LAN-90® Process Control View (PCV®)
Custom Application Toolkit
(Software Release 5.2)
**WARNING** notices as used in this instruction apply to hazards or unsafe practices that could result in personal injury or death.

**CAUTION** notices apply to hazards or unsafe practices that could result in property damage.

**NOTES** highlight procedures and contain information that assists the operator in understanding the information contained in this instruction.

---

**WARNING**

**INSTRUCTION MANUALS**

DO NOT INSTALL, MAINTAIN, OR OPERATE THIS EQUIPMENT WITHOUT READING, UNDERSTANDING, AND FOLLOWING THE PROPER Elsag Bailey INSTRUCTIONS AND MANUALS; OTHERWISE, INJURY OR DAMAGE MAY RESULT.

**RADIO FREQUENCY INTERFERENCE**

MOST ELECTRONIC EQUIPMENT IS INFLUENCED BY RADIO FREQUENCY INTERFERENCE (RFI). CAUTION SHOULD BE EXERCISED WITH REGARD TO THE USE OF PORTABLE COMMUNICATIONS EQUIPMENT IN THE AREA AROUND SUCH EQUIPMENT. PRUDENT PRACTICE DICTATES THAT SIGNS SHOULD BE POSTED IN THE VICINITY OF THE EQUIPMENT CAUTIONING AGAINST THE USE OF PORTABLE COMMUNICATIONS EQUIPMENT.

**POSSIBLE PROCESS UPSETS**

MAINTENANCE MUST BE PERFORMED ONLY BY QUALIFIED PERSONNEL AND ONLY AFTER SECURING EQUIPMENT CONTROLLED BY THIS PRODUCT. ADJUSTING OR REMOVING THIS PRODUCT WHILE IT IS IN THE SYSTEM MAY UPSET THE PROCESS BEING CONTROLLED. SOME PROCESS UPSETS MAY CAUSE INJURY OR DAMAGE.

---

**NOTICE**

The information contained in this document is subject to change without notice.

Elsag Bailey, its affiliates, employees, and agents, and the authors and contributors to this publication specifically disclaim all liabilities and warranties, express and implied (including warranties of merchantability and fitness for a particular purpose), for the accuracy, currency, completeness, and/or reliability of the information contained herein and/or for the fitness for any particular use and/or for the performance of any material and/or equipment selected in whole or part with the user of/or in reliance upon information contained herein. Selection of materials and/or equipment is at the sole risk of the user of this publication.

This document contains proprietary information of Elsag Bailey, Elsag Bailey Process Automation, and is issued in strict confidence. Its use, or reproduction for use, for the reverse engineering, development or manufacture of hardware or software described herein is prohibited. No part of this document may be photocopied or reproduced without the prior written consent of Elsag Bailey.
This manual provides general information and specific instructions on configuring and using the LAN-90® PCV® 5.2 Custom Application Toolkit to develop software that uses the library functions provided with LAN-90 PCV data.

This manual can be used as a reference guide for software developers who are developing programs for use with LAN-90 PCV.

This manual assumes the reader has a general knowledge of CRT-based process control systems.
List of Effective Pages

Total number of pages in this instruction is 44, consisting of the following:

<table>
<thead>
<tr>
<th>Page No.</th>
<th>Change Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>Original</td>
</tr>
<tr>
<td>List of Effective Pages</td>
<td>Original</td>
</tr>
<tr>
<td>iii through vi</td>
<td></td>
</tr>
<tr>
<td>1-1</td>
<td>Original</td>
</tr>
<tr>
<td>2-1 through 2-2</td>
<td>Original</td>
</tr>
<tr>
<td>3-1 through 3-6</td>
<td>Original</td>
</tr>
<tr>
<td>4-1 through 4-24</td>
<td>Original</td>
</tr>
<tr>
<td>5-1 through 5-3</td>
<td>Original</td>
</tr>
<tr>
<td>Index-1 through Index-2</td>
<td>Original</td>
</tr>
</tbody>
</table>

When an update is received, insert the latest changed pages and dispose of the superseded pages.

