display. The count begins at zero (0 will display the first window).
- *cursor_row* is the current cursor row.
- *lins_per_pg* is the number of lines per page for paging up and down.
- *curs_st_col* is the cursor starting column for each field. The range is 0-39 for the teach pendant.
- *curs_en_col* is the cursor ending column for each field. The range is 0-39 for the teach pendant.
- *curr_field* is the current field in which the cursor is located. The count begins at zero (0 will set the cursor to the first field).
- *last_field* is the last field in the list.
- *curr_it_num* is the item number the cursor is on.
- *sob_it_num* is the item number of the first item in the array.
- *eob_it_num* is the item number of the last item in the array.
- *last_it_num* is the item number of the last item in the list.
- *menu_id* is the current menu identifier. Not implemented. May be left uninitialized.

**Example:** Refer to **Section B.12**, "Displaying a List From a Dictionary File" (DCLIST_EX.KL), for a detailed program example.
A.5.28 DOSFILE_INF Built-In Procedure

**Purpose:** Returns information for a device as a string in the value parameter.

**Syntax:** DOSFILE_INF(device, item, value_str, status)

Input/Output Parameters:

[in] device :STRING

[in] item :INTEGER

[out] value_str :STRING

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

**Details:**

- DOSFILE_INF is only available on R-30iB Controller
- `device` can be UD1:
- `item` is the constant of the data to be returned:
  - DF_MANUFNAME (Manufacture’s name)
  - DF_PRODNAME (Product name)
  - DF_SERIALNO (Serial number)
  - DF_PRODID (Product ID number as assigned by USB authority)
  - DF_VENDID (Vendor ID number as assigned by USB authority)
- `value_str` is the string where the data is returned.
- `status` explains the status of an attempted operation. If not equal to 0, then an error occurred.

**Example:** The following example gets information about the UD1 device.

DOSFILE_INF Built-In Procedure

```
program dfinfo
%NOLOCKGROUP
var
value: string[60]
status: integer
begin
value = ''
DOSFILE_INF('UD1:', DF_MANUFNAME, value, status)
```
A.6 - E - KAREL LANGUAGE DESCRIPTION

A.6.1 ENABLE CONDITION Action

**Purpose:** Enables the specified condition handler

**Syntax:** ENABLE CONDITION [cond_hand_no]

where:

cond_hand_no : an INTEGER expression

**Details:**

- **ENABLE CONDITION** has no effect when
  - The condition handler is not defined
  - The condition handler is defined but is already enabled
- **cond_hand_no** must be in the range of 1-1000. Otherwise, the program will be aborted with an error.
- When a condition handler is enabled, its conditions are tested each time the condition handler is scanned. If the conditions are satisfied, the corresponding actions are performed and the condition handler is deactivated. Issue an ENABLE CONDITION statement or action to reactivate it.
- Use the DISABLE CONDITION statement or action to deactivate a condition handler that has been enabled.
- Condition handlers are known only to the task which defines them. One task cannot enable another tasks condition.
See Also: DISABLE CONDITION Action, Chapter 6 CONDITION HANDLERS

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

### A.6.2 ENABLE CONDITION Statement

**Purpose:** Enables the specified condition handler

**Syntax:**

```
ENABLE CONDITION [cond_hand_no]
```

where:

cond_hand_no :an INTEGER expression

**Details:**

- ENABLE CONDITION has no effect when
  - The condition handler is not defined
  - The condition handler is defined but is already enabled
- `cond_hand_no` must be in the range of 1-1000. Otherwise, the program will be aborted with an error.
- When a condition handler is enabled, its conditions are tested each time the condition handler is scanned. If the conditions are satisfied, the corresponding actions are performed and the condition handler is deactivated. Issue an ENABLE CONDITION statement or action to reactivate it.
- Use the DISABLE CONDITION statement or action to deactivate a condition handler that has been enabled.
- Condition handlers are known only to the task which defines them. One task cannot enable another tasks condition.

See Also: DISABLE CONDITION Statement, Chapter 6 CONDITION HANDLERS, Appendix E, "Syntax Diagrams," for additional syntax information

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

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

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

### A.6.3 %ENVIRONMENT Translator Directive

**Purpose:** Loads environment file.
Syntax: %ENVIRONMENT path_name

- Used by the off-line translator to specify that the binary file, path_name.ev, should be loaded. Environment files contain definitions for predefined constants, ports, types, system variables, and built-ins.

- All .EV files are loaded upon installation of the controller software. Therefore, the controller’s translator will ignore %ENVIRONMENT statements since it already has the .EV files loaded.

- path_name can be one of the following:
  - BYNAM
  - CTDEF (allows program access to CRT/KB system variables)
  - ERRS
  - FDEV
  - FLBT
  - IOSETUP
  - KCLOP
  - MEMO
  - MIR
  - MOTN
  - MULTI
  - PATHOP
  - PBCORE
  - PBQMG
  - REGOPE
  - STRNG
  - SYSDEF (allows program access to most system variables)
  - SYSTEM
  - TIM
  - TPE
  - TRANS
  - UIF
  - VECTR

- If no %ENVIRONMENT statements are specified in your KAREL program, the off-line translator will load all the .EV files specified in TRMNEG.TX. The translator must be able to find these files in the current directory or in one of the PATH directories.
If at least one %ENVIRONMENT statement is specified, the off-line translator will only load the files you specify in your KAREL program. Specifying your own %ENVIRONMENT statements will reduce the amount of memory required to translate and will be faster, especially if you do not require system variables since SYSDEF.EV is the largest file.

SYSTEM.EV and PBCORE.EV are automatically loaded by the translator and should not be specified in your KAREL program. The off-line translator will print the message "Continuing without system defined symbols" if it cannot find SYSTEM.EV. Do not ignore this message. Make sure the SYSTEM.EV file is loaded.

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

Section B.3, "Saving Data to the Default Device" (SAVE_VR.KL)
Section B.6, "Path Variables and Condition Handlers Program" (PTH_MOVE.KL)
Section B.14, "Applying Offsets to a Copied Teach Pendant Program" (CPY_TP.KL)

### A.6.4 ERR_DATA Built-In Procedure

**Purpose:** Reads the requested error from the error history and returns the error

**Syntax:**

```
ERR_DATA(seq_num, error_code, error_string, cause_code, cause_string, time_int, severity, prog_nam)
```

**Input/Output Parameters:**

- [in,out] seq_num :INTEGER
- [out] error_code :INTEGER
- [out] error_string :STRING
- [out] cause_code :INTEGER
- [out] cause_string :STRING
- [out] time_int :INTEGER
- [out] severity :INTEGER
- [out] prog_nam :STRING

%ENVIRONMENT Group :ERRS
• \textit{seq\_num} is the sequence number of the previous error requested. \textit{seq\_num} should be set to 0 if the oldest error in the history is desired. \textit{seq\_num} should be set to MAXINT if the most recent error is desired.

• \textit{seq\_num} is set to the sequence number of the error that is returned.
  — If the initial value of \textit{seq\_num} is greater than the sequence number of the newest error in the log, \textit{seq\_num} is returned as zero and no other data is returned.
  — If the initial value of \textit{seq\_num} is less than the sequence number of the oldest error in the log, the oldest error is returned.

• \textit{error\_code} returns the error code and \textit{error\_string} returns the error message. \textit{error\_string} must be at least 40 characters long or the program will abort with an error.

• \textit{cause\_code} returns the reason code if it exists and \textit{cause\_string} returns the message. \textit{cause\_string} must be at least 40 characters long or the program will abort with an error.

• \textit{error\_code} and \textit{cause\_code} are in the following format:

\[ ffccc \]

where \textit{ff} represents the facility code of the error.

\[ cccc \]

c\textit{ccc} represents the error code within the specified facility.

Refer to Chapter 6, "Condition Handlers," for the error facility codes.

• \textit{time\_int} returns the time that \textit{error\_code} was posted. The time is in encoded format, and \text{CNV\_TIME\_STR} Built-In should be used to get the date-and-time string.

• \textit{severity} returns one of the following \textit{error\_codes}: 0 : WARNING 1 : PAUSE 2 : ABORT

• If the error occurs in the execution of a program, \textit{prog\_nam} specifies the name of the program in which the error occurred.

• If the error is posted by POST\_ERR, or if the error is not associated with a particular program (e.g., E-STOP), \textit{prog\_nam} is returned as ‘”’.

• Calling ERR\_DATA immediately after POST\_ERR may not return the error just posted since POST\_ERR returns before the error is actually in the error log.

\textbf{See Also:} POST\_ERR Built-In Procedure

\section*{A.6.5 ERROR Condition}

\textbf{Purpose:} Specifies an error as a condition

\textbf{Syntax :} \text{ERROR[n]}

where:
n : an INTEGER expression or asterisk (*)

Details:

- If n is an INTEGER, it represents an error code number. The condition is satisfied when the specified error occurs.
- If n is an asterisk (*), it represents a wildcard. The condition is satisfied when any error occurs.
- The condition is an event condition, meaning it is satisfied only for the scan performed when the error was detected. The error is not remembered on subsequent scans.

See Also: Chapter 6 CONDITION HANDLERS, for more information on using conditions. The appropriate application-specific FANUC Robotics Setup and Operations Manual for a list of all error codes

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

### A.6.6 EVAL Clause

**Purpose:** Allows expressions to be evaluated in a condition handler definition

**Syntax:** EVAL(expression)

where:

expression : a valid KAREL expression

**Details:**

- expression is evaluated when the condition handler is defined, rather than dynamically during scanning.
- expression can be any valid expression that does not contain a function call.

See Also: Chapter 6 CONDITION HANDLERS, for more information on using conditions

Example: The following example causes delay until AIN[ force ] is greater than the evaluated expression (10 * f_scale).

```
EVAL Clause
WRITE ('Enter force scale: ')  
READ (f_scale)  
REPEAT  
   DELAY(1000)  
UNTIL AIN[force] > EVAL(10 * f_scale)
```
A.6.7 EVENT Condition

**Purpose:** Specifies the number of an event that satisfies a condition when a SIGNAL EVENT statement or action with that event number is executed.

**Syntax:** EVENT[event_no]

where:

event_no : is an INTEGER expression

**Details:**

- Events can be used as user-defined event codes that become TRUE when signaled.
- The SIGNAL EVENT statement or action is used to signal that an event has occurred.
- event_no must be in the range of -32768 to 32767.

**See Also:** SIGNAL EVENT Action, and CONDITION or SIGNAL EVENT Statement

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

A.6.8 EXP Built-In Function

**Purpose:** Returns a REAL value equal to e (approximately 2.71828) raised to the power specified by a REAL argument.

**Syntax:** EXP(x)

Function Return Type : REAL

Input/Output Parameters :

[in] x : REAL

%ENVIRONMENT Group : SYSTEM

**Details:**

- EXP returns e (base of the natural logarithm) raised to the power x.
- x must be less than 80. Otherwise, the program will be paused with an error.

**Example:** The following example uses the EXP Built-In to evaluate the exponent of the expression (-6.44 + timevar/(timevar + 20)).
EXP Built-In Function

WRITE (CR, ‘Enter time needed for move:’)  
READ (timevar)  
distance = timevar *  
    EXP(-6.44 + timevar/(timevar + 20))  
WRITE (CR, CR, ‘Distance for move:’, distance::10::3)

A.7  - F - KAREL LANGUAGE DESCRIPTION

A.7.1 FILE Data Type

Purpose: Defines a variable as FILE data type

Syntax : file

Details:

• FILE allows you to declare a static variable as a file.
• You must use a FILE variable in OPEN FILE, READ, WRITE, CANCEL FILE, and CLOSE FILE statements.
• You can pass a FILE variable as a parameter to a routine.
• Several built-in routines require a FILE variable as a parameter, such as BYTES_LEFT, CLR_IO_STAT, GET_FILE_POS, IO_STATUS, SET_FILE_POS.
• FILE variables have these restrictions:
  — FILE variables must be a static variable.
  — FILE variables are never saved.
  — FILE variables cannot be function return values.
  — FILE types are not allowed in structures, but are allowed in arrays.
  — No other use of this variable data type, including assignment to one another, is permitted.

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)

A.7.2 FILE_LIST Built-In Procedure

Purpose: Generates a list of files with the specified name and type on the specified device.
Syntax: FILE_LIST(file_spec, n_skip, format, ary_nam, n_files, status)

Input/Output Parameters:

[in] file_spec :STRING
[in] n_skip :INTEGER
[in] format :INTEGER
[out] ary_nam :ARRAY of STRING
[out] n_files :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :BYNAM

Details:

• file_spec specifies the device, name, and type of the list of files to be found. file_spec can be specified using the wildcard (*) character.

• n_skip is used when more files exist than the declared length of ary_nam. Set n_skip to 0 the first time you use FILE_LIST. If ary_nam is completely filled with variable names, copy the array to another ARRAY of STRINGs and execute the FILE_LIST again with n_skip equal to n_files. The second call to FILE_LIST will skip the files found in the first pass and only locate the remaining files.

• format specifies the format of the file name and file type. The following values are valid for format:

  1 file_name only, no blanks 2 file_type only, no blanks 3 file_name.file_type , no blanks 4 filename.ext size date time The total length is 40 characters.

    — The file_name starts with character 1.
    — The file_type (extension) starts with character 10.
    — The size starts with character 21.
    — The date starts with character 26.
    — The time starts with character 36.

    Date and time are only returned if the device supports time stamping; otherwise just the filename.ext size is stored.

• ary_nam is an ARRAY of STRINGs to store the file names. If the string length of ary_nam is not large enough to store the formatted information, an error will be returned.

• n_files is the number of files stored in ary_name.
• *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

**See Also:** VAR_LIST, PROG_LIST Built-In Procedures

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

### A.7.3 FOR...ENDFOR Statement

**Purpose:** Looping construct based on an INTEGER counter

**Syntax:** FOR count = initial || TO | DOWNTO || final

DO{stmt} ENDFOR

where:

[in]count :INTEGER variable

[in]initial :INTEGER expression

[in]final :INTEGER expression

[in]stmt :executable KAREL statement

**Details:**

- Initially, *count* is set to the value of *initial* and *final* is evaluated. For each iteration, *count* is compared to *final*.
- If TO is used, *count* is incremented for each loop iteration.
- If DOWNTO is used, *count* is decremented for each loop iteration.
- If *count* is greater than *final* using TO, *stmt* is never executed.
- If *count* is less than *final* using DOWNTO, *stmt* is never executed on the first iteration.
- If the comparison does not fail on the first iteration, the FOR loop will be executed for the number of times that equals \( \text{ABS(final - initial)} + 1 \).
- If *final* = *initial*, the loop is executed once.
- *initial* is evaluated prior to entering the loop. Therefore, changing the values of *initial* and *final* during loop execution has no effect on the number of iterations performed.
- The value of *count* on exit from the loop is uninitialized.
- Never issue a GO TO statement in a FOR loop. If a GO TO statement causes the program to exit a FOR loop, the program might be aborted with a “Run time stack overflow” error.
• Never include a GO TO label in a FOR loop. Entering a FOR loop by a GO TO statement usually causes the program to be aborted with a “Run time stack underflow” error when the ENDFOR statement is encountered.

• The program will not be translated if count is a system variable or ARRAY element.

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

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.7 , "Listing Files and Programs and Manipulating Strings" (LIST_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)
Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.7.4 FORCE_SPMENU Built-In Procedure

Purpose: Forces the display of the specified menu

Syntax : FORCE_SPMENU(device_code, spmenu_id, screen_no)

Input/Output Parameters :

[in] device_code :INTEGER

[in] spmenu_id :INTEGER

[in] screen_no :INTEGER

%ENVIRONMENT Group :pbcore

Details:

• device_code specifies the device and should be one of the following predefined constants:
  
  tp_panel Teach pendant device
  
  crt_panel CRT device
• `spmenu_id` and `screen_no` specify the menu to force. The predefined constants beginning
  with SPI_ define the `spmenu_id` and the predefined constants beginning with SCR_ define the
  `screen_no`. If no SCR_ is listed, use 1.

  SPI_TPHINTS — UTILITIES Hints
  SPI_TPPRGADJ — UTILITIES Prog Adjust
  SPI_TPMIRROR — UTILITIES Mirror Image
  SPI_TPSHIFT — UTILITIES Program Shift
  SPI_TPTSTRUN — TEST CYCLE
  SPI_TPMANUAL, SCR_MACMAN — MANUAL Macros
  SPI_TPOTREL — MANUAL OT Release
  SPI_TPALARM, SCR_ALM_ALL — ALARM Alarm Log
  SPI_TPALARM, SCR_ALM_MOT — ALARM Motion Log
  SPI_TPALARM, SCR_ALM_SYS — ALARM System Log
  SPI_TPALARM, SCR_ALM_APPL — ALARM Appl Log
  SPI_TPDIGIO — I/O Digital
  SPI_TPANAIO — I/O Analog
  SPI_TPGRPIO — I/O Group
  SPI_TPROBIO — I/O Robot
  SPI_TPUOPIO — I/O UOP
  SPI_TPSOPIO — I/O SOP
  SPI_TPLCIO — I/O PLC
  SPI_TPSETGEN — SETUP General
  SPI_TPFRAM — SETUP Frames
  SPI_TPPORT — SETUP Port Init
  SPI_TPMACRO, SCR_MACSETUP — SETUP Macro
  SPI_TPREFPOS — SETUP Ref Position
SPI_TPPWORD — SETUP Passwords

SPI_TPHCCOMM — SETUP Host Comm

SPI_TPSYRSR — SETUP RSR/PNS

SPI_TPFILS — FILE

SPI_TPSTATUS, SCR_AXIS — STATUS Axis

SPI_TPMEMORY — STATUS Memory

SPI_TPVERSN — STATUS Version ID

SPI_TPPRGSTS — STATUS Program

SPI_TPSFTY — STATUS Safety Signals

SPI_TPUUSER — USER

SPI_TPSELECT — SELECT

SPI_TPTCH — EDIT

SPI_TPREGIS, SCR_NUMREG — DATA Registers

SPI_SFMPREG, SCR_POSREG — DATA Position Reg

SPI_TPSYSV, SCR_NUMVAR — DATA KAREL Vars

SPI_TPSYSV, SCR_POSVAR — DATA KAREL Posns

SPI_TPOSN — POSITION

SPI_TPSYSV, SCR_CLOCK — SYSTEM Clock

SPI_TPSYSV, SCR_SYSVAR — SYSTEM Variables

SPI_TPMASCAL — SYSTEM Master/Cal

SPI_TPBRCRTCTR — SYSTEM Brake Cntrl

SPI_TPAAXLM — SYSTEM Axis Limits

SPI_CRTKCL, SCR_KCL — KCL> (crt_panel only)

SPI_CRTKCL, SCR_CRT — KAREL EDITOR (crt_panel only)

SPI_TPUUSER2 — Menu for form/table managers
See Also: ACT_SCREEN Built-In Procedure

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

Section B.4 , "Standard Routines" (ROUT_EX.KL)
Section B.5 , "Using Register Built-ins" (REG_EX.KL)
Section B.12 , "Dictionary Files" (DCLISTEG.UTX)
Section B.13 , "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)
Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

A.7.5 FORMAT_DEV Built-In Procedure

Purpose: Deletes any existing information and records a directory and other internal information on
the specified device.

Syntax : FORMAT_DEV(device, volume_name, nowait_sw, status)

Input/Output Parameters :

[in] device :STRING
[in] volume_name :STRING
[in] nowait_sw :BOOLEAN
[out] status :INTEGER

%ENVIRONMENT Group :FDEV

Details:

• \textit{device} specifies the device to initialize.

• \textit{volume_name} acts as a label for a particular unit of storage media. \textit{volume_name} can be a
maximum of 11 characters and will be truncated to 11 characters if more are specified.

• If \textit{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 is complete.
  If you have time critical condition handlers in the program, put them in another program that
  executes as a separate task.

\textbf{Note} \textit{nowait_sw} is not available in this release and should be set to FALSE.

• \textit{status} explains the status of the attempted operation. If not equal to 0, then an error occurred.
See Also: MOUNT_DEV, DISMOUNT_DEV Built-In Procedures

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

A.7.6 FRAME Built-In Function

Purpose: Returns a frame with a POSITION data type representing the transformation to the coordinate frame specified by three (or four) POSITION arguments.

Syntax: FRAME(pos1, pos2, pos3 <,pos4>)

Function Return Type: Position

Input/Output Parameters:

[in]pos1 :POSITION
[in]pos2 :POSITION
[in]pos3 :POSITION
[in]pos4 :POSITION

%ENVIRONMENT Group: SYSTEM

Details:

- The returned value is computed as follows:
  - pos1 is assumed to be the origin unless a pos4 argument is supplied. See Figure A–2.
  - If pos4 is supplied, the origin is shifted to pos4, and the new coordinate frame retains the same orientation in space as the first coordinate frame. See Figure A–2.
  - The x-axis is parallel to a line from pos1 to pos2.
  - The xy-plane is defined to be that plane containing pos1, pos2, and pos3, with pos3 in the positive half of the plane.
  - The y-axis is perpendicular to the x-axis and in the xy-plane.
  - The z-axis is through pos1 and perpendicular to the xy-plane. The positive direction is determined by the right hand rule.
  - The configuration of the result is set to that of pos1, or pos4 if it is supplied.

- pos1 and pos2 arguments must be at least 10 millimeters apart and pos3 must be at least 10 millimeters away from the line connecting pos1 and pos2.

If either condition is not met, the program is paused with an error.
**Example:** The following example allows the operator to set a frame to a pallet so that a palletizing routine will be able to move the TCP along the x, y, z direction in the pallet’s coordinate frame.

**FRAME Built-In Function**

```plaintext
WRITE('Teach corner_1, corner_2, corner_3',CR)
RELEASE --Allows operator to turn on teach pendant
--and teach positions
```
ATTACH  --Returns motion control to program
$UFRAME = FRAME (corner_1, corner_2, corner_3)

A.7.7  FROM Clause

**Purpose:** Indicates a variable or routine that is external to the program, allowing data and/or routines to be shared among programs

**Syntax:** FROM prog_name

where:

prog_name : any KAREL program identifier

**Details:**

- The FROM clause can be part of a type, variable, or routine declaration.
- The type, variable, or routine belongs to the program specified by prog_name.
- In a FROM clause, prog_name can be the name of any program, including the program in which the type, variable, or routine is declared.
- If the FROM clause is used in a routine declaration and is called during program execution, the body of the declaration must appear in the specified program and that program must be loaded.
- The FROM clause cannot be used when declaring variables in the declaration section of a routine.

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

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.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)
Section B.1 , "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)
A.8  - G - KAREL LANGUAGE DESCRIPTION

A.8.1 GET_ATTR_PRG Built-In Procedure

**Purpose:** Gets attribute data from the specified teach pendant or KAREL program

**Syntax:** GET_ATTR_PRG(program_name, attr_number, int_value, string_value, status)

Input/Output Parameters :

[in] program_name :STRING

[in] attr_number :INTEGER

[out] int_value :INTEGER

[out] string_value :STRING

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

**Details:**

- *program_name* specifies the program from which to get attribute.
- *attr_number* is the attribute whose value is to be returned. 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 A

AT_DEF_GROUP : Default motion group mask (for task attribute). See Table A-16.

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)

AT_CONTROL : Control code (for task attribute)

• The program type returned for AT_PROG_TYPE will be one of the following constants:

  PT_KRLPRG : Karel program

  PT_MNE_UNDEF : Teach pendant program of undefined sub type

  PT_MNE_JOB : Teach pendant job

  PT_MNE_PROC : Teach pendant process

  PT_MNE_MACRO : Teach pendant macro

• If the attribute data is a number, it is returned in int_value and string_value is not modified.

• If the attribute data is a string, it is returned in string_value and int_value is not modified.

• 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

  17027 string_value is not large enough to contain the attribute string. The value has been truncated to fit.

  17033 attr_number has an illegal value

See Also: SET_ATTR_PRG, GET_TSK_INFO, SET_TSK_ATTR Built-In Procedures
A.8.2 GET_FILE_POS Built-In Function

**Purpose:** Returns the current file position (where the next READ or WRITE operation will take place) in the specified file.

**Syntax:** GET_FILE_POS(file_id)

Function Return Type : INTEGER

Input/Output Parameters :

[in] file_id :FILE

%ENVIRONMENT Group : FLBT

**Details:**

- GET_FILE_POS returns the number of bytes before the next byte to be read or written in the file.
- Line terminators are counted in the value returned.
- The file associated with file_id must be open. Otherwise, the program is aborted with an error.
- If the file associated with file_id is open for read-only, it cannot be on the FROM or RAM disks as a compressed file.

**Warning**

GET_FILE_POS is only supported for files opened on the RAM Disk device. Do not use GET_FILE_POS on another device; otherwise, you could injure personnel and damage equipment.

**Example:** The following example opens the filepos.dt data file, stores the positions in my_path in the file, and builds a directory to access them.

**GET_FILE_POS Built-In Function**

```
OPEN FILE file_id ('RW','filepos.dt')
FOR i = 1 TO PATH_LEN(my_path) DO
    temp_pos = my_path[i].node_pos
    pos_dir[i] = GET_FILE_POS(file_id)
    WRITE file_id (temp_pos)
ENDFOR
```
### A.8.3 GET_JPOS_REG Built-In Function

**Purpose:** Gets a JOINTPOS value from the specified register

**Syntax:** GET_JPOS_REG(register_no, status <,group_no>)

Function Return Type :REGOPE

Input/Output Parameters :

- [in] register_no :INTEGER
- [out] status :INTEGER
- [in] group_no :INTEGER

%ENVIRONMENT Group :REGOPE

**Details:**

- *register_no* specifies the position register to get.
- 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.
- GET_JPOS_REG returns the position in JOINTPOS format. Use POS_REG_TYPE to determine the position representation.
- *status* explains the status of the attempted operation. If not equal to 0, then an error occurred.

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

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

### A.8.4 GET_JPOS_TPE Built-In Function

**Purpose:** Gets a JOINTPOS value from the specified position in the specified teach pendant program

**Syntax:** GET_JPOS_TPE(open_id, position_no, status <, group_no>)

Function Return Type :JOINTPOS

Input/Output Parameters :

- [in] open_id :INTEGER
[in] position_no :INTEGER

[out] status :INTEGER

[in] group_no :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

• *open_id* specifies the teach pendant program. A program must be opened before calling this built-in.

• *position_no* specifies the position in the program to get.

• 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.

• No conversion is done for the position representation. The position data must be in JOINTPOS format. If the stored position is not in JOINTPOS, an error status is returned. Use GET_POS_TYP to get the position representation.

• If the specified position in the program is uninitialized, the returned JOINTPOS value is uninitialized and the status is set to 17038, "Uninitialized TPE position".

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

See Also: SET_JPOS_TPE, GET_POS_TPE, SET_POS_TPE Built-ins

Example: Refer to Section B.14, "Applying Offsets to a Copied Teach Pendant Program" (CPY_TPE.KL), for a detailed program example.
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION  MARRC75KR07091E Rev D

[out] slot_no :INTEGER
[out] phy_port_type :INTEGER
[out] phy_port_no :INTEGER
[out] n_ports :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

• *log_port_type* specifies the code for the type of port whose assignment is being accessed. Codes are defined in FR:KLIOTYS.KL.

• *log_port_no* specifies the number of the port whose assignment is being accessed.

• *rack_no* is returned with the rack containing the port module. For process I/O boards, memory-image, and dummy ports, this is zero; for Allen-Bradley ports, this is 16.

• *phy_port_type* is returned with the type of port 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 returned with the number of the port 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 returned with the number of physical ports assigned to the logical port. This will be 1 in all cases except when *log_port_type* is a group type. In this case, *n_ports* indicates the number of bits in the group.

• *status* is returned with zero if the parameters are valid and the specified port is assigned. Otherwise, it is returned with an error code.

Example: The following example returns to the caller the module rack and slot number, port_number, and number of bits assigned to a specified group input port. A boolean is returned indicating whether the port is assigned to a DIN port. If the port is not assigned, a non-zero status is returned.

**GET_PORT_ASG Built-In Procedure**

```karel
PROGRAM getasgprog
%ENVIRONMENT IOSETUP
%INCLUDE FR:\kliotyps
ROUTINE get_gin_asg(gin_port_no: INTEGER;
    rack_no: INTEGER;
    slot_no: INTEGER;
    frst_port_no: INTEGER;
    n_ports: INTEGER;
    asgd_to_din: BOOLEAN): INTEGER

VAR
```
phy_port_typ: INTEGER
status: INTEGER

BEGIN
GET_PORT_ASG(io_gpin, gin_port_no, rack_no, slot_no,
phy_port_typ, frst_port_no, n_ports, status)
IF status <> 0 THEN
    RETURN (status)
ENDIF
asgd_to_din = (phy_port_typ = io_din)
END get_gin_asg
BEGIN
END getasgprog

A.8.6 GET_PORT_ATR Built-In Function

Purpose: Gets an attribute from the specified port

Syntax : GET_PORT_ATR(port_id, atr_type, atr_value)

Function Return Type :INTEGER

Input/Output Parameters :

[in] port_id :INTEGER

[in] atr_type :INTEGER

[out] atr_value :INTEGER

%ENVIRONMENT Group :FLBT

Details:

- *port_id* specifies which port is to be queried. Use one of the following predefined constants:

    port_1
    port_2
    port_3
    port_4
    port_5
• *atr_type* specifies the attribute whose current setting is to be returned. Use one of the following predefined constants:

- atr_readahd : Read ahead buffer
- atr_baud : Baud rate
- atr_parity : Parity
- atr_sbits : Stop bit
- atr_dbits : Data length
- atr_xonoff : Xon/Xoff
- atr_col : End of line
- atr_modem : Modem line

• *atr_value* receives the current value for the specified attribute.

• GET_PORT_ATR returns the status of this action to the port.

**See Also:** SET_PORT_ATR Built-In Function, Chapter 7 *FILE INPUT/OUTPUT OPERATIONS*

**Example:** The following example sets up the port to a desired configuration, if it is not already set to the specified configuration.

**GET_PORT_ATR Built-In Function**

```karel
PROGRAM port_atr
%ENVIRONMENT FLBT
VAR
  stat: INTEGER
  atr_value: INTEGER
BEGIN
  -- sets read ahead buffer to desired value, if not already correct
  stat=GET_PORT_ATR(port_2, atr_readahd, atr_value)
  IF(atr_value <> 2) THEN
    stat=SET_PORT_ATR(port_2, atr_readahd, 2) --set to 256 bytes
  ENDIF
  -- sets the baud rate to 9600, if not already set
  stat=GET_PORT_ATR(port_2, atr_baud, atr_value)
  IF(atr_value <> BAUD_9600) THEN
    stat=SET_PORT_ATR(port_2, atr_baud, baud_9600)
  ENDIF
  -- sets parity to even, if not already set
  stat=GET_PORT_ATR(port_2, atr_parity, atr_value)
  IF(atr_value <> PARITY_EVEN) THEN
    stat=SET_PORT_ATR(port_2, atr_parity, PARITY_EVEN)
```

A–152
A.8.7 GET_PORT_CMT Built-In Procedure

**Purpose:** Allows a KAREL program to determine the comment that is set for a specified logical port

**Syntax:** GET_PORT_CMT(port_type, port_no, comment_str, status)

**Input/Output Parameters:**

- [in] port_type :INTEGER
- [in] port_no :INTEGER
- [out] comment_str :STRING
- [out] status :INTEGER

**Details:**

- *port_type* specifies the code for the type of port whose comment is being returned. Codes are defined in FR:KLIOTYPs.KL.
• \textit{port\_no} specifies the port number whose comment is being set.

• \textit{comment\_str} is returned with the comment for the specified port. This should be declared as a \texttt{STRING} with a length of at least 16 characters.

• \textit{status} is returned with zero if the parameters are valid and the comment is returned for the specified port.

\textbf{See Also:} GET\_PORT\_VAL, GET\_PORT\_MOD, SET\_PORT\_CMT, SET\_PORT\_VAL, SET\_PORT\_MOD Built-in Procedures.

\section*{A.8.8 GET\_PORT\_MOD Built-In Procedure}

\textbf{Purpose:} Allows a KAREL program to determine what special port modes are set for a specified logical port

\textbf{Syntax :} GET\_PORT\_MOD(port\_type, port\_no, mode\_mask, status)

\textbf{Input/Output Parameters :}

\begin{itemize}
  \item \texttt{[in] port\_type :INTEGER}
  \item \texttt{[in] port\_no :INTEGER}
  \item \texttt{[out] mode\_mask :INTEGER}
  \item \texttt{[out] status :INTEGER}
\end{itemize}

%ENVIRONMENT Group :IOSETUP

\textbf{Details:}

\begin{itemize}
  \item \textit{port\_type} specifies the code for the type of port whose mode is being returned. Codes are defined in FR:KLIOTYPS.KL.
  \item \textit{port\_no} specifies the port number whose mode is being set.
  \item \textit{mode\_mask} is returned with a mask specifying which modes are turned on. The following modes are defined:
    \begin{itemize}
      \item 1 :reverse mode
        Sense of the port is reversed; if the port is set to TRUE, the physical output is set to FALSE. 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.
      \item 2 :complementary mode
    \end{itemize}
\end{itemize}
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, then port n is set to TRUE and port n + 1 is set to FALSE. If port n is set to FALSE, then port n is set to FALSE and port n + 1 is set to TRUE. This is effective only for output ports.

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

**Example:** The following example gets the mode(s) for a specified port.

**GET_PORT_MOD_Built-In Procedure**

```plaintext
PROGRAM getmodprog
%ENVIRONMENT IOSETUP
%INCLUDE FR:\kliotypes
ROUTINE get_mode( port_type: INTEGER;
port_no: INTEGER;
reverse: BOOLEAN;
complementary: BOOLEAN): INTEGER
VAR
mode: INTEGER
status: INTEGER
BEGIN
GET_PORT_MOD(port_type, port_no, mode, status)
IF (status <>0) THEN
   RETURN (status)
ENDIF
IF (mode AND 1) <> 0 THEN
   reverse = TRUE
ELSE
   reverse = FALSE
ENDIF
IF (mode AND 2) <> 0 THEN
   complementary = TRUE
ELSE
   complementary = FALSE
ENDIF
RETURN (status)
END get_mode
BEGIN
END getmodprog
```

**A.8.9 GET_PORT_SIM Built-In Procedure**

**Purpose:** Gets port simulation status
Syntax : GET_PORT_SIM(port_type, port_no, simulated, status)

Input/Output Parameters:

[in] port_type :INTEGER
[in] port_no :INTEGER
[out] simulated :INTEGER
[out] status :INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

• port_type specifies the code for the type of port to get. Codes are defined in FRS:KLIOTYPS.KL.
• port_no specifies the number of the port whose simulation status is being returned.
• simulated returns TRUE if the port is being simulated, FALSE otherwise.
• status is returned with zero if the port is valid.

See Also: GET_PORT_MOD, SET_PORT_SIM, SET_PORT_MOD Built-in Procedures.

A.8.10 GET_PORT_VAL Built-In Procedure

Purpose: Allows a KAREL program to determine the current value of a specified logical port

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

Input/Output Parameters:

[in] port_type :INTEGER
[in] port_no :INTEGER
[out] value :STRING
[out] status :INTEGER

%ENVIRONMENT Group :IOSETUP

Details:

• port_type specifies the code for the type of port whose comment is being returned. Codes are defined in FR:KLIOTYPS.KL.
• `port_no` specifies the port number whose comment is being set.
• `value` is returned with the current value (status) of the specified port. For BOOLEAN port types, (i.e. DIN), this will be 0 = OFF, or 1 = ON.
• `status` is returned with zero if the parameters are valid and the comment is returned for the specified port.

See Also: GET_PORT_CMT, GET_PORT_MOD, SET_PORT_CMT, SET_PORT_VAL, SET_PORT_MOD Built-in Procedures.

### A.8.11 GET_POS_FRM Built-In Procedure

**Purpose:** Gets the uframe number and utool number of the specified position in the specified teach pendant program.

**Syntax:**
`GET_POS_FRM(open_id, position_no, gnum, ufram_no, utool_no, status)`

**Input/Output Parameters:**
- `[in]` `open_id` :INTEGER
- `[in]` `position_no` :INTEGER
- `[in]` `gnum` :INTEGER
- `[out]` `ufram_no` :INTEGER
- `[out]` `utool_no` :INTEGER
- `[out]` `status` :INTEGER

**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 teach pendant program.
- `gnum` specifies the group number of position.
- `ufram_no` is returned with the frame number of position_no.
- `utool_no` is returned with the tool number of position_no.
- If the specified position, `position_no`, is uninitialized, the `status` is set to 17038, "Uninitialized TPE position."
- `status` indicates the status of the attempted operation. If not equal to 0, then an error occurred.
A.8.12 GET_POS_REG Built-In Function

**Purpose:** Gets an XZWPR value from the specified register

**Syntax:** GET_POS_REG(register_no, status <,group_no>)

**Function Return Type:** XZWPREXT

**Input/Output Parameters:**
- [in] register_no :INTEGER
- [out] status :INTEGER
- [in] group_no :INTEGER

%ENVIRONMENT Group :REGOPE

**Details:**
- register_no specifies the position register to get.
- 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.
- GET_POS_REG returns the position in XZWPREXT format. Use POS_REG_TYPE to determine the position representation.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

**See Also:** GET_JPOS_REG, SET_JPOS_REG, SET_POS_REG, GET_REG Built-in Procedures.

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

A.8.13 GET_POS_TPE Built-In Function

**Purpose:** Gets an XZWPREXT value from the specified position in the specified teach pendant program

**Syntax:** GET_POS_TPE(open_id, position_no, status <, group_no>)

**Function Return Type:** XZWPREXT
Input/Output Parameters:

[in] open_id : INTEGER
[in] position_no : INTEGER
[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 to get.
• No conversion is done for the position representation. The positional data must be in XYZWPR or XYZWPREXT, otherwise, an error status is returned. Use GET_POS_TYP to get the position representation.
• If the specified position in the program is uninitialized, the returned XYZWPR value is uninitialized and status is set to 17038, "Uninitialized TPE Position."
• 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.
• status explains the status of the attempted operation. If not equal to 0, then an error has occurred.

See Also: GET_JPOS_TPE, SET_JPOS_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.

A.8.14 GET_POS_TYP Built-In Procedure

Purpose: Gets the position representation of the specified position in the specified teach pendant program

Syntax : GET_POS_TYP(open_id, position_no, group_no, posn_typ, num_axs, status)

Input/Output Parameters:

[in] open_id :INTEGER
[in] position_no :INTEGER
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION  MARRC75KR07091E Rev D

[in] group_no :INTEGER

[out] posn_typ :INTEGER

[out] num_axs :INTEGER

[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.

• group_no specifies the group number.

• Position type is returned by posn_typ. posn_typ is defined as follows:2 :XYZWPR6 :XYZWPREXT9 :JOINTPOS

• If it is in joint position, the number of the axis in the representation is returned by num_axs.

• If the specified position in the program is uninitialized, then a status is set to 17038, "Uninitialized TPE Position."

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

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

A.8.15 GET_PREG_CMT Built-In-Procedure

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

Syntax: GET_PREG_CMT (register_no, comment_string, status)

Input/Output Parameters:

[in] register_no: INTEGER

[out] comment_string: STRING

[out] status: INTEGER

%ENVIRONMENT group: REGOPE

Details:
• Register_no specifies which position register to retrieve the comments from. The comment of
the given position register is returned in the comment_string.

A.8.16 GET_QUEUE Built-In Procedure

Purpose: Retrieves the specified oldest entry from a queue

Syntax: GET_QUEUE(queue, queue_data, value, status, sequence_no)

Input/Output Parameters:

[in,out] queue_t : QUEUE_TYPE
[in,out] queue_data : ARRAY OF INTEGER
[out] value : INTEGER
[out] sequence_no : INTEGER
[out] status : INTEGER

%ENVIRONMENT Group : PBQMGR

Details:

• queue_t specifies the queue variable for the queue from which the value is to be obtained.
• queue_data specifies the array variable with the queue data.
• value is returned with the oldest entry obtained from the queue.
• sequence_no is returned with the sequence number of the returned entry.
• status is returned with the zero if an entry is successfully obtained from the queue. Otherwise, a
  value of 61002, “Queue is empty,” is returned.

See Also: MODIFY_QUEUE Built-In Procedure, Section 15.8, "Using Queues for Task
Communication

Example: In the following example the routine get_nxt_err returns the oldest entry from the error
queue, or zero if the queue is empty.

GET_QUEUE Built-In Procedure

PROGRAM get_queue_x
%environment PBQMGR
VAR
  error_queue FROM global_vars: QUEUE_TYPE
  error_data FROM global_vars: ARRAY[100] OF INTEGER
ROUTINE get_nxt_err: INTEGER
VAR
    status: INTEGER
    value: INTEGER
    sequence_no: INTEGER
BEGIN
    GET_QUEUE(error_queue, error_data, value, sequence_no, status)
    IF (status = 0) THEN
        RETURN (value)
    ELSE
        RETURN (0)
    ENDIF
END get_nxt_err
BEGIN
END get_queue_x

A.8.17 GET_REG Built-In Procedure

Purpose: Gets an INTEGER or REAL value from the specified register

Syntax: GET_REG(register_no, real_flag, int_value, real_value, status)

Input/Output Parameters:

[in] register_no :INTEGER

[out] real_flag :BOOLEAN

[out] int_value :INTEGER

[out] real_value :REAL

[out] status :INTEGER

%ENVIRONMENT Group :REGOPE

Details:

- register_no specifies the register to get.
- real_flag is set to TRUE and real_value to the register content if the specified register has a real value. Otherwise, real_flag is set to FALSE and int_value is set to the contents of the register.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.
Example: Refer to Section B.5, "Using Register Built-ins" (REG_EX.KL), for a detailed program example.

### A.8.18 GET_REG_CMT

**Purpose:** To retrieve the comment information of a KAREL register based on a given register number.

**Syntax:** GET_REG_CMT (register_no, comment_string, status)

**Input/Output Parameters:**
- [in] register_no: INTEGER
- [out] comment_string: STRING
- [out] status: INTEGER

**Details:**
- Register_no specifies which register to retrieve the comments from. The comment of the given register is returned in comment_string.

### A.8.19 GET_SREG_CMT Builtin Procedure

**Purpose:** Gets the comment from the specified string register.

**Syntax:** GET_SREG_CMT(register_no, comment, status)

**Input/Output Parameters:**
- [in] register_no :INTEGER
- [out] comment :STRING[254]
- [out] status :INTEGER

**Details:**
- register_no specifies the string register to get.
- comment contains the comment of 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, SET_STR_REG, SET_SREG_CMT Built-in Procedures.

A.8.20 GET_STR_REG Built-In Procedure

Purpose: Gets the value from the specified string register.

Syntax: GET_STR_REG(register_no, value, status)

Input/Output Parameters:
[in] register_no :INTEGER
[out] value :STRING[254]
[out] status :INTEGER

%ENVIRONMENT Group :REGOPE

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

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

A.8.21 GET_TIME Built-In Procedure

Purpose: Retrieves the current time (in integer representation) from within the KAREL system.

Syntax: GET_TIME(i)

Input/Output Parameters:
[out] i :INTEGER

%ENVIRONMENT Group :TIM

Details:
• i holds the INTEGER representation of the current time stored in the KAREL system. This value is represented in 32-bit INTEGER format as follows:
The contents of the individual fields are as follows:

— **DATE:**

  - Bits 31-25 — Year since 1980
  - Bits 24-21 — Month (1-12)
  - Bits 20-16 — Day of the month

— **TIME:**

  - Bits 15-11 — Number of hours (0-23)
  - Bits 10-5 — Number of minutes (0-59)
  - Bits 4-0 — Number of 2-second increments (0-29)

- INTEGER values can be compared to determine if one time is more recent than another.
- Use the CNV_TIME_STR built-in procedure to convert the INTEGER into the “DD-MMM-YYY HH:MM:SS” STRING format.

**See Also:** CNV_TIME_STR Built-In Procedure

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

### A.8.22 GET_TPE_CMT Built-in Procedure

**Purpose:** This built-in provides the ability for a KAREL program to read the comment associated with a specified position in a teach pendant program.