**NOTE:** On an update page, the changed text or table is indicated by a vertical bar in the outer margin of the page adjacent to the changed area. A changed figure is indicated by a vertical bar in the outer margin next to the figure caption. The date the update was prepared will appear beside the page number.
# Table of Contents

**SECTION 1 - INTRODUCTION**
- INTRODUCTION ........................................................................................................... 1-1

**SECTION 2 - INSTALLATION**
- OVERVIEW ................................................................................................................. 2-1
- INSTALLING CAT ON AN EXISTING SYSTEM ........................................................... 2-1
- LOADING CAT ............................................................................................................. 2-2

**SECTION 3 - USING THE CUSTOM APPLICATION TOOLKIT**
- PROGRAMMING WITH THE CAT LIBRARIES .......................................................... 3-1
- Methods To Create Custom Software Programs ....................................................... 3-1
- CAT Library Routines ............................................................................................... 3-1
- On-Line CAT Documentation .................................................................................... 3-2
- Tutorial - Usage and Compilation .......................................................................... 3-2
- Compiling the Tutorial Program ............................................................................. 3-2
- RUNNING THE TUTORIAL .......................................................................................... 3-3
- Example Program for LAN-90 Custom Applications Toolkit ................................. 3-3
- WRITING YOUR OWN APPLICATION PROGRAM .................................................. 3-3
  - Adding Your Own Application Software ............................................................... 3-4
  - USING STATION TAGS WITH CAT ........................................................................ 3-5

**SECTION 4 - CAT LIBRARY REFERENCE**
- CAT LIBRARY REFERENCE ...................................................................................... 4-1
- pcvGetAlmDesc() ...................................................................................................... 4-2
- pcvGetAlmcmm() .................................................................................................... 4-3
- pcvGetCiudata() ..................................................................................................... 4-3
- pcvGetDdValues() .................................................................................................. 4-4
- pcvGetHaddr() ....................................................................................................... 4-4
- pcvGetMsdValues() ............................................................................................... 4-5
- pcvGetPriDisp() ..................................................................................................... 4-6
- pcvGetRcmValues() ............................................................................................... 4-7
- pcvGetStatModes() ............................................................................................... 4-8
- pcvGetStatics() ..................................................................................................... 4-9
- pcvGetTagAddr() ................................................................................................... 4-9
- pcvGetTagAlmval() ............................................................................................... 4-10
- pcvGetTagDesc() ................................................................................................. 4-11
- pcvGetTagEu() ...................................................................................................... 4-12
- pcvGetTagIndex() ............................................................................................... 4-12
- pcvGetTagLs() ...................................................................................................... 4-13
- pcvGetTagName() ................................................................................................. 4-14
- pcvGetTagType() ................................................................................................. 4-15
- pcvGetTagValue() ............................................................................................... 4-16
- pcvGetTrdIndex() ............................................................................................... 4-17
- pcvGetValues() .................................................................................................. 4-18
- pcvGetVitals() ..................................................................................................... 4-19
- pcvOutputToStn() ............................................................................................... 4-21
- pcvReadTrendData() ........................................................................................... 4-22
- pcvSendStaMode() .............................................................................................. 4-22
- pcvSendTagValue() ............................................................................................. 4-23
# Table of Contents (continued)

## SECTION 5 - PORTING CAT APPLICATIONS

- PORTING FROM QNX2 TO QNX4 .......................................................... 5-1
  - Translating CAT Header Files and Functions .................................. 5-1
  - Porting Utility ................................................................. 5-2

---

## List of Figures

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Accessing the PCV Install Utility</td>
<td>2-1</td>
</tr>
<tr>
<td>3-1</td>
<td>Main Display for C Tutorial Program</td>
<td>3-3</td>
</tr>
<tr>
<td>3-2</td>
<td>Example of a C Program</td>
<td>3-4</td>
</tr>
<tr>
<td>3-3</td>
<td>Module Logic</td>
<td>3-6</td>
</tr>
</tbody>
</table>