**Syntax:** GET_TPE_CMT(open_id, pos_no, comment, status)

Input/Output Parameters:

[in] open_id :INTEGER

[in] pos_no :INTEGER
[out] comment :STRING

[out] status :INTEGER

%ENVIRONMENT Group :TPE

Details:

• open_id specifies the open_id returned from a previous call to OPEN_TPE.
• pos_no specifies the number of the position in the TPP program to get a comment from.
• comment is associated with specified positions and is returned with a zero length string if the position has no comment. If the string variable is too short for the comment, an error is returned and the string is not changed.
• status indicates zero if the operation was successful, otherwise an error code will be displayed.

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

A.8.23 GET_TPE_PRM Built-in Procedure

Purpose: Gets the values of the parameters when parameters are passed in a TPE CALL or MACRO instruction.

Syntax: GET_TPE_PRM(param_no, data_type, int_value, real_value, str_value, status)

Input/Output Parameters:

[in] param_no :INTEGER

[out] data_type :INTEGER

[out] int_value :INTEGER

[out] real_value :REAL

[out] str_value :STRING

[out] status :INTEGER

%ENVIRONMENT Group :PBCORE

Details:

• param_no indicates the number of the parameter. There can be at most ten parameters.
• data_type indicates the data type for the parameter, as follows:
— 1 : INTEGER
— 2 : REAL
— 3 : STRING

• int_value is the value of the parameter if the data_type is INTEGER.
• real_value is the value of the parameter if the data_type is REAL.
• str_value is the value of the parameter if the data_type is STRING.
• status explains the status of the attempted operation. If not equal to 0, then an error has occurred.
• If the parameter designated by param_no does not exist, a status of 17042 is returned, which is the error message: "ROUT-042 WARN TPE parameters do not exist." If this error is returned, confirm the param_no and the parameter in the CALL or MACRO command in the main TPE program.

See Also: Application-Specific FANUC Robotics Setup and Operations Manual, for information on using parameters in teach pendant CALL or MACRO instructions.

Example: The following example shows the implementation of a macro (Send Event) with CALL parameters that are retrieved by a KAREL program that uses the GET_TPE_PRM built-in.

GET_TPE_PRM Built-In Procedure

Macro table entry for the Send Event macro:
109 [Send Event ] [SENDEVNT]--[ 0]

Teach pendant program, TEST1.TP, which uses the Send Event macro:
1: ! Send Event 7
  2: ! Wait for PC answer
  3: ! Answer in REG 5
  4: Send Event(7,1,5)
  6: ! Error in macro
  7: !
  8: LBL[10]

Teach pendant program SENDEVNT.TP, which implements the Send Event macro by calling the GESNDEVT KAREL program and passing the CALL parameters from Send Event:
1: !Send Event Macro
  2: CALL GESNDEVT(AR[1],AR[2],AR[3])

Snippet of the KAREL program GESNDEVT.KL, which gets the parameter information using the GET_TPE_PRM built-in:
PROGRAM GESNDEVT
  ...
BEGIN
  -- Send Event(event_no [,wait_sw [,status_reg]] )
  -- get parameter 1 (mandatory parameter)
Get_tpe_prm(1, data_type, event_no,real_value,string_value,status)
IF status<>0 THEN -- 17042 "ROUT-042 TPE parameters do not exist"
    POST_ERR(status, '', 0, erAbort)
ELSE
    IF data_type <> PARM_INTEGER THEN -- make sure parm is an integer
        POST_ERR(er_pceventer, '1', 0, erAbort)
    ELSE
        IF (event_no < MIN_EVENT) OR (event_no > MAX_EVENT) THEN
            POST_ERR(er_illevent, '', 0, erAbort)
        ENDIF
    ENDIF
ENDIF

-- get second parameter (optional)
Get_tpe_prm(2, data_type, wait_sw,real_value,string_value,status)
IF status<>0 THEN
    IF status = ER17042 THEN -- "ROUT-142 Parameter doesn’t exist"
        wait_sw = 0 -- DEFAULT no wait
    ELSE
        POST_ERR(status, '', 0, erWarn) -- other error
    ENDIF
ENDIF

A.8.24 GET_TSK_INFO Built-In Procedure

Purpose: Get the value of the specified task attribute

Syntax: GET_TSK_INFO(task_name, task_no, attribute, value_int, value_str, status)

Input/Output Parameters:
[in,out] task_name :STRING
[in,out] task_no :INTEGER
[in] attribute :INTEGER
[ out] value_int :INTEGER
[ out] value_str :STRING
[ out] status :INTEGER

%ENVIRONMENT Group :PBCORE
Details:

- *task_name* is the name of the task of interest. *task_name* is used as input only if *task_no* is uninitialized or set to 0, otherwise, *task_name* is considered an output parameter.
- *task_no* is the task number of interest. If *task_no* is uninitialized or set to 0, it is returned as an output parameter.
- *attribute* is the task attribute whose value is to be returned. It will be returned in *value_int* unless otherwise specified. The following attributes are valid:

  TSK_HOLDCOND : Task hold conditions
  TSK_LINENUM : Current executing line number
  TSK_LOCKGRP : Locked group
  TSK_MCTL : Motion controlled groups
  TSK_NOABORT : Ignore abort request
  TSK_NOBUSY : Busy lamp off
  TSK_NOPAUSE : Ignore pause request
  TSK_NUMCLDS : Number of child tasks
  TSK_PARENT : Parent task number
  TSK_PAUSESFT : Pause on shift release
  TSK_PRIORITY : Task priority
  TSK_PROGNAME : Current program name returned in *value_str*
  TSK_PROGTYPE : Program type - refer to description below
  TSK_ROUTNAME : Current routine name returned in *value_str*
  TSK_STACK : Stack size
  TSK_STATUS : Task status — refer to description below
  TSK_STEP : Single step task
  TSK_TIMESLIC : Time slice duration in ms
  TSK_TPMOTION : TP motion enable
  TSK_TRACE : Trace enable
TSK_TRACELEN : Length of trace array

- TSK_STATUS is the task status: The return values are:
  
  - PG_RUNACCEPT : Run request has been accepted
  
  - PG_ABORTING : Abort has been accepted
  
  - PG_RUNNING : Task is running
  
  - PG_PAUSED : Task is paused
  
  - PG_ABORTED : Task is aborted

- TSK_PROGTYPE is the program type. The return values are:
  
  - PG_NOT_EXEC : Program has not been executed yet
  
  - PG_MNEMONIC : Teach pendant program is or was executing
  
  - PG_AR_KAREL : KAREL program is or was executing

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

See Also: Chapter 15 MULTI-TASKING

Example: See examples in Chapter 15 MULTI-TASKING

### A.8.25 GET_USEC_SUB Built-In Procedure

**Purpose:** Returns an INTEGER value indicating the elapsed time in microseconds (1/1,000,000).

**Syntax:**

```
us_delta = GET_USEC_SUB(us2, us1)
```

**Function Return Type:** INTEGER

**Input/Output Parameters:**

- **[in]** `us2`: INTEGER
- **[in]** `us1`: INTEGER

**%ENVIRONMENT Group:** TIM

**Details:**

- `us2` is the second time returned from GET_USEC_TIM.
- `us1` is the first time returned from GET_USEC_TIM.
The returned value is the INTEGER representation of the elapsed time \( \text{us2} - \text{us1} \) in microseconds.

This is intended to measure fast operations. The result will wrap after 2 minutes and will no longer be valid.

Example: The following example measures the amount of time in microseconds to increment a number.

**Figure A–3. GET_USEC_SUB Built-In Function**

```karel
i = 0
us1 = GET_USEC_TIM
i = i + 1
us_delta = GET_USEC_SUB(GET_USEC_TIM, us1)
WRITE ('Time to increment a number: ', us_delta, ' us', CR)
```

**A.8.26 GET_USEC_TIM Built-In Function**

**Purpose:** Returns an INTEGER value indicating the current time in microseconds (1/1,000,000) from within the KAREL system.

**Syntax:** \( \text{us} = \text{GET_USEC_TIM} \)

Function Return Type: INTEGER

Input/Output Parameters:

None

%ENVIRONMENT Group: TIM

**Details:**

- The returned value is the INTEGER representation of the current time in microseconds stored in the KAREL system.
- This function is used with the GET_USEC_SUB built-in function to determine the elapsed time of an operation.

**A.8.27 GET_VAR Built-In Procedure**

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

**Syntax:** GET_VAR(entry, prog_name, var_name, value, status)
Input/Output Parameters:

[in,out] entry :INTEGER
[in] prog_name :STRING
[in] var_name :STRING
[out] 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 blank, it will default to the current task name being executed. Set the prog_name to ‘**SYSTEM**’ to get a system variable. prog_name can also access a system variable on a robot in a ring.

• var_name must refer to a static, program 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.

• value is the value of var_name.

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

• If the value of var_name is uninitialized, then value will be set to uninitialized and status will be set to 12311.

• 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: SET_VAR Built-In Procedure, Internet Options Manual for information on accessing system variables on a robot in a ring.
Caution

Using GET_VAR to modify system variables could cause unexpected results.

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

**Accessing $TP_DEFPROG on MHROB03**

GET_VAR(entry, ‘\\MHROB03\*system*’, ‘$TP_DEFPROG’, strvar, status)

**Example 2:** GET_VAR Built-In Procedure displays two programs, **util prog** and **task**. The program **util prog** uses a FOR loop to increment the value of the INTEGER variable **num_of_parts**. **util prog** also assigns values to the ARRAY **part_array**. The program **task** uses two GET_VAR statements to retrieve the values of **num_of Parts** and **part_array[3]**. The value of **num_of_parts** is assigned to the INTEGER variable **count** and **part_array[3]** is assigned to the STRING variable **part_name**. The last GET_VAR statement places the value of **count** into another INTEGER variable **newcount**.

**GET_VAR Built-In Procedure**

PROGRAM **util prog**
VAR
  j, num_of_parts : INTEGER
BEGIN
  num_of_parts = 0
  FOR j = 1 to 20 DO
    num_of_parts = num_of_parts + 1
  ENDFOR
  part_array[1] = 10
  part_array[2] = 20
  part_array[3] = 30
  part_array[4] = 40
  part_array[5] = 50
END util prog

PROGRAM **task**
VAR
  entry, status : INTEGER
  count, new_count : INTEGER
  part_name : STRING[20]
BEGIN
  GET_VAR(entry, ‘util prog’, ‘part_array[3]’, part_name, status)
  WRITE(‘Part Name is Now....>’, part_name, cr)
  GET_VAR(entry, ‘util prog’, ‘num_of_parts’, count, status)
WRITE(‘COUNT Now Equals....>’, count, cr)

GET_VAR(entry,’task’,’count’,new_count,status)

END task

In **GET_VAR** **SET_VAR** Built-In Procedure, 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** Built-In Procedure

FOR idx = 1 TO $PH_ROSIP.$NUM_MEMBERS DO
  IF idx = $PH_ROSIP.$MY_INDEX THEN
    -- This will work but 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])
  ENDSELECT
  ENDIF
ENDFOR

**Example 3** **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 GET_VAR on numeric registers, the type of the KAREL variable must match the type of the register or the error, “Incompatible value” is returned. In the example below if numeric register 10 is a real value an error will be returned and the real value will be set in the error case.

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.

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
```

A.8.28 GO TO Statement

**Purpose:** Transfers control to a specified statement

**Syntax:** `|| GO TO | GOTO || stmt_label`

where:

stmt_label: A valid KAREL identifier
Details:

- `stmt_label` must be defined in the same routine or program body as the GO TO statement.
- Label identifiers are followed by double colons (::). Executable statements may or may not follow on the same line.
- GOTO should only be used in special circumstances where normal control structures such as WHILE, REPEAT, and FOR loops would be awkward or difficult to implement.

See Also: Section 2.1.5, “Labels,” for more information on rules for labels, Appendix E, “Syntax Diagrams,” for additional syntax information

Example: The following example moves the TCP from one position to another depending on the status of DIN[1].

**GO TO Statement**

```
BEGIN
  IF NOT DIN[1] THEN
    move_to_p1 —— Call TP program to do move
  ELSE
    GO TO end_it
  ENDIF
  IF NOT DIN[1] THEN
    move_to_p2 —— Call TP program to do move
  ELSE
    GO TO end_it
  ENDIF
END_IT::
```

**A.9 - H - KAREL LANGUAGE DESCRIPTION**

**A.9.1 HOLD Action**

**Purpose:** Causes the current motion to be held and prevents subsequent motions from starting

**Syntax:** HOLD <GROUP[n,\{,n\}>

**Details:**

- Any motion in progress is held. Robot and auxiliary or extended axes decelerate to a stop.
- An attempted motion after a HOLD is executed is also held. HOLD cannot be overridden by a condition handler which issues a motion.
- HOLD is released using the UNHOLD statement or action.
• If the group clause is not present, all groups for which the task has control (when the condition is defined) will be canceled.

• If a motion that is held is part of a SIMULTANEOUS or COORDINATED motion with other groups, the motions for all groups are held.

• Motion cannot be held for a different task.

See Also: Chapter 8 POSITION DATA, for more information on starting and stopping motions.

Example: The following example shows a Condition Handler that holds motion when DIN[1] turns on.

**HOLD Action**

```
CONDITION[1]:
WHEN DIN[1] = ON DO
    HOLD
ENDCONDITION
```

**A.9.2 HOLD Statement**

**Purpose:** Causes the current motion to be held and prevents subsequent motions from starting.

**Syntax:** HOLD <GROUP[n]>

**Details:**

• Any motion in progress is held. Robot and auxiliary or extended axes decelerate to a stop.

• An attempted motion after a HOLD is executed is also held. HOLD cannot be overridden by a condition handler which issues a motion.

• HOLD is released using the UNHOLD statement or action.

• All held motions are canceled if a RELEASE statement is executed while motion is held.

• If the group clause is not present, all groups for which the task has control will be canceled.

• If a 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 held for a different task.

See Also: Chapter 8 POSITION DATA, for more information on starting and stopping motions, Appendix E, “Syntax Diagrams,” for additional syntax information.
A.10 - I - KAREL LANGUAGE DESCRIPTION

A.10.1 IF ... ENDIF Statement

**Purpose:** Executes a sequence of statements if a BOOLEAN expression is TRUE; an alternate sequence can be executed if the condition is FALSE.

**Syntax:** IF bool_exp THEN

{ true_stmt } < ELSE

{ false_stmt } >ENDIF

where:

bool_exp : BOOLEAN

true_stmt : An executable KAREL statement

false_stmt : An executable KAREL statement

**Details:**

- If `bool_exp` evaluates to TRUE, the statements contained in the `true_stmt` are executed. Execution then continues with the first statement after the ENDIF.
- If `bool_exp` evaluates to FALSE and no ELSE clause is specified, execution skips directly to the first statement after the ENDIF.
- If `bool_exp` evaluates to FALSE and an ELSE clause is specified, the statements contained in the `false_stmt` are executed. Execution then continues with the first statement after the ENDIF.
- IF statements can be nested in either `true_stmt` or `false_stmt`.

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

**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.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.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.10.2 IN Clause

Purpose: Specifies where a variable will be created

Syntax: IN (CMOS | DRAM | SHADOW)

Details:

- The IN clause can be part of a variable declaration. It should be specified before the FROM clause.
- IN CMOS specifies that the variable will be created in permanent memory.
- IN DRAM specifies that the variable will be created in temporary memory.
- IN SHADOW specifies that any changes made to the variable will be maintained in CMOS. Writes to this type of variable are slower but reads are much faster. This a good memory type to use for configuration parameters that are currently in CMOS.
- IN UNINIT_DRAM specifies that a DRAM variable is UNINITIALIZED at startup.
- If the IN clause is not specified all variables are created in temporary memory; unless the %CMOSVARS or %SHADOW directive is specified, in which case all variables will be created in permanent memory.
- The IN clause cannot be used when declaring variables in the declaration section of a routine.

See Also: Section 1.4.1, %CMOSVARS Translator Directive, %SHADOW Translator Directive, %SHADOWCMOS Translator Directive

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

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

In CMOS, Section B.10, "Using Dynamic Display Built-ins" (DYN_DISP.KL) or Section B.11, "Manipulating Values of Dynamically Displayed Variables" (CHG_DATA.KL)
A.10.3 %INCLUDE Translator Directive

**Purpose:** Inserts other files in a program at translation time.

**Syntax:** `%INCLUDE file_spec`

**Details:**

- `file_spec` is the name of the file to include. It has the following details:
  - The file name specified must be no longer than 12 characters.
  - The file type defaults to .KL, and so it does not appear in the directive.
- The `%INCLUDE` directive must appear on a line by itself.
- The specified files usually contain declarations, such as CONST or VAR declarations. However, they can contain any portion of a program including executable statements and even other `%INCLUDE` directives.
- Included files can themselves include other files up to a maximum depth of three nested included files. There is no limit on the total number of included files.
- When the KAREL language translator encounters a `%INCLUDE` directive during translation of a file, it begins translating the included file just as though it were part of the original file. When the entire file has been included, the translator resumes with the original file.
- Some examples in Appendix A reference the following include files:

  ```
  %INCLUDE FR:\klevkmsk
  %INCLUDE FR:\klevkeys
  %INCLUDE FR:\klevccdf
  %INCLUDE FR:\kliotyps
  ```

  These files contain constants that can be used in your KAREL programs. If you are translating on the controller, you can include them directly from the FROM disk.

  The include files are also available on the OLPC disks, and are copied to the hard disk as part of the installation process.

  **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)
  - **Section B.13**, "Using the DISCTRL_ALPHA Built-in" (DCALP_EX.KL)
Section B.1, "Setting Up Digital Output Ports for Monitoring" (DOUT_EX.KL)

### A.10.4 INDEX Built-In Function

**Purpose:** Returns the index for the first character of the first occurrence of a specified STRING argument in another specified STRING argument. If the argument is not found, a 0 value is returned.

**Syntax:** INDEX(main, find)

Function Return Type: INTEGER

Input/Output Parameters:

[in] main : STRING

[in] find : STRING

%ENVIRONMENT Group : SYSTEM

**Details:**

- The returned value is the index position in main corresponding to the first character of the first occurrence of find or 0 if find does not occur in main.

**Example:** The following example uses the INDEX built-in function to look for the first occurrence of the string “Old” in part_desc.