---

## List of Tables

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>Configuration Information Functions</td>
<td>4-1</td>
</tr>
<tr>
<td>4-2</td>
<td>Data Collection Functions</td>
<td>4-1</td>
</tr>
<tr>
<td>4-3</td>
<td>Data Sending Functions</td>
<td>4-2</td>
</tr>
<tr>
<td>4-4</td>
<td>Good State Value</td>
<td>4-6</td>
</tr>
<tr>
<td>4-5</td>
<td>Requested State Value</td>
<td>4-6</td>
</tr>
<tr>
<td>4-6</td>
<td>Station Modes</td>
<td>4-8</td>
</tr>
<tr>
<td>4-7</td>
<td>Analog Alarm States</td>
<td>4-10</td>
</tr>
<tr>
<td>4-8</td>
<td>Database Tag Types</td>
<td>4-15</td>
</tr>
<tr>
<td>4-9</td>
<td>Tag Status Codes Set By pcvGetTagValue()</td>
<td>4-16</td>
</tr>
<tr>
<td>4-10</td>
<td>Tag Quality Codes Set By pcvGetTagValue()</td>
<td>4-17</td>
</tr>
<tr>
<td>4-11</td>
<td>Tag Status Codes Set By pcvGetValues()</td>
<td>4-18</td>
</tr>
<tr>
<td>4-12</td>
<td>Tag Quality Codes Set By pcvGetValues()</td>
<td>4-19</td>
</tr>
<tr>
<td>4-13</td>
<td>Tag Status Codes Set By pcvGetVitals()</td>
<td>4-20</td>
</tr>
<tr>
<td>4-14</td>
<td>Tag Quality Codes Set By pcvGetVitals()</td>
<td>4-21</td>
</tr>
<tr>
<td>4-15</td>
<td>Station Modes Set By pcvSendStaMode()</td>
<td>4-23</td>
</tr>
<tr>
<td>5-1</td>
<td>Header File Translations</td>
<td>5-1</td>
</tr>
<tr>
<td>5-2</td>
<td>Function Name Translations</td>
<td>5-2</td>
</tr>
</tbody>
</table>
Safety Summary

**SPECIFIC CAUTIONS**

If you are running the example programs while your system is on-line and connected to a live process, you must use extreme care. Executing data sending functions, such as “pcvSendTagValue()”, will actually send values to the process. Only use data sending functions in test scenarios. The configuration and data retrieval functions, such as “pcvGetTagValue()”, will not affect your process. (p. 3-3)

Adding your own application program may adversely affect the performance of the system. Also, if your application writes data to a hard disk or peripheral device, make sure adequate storage is available for both LAN-90 PCV and your own application programs. (p. 3-4)

Applications built with CAT should specify the CFLAGS and LDFLAGS options as is done in the CAT sample application (see “/user/cat/example/makefile”):

i.e.,

\[
\text{CFLAGS} = \text{ -2 -fp3 - Wc, “ei” - w1 - O - W3 - M$ (MODEL) - I$ (I)} \\
\text{LFLAGS} = \text{ -2 - M$ (MODEL) - I$ (P) - I$ (L) - I$ (M) - I$ (M) - I$ (N)} \\
\text{ - I$(T) - I$(u) (p. 3-4)}
\]

This function should not be used with station type tags. See the pcvOutputToStn() function. Also see USING STATION TAGS WITH CAT in Section 3. (p. 4-24)

Always check the source code for your program after running portfix, to make sure no incorrect translations have taken place. You may still have to make additional changes to your software before it will compile and run properly. (p. 5-3)
Trademarks and Registrations

Registrations and trademarks in this document include:

- C86 Registered trademark of Computer Innovations, Inc.
- Elsag Bailey Registered trademark of Elsag Bailey Process Automation.
- INFI 90 Registered trademark of Elsag Bailey Process Automation.
- LAN-90 Registered trademark of Elsag Bailey Process Automation.
- PCV Registered trademark of Elsag Bailey Process Automation.
- QNX Registered trademark of QNX Software Systems.
The Custom Applications Toolkit (CAT) allows you to write application software programs for use with the LAN-90 PCV system. The Toolkit allows you unlimited access to the database via easy to use library routines. This means you can add application software (reporting, data storage, calculations, alarm management) specifically tailored for your plant’s requirements. The Custom Applications Toolkit consists of all the necessary tools for creating your own applications. The tools include a programming language (WATCOM’s “C” language compiler) and database access library functions.