```karel
INDEX Built-In Function

class_key = 'Old'
part_desc = 'Refurbished Old Part'
IF INDEX(part_desc, class_key) > 0 THEN
  in_class = TRUE
ENDIF
```

### A.10.5 INI_DYN_DISB Built-In Procedure

**Purpose:** Initiates the dynamic display of a BOOLEAN variable. This procedure displays elements of a STRING ARRAY depending on the current value of the BOOLEAN variable.

**Syntax:** INI_DYN_DISB (b_var, window_name, field_width, attr_mask, char_size, row, col, interval, strings, status)

Input/Output Parameters:
[in] b_var :BOOLEAN
[in] window_name :STRING
[in] field_width :INTEGER
[in] attr_mask :INTEGER
[in] char_size :INTEGER
[in] row :INTEGER
[in] col :INTEGER
[in] interval :INTEGER
[in] strings :ARRAY OF STRING
[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

• The dynamic display is initiated based on the value of b_var. If b_var is FALSE, strings[1] is displayed; if b_var is TRUE, strings[2] is displayed. If b_var is uninitialized, a string of *'s is displayed. Both b_var and strings must be static (not local) variables.

• window_name must be a previously defined window name. See Section 7.10.1 and Section 7.10.2 for predefined window names.

• If field_width is non-zero, the display is extended with blanks if the element of strings[n] is shorter than this specified width. The area is cleared when the dynamic display is canceled.

• attr_mask is a bit-wise mask that indicates character display attributes. This should be one of the following constants:

0 :Normal
1 :Bold (Supported only on the CRT)
2 :Blink (Supported only on the CRT)
4 :Underline
8 :Reverse video

• To have multiple display attributes, use the OR operator to combine the constant attribute values together. For example, to have the text displayed as bold and underlined use 1 OR 4.
• *char_size* specifies whether data is to be displayed in normal, double-wide, or double-high, double-wide sizes. This should be one of the following constants:

0 : Normal

1 : Double-wide (Supported only on the CRT)

2 : Double-high, double-wide

• *row* and *col* specify the location in the window in which the data is to be displayed.

• *interval* indicates the minimum time interval, in milliseconds, between updates of the display. This must be greater than zero. The actual time might be greater since the task that formats the display runs at a low priority.

• *strings[n]* contains the text that will be displayed.

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

**See Also:** CNC_DYN_DISB built-in procedure

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

### A.10.6 INI_DYN_DISE Built-In Procedure

**Purpose:** Initiates the dynamic display of an INTEGER variable. This procedure displays elements of a STRING ARRAY depending on the current value of the INTEGER variable.

**Syntax:** 
INI_DYN_DISE (e_var, window_name, field_width, attr_mask, char_size, row, col, interval, strings, status)

**Input/Output Parameters:**

[in] e_var : INTEGER

[in] window_name : STRING

[in] field_width : INTEGER

[in] attr_mask : INTEGER

[in] char_size : INTEGER

[in] row : INTEGER

[in] col : INTEGER

[in] interval : INTEGER
[in] strings :ARRAY OF STRING

[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

• The dynamic display is initiated based on the value of \textit{e\_var}. If \textit{e\_var} has a value of \textit{n}, \textit{strings[n+1]} is displayed; if \textit{e\_var} has a negative value, or a value greater than or equal to the length of the array of \textit{strings}, a string of '?'s is displayed. Both \textit{e\_var} and \textit{strings} must be static (not local) variables.

• Refer to the \texttt{INI\_DYN\_DISB} built-in procedure for a description of the other parameters listed above.

See Also: CNC\_DYN\_DISE built-in procedure

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

\textbf{A.10.7 INI\_DYN\_DISI Built-In Procedure}

\textbf{Purpose:} Initiate the dynamic display of an INTEGER variable in a specified window.

\textbf{Syntax:} INI\_DYN\_DISI(i\_var, window\_name, field\_width, attr\_mask, char\_size, row, col, interval, buffer\_size, format, status)

Input/Output Parameters:

[in] \texttt{i\_var} :INTEGER

[in] \texttt{window\_name} :STRING

[in] \texttt{field\_width} :INTEGER

[in] \texttt{attr\_mask} :INTEGER

[in] \texttt{char\_size} :INTEGER

[in] \texttt{row} :INTEGER

[in] \texttt{col} :INTEGER

[in] \texttt{interval} :INTEGER

[in] \texttt{buffer\_size} :INTEGER
[in] format :STRING
[out] status :INTEGER

%ENVIRONMENT Group :UIF

Details:

- $i\_var$ is the integer whose dynamic display is to be initiated.
- If $field\_width$ is non-zero, the display is extended with blanks if $i\_var$ is shorter than this specified width. The area is cleared when the dynamic display is canceled.
- $buffer\_size$ is not implemented.
- $format$ is used to print out the variable. This can be passed as a literal enclosed in single quotes. The format string begins with a % and ends with a conversion character. Between the % and the conversion character there can be, in order:
  - Flags (in any order), which modify the specification:
    - - : specifies left adjustment of this field.
    - + : specifies that the number will always be printed with a sign.
    - 0 specifies padding a numeric field width with leading zeroes.
  - A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
  - A period, which separates the field width from the precision.
  - A number, the precision, that specifies the maximum number of characters to be printed from a string, or the number of digits after the decimal point of a floating-point value, or the minimum number of digits for an integer.

The format specifier must contain one of the conversion characters in Table A–12.

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type; Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>INTEGER; decimal number</td>
</tr>
<tr>
<td>o</td>
<td>INTEGER; unsigned octal notation (without a leading zero).</td>
</tr>
<tr>
<td>x,X</td>
<td>INTEGER; unsigned hexadecimal notation (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>u</td>
<td>INTEGER; unsigned decimal notation.</td>
</tr>
<tr>
<td>s</td>
<td>STRING; print characters from the string until end of string or the number of characters given by the precision.</td>
</tr>
</tbody>
</table>
Table A–12. Conversion Characters (Cont’d)

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type; Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>REAL; decimal notation of the form [-]mmm.dddddd, where the number of d's is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>e,E</td>
<td>REAL; decimal notation of the form [-]m.dddddde+-xx or [-]m.ddddddE+-xx, where the number of d's is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>g,G</td>
<td>REAL; %e or %E is used if the exponent is less than -4 or greater than or equal to the precision; otherwise %f is used. Trailing zeros and a trailing decimal pointer are not printed.</td>
</tr>
<tr>
<td>%</td>
<td>no argument is converted; print a %.</td>
</tr>
</tbody>
</table>

- Refer to the INI_DYN_DISP built-in procedure for a description of the other parameters listed above.

**See Also:** CNC_DYN_DISI, DEF_WINDOW Built-In Procedure

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

### A.10.8 INI_DYN_DISP Built-In Procedure

**Purpose:** Initiates the dynamic display of a value of a port in a specified window, based on the port type and port number.

**Syntax:** INI_DYN_DISP (port_type, port_no, window_name, field_width, attr_mask, char_size, row, col, interval, strings, status)

**Input/Output Parameters:**

[in] port_type :INTEGER

[in] port_no :INTEGER

[in] window_name :STRING

[in] field_width :INTEGER

[in] attr_mask :INTEGER

[in] char_size :INTEGER
Details:

- **port_type** specifies the type of port to be displayed. Codes are defined in FROM: KLIOTYPES.KL.
  - If the **port_type** is a BOOLEAN port (e.g., DIN), if the is FALSE, strings[1] is displayed; If variable is TRUE, strings[2] is displayed.
  - If the **port_type** is an INTEGER port (e.g., GIN), if the value of the port is n, strings[n+1] will be displayed. If the value of the port is greater than or equal to the length of the array of strings, a string of ‘?’s is displayed.

- **port_no** specifies the port number to be displayed.
- Refer to the INI_DYN_DISR built-in procedure for a description of other parameters listed above.

**See Also:** CNC_DYN_DISP Built-In procedure

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

### A.10.9 INI_DYN_DISR Built-In Procedure

**Purpose:** Initiates the dynamic display of a REAL variable in a specified window.

**Syntax:** INI_DYN_DISR(r_var, window_name, field_width, attr_mask, char_size, row, col, interval, buffer_size, format, status)

**Input/Output Parameters:**

- [in] r_var :REAL
- [in] window_name :STRING
- [in] field_width :INTEGER
- [in] attr_mask :INTEGER
- [in] char_size :INTEGER
[in] row :INTEGER
[in] col :INTEGER
[in] interval :INTEGER
[in] buffer_size :INTEGER
[in] format :STRING
[out] status :INTEGER

%ENVIRONMENT Group :UIF

**Details:**

- *r_var* is the REAL variable whose dynamic display is to be initiated.
- If *field_width* is non-zero, the display is extended with blanks if *r_var* is shorter than this specified width. The area is cleared when the dynamic display is canceled.
- Refer to the INI_DYN_DISI built-in procedure for a description of other parameters listed above.

**See Also:** CNC_DYN_DISR Built-In Procedure

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

### A.10.10 INI_DYN_DISS Built-In Procedure

**Purpose:** Initiates the dynamic display of a STRING variable in a specified window.

**Syntax:** INI_DYN_DISS(s_var, window_name, field_width, attr_mask, char_size, row, col, interval, buffer_size, format, status)

**Input/Output Parameters:**

- [in] s_var :STRING
- [in] window_name :STRING
- [in] field_width :INTEGER
- [in] attr_mask :INTEGER
- [in] char_size :INTEGER
- [in] row :INTEGER
[in] col :INTEGER
[in] interval :INTEGER
[in] buffer_size :INTEGER
[in] format :STRING
[out] status :INTEGER

%ENVIRONMENT Group :UIF

**Details:**

- `s_var` is the STRING variable whose dynamic display is to be initiated.
- If `field_width` is non-zero, the display is extended with blanks if `s_var` is shorter than this specified width. The area is cleared when the dynamic display is canceled.
- Refer to the INI_DYN_DISI built-in procedure for a description of other parameters listed above.

**See Also:** CNC_DYN_DISI, INI_DYN_DISI Built-In Procedures

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

### A.10.11 INIT_QUEUE Built-In Procedure

**Purpose:** Sets a queue variable entry to have no entries in the queue

**Syntax:** INIT_QUEUE(queue)

Input/Output Parameters:

[out] queue_t :QUEUE_TYPE

%ENVIRONMENT Group : PBQMGR

**Details:**

- `queue_t` is the queue to be initialized

**See Also:** GET_QUEUE, MODIFY_QUEUE Built-In Procedures, QUEUE_TYPE Data Type, Section 15.8, "Using Queues for Task Communication"

**Example:** The following example initializes a queue called `job_queue`.

A.10.12 INIT_TBL Built-In Procedure

**Purpose:** Initializes a table on the teach pendant

**Syntax:** INIT_TBL(dict_name, ele_number, num_rows, num_columns, col_data, inact_array, change_array, value_array, vptr_array, table_data, status)

**Input/Output Parameters:**

[in] dict_name :STRING

[in] ele_number :INTEGER

[in] num_rows :INTEGER

[in] num_columns :INTEGER

[in] col_data :ARRAY OF COL_DESC_T

[in] inact_array :ARRAY OF BOOLEAN

[in] change_array :ARRAY OF ARRAY OF BOOLEAN

[in] value_array :ARRAY OF STRING

[out] vptr_array :ARRAY OF ARRAY OF INTEGER

[in,out] table_data :XWORK_T

[out] status :INTEGER

%ENVIRONMENT Group :UIF

**Details:**

- The INIT_TBL and ACT_TBL built-in routines should only be used instead of DISCTRL_TBL
if special processing needs to be done with each keystroke or if function key processing needs to be done without exiting the table menu.

- INIT_TBL must be called before using the ACT_TBL built-in. INIT_TBL does not need to be called if using the DISCTRL_TBL built-in.

- dict_name is the four-character name of the dictionary containing the table header.

- ele_number is the element number of the table header.

- num_rows is the number of rows in the table.

- num_columns is the number of columns in the table.

- col_data is an array of column descriptor structures, one for each column in the table. It contains the following fields:
  - item_type : Data type of values in this column. The following data type constants are defined:
    TPX_INT_ITEM — Integer type
    TPX_REL_ITEM — Real type
    TPX_FKY_ITEM — Function key enumeration type
    TPX_SUB_ITEM — Subwindow enumeration type
    TPX_KST_ITEM — KAREL string type
    TPX_KSL_ITEM — KAREL string label type (can select, not edit)
    TPX_KBL_ITEM — KAREL boolean type
    TPX_BYT_ITEM — Byte type
    TPX_SHT_ITEM — Short type
    TPX_PBL_ITEM — Port boolean type
    TPX_PIN_ITEM — Port integer type
  - start_col : Starting character column (1..40) of the display field for this data column.
  - field_width : Width of the display field for this data column.
  - num_ele : Dictionary element used to display values for certain data types. The format of the dictionary elements for these data types are as follows:
    - TPX_FKY_ITEM: The enumerated values are placed on function key labels. There can be up to 2 pages of function key labels, for a maximum of 10 labels. Each label is a string of up to 8 characters. However, the last character of a label which is followed by another label should be left blank or the two labels will run together.
A single dictionary element defines all of the label values. Each value must be put on a separate line using &new_line. The values are assigned to the function keys F1..F5, F6..F10 and the numeric values 1..10 in sequence. Unlabeled function keys should be left blank. If there are any labels on the second function key page, F6..F10, the labels for keys 5 and 10 must have the character “>” in column 8. If there are no labels on keys F6..F10, lines do not have to be specified for any key label after the last non-blank label.

Example:

```
$ example_fkey_label_c
  "" &new_line
  "F2" &new_line
  "F3" &new_line
  "F4" &new_line
  "F5 >" &new_line
  "F6" &new_line
  "F7" &new_line
  "" &new_line
  "">
```

-- TPX_SUB_ITEM: The enumerated values are selected from a subwindow on the display device. There can be up to 5 subwindow pages, for a maximum of 35 values. Each value is a string of up to 16 characters.

A sequence of consecutive dictionary elements, starting with enum_dict, define the values. Each value must be put in a separate element, and must not end with &new_line. The character are assigned the numeric values 1..35 in sequence. The last dictionary element must be "\a".

Example:

```
$ example_sub_win_enum_c
  "Red"
  $
  "Blue"
  $
  "Green"
  $
  "Yellow"
  $
  "\a"
```

-- TPX_KBL_ITEM, TPX_PBL_ITEM: The "true" and "false" values are placed on function key labels F4 and F5, in that order. Each label is a string of up to 8 characters. However, the last character of the "true" label should be left blank or the two labels will run together.
A single dictionary element the label values. Each value must be put on a separate line using &new_line, with the “false” value first.

**Example:**
```
$ example_boolean_c
"OFF" &new_line
"ON"
```

— `enum_dict`: Dictionary name used to display data types TPX_FKY_ITEM, TPX_SUB_ITEM, TPX_KBL_ITEM, or TPX_PBL_ITEM

— `format_spec`: Format string is used to print out the data value. The format string contains a format specifier. The format string can also contain any desired characters before or after the format specifier. The format specifier itself begins with a % and ends with a conversion character. Between the % and the conversion character there may be, in order:

- Flags (in any order), which modify the specification:
  - `:` specifies left adjustment of this field.
  - `+`: specifies that the number will always be printed with a sign.
  - `space`: if the first character is not a sign, a space will be prefixed.
  - `0`: specifies padding a numeric field width with leading zeroes.

- A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.

- A period, which separates the field width from the precision.

- A number, the precision, that specifies the maximum number of characters to be printed from a string, or the number of digits after the decimal point of a floating-point value, or the minimum number of digits for an integer.

• The format specifier must contain one of the conversion characters in the following table:

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type; Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>INTEGER; decimal number.</td>
</tr>
<tr>
<td>o</td>
<td>INTEGER; unsigned octal notation (without a leading zero).</td>
</tr>
<tr>
<td>x,X</td>
<td>INTEGER; unsigned hexadecimal notation (without a leading 0x or 0X), using abcddef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>u</td>
<td>INTEGER; unsigned decimal notation.</td>
</tr>
</tbody>
</table>
### Table A-13. Conversion Characters (Cont’d)

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument Type; Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>STRING; print characters from the string until end of string or the number of characters given by the precision.</td>
</tr>
<tr>
<td>f</td>
<td>REAL; decimal notation of the form [-]mmm.dddddd, where the number of d's is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.</td>
</tr>
<tr>
<td>e,E</td>
<td>REAL; decimal notation of the form [-]m.dddddde+-xx or [-]m.ddddddE+-xx, where the number of d's is given by the precision. The default precision is 6; a precision of 0</td>
</tr>
<tr>
<td>g,G</td>
<td>REAL; %e or %E is used if the exponent is less than -4 or greater than or equal to the precision; otherwise %f is used. Trailing zeros and a trailing decimal pointer are not printed.</td>
</tr>
<tr>
<td>%</td>
<td>no argument is converted; print a %.</td>
</tr>
</tbody>
</table>

**Example:** "%d" or "%-10s"

The format specifiers which can be used with the data types specified in the item_type field in `col_data` are as follows:

- TPX_INT_ITEM %d, %o, %x, %X, %u
- TPX_REL_ITEM %f, %e, %E, %g, %G
- TPX_FKY_ITEM %s
- TPX_SUB_ITEM %s
- TPX_KST_ITEM %s
- TPX_KSL_ITEM %s
- TPX_KBL_ITEM %s
- TPX_BYT_ITEM %d, %o, %x, %X, %u, %c
- TPX_SHT_ITEM %d, %o, %x, %X, %u
- TPX_PBL_ITEM %s
- TPX_PIN_ITEM %d, %o, %x, %X, %u

— *max_integer*: Maximum value if data type is TPX_INT_ITEM, TPX_BYT_ITEM, or TPX_SHT_ITEM.
— *min_integer*: Minimum value if data type is TPX_INT_ITEM, TPX_BYT_ITEM, or TPX_SHT_ITEM.

— *max_real*: Maximum value for reals.

— *min_real*: Minimum value for reals.

— *clear_flag*: If data type is TPX_KST_ITEM, 1 causes the field to be cleared before entering characters and 0 causes it not to be cleared.

— *lower_case*: If data type is TPX_KST_ITEM, 1 allows the characters to be input to the string in upper or lower case and 0 restricts them to upper case.

• *inact_array* is an array of booleans that corresponds to each column in the table.

  — You can set each boolean to TRUE which will make that column inactive. This means the column cannot be cursorred to.

  — The array size can be less than or greater than the number of items in the table.

  — If *inact_array* is not used, then an array size of 1 can be used, and the array does not need to be initialized.

• *change_array* is a two dimensional array of booleans that corresponds to formatted data item in the table.

  — If the corresponding value is set, then the boolean will be set to TRUE, otherwise it is set to FALSE. You do not need to initialize the array.

  — The array size can be less than or greater than the number of data items in the table.

  — If *change_array* is not used, then an array size of 1 can be used.

• *value_array* is an array of variable names that correspond to the columns of data in the table. 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 *[prog_name]* is not specified, then the current program being executed is used.

  — *var_name* must refer to a static, global program variable.

  — *var_name* can contain node numbers, field names, and/or subscripts.

  — Each of these named variables must be a KAREL array of length *num_rows*. Its data type and values should be consistent with the value of the item_type field in *col_data* for the corresponding column, as follows:

    — TPX_INT_ITEM: ARRAY OF INTEGER containing the desired values.

    — TPX_REL_ITEM: ARRAY OF REAL containing the desired values.

    — TPX_FKY_ITEM: ARRAY OF INTEGER with values referring to items in the dictionary element specified in the enum_ele field in *col_data*. There can be at most 2 function key pages, or 10 possible function key enumeration values.

    — TPX_SUB_ITEM: ARRAY OF INTEGER with values referring to items in the dictionary element specified in the enum_ele field in *col_data*. There can be at most 28 subwindow enumeration values.
– TPX_KST_ITEM: ARRAY OF STRING containing the desired values.
– TPX_KST_ITEM: ARRAY OF STRING containing the desired values.
– TPX_KSL_ITEM: ARRAY OF STRING containing the desired values. These values cannot be edited by the user. If one is selected, ACT_TBL will return.
– TPX_KBL_ITEM: ARRAY OF BOOLEAN containing the desired values. The dictionary element specified by the enum_ele field in col_data should have exactly two elements, with the false item first and the true item second. TPX_BYT_ITEM: ARRAY OF BYTE containing the desired values. "--" TPX_SHT_ITEM: ARRAY OF SHORT containing the desired values. "--" TPX_PBL_ITEM: ARRAY OF STRING containing the names of the ports, for example “DIN[5]”. "--" TPX_PIN_ITEM: ARRAY OF STRING containing the names of the ports, for example “GOUT[3]”.
– TPX_BYT_ITEM: ARRAY OF BYTE containing the desired values.
– TPX_SHT_ITEM: ARRAY OF SHORT containing the desired values.
– TPX_PBL_ITEM: ARRAY OF STRING containing the names of the ports, for example “DIN[5]”.
– TPX_PIN_ITEM: ARRAY OF STRING containing the names of the ports, for example “GOUT[3]”.

• vptr_array is an array of integers that corresponds to each variable name in value_array. Do not change this data; it is used internally.
• table_data is used to display and control the table. Do not change this data; it is used internally.
• status explains the status of the attempted operation. If not equal to 0, then an error occurred.

Example: In this example, TPXTABEG.TX is loaded as ‘XTAB’ on the controller. TPEXTBL calls INIT_TBL to initialize a table with five columns and four rows. It calls ACT_TBL in a loop to read and process each key pressed.

INIT_TBL Built-In Procedure

```
TPXTABEG.TX
----------------------------------------------
$title
&reverse "DATA Test Schedule" &standard &new_line
"E1: " &new_line
" W(mm) TEST C(%%) G(123456) COLOR"
^1
?2
$function_keys
"f1" &new_line
"f2" &new_line
"f3" &new_line
"f4" &new_line
" HELP >" &new_line
"f6" &new_line
```

A–196
"f7" & new_line
"f8" & new_line
"f9" & new_line
"f10 >"
$help_text "Help text goes here...
"$enum1
"" & new_line
"" & new_line
"TRUE" & new_line
"FALSE" & new_line
"
$enum2
"Red"
$
"Blue"
$
"Green"
$
"Yellow"
$
"Brown"
$
"Pink"
$
"Mauve"
$
"Black"
$
"......"
--------------------------------------------
TPEXTBL.KL
--------------------------------------------
PROGRAM tpextbl
%ENVIRONMENT uif
%INCLUDE FROM:\klevccdf
%INCLUDE FROM:\klevkeys
VAR
  dict_name: STRING[6]
  ele_number: INTEGER
  num_rows: INTEGER
  num_columns: INTEGER
  col_data: ARRAY[5] OF COL_DESC_T
  change_array: ARRAY[4,5] OF BOOLEAN
  vpstr_array: ARRAY[4,5] OF INTEGER
  table_data: XWORK_T
  status: INTEGER
action: INTEGER
def_item: INTEGER
term_char: INTEGER
attach_sw: BOOLEAN
save_action: INTEGER
done: BOOLEAN
value1: ARRAY[4] OF INTEGER
value2: ARRAY[4] OF INTEGER
value5: ARRAY[4] OF INTEGER

BEGIN
  def_item = 1
  value_array[1] = ‘value1’
  value_array[2] = ‘value2’
  value_array[3] = ‘value3’
  value_array[4] = ‘value4’
  value_array[5] = ‘value5’
  value1[1] = 21
  value1[2] = 16
  value1[3] = 1
  value1[4] = 4
  value2[1] = 3
  value2[2] = 2
  value2[3] = 3
  value2[4] = 2
  value3[1] = -13
  value3[2] = 4.1
  value3[3] = 23.9
  value3[4] = -41
  value4[1] = ‘XXX---’
  value4[2] = ‘--X-X--’
  value4[3] = ‘XXX-XX’
  value4[4] = ‘-X-X--’
  value5[1] = 1
  value5[2] = 1
  value5[3] = 2
  value5[4] = 3
  inact_array[1] = FALSE
  inact_array[2] = FALSE
  inact_array[3] = FALSE
  inact_array[5] = FALSE
  col_data[1].item_type = TPX_INT_ITEM
  col_data[1].start_col = 6
  col_data[1].field_width = 4
  col_data[1].format_spec = ‘%3d’
col_data[1].max_integer = 99
col_data[1].min_integer = -99
col_data[2].item_type = TPX_FKY_ITEM
col_data[2].start_col = 12
col_data[2].field_width = 5
col_data[2].format_spec = '%s'
col_data[2].enum_ele = 3 -- enum1 element number
col_data[2].enum_dict = 'XTAB'
col_data[3].item_type = TPX_REL_ITEM
col_data[3].start_col = 18
col_data[3].field_width = 5
col_data[3].format_spec = '%3.1f'
col_data[4].item_type = TPX_KST_ITEM
col_data[4].start_col = 26
col_data[4].field_width = 6
col_data[4].format_spec = '%s'
col_data[5].item_type = TPX_SUB_ITEM
col_data[5].start_col = 34
col_data[5].field_width = 6
col_data[5].format_spec = '%s'
col_data[5].enum_ele = 4 -- enum2 element number
col_data[5].enum_dict = 'XTAB'
dict_name = 'XTAB'
ele_number = 0 -- title element number
num_rows = 4
num_columns = 5
def_item = 1
attach_sw = TRUE
INIT_TBL(dict_name, ele_number, num_rows, num_columns, col_data,
        inact_array, change_array, value_array, vptr_array,
        table_data, status)
IF status <> 0 THEN
    WRITE('INIT_TBL status = ', status, CR);
ELSE
    def_item = 1
    -- Initial display of table
    ACT_TBL(ky_disp_updt, def_item, table_data, term_char,
            attach_sw, status)
    IF status <> 0 THEN
        WRITE(CR, 'ACT_TBL status = ', status)
    ENDIF
ENDIF
IF status = 0 THEN
    -- Loop until a termination key is selected.
done = FALSE
    action = ky_reissue -- read new key
    WHILE NOT done DO
        A–199
-- Read new key, act on it, and return it
ACT_TBL(action, def_item, table_data, term_char,
    attach_sw, status) save_action = action
action = ky_reissue -- read new key
-- Debug only
WRITE TPERROR (CHR(cc_home) + CHR(cc_clear_win))
-- Process termination keys.
SELECT (term_char) OF
    CASE (ky_select, ky_new_menu):
        done = TRUE;
    CASE (ky_f1):
        -- Perform F1
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F1 pressed’)
    CASE (ky_f2):
        -- Perform F2
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F2 pressed’)
    CASE (ky_f3):
        -- Perform F3
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F3 pressed’)
    CASE (ky_f4):
        -- Perform F4
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F4 pressed’)
    CASE (ky_f5):
        -- Perform F5 Help
        action = ky_help
    CASE (ky_f6):
        -- Perform F6
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F6 pressed’)
    CASE (ky_f7):
        -- Perform F7
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F7 pressed’)
    CASE (ky_f8):
        -- Perform F8
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F8 pressed’)
    CASE (ky_f9):
        -- Perform F9
        SET_CURSOR(TPERROR, 1, 1, status)
        WRITE TPERROR (‘F9 pressed’)
    CASE (ky_f10):
        -- Perform F10
SET_CURSOR(TPERROR, 1, 1, status)
WRITE TPERROR ('F10 pressed')
CASE (ky_undef):
  -- Process special keys.
SELECT (save_action) OF
  CASE (ky_f1_s):
    -- Perform Shift F1
    SET_CURSOR(TPERROR, 1, 1, status)
    WRITE TPERROR ('F1 shifted pressed')
  ELSE:
    ENDSELECT
ELSE:
  action = term_char -- act on this key
ENDSELECT
ENDWHILE
IF term_char <> ky_new_menu THEN
  -- Cancel the dynamic display
  ACT_TBL(ky_cancel, def_item, table_data, term_char,
          attach_sw, status)
ENDIF
ENDIF
END tpextbl

A.10.13  IN_RANGE Built-In Function

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

Syntax : IN_RANGE(posn)

Function Return Type : BOOLEAN

Input/Output Parameters:

[in] posn : XYZWPREXT

%ENVIRONMENT Group :SYSTEM

Details:

- The returned value is TRUE if posn is within the work envelope of the group of axes; otherwise, FALSE is returned.
- The current $UFRAME and $UTOOL are applied to posn.
See Also: CHECK_EPROS Built-in procedure

Example: The following example checks to see if the new position is in the work envelope before moving the TCP to it.

**IN_RANGE Built-In Function**

```plaintext
IF IN_RANGE(part_slot) THEN
  SET_POS_REG(1, part_slot, status)
  move_to_pr — Call TP program to move to PR[1]
ELSE WRITE('I can’t get there!',CR)
ENDIF
```

### A.10.14 INSERT_NODE Built-In Procedure

**Purpose:** Inserts an uninitialized node in the specified PATH argument preceding the specified path node number

**Syntax:** INSERT_NODE(path_var, node_num, status)

Input/Output Parameters:

- [in] path_var :PATH
- [in] node_num :INTEGER
- [out] status :INTEGER

%ENVIRONMENT Group :PATHOP

**Details:**

- `node_num` specifies the index number of the path node before which the new uninitialized node is to be inserted.
- The new node can be assigned values by directly referencing its NODEDATA structure.
- All nodes following the inserted node are renumbered.
- Valid `node_num` values are in the range `node_num => 1` and `node_num <= PATH_LEN(path_var)`.
- If `node_num` is not a valid node number, status is returned with an error.
- If the program does not have enough RAM for an INSERT_NODE request, an error will occur.
- If the program is paused, the INSERT_NODE request is NOT retried.

See Also: DELETE_NODE, APPEND_NODE Built-in Procedures
Example: In the following example, the PATH_LEN built-in is used to set the variable length equal to the number of nodes in path_var. INSERT_NODE inserts a new path node before the last node in path_var.

**INSERT_NODE Built-In Procedure**

```plaintext
length = PATH_LEN(path_var)
INSERT_NODE(path_var, length, status)
```

**A.10.15 INSERT_QUEUE Built-In Procedure**

**Purpose:** Inserts an entry into a queue if the queue is not full

**Syntax:** INSERT_QUEUE(value, sequence_no, queue, 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

**Details:**

- **value** specifies the value to be inserted into the queue, queue_t.
- **sequence_no** specifies the sequence number of the entry before which the new entry is to be inserted.
- **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 61002, “Queue is full,” if there is no room for the entry in the queue, with 61003, “Bad sequence no,” if the specified sequence number is not in the queue.

**See Also:** MODIFY_QUEUE, APPEND_QUEUE, DELETE_QUEUE Built-In Procedures, Section 15.8, "Using Queues for Task Communication"
Example: In the following example, the routine `ins_in_queue` adds an entry `value` to a queue (`queue_t` and `queue_data`) following the specified entry `sequence_no`; it returns TRUE if this was successful; otherwise it returns FALSE.

**INSERT_QUEUE Built-In Procedure**

```karel
PROGRAM ins_queue_x
%environment PBQMGR
ROUTINE ins_in_queue(value: INTEGER;
    sequence_no: INTEGER;
    queue_t: QUEUE_TYPE;
    queue_data: ARRAY OF INTEGER): BOOLEAN

VAR
    status: INTEGER
BEGIN
    INSERT_QUEUE(value, sequence_no, queue_t, queue_data, status)
    return (status = 0)
END ins_in_queue
END ins_queue_x
```

### A.10.16 INTEGER Data Type

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

**Syntax:** INTEGER

**Details:**

- An INTEGER variable or parameter can assume whole number values in the range -2147483648 through +2147483646.
- INTEGER literals consist of a series of digits, optionally preceded by a plus or minus sign. They cannot contain decimal points, commas, spaces, dollar signs ($), or other punctuation characters. (See Table A–14)

#### Table A–14. Valid and Invalid INTEGER Literals

<table>
<thead>
<tr>
<th>Valid</th>
<th>Invalid</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
<td>Decimal point not allowed (must be a whole number)</td>
</tr>
<tr>
<td>-2500450</td>
<td>-2,500,450</td>
<td>Commas not allowed</td>
</tr>
<tr>
<td>+65</td>
<td>+6 5</td>
<td>Spaces not allowed</td>
</tr>
</tbody>
</table>

- If an INTEGER argument is passed to a routine where a REAL parameter is expected, it is treated as a REAL and passed by value.
Only INTEGER expressions can be assigned to INTEGER variables, returned from INTEGER function routines, or passed as arguments to INTEGER parameters.

Valid INTEGER operators are:

- Arithmetic operators (+, -, *, /, DIV, MOD)
- Relational operators (>, >=, =, <>, <, <=)
- Bitwise operations (AND, OR, NOT)

**See Also:** Chapter 5 ROUTINES, for more information on passing by value, Chapter 7 FILE INPUT/OUTPUT OPERATIONS, for more information on format specifiers

**Example:** Refer to Appendix B for detailed program examples.

### A.10.17 INV Built-In Function

**Purpose:** Used in coordinate frame transformations with the relative position operator (:) to determine the coordinate values of a POSITION in a frame that differs from the frame in which that POSITION was recorded

**Syntax:** INV(pos) Function Return Type :POSITION

Input/Output Parameters:

[in] pos :POSITION

%ENVIRONMENT Group :SYSTEM

**Details:**

- The returned value is the inverse of the pos argument.
- The configuration of the returned POSITION will be that of the pos argument.

**Example:** The following example uses the INV built-in to determine the POSITION of part_pos with reference to the coordinate frame that has rack_pos as its origin. Both part_pos and rack_pos were originally taught and recorded in User Frame. The robot is then instructed to move to that position.

#### INV Built-In Function

```karel
PROGRAM p_inv
VAR
  rack_pos, part_pos, p1 : POSITION
BEGIN
  p1 = INV(rack_pos):part_pos
  SET_POS_REG(1, p1, status)
  move_to_pr1 -- Call TP program to move to PR[1]
END p_inv
```

---

---
A.10.18 IO_MOD_TYPE Built-In Procedure

**Purpose:** Allows a KAREL program to determine the type of module in a specified rack/slot

**Syntax:** IO_MOD_TYPE(rack_no, slot_no, mod_type, status)

Input/Output Parameters:

- [in] rack_no :INTEGER
- [in] slot_no :INTEGER
- [out] mod_type :INTEGER
- [out] status :INTEGER

**Details:**

- **rack_no** is the rack containing the port module. For process I/O boards, 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 position of the board in the SLC-2 chain.
- **mod_type** is the module type.
- 6 A16B-2202-470
- 7 A16B-2202-472
- 8 A16B-2202-480
- **status** is returned with zero if the parameters are valid and there is a module or board with the specified rack/slot number as follows:

**Example:** The following example returns to the caller the module in the specified rack and slot number.

**IO_MOD_TYPE Built-In Procedure**

```karel
PROGRAM iomodtype
%ENVIRONMENT IOSETUP
ROUTINE get_mod_type(rack_no: INTEGER;
                      slot_no: INTEGER;
                      mod_type: INTEGER): INTEGER
```
VAR
    status: INTEGER
BEGIN
    IO_MOD_TYPE(rack_no, slot_no, mod_type, status)
    RETURN (status)
END get_mod_type
BEGIN
END iomodtype

A.10.19 IO_STATUS Built-In Function

**Purpose:** Returns an INTEGER value indicating the success or type of failure of the last operation on the file argument

**Syntax:** IO_STATUS(file_id)

Function Return Type :INTEGER

Input/Output Parameters:

[in] file_id :FILE

%ENVIRONMENT Group :PBCORE

**Details:**

- IO_STATUS can be used after an OPEN FILE, READ, WRITE, CANCEL FILE, or CLOSE FILE statement. Depending on the results of the operation, it will return 0 if successful or one of the errors listed in the application-specific *FANUC Robotics Setup and Operations Manual*. Some of the common errors are shown in Table A–15.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Last operation on specified file was successful</td>
</tr>
<tr>
<td>2021</td>
<td>End of file for RAM disk device</td>
</tr>
<tr>
<td>10006</td>
<td>End of file for floppy device</td>
</tr>
<tr>
<td>12311</td>
<td>Uninitialized variable</td>
</tr>
<tr>
<td>12324</td>
<td>Illegal open mode string</td>
</tr>
<tr>
<td>12325</td>
<td>Illegal file string</td>
</tr>
<tr>
<td>12326</td>
<td>File var is already used</td>
</tr>
<tr>
<td>12327</td>
<td>Open file failed</td>
</tr>
</tbody>
</table>
Table A–15. IO_STATUS Errors (Cont’d)

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

- 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.
- CVN_REL_JPOS and CVN_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 \textit{in\_jnt} is defined as the joint angles to be converted to the Cartesian position.
• The input \textit{uframe} is the user frame for the Cartesian position.
• The input \textit{utool} is defined as the corresponding tool frame.
• The input \textit{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 \textit{out\_pos} is the Cartesian position corresponding to the input joint angles.
• The output \textit{wjnt\_cfg} is the wrist joint configuration. The value will be output when \textit{config\_ref} corresponds to HALF\_SOLN.
• 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`

**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.

```karel
KCL Built-In Procedure
PROGRAM kcl_test
VAR
  command :STRING[20]
  status :INTEGER
BEGIN
```
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
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**

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

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

**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-30iB 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)

### 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

**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:
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

**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>
<thead>
<tr>
<th>GROUP</th>
<th>DECIMAL</th>
<th>BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Group 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Group 3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

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.
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
```

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**

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

---

A–218
MARRC75KR07091E Rev D     A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

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).

<table>
<thead>
<tr>
<th>GROUP</th>
<th>DECIMAL</th>
<th>BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Group 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Group 3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

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.

```karel
MOVE_FILE Built-In Procedure

PROGRAM move_files
```
%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 tcpser8
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
    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 = i/o_status(file_var)
        ENDFOR
        CLOSE FILE file_var
        write ('Disconnecting..',cr)
        MSG_DISCO('S8:',status)
        write ('Done.',cr)
    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.

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
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:
• 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**

```plaintext
PROGRAM read_and_mov
VAR my_path: PATH
  xyz_pos: XYZWR
  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
```
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)
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION MARRC75KR07091E Rev D

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)
```

A–230
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

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. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

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)
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`.

```karel
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
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.
- \textit{str\_index} specifies the indexed position of a character in the argument \textit{str}. A value of 1 indicates the first character.
- If \textit{str\_index} is less than one or greater than the current length of \textit{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.

\section*{A.16.5 ORIENT Built-In Function}

\textbf{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 \textit{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**

```plaintext
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**

```plaintext
-- 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
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

### 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
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

**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.

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

**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)
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
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

[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.

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.

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`

**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:**
The input \textit{ref\_int} are the reference joint angles that represent the robot's position just before moving to the current position.

The input \textit{in\_pos} is the robot Cartesian position to be converted to joint angles.

The input \textit{uframe} is the user frame for the Cartesian position.

The input \textit{utool} is the corresponding tool frame.

The input \textit{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: \texttt{config\_ref = HALF\_SOLN + CONFIG\_TCP}.

- \texttt{0} :(FULL\_SOLN) = Default
- \texttt{1} : (HALF\_SOLN) = Wrist joint (XYZ456). This does not calculate/use WPR.
- \texttt{2} :(CONFIG\_TCP) = The Wrist Joint Config (up/down) is based on the fixed wrist.
- \texttt{4} :(APPROX\_SOLN) = Approximate solution. Reduce calculation time for some robots.
- \texttt{8} :(NO\_TURNS) = Ignore wrist turn numbers. Use the closest path for joints 4, 5 and 6 (uses \texttt{ref\_int}).
- \texttt{16} :(NO\_M\_TURNS) = Ignore major axis (J1 only) turn number. Use the closest path.

The input \textit{wjnt\_cfg} is the wrist joint configuration. This value must be input when \textit{config\_ref} corresponds to HALF\_SOLN.

The input \textit{ext\_ang} contains the values of the joint angles for the extended axes if they exist.

The output \textit{out\_int} 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

**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
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

```karel
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:**
• 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 ( \geq < ) 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:

  **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
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
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.

### 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

**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”.

%ENVIRONMENT Group: CORE
• 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.
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.
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.
• 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.

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

**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.

```karel
VAR status: INTEGER
BEGIN
  PROG_CLEAR('ANS00003.TP', PBR_MNTYPE, status)
END
```
**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.
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.TP', 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)
```
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
• **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:

```pascal
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.
• 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 \textit{CONDITION HANDLERS} , Chapter 7 \textit{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.

\section*{A.17.27 PULSE Statement}

\textbf{Purpose:} Pulses a digital output port for a specified number of milliseconds.

\textbf{Syntax :} PULSE DOUT[port\_no]\ FOR time\_in\_ms < NOWAIT >

where:

port\_no : an INTEGER variable or literal

time\_in\_ms : an INTEGER

\textbf{Details:}

• \textit{port\_no} must be a valid digital output port number.

• \textit{time\_in\_ms} specifies the duration of the pulse in milliseconds.

• If \textit{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:

\begin{verbatim}
DOUT[n] = FALSE
DELAY x
DOUT[n] = TRUE
\end{verbatim}
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[5] will be executed before DOUT[start_air] is completed.

PULSE Statement

\[
\begin{align*}
PULSE & \text{ DOUT}[5] \text{ FOR (seconds} \times 1000) \text{ NOWAIT} \\
PULSE & \text{ DOUT}[\text{start_air}] \text{ FOR 50 NOWAIT}
\end{align*}
\]

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

\[
\begin{align*}
& \text{IF ignore cond THEN} \\
& \text{PURGE CONDITION}[1]
\end{align*}
\]
### 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.
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

**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)
• `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.

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)
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

**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**

```
$ "Maximum number of tasks = %d"
```

**UTILITY PROGRAM:**

```
PROGRAM util_prog
  %ENVIRONMENT uif
  VAR
    ksta: ARRAY[1] OF STRING[40]
    status: INTEGER
  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] status : 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 STP_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 klevkmstk.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)

- 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.
• `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:

```plaintext
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**

```plaintext
PROGRAM readkb
%NOLOCKGROUP
%ENVIRONMENT flbt
%ENVIRONMENT uif
%INCLUDE FR:eklevkmsk
VAR
  file_var: FILE
  key: INTEGER
```
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 \(-3.4028236 \times 10^{38}\) through \(-1.175494 \times 10^{-38}\), 0.0, and from \(+1.175494 \times 10^{-38}\) through \(+3.4028236 \times 10^{38}\), 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 \(-1.175494 \times 10^{-38}\) and \(+1.175494 \times 10^{-38}\) 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**

<table>
<thead>
<tr>
<th>VALID</th>
<th>INVALID</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>15</td>
<td>Decimal point is required (15 is an INTEGER not a REAL)</td>
</tr>
<tr>
<td>1</td>
<td>.</td>
<td>Must include an INTEGER or a fractional part</td>
</tr>
<tr>
<td>+2500.450</td>
<td>+2,500.450</td>
<td>Commas not allowed</td>
</tr>
<tr>
<td>1.25E-4</td>
<td>1.25E -4</td>
<td>Spaces not allowed</td>
</tr>
</tbody>
</table>

**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)
A.19.6 Relational Condition

**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.

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

**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`

**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

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.
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}]

**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.
- Those motions 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 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

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 rtin.”

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)

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 >

- 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)
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`

**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
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>
<thead>
<tr>
<th>GROUP</th>
<th>DECIMAL</th>
<th>BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Group 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Group 3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

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.

### 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)
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

**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
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:
• `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

```karel
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
[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**

```plaintext
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
%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)
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

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:
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**

```plaintext
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

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`

**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.
• 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"

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.

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.

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 ] 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 be set 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

```karel
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
```

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 KLIOTYPES.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>
<thead>
<tr>
<th>ATR_TYPE</th>
<th>ATR_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>atr_readahd</td>
<td>any integer, represents multiples of 128 bytes (for example: atr_value=1 means the buffer length is 128 bytes.)</td>
</tr>
</tbody>
</table>
| atr_baud    | baud_9600
              | baud_4800
              | baud_2400
              | baud_1200 |
| atr_parity  | parity_none
              | parity_even
              | parity_odd |
### Table A–20. Attribute Values (Cont’d)

<table>
<thead>
<tr>
<th>ATR_TYPE</th>
<th>ATR_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>atr_sbits</td>
<td>sbits_1</td>
</tr>
<tr>
<td></td>
<td>sbits_15</td>
</tr>
<tr>
<td></td>
<td>sbits_2</td>
</tr>
<tr>
<td>atr_dbits</td>
<td>dbits_5</td>
</tr>
<tr>
<td></td>
<td>dbits_6</td>
</tr>
<tr>
<td></td>
<td>dbits_7</td>
</tr>
<tr>
<td></td>
<td>dbits_8</td>
</tr>
<tr>
<td>atr_xonoff</td>
<td>xf_not_used</td>
</tr>
<tr>
<td></td>
<td>xf_used</td>
</tr>
<tr>
<td>atr_eol</td>
<td>an ASCII code value, Refer to Appendix D, &quot;Character Codes&quot;</td>
</tr>
<tr>
<td>atr_modem</td>
<td>md_not_used</td>
</tr>
<tr>
<td></td>
<td>md_use_dsr</td>
</tr>
<tr>
<td></td>
<td>md_nouse_dsr</td>
</tr>
<tr>
<td></td>
<td>md_use_dtr</td>
</tr>
<tr>
<td></td>
<td>md_nouse_dtr</td>
</tr>
<tr>
<td></td>
<td>md_use_rts</td>
</tr>
<tr>
<td></td>
<td>md_nouse_rts</td>
</tr>
</tbody>
</table>

- 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.

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 to FALSE. 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.

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.

### 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

**Details:**

- `port_type` specifies the code for the type of port to simulate. Codes are defined in KLIOTYPES.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

**Details:**

- *port_type* specifies the code for the type of port whose mode is being set. Codes are defined in KLIOTYPES.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

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

**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

**Purpose:** Stores an XZWPWR 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 : XZWPWR
- **[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 XZWPWR 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

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

[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.
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

**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

**Details:**
- *i* holds the INTEGER representation of time within the KAREL system. This value is represented in 32-bit INTEGER format as follows:
Table A–21. 32–Bit INTEGER Format of Time

<table>
<thead>
<tr>
<th>31–25</th>
<th>24–21</th>
<th>20–16</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>month</td>
<td>day</td>
</tr>
<tr>
<td>15–11</td>
<td>10–5</td>
<td>4–0</td>
</tr>
<tr>
<td>hour</td>
<td>minute</td>
<td>second</td>
</tr>
</tbody>
</table>

• 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_no* 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
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION  MARRC75KR07091E Rev D

[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
[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

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.

<table>
<thead>
<tr>
<th>Caution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using SET_VAR to modify system variables could cause unexpected results.</td>
</tr>
</tbody>
</table>

- 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

```plaintext
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 .
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
```

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.
• \( \text{angle} \) must be in the range of \( \pm 18000 \) 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**

\[
c = \text{SQRT}(a*a+b*b)
\]

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).
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:**

\[
\text{STD\_PTH\_NODE} = \text{STRUCTURE}
\]

- 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.

A.20.48 STOP Action

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

**Syntax:**

\[
\text{STOP } \langle \text{GROUP}[n{,n}] \rangle
\]

**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>
<thead>
<tr>
<th>VALID</th>
<th>INVALID</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>'123456'</td>
<td>123456</td>
<td>Without quotes 123456 is an INTEGER literal</td>
</tr>
</tbody>
</table>

**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)
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

...
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.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)
IF ratio = TAN(angle) THEN
    WRITE (‘ratio is correct’,CR)
ENDIF

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
listing_sw : BOOLEAN
status : INTEGER

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
```


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

```karel
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.

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
```

### 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

<table>
<thead>
<tr>
<th>GROUP</th>
<th>DECIMAL</th>
<th>BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Group 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Group 3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

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.
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

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, 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.

A.23  V - KAREL LANGUAGE DESCRIPTION

A.23.1 V_CAM_CALIB iRVision 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

<table>
<thead>
<tr>
<th>Calibration Type</th>
<th>Function Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Pattern Calibration</td>
<td>Specify the index of the calibration plane: 1 or 2.</td>
</tr>
<tr>
<td>3DL Calibration</td>
<td>Specify the index of the calibration plane: 1 or 2.</td>
</tr>
<tr>
<td>Visual Tracking Calibration</td>
<td>Not supported</td>
</tr>
<tr>
<td>Simple 2D Calibration</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

- `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 iRVision KAREL Interface Software Option (J870).

### Example:

```plaintext
---------------------
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
```
V_GET_OFFSET /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 /RVision KAREL Interface Software Option (J870).

Example Program:

--------------------------------------------
PROGRAM vision
--------------------------------------------

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 /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 iRVision 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)
```

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

A.23.4 V_GET_QUEUE iRVision 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 iRVision KAREL Interface Software Option (J870).