This manual is intended for a Software Developer. You must be familiar with a variety of computer programming tools to use CAT. This manual assumes you have a working knowledge of text editors, compilers, and file utilities.
SECTION 2 - INSTALLATION

OVERVIEW

Usually CAT is installed with the base system. Refer to the **LAN-90 PCV Installation manual** for information on installing the Custom Application Toolkit.

These instructions are intended for users who purchase the CAT option as a separate package and install it after the base system has been installed.

To install the compiler, follow the instructions which accompany the WATCOM “C” compiler software package.

INSTALLING CAT ON AN EXISTING SYSTEM

To install the CAT package, follow the path depicted in Figure 2-1.

Figure 2-1. Accessing the PCV Install Utility
LOADING CAT

CAT is shipped as an archived volume.

When the installation program asks you to place the first disk in the floppy drive, insert the CAT disk and press <Enter>.

When the program asks if you have other disks to install, press <F10> to continue to the next step.
SECTION 3 - USING THE CUSTOM APPLICATION TOOLKIT

PROGRAMMING WITH THE CAT LIBRARIES

The Custom Applications Toolkit (CAT) allows you to write application software programs for use with the LAN-90 PCV system.

Methods To Create Custom Software Programs

While developing your applications, there are several tools you must use. Some of the tools required are:

- Text Editors.
- Compilers.
- File Utilities.

A text editor is used to create the program source code. A text editor (vedit) is provided with the QNX® operating system.

Compilers translate the program source code into executable files. WATCOM's "C" compiler is included with CAT.

File utilities are used to back up source and executable files to removable media (e.g., floppy disks). Some of the utilities that can be useful during software development are: "cp", "vol", "pax", "ls", and "size". See your QNX Operating System Manual for details.

You must be familiar with the above tools before you can begin developing software.

Plan and implement your programs using top down design methods. You can include in your program the library calls to receive and send process information. For example, your application could gather process data from many tags (e.g., ten temperature input tags configured in the database), perform a complex calculation, then send the result of the calculation back through another tag. You could implement this program as a background process that executes periodically (e.g., once an hour), or as a foreground process that executes whenever selected by the operator (from the Application menu).

CAT Library Routines.

The Custom Applications Toolkit provides several library routines that application programs can use to access the database. Note that each library call accesses the database administrator programs. The library routine calls are inserted
into the application source code, and are linked into the executable modules during the compile/link phases.

Section 4 contains a complete reference to all the CAT library routines.

On-Line CAT Documentation

The CAT library routine descriptions are available on-line to the developer. To access this on-line “manual”, the catman function is provided. To use it, simply type catman, followed by the name of the function you wish to see described. For example, typing:

```
catman pcvGetTagValue
```

will show a description of the pcvGetTagValue function.

Tutorial - Usage and Compilation

The CAT package comes complete with a tutorial program. The tutorial program illustrates the use of the library routines. The example program is in the “/usr/cat/example” directory and is called “testc.c” (C language tutorial program).

A “makefile” is also included with the example program. The “makefile” is used to compile and link the example program. You can add your application programs to the existing “makefile” or create your own.

Compiling the Tutorial Program

To compile the C tutorial program, enter the following commands at the QNX prompt:

```
cd /usr/cat/example
make testc
```

The program source directory.

Compile and link “testc.c”.

RUNNING THE TUTORIAL

The tutorial program is designed to show you how each CAT function works, and what parameters are needed by each function. To run the C tutorial program, type:

```
cd /usr/cat/example
testc
```

The tutorial program allows you to execute each function and enter the various parameters (Figure 3-1).
Example Program for LAN-90 Custom Applications Toolkit