A.23.5 V_INIT_QUEUE iRVision Built-in Procedure

**Purpose:** Initializes the specified queue.

**Syntax:** V_INIT_QUEUE (queue_num, status)

- [in] queue_num: INTEGER
A.23.6 V_RALC_QUEUE iRVision 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

**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 iRVision KAREL Interface Software Option (J870).

A.23.7 V_RUN_FIND iRVision Built-In Procedure

**Purpose:** Starts an iRVision 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:

• \textit{vp\_name} is the name of the vision process created in setup mode
• \textit{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.
• \textit{status} explains the status of the attempted operation. If not equal to 0, then an error occurred.
• This built-in requires the \textit{iRVision 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.23.8  V_SET_REF  iRVision 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 iRVision KAREL Interface Software Option (J870).

A.23.9 V_START_VTRK iRVision 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:

• \textit{vp\_name} is the name of the visual tracking vision process.

• \textit{status} explains the status of the attempted operation. If not equal to 0, then an error occurred.

• After this command is executed, iRVision monitors a specified condition (such as a conveyor move distance). iRVision 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 iRVision resides.

• This built-in requires the iRVision KAREL Interface Software Option (J870).

A.23.10 V_STOP_VTRK iRVision 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:

- \textit{Vp\_name} is the name of the visual tracking vision process.
- \textit{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 \textit{iRVision} resides.
- This built-in requires the \textit{iRVision} KAREL Interface Software Option (J870).

\subsection{A.23.11 VAR\_INFO Built-In Procedure}

\textbf{Purpose:} Allows a KAREL program to determine data type and numerical information regarding internal or external program variables

\textbf{Syntax:} \texttt{VAR\_INFO(prog\_name, var\_name, uninit, type\_nam, type\_value, dims, slen, status)}

\textbf{Input/Output Parameters:}

[in] \texttt{prog\_name}: STRING

[in] \texttt{var\_name}: STRING

[out] \texttt{uninit\_b} : BOOLEAN

[out] \texttt{type\_nam} : STRING

[out] \texttt{dims} : ARRAY[3] OF INTEGER

[out] \texttt{type\_value} : INTEGER

[out] \texttt{status} : INTEGER

[out] \texttt{slen} : INTEGER

%ENVIRONMENT Group : BYNAM

\textbf{Details:}

- \texttt{prog\_name} specifies the name of the program that contains the specified variable. If \texttt{prog\_name} is blank, then it defaults to the current program being executed.
- \texttt{var\_name} must refer to a static program variable.
- \texttt{var\_name} can contain node numbers, field names, and/or subscripts.
- \texttt{uninit\_b} will return a value of TRUE if the variable specified by \texttt{var\_name} is uninitialized and FALSE if the variable specified by \texttt{var\_name} is initialized.
• `type_name` 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>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>1</td>
</tr>
<tr>
<td>XYZWPR</td>
<td>2</td>
</tr>
<tr>
<td>XYZWPREXT</td>
<td>6</td>
</tr>
<tr>
<td>INTEGER</td>
<td>16</td>
</tr>
<tr>
<td>REAL</td>
<td>17</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>18</td>
</tr>
<tr>
<td>VECTOR</td>
<td>19</td>
</tr>
<tr>
<td>VIS_PROCESS</td>
<td>21</td>
</tr>
<tr>
<td>MODEL</td>
<td>22</td>
</tr>
<tr>
<td>SHORT</td>
<td>23</td>
</tr>
<tr>
<td>BYTE</td>
<td>24</td>
</tr>
<tr>
<td>JOINTPOS1</td>
<td>25</td>
</tr>
<tr>
<td>CONFIG</td>
<td>28</td>
</tr>
<tr>
<td>FILE</td>
<td>29</td>
</tr>
<tr>
<td>PATH</td>
<td>31</td>
</tr>
<tr>
<td>CAM_SETUP</td>
<td>32</td>
</tr>
<tr>
<td>JOINTPOS2</td>
<td>41</td>
</tr>
<tr>
<td>JOINTPOS3</td>
<td>57</td>
</tr>
<tr>
<td>JOINTPOS4</td>
<td>73</td>
</tr>
<tr>
<td>JOINTPOS5</td>
<td>89</td>
</tr>
<tr>
<td>JOINTPOS6</td>
<td>105</td>
</tr>
<tr>
<td>JOINTPOS7</td>
<td>121</td>
</tr>
<tr>
<td>JOINTPOS8</td>
<td>137</td>
</tr>
<tr>
<td>JOINTPOS9</td>
<td>153</td>
</tr>
<tr>
<td>JOINTPOS</td>
<td>153</td>
</tr>
<tr>
<td>STRING</td>
<td>209</td>
</tr>
<tr>
<td>user-defined type</td>
<td>210</td>
</tr>
</tbody>
</table>
• `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**

```karel
PROGRAM util_prog
VAR
  counter, i : INTEGER
BEGIN
  counter = 0
  FOR i = 1 TO 10 DO
    counter = counter + 1
  ENDFOR
END util_prog
END task
```

**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**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All variable types</td>
<td>0</td>
</tr>
<tr>
<td>POSITION</td>
<td>1</td>
</tr>
<tr>
<td>XYZWPR</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER</td>
<td>16</td>
</tr>
<tr>
<td>REAL</td>
<td>17</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>18</td>
</tr>
<tr>
<td>VECTOR</td>
<td>19</td>
</tr>
<tr>
<td>VIS_PROCESS</td>
<td>21</td>
</tr>
<tr>
<td>MODEL</td>
<td>22</td>
</tr>
<tr>
<td>SHORT</td>
<td>23</td>
</tr>
<tr>
<td>BYTE</td>
<td>24</td>
</tr>
</tbody>
</table>
### Valid Data Types (Cont’d)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOINTPOS1</td>
<td>25</td>
</tr>
<tr>
<td>CONFIG</td>
<td>28</td>
</tr>
<tr>
<td>FILE</td>
<td>29</td>
</tr>
<tr>
<td>PATH</td>
<td>31</td>
</tr>
<tr>
<td>CAM_SETUP</td>
<td>32</td>
</tr>
<tr>
<td>JOINTPOS2</td>
<td>41</td>
</tr>
<tr>
<td>JOINTPOS3</td>
<td>57</td>
</tr>
<tr>
<td>JOINTPOS4</td>
<td>73</td>
</tr>
<tr>
<td>JOINTPOS5</td>
<td>89</td>
</tr>
<tr>
<td>JOINTPOS6</td>
<td>105</td>
</tr>
<tr>
<td>JOINTPOS7</td>
<td>121</td>
</tr>
<tr>
<td>JOINTPOS8</td>
<td>137</td>
</tr>
<tr>
<td>JOINTPOS9</td>
<td>153</td>
</tr>
<tr>
<td>JOINTPOS</td>
<td>153</td>
</tr>
<tr>
<td>STRING</td>
<td>209</td>
</tr>
<tr>
<td>user-defined type</td>
<td>210</td>
</tr>
</tbody>
</table>

- `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.
• \( n_{\text{vars}} \) is the number of variables stored in \( \text{ary}_\text{name} \).
• \( \text{status} \) 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.
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

**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.
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**

```karel
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('f:', 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 iRVision 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 iRVision KAREL Interface Software Option (J870).

A.23.16 VREG_OFFSET iRVision 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 iRVision 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)

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.

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.
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
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
• **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.

### 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
A. KAREL LANGUAGE ALPHABETICAL DESCRIPTION

- 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
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:

```plaintext
xmlext_t = STRUCTURE
    first: integer
    second: real
    third: BOOLEAN
    fourth: string[20]
ENDSTRUCTURE
```

Can be set via the XML:

```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.
### A.25.6 XZWPR Data Type

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

**Syntax:** XZWPR <IN GROUP [n]>

**Details:**

- An XZWPR 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 XZWPR variables can be accessed or set as if they were defined as follows:

```plaintext
XZWPR = 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)
A.25.7 XZWPREXT Data Type

**Purpose:** Defines a variable, function return type, or routine parameter as an XZWPREXT

**Syntax:** XZWPREXT <IN GROUP [n]>

**Details:**

- An XZWPREXT 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 XZWPREXT variables can be accessed or set as if they were defined as follows:

```plaintext
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
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>
<thead>
<tr>
<th>Program Name</th>
<th>Program Function</th>
<th>Built-ins Used</th>
<th>Section to Refer</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPY_PTH.KL</td>
<td>Copies path variables.</td>
<td>APPEND_NODE</td>
<td>Section B.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BY_NAME</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL_PROG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNV_INT_STR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY_PATH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CREATE_VAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CURPOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE_NODE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PATH_LEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROG_LIST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ_KB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_CURSOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_FILE_ATR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_VAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUB_STR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VAR_LIST</td>
<td></td>
</tr>
<tr>
<td>SAVE_VRS.KL</td>
<td>Saves data to the default device.</td>
<td>DELETE_FILE</td>
<td>Section B.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>Program Name</td>
<td>Program Function</td>
<td>Built-ins Used</td>
<td>Section to Refer</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ROUT_EX.KL</td>
<td>Contains standard routines that are used throughout the program examples.</td>
<td>CHR</td>
<td>Section B.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORCE_SPMENU</td>
<td></td>
</tr>
<tr>
<td>REG_EX.KL</td>
<td>Uses Register built-ins.</td>
<td>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</td>
<td>Section B.5</td>
</tr>
<tr>
<td>PTH_MOVE.KL</td>
<td>Teaches and moves along a path. Also uses condition handlers.</td>
<td>CHR, CNV_REL_JPOS, PATH_LEN, SET_CURSOR</td>
<td>Section B.6</td>
</tr>
<tr>
<td>LIST_EX.KL</td>
<td>Lists files and programs, and manipulate strings.</td>
<td>ABS, ARRAY_LEN, CNV_INT_STR, FILE_LIST, LOAD, LOAD_STATUS, PROG_LIST, ROUND, ROUND, SUB_STR</td>
<td>Section B.7</td>
</tr>
<tr>
<td>GEN_HEX.KL</td>
<td>Generates a hexagon, and moves along the path.</td>
<td>CNV_REL_JPOS, COS, CURPOS</td>
<td>Section B.8</td>
</tr>
<tr>
<td>FILE_EX.KL</td>
<td>Uses the File and Device built-ins.</td>
<td>CNV_TIME_STR, COPY_FILE, DISMOUNT_DEV, FORMAT_DEV, GET_TIME, MOUNT_DEV, SUB_STR</td>
<td>Section B.9</td>
</tr>
</tbody>
</table>

Table B–1. KAREL Example Programs (Cont’d)
<table>
<thead>
<tr>
<th>Program Name</th>
<th>Program Function</th>
<th>Built-ins Used</th>
<th>Section to Refer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYN_DISP.KL</td>
<td>Uses Dynamic Display built-ins.</td>
<td>ABORT_TASK</td>
<td>Section B.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNC_DYN_DISB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNC_DYN_DISE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNC_DYN_DISP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNC_DYN_DISI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNC_DYN_DISR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INI_DYN_DISB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INI_DYN_DISE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INI_DYN_DISP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INI_DYN_DISI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INI_DYN_DISR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOAD_STATUS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RUN_TASK</td>
<td></td>
</tr>
<tr>
<td>CHG_DATA.KL</td>
<td>Processes and changes values of dynamically displayed variables.</td>
<td></td>
<td>Section B.11</td>
</tr>
<tr>
<td>DCLST_EX.KL</td>
<td>Displays a list from a dictionary file.</td>
<td>ADD_DICT</td>
<td>Section B.12.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACT_SCREEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATT_WINDOW_S</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHECK_DICT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLR_IO_STAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNV_STR_INT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEF_SCREEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISCTRL_LIST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORCE_SPENU</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IO_STATUS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ_DICT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>REMOVE_DICT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_FILE_ATR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_WINDOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STR_LEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNINIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE_DICT</td>
<td></td>
</tr>
<tr>
<td>DCLISTEG.UTX</td>
<td>Dictionary file.</td>
<td>N/A</td>
<td>Section B.12</td>
</tr>
</tbody>
</table>
### Table B–1. KAREL Example Programs (Cont’d)

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Program Function</th>
<th>Built-ins Used</th>
<th>Section to Refer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCALP_1X.KL</td>
<td>Uses the DISCTRL_ALPHA Built-in.</td>
<td>ADD_DICT</td>
<td>Section B.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISCTRL_ALPH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORCE_SPEMU</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>POST_ERR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_CURSOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_LANG</td>
<td></td>
</tr>
<tr>
<td>DCALPHEG.UTX</td>
<td>Dictionary file.</td>
<td>N/A</td>
<td>Section B.13.1</td>
</tr>
<tr>
<td>CPY_TP.KL</td>
<td>Applies offsets to copied teach pendant programs.</td>
<td>AVL_POS_NUM</td>
<td>Section B.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLOSE_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNV_JPOS_REL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CNV_REL_JPOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GET_JPOS_TYP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GET_POS_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GET.Pos_TYP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPEN TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROG_LIST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_JPOS_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_POS_TPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET_Pos_TPE</td>
<td></td>
</tr>
</tbody>
</table>
Table B–1. KAREL Example Programs (Cont’d)

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Program Function</th>
<th>Built-ins Used</th>
<th>Section to Refer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUT_EX.KL</td>
<td>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 hooked 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.</td>
<td>CHRPATH_LEN, CURPOS, DELAY, FORCE_SPMENU, RESET, SET_PORT_ASG, SET_PORT_CMT, SET_PORT_MOD, SET_PORT_SIM</td>
<td>Section B.1</td>
</tr>
</tbody>
</table>

B.1 SETTING UP DIGITAL OUTPUT PORTS FOR PROCESS MONITORING

This program sets up digital output ports for process monitoring. The DOUTs 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 they are hooked up to the actual external equipments as a feedback response. The robot is moved along a path until external equipment needs servicing, 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

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_CNT  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...ENDCONDITION 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

----- 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 )

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 )
---

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,
lst_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 :XYZWR -- Perch Position
tmp_xyz :XYZWR -- XYZWR 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
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 5 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
ENDFOR
ENDFOR
Setting Up Digital Output Ports for Process Monitoring - Declare Routines

```plaintext
-- 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
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 ('Please Fix the equipment then',CR)
WRITE ('SET DOUT[',EQIP_FIXED,'] = TRUE ',CR)
--Display the following message in reverse video
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[EQUIP_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 ;

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
    NOMESSAGE
    UNPAUSE
    init_port
  -- Trap UNINITIALIZATION error
  -- Supress the error message
  -- UNPAUSE
  -- Allow ports to be initialized.
ENDCONDITION

ENABLE CONDITION[UNINIT_CH]
CONNECT TIMER to cont_timer
CONDITION [CONT_CH]:
  WHEN cont_timer > 1000 DO
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
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[EQUIP_ERROR] is set to TRUE. Which will stop the motion and require the equipment to be fixed before motion can be resumed.
CONDITION[EQUIP_FAIL]:
  WHEN DOUT[EQUIP_ERROR] DO
    STOP
    DOUT[EQUIP_FIXED] = FALSE
    DOUT[EQUIP_READY] = FALSE
    ENABLE CONDITION[RESTART]
    service_rtn
ENDCONDITION
ENABLE CONDITION[EQUIP_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
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
  END FOR
  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
BYPNAME 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
--- 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

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
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.
  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
--- parameter. 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
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 ;

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
    -- Copy without saving variables.
  ENDIF
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_index) -- 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 unssccessful: Status = ’,status,cr)
    ENDIF
  ENDFOR
ENDIF

--- 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 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
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, status, mem_loc)
IF (status <> SUCCESS) THEN
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
--- 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

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:
--- CRERROR Sec 3-A
--- CRTFUNC Sec 3-A
--- CRTPROMPT Sec 3-A
--- CRTSTATUS Sec 3-A
--- OUTPUT Sec 3-A
--- TERROR 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, CRERROR, 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 CRERROR (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

B-32
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).
- 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] ;
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.
• 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.KI

--- 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
%ncomment = 'Reg-ops'
%alphabetize

--- Section 2: Variable Declaration

CONST
c_c_success = 0 -- Success status
c_c_xyzwpr = 2 -- Position Register has an XYZWPR
c_c_jntpos = 9 -- Position Register has a JOINTPOS

VAR
xyz :XYZWPR
jpos :JOINTPOS
r_val :REAL
prg_index,
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.TP starting on first line.
--Do not pause on entry of PROG_1.
B. KAREL EXAMPLE PROGRAMS

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
  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
---- 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 Sec 3-A, B, 4-D
---- VAR Sec 2 Sec 2
---- PROGRAM Sec 4-D
---- 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.
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 index = 1 to 6 DO
   real_ary[index] = 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
   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
   via_pos.x = via_pos.x +200
   via_pos.y = via_pos.y +200
   -- 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
   des_pos.x = des_pos.x + 400
   -- 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.
   --

Path Variables and Condition Handlers Program - Move Along Path

   tmp_xyz = strt_jnt
   SET_POS_REG(1, tmp_xyz, status)
   move_to_pr
   tmp_xyz = des_pos
   SET_POS_REG(1, tmp_xyz, status)
   move_circ

   tmp_xyz = via_pos
   SET_POS_REG(2, tmp_xyz, status)
   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
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

B.7 LISTING FILES AND PROGRAMS AND MANIPULATING STRINGS

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: Sec 3-B
  Clauses:
  FROM
  Conditions:
  Data types:
  ARRAY OF STRING
  BOOLEAN
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 ‘+’
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] -- pseudo 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
  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
          -- from FILE_LIST & PROG_LIST
  prog_name : STRING[10] -- Program names to list from PROG_LIST
  prog_type : INTEGER -- Program types to list from 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
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

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
--- Display a directory listing of files on the F1py:

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.
Generate and Move Along Hexagon Path - Overview

--- 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'
--- 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
vertex : INTEGER -- the index used specify each vertex 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 vertex = 2 TO NUM_AXES DO -- start at 2 since 1 is already set
  p_xyzwpr[vertex] = p_xyzwpr[1] -- Initialize all vertices
-- 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_arr : ARRAY[NUM_AXES] OF REAL
BEGIN
  -- Initialize the center position array to zero
  FOR indx = 1 TO NUM_AXES DO
    p_cntr_arr[indx] = 0
  ENDFOR
  -- Set JOINT 3 and 5 to -45 and 45 degrees
  p_cntr_arr[3] = -45
  p_cntr_arr[5] = 45 -- Convert the REAL array to a joint position,
  -- p_cntr
  CNV_REL_JPOS(p_cntr_arr, 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
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

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[1], 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

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
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:
--- Action:
--- FROM Sec 3
--- Clauses:
--- 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
%c obment = ‘COPY FILES’

---- Section 2: Variable Declaration

CONST
  SUCCESS = 0 -- Success status from builtins
  FINISHED = TRUE -- Finished Copy
  TRYAGAIN = 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)
    ENDIF
    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
  ELSE
    cpy_stat = TRY_AGAIN
  ENDIF
ENDIF
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
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.

Using Dynamic Display Built-ins - Overview

--- DYN_DISP.KL
--- Section 0: Detail about DYN_DISP.KL
--- Elements of KAREL Language Covered: In Section:
---- Actions:
---- FROM Sec 3-C
---- IN CMOS Sec 2
---- WHEN Sec 4
---- Clauses:
---- 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
Using Dynamic Display Built-ins - Declaration Section

--- Section 1: Program and Environment Declaration

PROGRAM DYN_DISP
%noLockgroup
%comment = 'Dynamic Disp'
%alphabetize
%INCLUDE KLIOTYPES

--- 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]
    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_PK' 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:
--- ---------
--- |xxxxxxxxx|
--- ---------
--- 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

--- With the format and field_width specification the REAL will be
--- displayed as follows:
--- ------------
--- |xxxxxx.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
--- 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 Pendent
  -- 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

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 the predefined display window
Field_Width = 10 -- Use the minimum width necessary
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

--- 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_Str
Buffer_Size = 10 -- Minimum value required.
Format = '%10s' -- 10 character minimum field width
--- With this specification the STRING will be displayed as follows:
--- -----------
--- |xxxxxxxxx|
--- -----------
--- 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
STATUS = cc_success -- Force display of the TP USER screen
-- Initialize the dynamically displayed variables
Int1 = 1
Real1 = 1.0
Bool1 = FALSE
Enum1 = 0
Str1 = ''
-- 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

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

Using Dynamic Display Built-ins - Initiate Dynamic Displays
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)
B. KAREL EXAMPLE PROGRAMS

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: FROM Sec 2
--- Clauses:
----------------------------------------

ENDIF
CNC_DYN_DISI(Int1, Int_Wind,Status) -- Cancel display of Int1
IF Status <> 0 THEN -- Check the status
  WRITE(’ CNC_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(’ CNC_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(’ CNC_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(’ CNC_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(’ CNC_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(’ CNC_DYN_DISS failed, Status=’,status,CR)
ENDIF
END DYN_DISP
--- 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'

Manipulate Dynamically Displayed Variables - Declaration Section

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
Manipulate Dynamically Displayed Variables - 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 D7+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:

<table>
<thead>
<tr>
<th>Elements of KAREL Language Covered</th>
<th>In Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions:</td>
<td>Sec 3-E</td>
</tr>
<tr>
<td>Clauses:</td>
<td></td>
</tr>
<tr>
<td>FROM</td>
<td></td>
</tr>
<tr>
<td>Conditions:</td>
<td></td>
</tr>
<tr>
<td>Data types:</td>
<td></td>
</tr>
<tr>
<td>ARRAY OF STRING</td>
<td>Sec 2</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Sec 2</td>
</tr>
<tr>
<td>DISP_DAT_T</td>
<td>Sec 2</td>
</tr>
<tr>
<td>FILE</td>
<td>Sec 2</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Sec 2</td>
</tr>
<tr>
<td>STRING</td>
<td>Sec 2</td>
</tr>
<tr>
<td>Directives:</td>
<td></td>
</tr>
<tr>
<td>ALPHABETIZE</td>
<td>Sec 1</td>
</tr>
<tr>
<td>COMMENT</td>
<td>Sec 1</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Sec 1</td>
</tr>
<tr>
<td>NOLOCKGROUP</td>
<td>Sec 1</td>
</tr>
<tr>
<td>Built-in Functions &amp; Procedures:</td>
<td></td>
</tr>
<tr>
<td>ADD_DICT</td>
<td>Sec 3-B</td>
</tr>
<tr>
<td>ACT_SCREEN</td>
<td>Sec 4-I</td>
</tr>
<tr>
<td>ATT_WINDOW_S</td>
<td>Sec 4-C</td>
</tr>
<tr>
<td>CHECK_DICT</td>
<td>Sec 3-B</td>
</tr>
<tr>
<td>CLR_IO_STAT</td>
<td>Sec 3-A,C</td>
</tr>
<tr>
<td>CNV_STR_INT</td>
<td>Sec 4-E</td>
</tr>
<tr>
<td>DEF_SCREEN</td>
<td>Sec 4-B</td>
</tr>
<tr>
<td>DET_WINDOW</td>
<td>Sec 4-I</td>
</tr>
<tr>
<td>DISCTRL_LIST</td>
<td>Sec 4-G,H</td>
</tr>
<tr>
<td>FORCE_SPMENU</td>
<td>Sec 4-A,B,C</td>
</tr>
<tr>
<td>IO_STATUS</td>
<td>Sec 3-A,</td>
</tr>
<tr>
<td>ORD</td>
<td>Sec 4-H</td>
</tr>
<tr>
<td>READ_DICT</td>
<td>Sec 4-E,F</td>
</tr>
</tbody>
</table>

Display List from Dictionary File - Overview Continued

| Statements:                        |            |
| 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|
| 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 |
--- 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
--- TTPROMPT 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_Cmd : INTEGER -- decide if any action is pending
act_pending : INTEGER
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
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.
  ENDIF
End Chk_Add_Dct
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
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(TPPFUNC, '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(TPPROJECT, '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.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
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')
      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)
         ENDIF
         IF act_pending = num_options THEN
            -- Exit the routine
            done = TRUE
         ENDIF
      ENDIF
   ENDSELECT

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)
   ENDIF
   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
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

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_PK1 &home" [TYPE]"
$-,TPTSSP_PK2 &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

B.13 USING THE DISCTRL_ALPHA BUILT-IN

This program shows three different ways to use the DISCTRL_ALPHA built-in. The DISCTRL_ALPHA 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:
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
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
  -- must be loaded to continue.
ENDIF
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**

---

**Elements of KAREL Language Covered: In Section:**

**Actions:**

**Clauses:**

FROM

**Conditions:**

**Data Types:**

ARRAY OF REAL

ARRAY OF STRING

BOOLEAN

INTEGER

JOINTPOS

REAL

STRING

XYZWPR

**Directives:**

ENVIRONMENT

**Built-in Functions & Procedures:**

AVL_POS_NUM

CHR

CLOSE_TPE

CNV_REL_JPOS

CNV_JPOS_REL

COPY_TPE

GET_JPOS_TPE

GET_POS_TYP

GET_POS_TPE

OPEN_TPE

PROG_LIST

SELECT_TPE

SET_JPOS_TPE

SET_POS_TPE

---

**Applying Offsets to Copied Teach Pendant Program - Overview Continued**

---

**Statements:**

---
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

<table>
<thead>
<tr>
<th>Section 1: Program and Environment Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM CPY_TP</td>
</tr>
<tr>
<td>%ENVIRONMENT TPE -- necessary for all xxx_TPE built-ins</td>
</tr>
<tr>
<td>%ENVIRONMENT BYNAM -- necessary for PROG_LIST built-in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 2: Constant and Variable Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST</td>
</tr>
<tr>
<td>ER_WARN = 0 -- warning constant for use in POST_ERR</td>
</tr>
<tr>
<td>SUCCESS = 0 -- success constant</td>
</tr>
<tr>
<td>JNT_POS = 9 -- constant for GET_POS_TYP</td>
</tr>
<tr>
<td>XYZ_POS = 2 -- constant for GET_POS_TYP</td>
</tr>
<tr>
<td>MAX_AXS = 9 -- Maximum number of axes JOINTPOS has</td>
</tr>
<tr>
<td>VAR</td>
</tr>
<tr>
<td>from_prog: STRING[13] -- TP program name to be copied FROM</td>
</tr>
<tr>
<td>to_prog : STRING[13] -- TP program name to be copied TO</td>
</tr>
<tr>
<td>over_sw : BOOLEAN -- Decides whether to overwrite an existing program when performing COPY_TPE</td>
</tr>
<tr>
<td>status : INTEGER -- Holds error status from the builtin calls</td>
</tr>
<tr>
<td>off_xyz : XZWPR -- Offset amount for the XZWPR positions</td>
</tr>
<tr>
<td>jp_off : ARRAY [9] of REAL -- Offset amount for the JOINT positions</td>
</tr>
<tr>
<td>new_xyz : XZWPR -- XZWPR which has offset applied</td>
</tr>
<tr>
<td>org_xyz : XZWPR -- Original XZWPR from to_prog</td>
</tr>
</tbody>
</table>
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
  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
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_progs > 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)
        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
---------------------------------------------

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)
  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

--- 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)

  --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
      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

Applying Offsets to Copied Teach Pendant Program - Apply Offsets to Positions Cont.
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) -- 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
## Contents

<table>
<thead>
<tr>
<th>Appendix C</th>
<th>KCL COMMAND ALPHABETICAL DESCRIPTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>ABORT command</td>
<td>C–6</td>
</tr>
<tr>
<td>C.2</td>
<td>APPEND FILE command</td>
<td>C–6</td>
</tr>
<tr>
<td>C.3</td>
<td>APPEND NODE command</td>
<td>C–6</td>
</tr>
<tr>
<td>C.4</td>
<td>CHDIR command</td>
<td>C–7</td>
</tr>
<tr>
<td>C.5</td>
<td>CLEAR ALL command</td>
<td>C–7</td>
</tr>
<tr>
<td>C.6</td>
<td>CLEAR BREAK CONDITION command</td>
<td>C–8</td>
</tr>
<tr>
<td>C.7</td>
<td>CLEAR BREAK PROGRAM command</td>
<td>C–8</td>
</tr>
<tr>
<td>C.8</td>
<td>CLEAR DICT command</td>
<td>C–9</td>
</tr>
<tr>
<td>C.9</td>
<td>CLEAR PROGRAM command</td>
<td>C–9</td>
</tr>
<tr>
<td>C.10</td>
<td>CLEAR VARS command</td>
<td>C–9</td>
</tr>
<tr>
<td>C.11</td>
<td>COMPRESS DICT command</td>
<td>C–10</td>
</tr>
<tr>
<td>C.12</td>
<td>COMPRESS FORM command</td>
<td>C–10</td>
</tr>
<tr>
<td>C.13</td>
<td>CONTINUE command</td>
<td>C–11</td>
</tr>
<tr>
<td>C.14</td>
<td>COPY FILE command</td>
<td>C–11</td>
</tr>
<tr>
<td>C.15</td>
<td>CREATE VARIABLE command</td>
<td>C–12</td>
</tr>
<tr>
<td>C.16</td>
<td>DELETE FILE command</td>
<td>C–13</td>
</tr>
<tr>
<td>C.17</td>
<td>DELETE NODE command</td>
<td>C–13</td>
</tr>
<tr>
<td>C.18</td>
<td>DELETE VARIABLE command</td>
<td>C–14</td>
</tr>
<tr>
<td>C.19</td>
<td>DIRECTORY command</td>
<td>C–14</td>
</tr>
<tr>
<td>C.20</td>
<td>DISABLE BREAK PROGRAM command</td>
<td>C–15</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.62 SET DEFAULT command</td>
<td>C–35</td>
</tr>
<tr>
<td>C.63 SET GROUP command</td>
<td>C–35</td>
</tr>
<tr>
<td>C.64 SET LANGUAGE command</td>
<td>C–36</td>
</tr>
<tr>
<td>C.65 SET LOCAL VARIABLE command</td>
<td>C–36</td>
</tr>
<tr>
<td>C.66 SET PORT command</td>
<td>C–37</td>
</tr>
<tr>
<td>C.67 SET TASK command</td>
<td>C–37</td>
</tr>
<tr>
<td>C.68 SET TRACE command</td>
<td>C–38</td>
</tr>
<tr>
<td>C.69 SET VARIABLE command</td>
<td>C–38</td>
</tr>
<tr>
<td>C.70 SET VERIFY command</td>
<td>C–39</td>
</tr>
<tr>
<td>C.71 SHOW BREAK command</td>
<td>C–39</td>
</tr>
<tr>
<td>C.72 SHOW BUILTINS command</td>
<td>C–40</td>
</tr>
<tr>
<td>C.73 SHOW CONDITION command</td>
<td>C–40</td>
</tr>
<tr>
<td>C.74 SHOW CLOCK command</td>
<td>C–40</td>
</tr>
<tr>
<td>C.75 SHOW CURPOS command</td>
<td>C–41</td>
</tr>
<tr>
<td>C.76 SHOW DEFAULT command</td>
<td>C–41</td>
</tr>
<tr>
<td>C.77 SHOW DEVICE command</td>
<td>C–41</td>
</tr>
<tr>
<td>C.78 SHOW DICTS command</td>
<td>C–41</td>
</tr>
<tr>
<td>C.79 SHOW GROUP command</td>
<td>C–41</td>
</tr>
<tr>
<td>C.80 SHOW HISTORY command</td>
<td>C–42</td>
</tr>
<tr>
<td>C.81 SHOW LANG command</td>
<td>C–42</td>
</tr>
<tr>
<td>C.82 SHOW LANGS command</td>
<td>C–42</td>
</tr>
<tr>
<td>C.83 SHOW LOCAL VARIABLE command</td>
<td>C–42</td>
</tr>
<tr>
<td>C.84 SHOW LOCAL VARS command</td>
<td>C–44</td>
</tr>
<tr>
<td>C.85 SHOW MEMORY command</td>
<td>C–44</td>
</tr>
<tr>
<td>C.86 SHOW PROGRAM command</td>
<td>C–44</td>
</tr>
<tr>
<td>C.87 SHOW PROGRAMS command</td>
<td>C–45</td>
</tr>
<tr>
<td>C.88 SHOW SYSTEM command</td>
<td>C–45</td>
</tr>
<tr>
<td>C.89 SHOW TASK command</td>
<td>C–45</td>
</tr>
<tr>
<td>C.90 SHOW TASKS command</td>
<td>C–46</td>
</tr>
<tr>
<td>C.91 SHOW TRACE command</td>
<td>C–46</td>
</tr>
<tr>
<td>C.92 SHOW TYPES command</td>
<td>C–46</td>
</tr>
<tr>
<td>C.93 SHOW VARIABLE command</td>
<td>C–47</td>
</tr>
<tr>
<td>C.94 SHOW VARS command</td>
<td>C–47</td>
</tr>
<tr>
<td>C.95 SHOW data_type command</td>
<td>C–48</td>
</tr>
<tr>
<td>C.96 SIMULATE command</td>
<td>C–48</td>
</tr>
<tr>
<td>C.97 SKIP command</td>
<td>C–49</td>
</tr>
<tr>
<td>C.98 STEP OFF command</td>
<td>C–50</td>
</tr>
<tr>
<td>C.99 STEP ON command</td>
<td>C–50</td>
</tr>
<tr>
<td>C.100 TRANSLATE command</td>
<td>C–50</td>
</tr>
<tr>
<td>C.101 TYPE command</td>
<td>C–51</td>
</tr>
<tr>
<td>C.102 UNSIMULATE command</td>
<td>C–51</td>
</tr>
</tbody>
</table>
C.103  WAIT command ........................................................................................................................................ C–52
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, ‘C:\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.
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

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 flypy: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_name>`

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.
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
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

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

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
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

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 *.k 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:

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
C. KCL COMMAND ALPHABETICAL DESCRIPTION

Examples:  KCL> CREATE VAR [testProg]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.


KCL> DELETE NODE weld_pth[3]
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

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 command or
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
**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
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

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.

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

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
**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 $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 SOLD 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 SOLD 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
**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:
• 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."

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.
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

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
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.
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.

### 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

**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.
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 >
```
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

---

**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
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
**C. KCL COMMAND ALPHABETICAL DESCRIPTION**

**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.
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'

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.
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.
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
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 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:
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
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
C. KCL COMMAND ALPHABETICAL DESCRIPTION

**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
**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.
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.

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

### 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
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.
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.
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

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.
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:
DIN, DOUT, WDI, WDOAIN, AOUT, GIN, GOUT

See Also: SIMULATE command

Examples: KCL> UNSIMULATE DIN[17]
KCL> UNSIM ALL

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
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

<table>
<thead>
<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>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(NUL)</td>
<td>032</td>
<td>SP</td>
<td>064</td>
<td>@</td>
<td>096</td>
<td>’</td>
</tr>
<tr>
<td>001</td>
<td>(SOH)</td>
<td>033</td>
<td>!</td>
<td>065</td>
<td>A</td>
<td>097</td>
<td>a</td>
</tr>
<tr>
<td>002</td>
<td>(STX)</td>
<td>034</td>
<td>”</td>
<td>066</td>
<td>B</td>
<td>098</td>
<td>b</td>
</tr>
<tr>
<td>003</td>
<td>(ETX)</td>
<td>035</td>
<td>#</td>
<td>067</td>
<td>C</td>
<td>099</td>
<td>c</td>
</tr>
<tr>
<td>004</td>
<td>(EOT)</td>
<td>036</td>
<td>$</td>
<td>068</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>005</td>
<td>(ENQ)</td>
<td>037</td>
<td>%</td>
<td>069</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>006</td>
<td>(ACK)</td>
<td>038</td>
<td>&amp;</td>
<td>070</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>007</td>
<td>(BEL)</td>
<td>039</td>
<td>’</td>
<td>071</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>008</td>
<td>(BS)</td>
<td>040</td>
<td>(</td>
<td>072</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>009</td>
<td>(HT)</td>
<td>041</td>
<td>)</td>
<td>073</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>010</td>
<td>(LF)</td>
<td>042</td>
<td>*</td>
<td>074</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>011</td>
<td>(VT)</td>
<td>043</td>
<td>+</td>
<td>075</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>012</td>
<td>(FF)</td>
<td>044</td>
<td>’</td>
<td>076</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>013</td>
<td>(CR)</td>
<td>045</td>
<td>-</td>
<td>077</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>014</td>
<td>(SO)</td>
<td>046</td>
<td>.</td>
<td>078</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>015</td>
<td>(SI)</td>
<td>047</td>
<td>/</td>
<td>079</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>016</td>
<td>(DLE)</td>
<td>048</td>
<td>0</td>
<td>080</td>
<td>P</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>017</td>
<td>(DC1)</td>
<td>049</td>
<td>1</td>
<td>081</td>
<td>Q</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>018</td>
<td>(DC2)</td>
<td>050</td>
<td>2</td>
<td>082</td>
<td>R</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>019</td>
<td>(DC3)</td>
<td>051</td>
<td>3</td>
<td>083</td>
<td>S</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>020</td>
<td>(DC4)</td>
<td>052</td>
<td>4</td>
<td>084</td>
<td>T</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>021</td>
<td>(NAK)</td>
<td>053</td>
<td>5</td>
<td>085</td>
<td>U</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>022</td>
<td>(SYN)</td>
<td>054</td>
<td>6</td>
<td>086</td>
<td>V</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>023</td>
<td>(ETB)</td>
<td>055</td>
<td>7</td>
<td>087</td>
<td>W</td>
<td>119</td>
<td>w</td>
</tr>
</tbody>
</table>
### Table D–1. ASCII Character Codes (Cont’d)

<table>
<thead>
<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>
</thead>
<tbody>
<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>&lt;</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>&gt;</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>
</tbody>
</table>

### Table D–2. Special ASCII Character Codes

<table>
<thead>
<tr>
<th>Decimal Code</th>
<th>Character Value</th>
<th>Decimal Code</th>
<th>Character Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Clear window</td>
<td>154</td>
<td>Turn Multinational mode on</td>
</tr>
<tr>
<td>129</td>
<td>Clear to end of line</td>
<td>155 48</td>
<td>Foreground color black</td>
</tr>
<tr>
<td>130</td>
<td>Clear to end of window</td>
<td>155 49</td>
<td>Foreground color red</td>
</tr>
<tr>
<td>131</td>
<td>Set cursor position</td>
<td>155 50</td>
<td>Foreground color green</td>
</tr>
<tr>
<td>132</td>
<td>Carriage return</td>
<td>155 51</td>
<td>Foreground color yellow</td>
</tr>
<tr>
<td>133</td>
<td>Line feed</td>
<td>155 52</td>
<td>Foreground color blue</td>
</tr>
<tr>
<td>134</td>
<td>Reverse line feed</td>
<td>155 53</td>
<td>Foreground color magenta</td>
</tr>
<tr>
<td>135</td>
<td>Carriage return &amp; line feed</td>
<td>155 54</td>
<td>Foreground color cyan</td>
</tr>
<tr>
<td>136</td>
<td>Back Space</td>
<td>155 55</td>
<td>Foreground color white</td>
</tr>
</tbody>
</table>
### Table D–2. Special ASCII Character Codes (Cont’d)

<table>
<thead>
<tr>
<th>Decimal Code</th>
<th>Character Value</th>
<th>Decimal Code</th>
<th>Character Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>Home cursor in window</td>
<td>155 127</td>
<td>Foreground color default</td>
</tr>
<tr>
<td>138</td>
<td>Blink video attribute</td>
<td>156 48</td>
<td>Background color black</td>
</tr>
<tr>
<td>139</td>
<td>Reverse video attribute</td>
<td>156 49</td>
<td>Background color red</td>
</tr>
<tr>
<td>140</td>
<td>Bold video attribute</td>
<td>156 50</td>
<td>Background color green</td>
</tr>
<tr>
<td>141</td>
<td>Underline video attribute</td>
<td>156 51</td>
<td>Background color yellow</td>
</tr>
<tr>
<td>142</td>
<td>Wide video size</td>
<td>156 52</td>
<td>Background color blue</td>
</tr>
<tr>
<td>143</td>
<td>Normal video attribute</td>
<td>156 53</td>
<td>Background color magenta</td>
</tr>
<tr>
<td>146</td>
<td>Turn Graphics mode on</td>
<td>156 54</td>
<td>Background color cyan</td>
</tr>
<tr>
<td>147</td>
<td>Turn ASCII mode on</td>
<td>156 55</td>
<td>Background color white</td>
</tr>
<tr>
<td>148</td>
<td>High/wide video size</td>
<td>156 127</td>
<td>Background color default</td>
</tr>
<tr>
<td>153</td>
<td>Normal video size</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table D–3. Multinational Character Codes

<table>
<thead>
<tr>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>032</td>
<td>064</td>
<td>Å</td>
<td>096</td>
<td>à</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>033</td>
<td>065</td>
<td>Å</td>
<td>097</td>
<td>á</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002</td>
<td>034</td>
<td>066</td>
<td>Å</td>
<td>098</td>
<td>à</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>035</td>
<td>067</td>
<td>Å</td>
<td>099</td>
<td>ä</td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>(IND)</td>
<td>068</td>
<td>Å</td>
<td>100</td>
<td>ä</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>(NEL)</td>
<td>069</td>
<td>Å</td>
<td>101</td>
<td>à</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006</td>
<td>(SSA)</td>
<td>070</td>
<td>Æ</td>
<td>102</td>
<td>æ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>(ESA)</td>
<td>071</td>
<td>Ç</td>
<td>103</td>
<td>ç</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Multinational Character Codes (Cont’d)

<table>
<thead>
<tr>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>(HTS)</td>
<td>040</td>
<td>□</td>
<td>072</td>
<td>É</td>
<td>104</td>
<td>ë</td>
</tr>
<tr>
<td>009</td>
<td>(HTJ)</td>
<td>041</td>
<td>©</td>
<td>073</td>
<td>É</td>
<td>105</td>
<td>ë</td>
</tr>
<tr>
<td>010</td>
<td>(VTS)</td>
<td>042</td>
<td>Ḃ</td>
<td>074</td>
<td>É</td>
<td>106</td>
<td>ë</td>
</tr>
<tr>
<td>011</td>
<td>(PLD)</td>
<td>043</td>
<td>«</td>
<td>075</td>
<td>É</td>
<td>107</td>
<td>ë</td>
</tr>
<tr>
<td>012</td>
<td>(PLU)</td>
<td>044</td>
<td></td>
<td>076</td>
<td>İ</td>
<td>108</td>
<td>i</td>
</tr>
<tr>
<td>013</td>
<td>(RI)</td>
<td>045</td>
<td></td>
<td>077</td>
<td>İ</td>
<td>109</td>
<td>i</td>
</tr>
<tr>
<td>014</td>
<td>(SS2)</td>
<td>046</td>
<td></td>
<td>078</td>
<td>İ</td>
<td>110</td>
<td>i</td>
</tr>
<tr>
<td>015</td>
<td>(SS3)</td>
<td>047</td>
<td></td>
<td>079</td>
<td>ĭ</td>
<td>111</td>
<td>ĭ</td>
</tr>
<tr>
<td>016</td>
<td>(DCS)</td>
<td>048</td>
<td></td>
<td>080</td>
<td></td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>017</td>
<td>(PU1)</td>
<td>049</td>
<td>±</td>
<td>081</td>
<td>Ń</td>
<td>113</td>
<td>Ń</td>
</tr>
<tr>
<td>018</td>
<td>(PU2)</td>
<td>050</td>
<td>²</td>
<td>082</td>
<td>Ö</td>
<td>114</td>
<td>Ö</td>
</tr>
<tr>
<td>019</td>
<td>(STS)</td>
<td>051</td>
<td>³</td>
<td>083</td>
<td>Ö</td>
<td>115</td>
<td>Ö</td>
</tr>
<tr>
<td>020</td>
<td>(CCH)</td>
<td>052</td>
<td></td>
<td>084</td>
<td>Ø</td>
<td>116</td>
<td>Ø</td>
</tr>
<tr>
<td>021</td>
<td>(MW)</td>
<td>053</td>
<td>µ</td>
<td>085</td>
<td>Ö</td>
<td>117</td>
<td>Ö</td>
</tr>
<tr>
<td>022</td>
<td>(SPA)</td>
<td>054</td>
<td>¶</td>
<td>086</td>
<td>Ö</td>
<td>118</td>
<td>Ö</td>
</tr>
<tr>
<td>023</td>
<td>(EPA)</td>
<td>055</td>
<td>•</td>
<td>087</td>
<td>ÖE</td>
<td>119</td>
<td>ÖE</td>
</tr>
<tr>
<td>024</td>
<td></td>
<td>056</td>
<td></td>
<td>088</td>
<td>Ø</td>
<td>120</td>
<td>Ø</td>
</tr>
<tr>
<td>025</td>
<td></td>
<td>057</td>
<td>¹</td>
<td>089</td>
<td>Ü</td>
<td>121</td>
<td>Ü</td>
</tr>
<tr>
<td>026</td>
<td></td>
<td>058</td>
<td></td>
<td>090</td>
<td>Ü</td>
<td>122</td>
<td>Ü</td>
</tr>
<tr>
<td>027</td>
<td>(CSI)</td>
<td>059</td>
<td>»</td>
<td>091</td>
<td>Ü</td>
<td>123</td>
<td>Ü</td>
</tr>
<tr>
<td>028</td>
<td>(ST)</td>
<td>060</td>
<td>¼</td>
<td>092</td>
<td>Ü</td>
<td>124</td>
<td>Ü</td>
</tr>
<tr>
<td>029</td>
<td>(OSC)</td>
<td>061</td>
<td>½</td>
<td>093</td>
<td>Y</td>
<td>125</td>
<td>Ь</td>
</tr>
<tr>
<td>030</td>
<td>(PM)</td>
<td>062</td>
<td></td>
<td>094</td>
<td></td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>031</td>
<td>(APC)</td>
<td>063</td>
<td>ğ</td>
<td>095</td>
<td>Ь</td>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>
### Table D-4. Graphics Character Codes (not available in R-30iB)

<table>
<thead>
<tr>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(NUL)</td>
<td>032</td>
<td>SP</td>
<td>064</td>
<td>@</td>
<td>096</td>
</tr>
<tr>
<td>001</td>
<td>(SOH)</td>
<td>033</td>
<td>!</td>
<td>065</td>
<td>A</td>
<td>097</td>
</tr>
<tr>
<td>002</td>
<td>(STX)</td>
<td>034</td>
<td>&quot;</td>
<td>066</td>
<td>B</td>
<td>098</td>
</tr>
<tr>
<td>003</td>
<td>(ETX)</td>
<td>035</td>
<td>#</td>
<td>067</td>
<td>C</td>
<td>099</td>
</tr>
<tr>
<td>004</td>
<td>(EOT)</td>
<td>036</td>
<td>$</td>
<td>068</td>
<td>D</td>
<td>100</td>
</tr>
<tr>
<td>005</td>
<td>(ENQ)</td>
<td>037</td>
<td>%</td>
<td>069</td>
<td>E</td>
<td>101</td>
</tr>
<tr>
<td>006</td>
<td>(ACK)</td>
<td>038</td>
<td>&amp;</td>
<td>070</td>
<td>F</td>
<td>102</td>
</tr>
<tr>
<td>007</td>
<td>(BEL)</td>
<td>039</td>
<td>'</td>
<td>071</td>
<td>G</td>
<td>103</td>
</tr>
<tr>
<td>008</td>
<td>(BS)</td>
<td>040</td>
<td>(</td>
<td>072</td>
<td>H</td>
<td>104</td>
</tr>
<tr>
<td>009</td>
<td>(HT)</td>
<td>041</td>
<td>)</td>
<td>073</td>
<td>I</td>
<td>105</td>
</tr>
<tr>
<td>010</td>
<td>(LF)</td>
<td>042</td>
<td>*</td>
<td>074</td>
<td>J</td>
<td>106</td>
</tr>
<tr>
<td>011</td>
<td>(VT)</td>
<td>043</td>
<td>+</td>
<td>075</td>
<td>K</td>
<td>107</td>
</tr>
<tr>
<td>012</td>
<td>(FF)</td>
<td>044</td>
<td>,</td>
<td>076</td>
<td>L</td>
<td>108</td>
</tr>
<tr>
<td>013</td>
<td>(CR)</td>
<td>045</td>
<td>-</td>
<td>077</td>
<td>M</td>
<td>109</td>
</tr>
<tr>
<td>014</td>
<td>(SO)</td>
<td>046</td>
<td>.</td>
<td>078</td>
<td>N</td>
<td>110</td>
</tr>
<tr>
<td>015</td>
<td>(SI)</td>
<td>047</td>
<td>/</td>
<td>079</td>
<td>O</td>
<td>111</td>
</tr>
<tr>
<td>016</td>
<td>(DLE)</td>
<td>048</td>
<td>0</td>
<td>080</td>
<td>P</td>
<td>112</td>
</tr>
<tr>
<td>017</td>
<td>(DC1)</td>
<td>049</td>
<td>1</td>
<td>081</td>
<td>Q</td>
<td>113</td>
</tr>
<tr>
<td>018</td>
<td>(DC2)</td>
<td>050</td>
<td>2</td>
<td>082</td>
<td>R</td>
<td>114</td>
</tr>
<tr>
<td>019</td>
<td>(DC3)</td>
<td>051</td>
<td>3</td>
<td>083</td>
<td>S</td>
<td>115</td>
</tr>
<tr>
<td>020</td>
<td>(DC4)</td>
<td>052</td>
<td>4</td>
<td>084</td>
<td>T</td>
<td>116</td>
</tr>
</tbody>
</table>
### Table D-4. Graphics Character Codes (not available in R-30iB) (Cont’d)

<table>
<thead>
<tr>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
<th>Decimal Codes</th>
<th>Character Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>021</td>
<td>(NAK)</td>
<td>053</td>
<td>5</td>
<td>085</td>
<td>U</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>(SYN)</td>
<td>054</td>
<td>6</td>
<td>086</td>
<td>V</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>023</td>
<td>(ETB)</td>
<td>055</td>
<td>7</td>
<td>087</td>
<td>W</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>024</td>
<td>(CAN)</td>
<td>056</td>
<td>8</td>
<td>088</td>
<td>X</td>
<td>120</td>
<td></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></td>
</tr>
<tr>
<td>026</td>
<td>(SUB)</td>
<td>058</td>
<td>:</td>
<td>090</td>
<td>Z</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>027</td>
<td>(ESC)</td>
<td>059</td>
<td>;</td>
<td>091</td>
<td>[</td>
<td>123</td>
<td>Π</td>
</tr>
<tr>
<td>028</td>
<td>(FS)</td>
<td>060</td>
<td>&lt;</td>
<td>092</td>
<td>&gt;</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>&gt;</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>(blank)</td>
<td>127</td>
<td>(DEL)</td>
</tr>
</tbody>
</table>

### Table D-5. Teach Pendant Input Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48</td>
<td>174</td>
<td>USER KEY 2</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>175</td>
<td>USER KEY 3</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>176</td>
<td>USER KEY 4</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>177</td>
<td>USER KEY 5</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>178</td>
<td>USER KEY 6</td>
</tr>
<tr>
<td>5</td>
<td>53</td>
<td>185</td>
<td>FWD</td>
</tr>
<tr>
<td>6</td>
<td>54</td>
<td>186</td>
<td>BWD</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>187</td>
<td>COORD</td>
</tr>
</tbody>
</table>
### Table D–5. Teach Pendant Input Codes (Cont’d)

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>56</td>
<td>188</td>
<td>+X</td>
</tr>
<tr>
<td>9</td>
<td>57</td>
<td>189</td>
<td>+Y</td>
</tr>
<tr>
<td>128</td>
<td>PREV</td>
<td>190</td>
<td>+Z</td>
</tr>
<tr>
<td>129</td>
<td>F1</td>
<td>191</td>
<td>+X rotation</td>
</tr>
<tr>
<td>131</td>
<td>F2</td>
<td>192</td>
<td>+Y rotation</td>
</tr>
<tr>
<td>132</td>
<td>F3</td>
<td>193</td>
<td>+Z rotation</td>
</tr>
<tr>
<td>133</td>
<td>F4</td>
<td>194</td>
<td>-X</td>
</tr>
<tr>
<td>134</td>
<td>F5</td>
<td>195</td>
<td>-Y</td>
</tr>
<tr>
<td>135</td>
<td>NEXT</td>
<td>196</td>
<td>-Z</td>
</tr>
<tr>
<td>143</td>
<td>SELECT</td>
<td>197</td>
<td>-X rotation</td>
</tr>
<tr>
<td>144</td>
<td>MENUS</td>
<td>198</td>
<td>-Y rotation</td>
</tr>
<tr>
<td>145</td>
<td>EDIT</td>
<td>199</td>
<td>-Z rotation</td>
</tr>
<tr>
<td>146</td>
<td>DATA</td>
<td>210</td>
<td>USER KEY</td>
</tr>
<tr>
<td>147</td>
<td>FCTN</td>
<td>212</td>
<td>Up arrow</td>
</tr>
<tr>
<td>148</td>
<td>ITEM</td>
<td>213</td>
<td>Down arrow</td>
</tr>
<tr>
<td>149</td>
<td>+%</td>
<td>214</td>
<td>Right arrow</td>
</tr>
<tr>
<td>150</td>
<td>-%</td>
<td>215</td>
<td>Left arrow</td>
</tr>
<tr>
<td>151</td>
<td>HOLD</td>
<td>147</td>
<td>DEADMAN switch, left</td>
</tr>
<tr>
<td>152</td>
<td>STEP</td>
<td>248</td>
<td>DEADMAN switch, right</td>
</tr>
<tr>
<td>153</td>
<td>RESET</td>
<td>249</td>
<td>ON/OFF switch</td>
</tr>
<tr>
<td>173</td>
<td>USER KEY 1</td>
<td>250</td>
<td>EMERGENCY STOP</td>
</tr>
</tbody>
</table>

### Table D–6. European Character Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Code</th>
<th>Value</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>A’</td>
<td>213</td>
<td>O~</td>
<td>234</td>
<td>e^</td>
</tr>
<tr>
<td>193</td>
<td>A’</td>
<td>214</td>
<td>O:</td>
<td>235</td>
<td>e:</td>
</tr>
<tr>
<td>194</td>
<td>A^</td>
<td>215</td>
<td>OE</td>
<td>236</td>
<td>i'</td>
</tr>
<tr>
<td>195</td>
<td>A~</td>
<td>216</td>
<td>O/</td>
<td>237</td>
<td>i'</td>
</tr>
<tr>
<td>196</td>
<td>A:</td>
<td>217</td>
<td>U'</td>
<td>238</td>
<td>i^</td>
</tr>
<tr>
<td>197</td>
<td>Ao</td>
<td>218</td>
<td>U'</td>
<td>239</td>
<td>i:</td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
<td>Code</td>
<td>Value</td>
<td>Code</td>
<td>Value</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>198</td>
<td>AE</td>
<td>219</td>
<td>U^</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>199</td>
<td>CC</td>
<td>220</td>
<td>U:</td>
<td>241</td>
<td>n~</td>
</tr>
<tr>
<td>200</td>
<td>E'</td>
<td>221</td>
<td>Y:</td>
<td>242</td>
<td>o'</td>
</tr>
<tr>
<td>201</td>
<td>E'</td>
<td>222</td>
<td></td>
<td>243</td>
<td>o'</td>
</tr>
<tr>
<td>202</td>
<td>E^</td>
<td>223</td>
<td>Bb</td>
<td>244</td>
<td>o^</td>
</tr>
<tr>
<td>203</td>
<td>E:</td>
<td>224</td>
<td>a'</td>
<td>245</td>
<td>o~</td>
</tr>
<tr>
<td>204</td>
<td>l'</td>
<td>225</td>
<td>a'</td>
<td>246</td>
<td>o:</td>
</tr>
<tr>
<td>205</td>
<td>l'</td>
<td>226</td>
<td>a^</td>
<td>247</td>
<td>oe</td>
</tr>
<tr>
<td>206</td>
<td>l^</td>
<td>227</td>
<td>a~</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>l:</td>
<td>228</td>
<td>a:</td>
<td>249</td>
<td>u'</td>
</tr>
<tr>
<td>208</td>
<td></td>
<td>229</td>
<td>ao</td>
<td>250</td>
<td>u'</td>
</tr>
<tr>
<td>209</td>
<td>N~</td>
<td>230</td>
<td>ae</td>
<td>251</td>
<td>u^</td>
</tr>
<tr>
<td>210</td>
<td>O'</td>
<td>231</td>
<td></td>
<td>252</td>
<td>u:</td>
</tr>
<tr>
<td>211</td>
<td>O'</td>
<td>232</td>
<td>e'</td>
<td>253</td>
<td>y:</td>
</tr>
<tr>
<td>212</td>
<td>O^</td>
<td>233</td>
<td>e'</td>
<td>254</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Decimal Value</th>
<th>ASCII Character</th>
<th>Graphic Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>a</td>
<td>solid box</td>
</tr>
<tr>
<td>102</td>
<td>f</td>
<td>diamond</td>
</tr>
<tr>
<td>103</td>
<td>g</td>
<td>plus/minus</td>
</tr>
<tr>
<td>106</td>
<td>j</td>
<td>lower-right box corner</td>
</tr>
<tr>
<td>107</td>
<td>k</td>
<td>upper-right box corner</td>
</tr>
<tr>
<td>108</td>
<td>l</td>
<td>upper-left box corner</td>
</tr>
<tr>
<td>109</td>
<td>m</td>
<td>lower-left box corner</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>intersection lines</td>
</tr>
<tr>
<td>111</td>
<td>o</td>
<td>pixel row 1 horizontal line</td>
</tr>
<tr>
<td>112</td>
<td>p</td>
<td>pixel row 2 horizontal line</td>
</tr>
<tr>
<td>113</td>
<td>q</td>
<td>pixel row 3 horizontal line</td>
</tr>
<tr>
<td>114</td>
<td>r</td>
<td>pixel row 4 horizontal line</td>
</tr>
<tr>
<td>115</td>
<td>s</td>
<td>pixel row 5 horizontal line</td>
</tr>
<tr>
<td>116</td>
<td>t</td>
<td>T from right</td>
</tr>
<tr>
<td>117</td>
<td>u</td>
<td>T from left</td>
</tr>
<tr>
<td>119</td>
<td>v</td>
<td>T from above</td>
</tr>
<tr>
<td>119</td>
<td>w</td>
<td>T from below</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>Vertical Line</td>
</tr>
<tr>
<td>121</td>
<td>y</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>122</td>
<td>z</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>123</td>
<td>{</td>
<td>Pi</td>
</tr>
<tr>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>}</td>
<td>British pound symbol</td>
</tr>
</tbody>
</table>
Appendix E

SYNTAX DIAGRAMS

Contents
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 lineend (; 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.
**Figure E–1.**

PROGRAM module definition

```
PROGRAM module definition

- directive list
- TYPE declaration
- VAR declaration
- CONST declaration
- ROUTINE declaration

BEGIN statement
```
Figure E–2.

Directive list

- %ALPHABETIZE
- %COMMENT
- %CMOSVARS
- %CRTDEVICE
- %DEFGROUP
- %DELAY
- %ENVIRONMENT
- %LOCKGROUP
- %NOABORT
- %NOPAUSE
- %NOBUSYLAMP
- %NOLOCKGROUP
- %NOPAUSESHFT
- %PRIORITY
- %STACKSIZE
- %TIMESLICE
- %TPMOTION

Quoted String

Integer

Identifier

Constant

ERROR

COMMAND

TPENABLE
Figure E–3.

**CONST -- constant declaration**

- `CONST` → constant name → `identifier` → `=` → constant

**TYPE -- type declaration**

- `TYPE` → type name → `identifier` → `FROM` → program name → `identifier` → `=` → user type
Figure E–4.

**user type**

```
data type

STRUCTURE 

field list

ENDSTRUCTURE
```

**field list**

```
field name

identifier : structure array type
```

**structure array type**

```
data type

BYTE

SHORT
```
Figure E–5.

data type

- type name
- identifier
- system type name
- system identifier
- INTEGER
- REAL
- BOOLEAN
- VECTOR
- VIS_PROCESS
- MODEL
- CAM_SETUP
- FILE
- CONFIG
- PATH

PATH HEADER

NODEDATA

structure type name

identifier
Figure E–6.

data type continued

- position type
- GROUP
- (integer) constant
- STRING
- string size
- (integer) constant
- ARRAY
- (integer) constant
- OF
- structure array type
Figure E–7.

position type

POSITION

XYZWPR

JOINTPOS1

JOINTPOS2

JOINTPOS3

JOINTPOS4

JOINTPOS5

JOINTPOS6

JOINTPOS7

JOINTPOS8

JOINTPOS9

JOINTPOS10

XYZWPREXT
Figure E–8.

VAR -- variable declaration

ROUTINE -- routine declaration
Figure E–9.

return data type

```
    type name
       identifier
       system type name
       system identifier

    INTEGER
    REAL
    BOOLEAN
    VECTOR
    CONFIG

    position type
       IN
       GROUP
       (integer)
       constant

    STRING

    ARRAY
       OF
       structure array
       type
```
Figure E–10.

parameter type

- type name
- system type name
  - system identifier

- INTEGER
- REAL
- BOOLEAN
- VECTOR
- VIS_PROCESS
- MODEL
- CAM_SETUP
- FILE
- CONFIG

PATH

PATHHEADER  \(\equiv\)  \(\land\)  \(\uparrow\)

NODEDATA  \(\equiv\)  \(\land\)  \(\uparrow\)
Figure E–11.

parameter type continued

![Diagram showing syntax diagram with nodes labeled position type, STRING, ARRAY, GROUP, constant, and OF, illustrating the structure array type.](image-url)
Figure E–12.

statement list

statement

label name

identifier

ABORT statement
assignment statement
ATTACH statement
call statement routine
CANCEL statement
CANCEL FILE statement
CLOSE FILE statement
CLOSE HAND statement
CONDITION statement
CONNECT TIMER statement
DELAY statement
DISABLE statement
DISCONNECT TIMER statement
ENABLE statement
FOR statement
GO TO statement
HOLD statement
IF statement
OPEN FILE statement
OPEN HAND statement
PAUSE statement
PULSE statement
PURGE statement
READ statement
RELAX statement
RELEASE statement
REPEAT statement
RESUME statement
RETURN statement
SELECT statement
SIGNAL statement
STOP statement
UNHOLD statement
USING statement
WAIT statement
WHILE statement
WRITE statement
Figure E-13.

- **ABORT** - statement
  - ABORT
  - PROGRAM
  - task number
  - (integer)
  - expression

- assignment - statement
  - variable access
  - =
  - expression

- **ATTACH** - statement
  - ATTACH

- **CALL** - routine
  - routine name
  - identifier
  - actual parameter
  - expression

- **CANCEL** - statement
  - CANCEL
  - GROUP
  - (integer)
  - constant

- **CANCEL FILE** - statement
  - CANCEL
  - FILE
  - file variable
  - variable access

- **CLOSE FILE** - statement
  - CLOSE
  - FILE
  - file variable
  - variable access

- **CLOSE HAND** - statement
  - CLOSE
  - HAND
  - hand spec
  - (integer)
  - expression
Figure E–15.

- **DISCONNECT TIMER** - statement
  - DISCONNECT
  - TIMER
  - clock variable (integer) identifier

- **ENABLE** - statement
  - ENABLE
  - CONDITION
  - condition handler-number (integer) expression

- **FOR** - statement
  - FOR
  - (integer) identifier
  - count variable
  - initial value (integer) expression
  - final value (integer) expression
  - TO
  - DOWNTO
  - DO
  - loop body statement list
  - ENDFOR

- **GO TO** - statement
  - GOTO
  - GO
  - TO
  - label name (destination) identifier

- **HOLD** - statement
  - HOLD
  - GROUP
  - (integer) constant

- **IF THEN** - statement
  - IF
  - (boolean) expression
  - THEN
  - statement list
  - ELSE
  - statement list
  - ENDF
Figure E–16.

OPEN FILE - statement

OPEN  FILE  variable access  usage spec (string) expression

OPEN HAND - statement

OPEN  HAND  hand spec (integer) identifier

PAUSE - statement

PAUSE  PROGRAM  task number (integer) expression

PULSE - statement

PULSE  DOUT  RDO  port number (integer) identifier  FOR time spec (ms) NOWAIT

PURGE - statement

PURGE  CONDITION  condition handler number (integer) expression
READ - statement

read item - statement

RELAX - statement

RELEASE - statement

REPEAT - statement

RESUME - statement
Figure E-18.

RETURN - statement

SELECT - statement

SIGNAL - statement

STOP - statement

UNHOLD - statement

USING - statement
Figure E–19.

WAIT - statement

WHILE - statement

WRITE - statement

write item

CR
Figure E–21.

Condition handler action

- HOLD
- UNHOLD
- RESUME
- STOP
- CANCEL
- NOABORT
- NOPAUSE
- UNPAUSE
- NOMESSAGE
- RESTORE
- ENABLE
- DISABLE
- SIGNAL
- PULSE
- EVENT
- SEMAPHORE
- GROUP
- CONDITION
- procedure name
- identifier
- DOUT
- RDO
- SIGNAL
- ENABLE
- DISABLE
- NOABORT
- NOPAUSE
- CANCEL
- UNPAUSE
- NOMESSAGE
- RESTORE
- STOP
- HOLD

Expression:
- GROUP (integer)
- CONDITION (integer)
- EVENT (integer)
- SEMAPHORE (integer)
- DOUT (integer)
- RDO (integer)
- FOR (integer)

Expression:
- constant
- expression
- time spec (ms)
- port index
- event/semaphore number
- condition handler number
- condition handler action
- procedure name
condition handler action continued

![Syntax Diagram](image_url)
Figure E–23.

variable access

system var or field name

array index

expression

sum
Figure E–24.

- **sum**
  - OR
  - +

- **product**
  - product
  - product
  - *
  - / (AND)
  - DIV (MOD)

- **factor**
  - factor
  - factor
  - primary
  - NOT
  - : @ ^
Figure E–25.

primary

- variable access
- constant
- function name
- identifier
- actual parameter
- expression

literal

- character
- unsigned numeric value
Glossary

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.
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.
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.
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.
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.
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.
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.

development
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.
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.
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.
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, and 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.
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.
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.

 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.
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.
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.
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
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.
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.
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.
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

tmilliampere
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.

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.
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.
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.
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.
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.

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.
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.
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.
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.
TOOL Frame
The Cartesian coordinate system that has the position of the TCP as its origin to set. 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.
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.
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

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
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