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EXIT THE PROGRAM</td>
</tr>
<tr>
<td>1</td>
<td>pcvGetTagIndex</td>
</tr>
<tr>
<td>2</td>
<td>pcvGetTagType</td>
</tr>
<tr>
<td>3</td>
<td>pcvGetTagName</td>
</tr>
<tr>
<td>4</td>
<td>pcvGetTagDesc</td>
</tr>
<tr>
<td>5</td>
<td>pcvGetTagDefault</td>
</tr>
<tr>
<td>6</td>
<td>pcvGetTagHaddr</td>
</tr>
<tr>
<td>7</td>
<td>pcvGetTagEu</td>
</tr>
<tr>
<td>8</td>
<td>pcvGetTagLs</td>
</tr>
<tr>
<td>9</td>
<td>pcvGetPriDisp</td>
</tr>
<tr>
<td>10</td>
<td>pcvGetStatics</td>
</tr>
<tr>
<td>11</td>
<td>pcvGetAlmval</td>
</tr>
<tr>
<td>12</td>
<td>pcvGetAlmDesc</td>
</tr>
<tr>
<td>13</td>
<td>pcvGetTagAddr</td>
</tr>
<tr>
<td>14</td>
<td>pcvGetTrdIndex</td>
</tr>
<tr>
<td>15</td>
<td>pcvGetTagValue</td>
</tr>
<tr>
<td>16</td>
<td>pcvGetRstValue</td>
</tr>
<tr>
<td>17</td>
<td>pcvGetDvValues</td>
</tr>
<tr>
<td>18</td>
<td>pcvGetMdValues</td>
</tr>
<tr>
<td>19</td>
<td>pcvGetStatModes</td>
</tr>
<tr>
<td>20</td>
<td>pcvGetVitals</td>
</tr>
<tr>
<td>21</td>
<td>pcvGetValues</td>
</tr>
<tr>
<td>22</td>
<td>pcvGetAemcmmt</td>
</tr>
<tr>
<td>23</td>
<td>pcvGetCiudata</td>
</tr>
<tr>
<td>24</td>
<td>pcvReadTrendData</td>
</tr>
<tr>
<td>25</td>
<td>pcvSendTagValue</td>
</tr>
<tr>
<td>26</td>
<td>pcvSendStaMode</td>
</tr>
<tr>
<td>27</td>
<td>pcvOutputToStn</td>
</tr>
</tbody>
</table>

Figure 3-1. Main Display for C Tutorial Program

CAUTION
If you are running the example programs while your system is on-line and connected to a live process, you must use extreme care. Executing data sending functions, such as “pcvSendTagValue()”, will actually send values to the process. Only use data sending functions in test scenarios. The configuration and data retrieval functions, such as “pcvGetTagValue()”, will not affect your process.

WRITING YOUR OWN APPLICATION PROGRAM

This section describes how you would develop your own application using a simple C program as an example. This simple program retrieves the value for a process tag (TANK LEVEL) and periodically writes the value to the screen.

Create your own directory to store your source programs and executable files by typing:

```
mkdir /usr/cat/mydir
```

To begin editing your program source code using the QNX text editor, type:

```
cd /usr/cat/mydir
vedit myprog.c
```

You can create an application program that includes the code shown in Figure 3-2, which uses some of the CAT library routines.
After creating the complete program (including headers, main function, etc.) and creating a "makefile", the executable program can be compiled. More details regarding include files and libraries are contained inside the tutorial programs.

Adding Your Own Application Software

The methods you use to add your application software to the system depend on the nature of the application program. If an application program is initiated by a user selection (i.e., a foreground operation), the program should be added using the Application menu (see Editing The Application Menu in the Configuration Manual).

If an application must run constantly (i.e., a background application), the program should be added to the system start-up file. Note that the system start-up file is in the "/etc/config" directory and is named "sysinit.#" where # is the node number (0 for non-networked systems). The application program start-up should be placed after the LAN-90 PCV system is started.

Figure 3-2. Example of a C Program

```c
/* get the internal tag index */
itag = pcvGetTagIndex("TANK LEVEL");
while (1) {
    return_code = pcvGetTagValue(itag, i, &dvalue, &dstatus);
    if (return_code) {
        printf("Error calling function pcvGetTagValue\n");
    } else {
        printf("Value for TANK LEVEL - %f", dvalue);
    }
    sleep(10); /* delay for 10 seconds */
}
```

CAUTION

Adding your own application program may adversely affect the performance of the system. Also, if your application writes data to a hard disk or peripheral device, make sure adequate storage is available for both LAN-90 PCV and your own application programs.

Applications built with CAT should specify the CFLAGS and LDFLAGS options as is done in the CAT sample application (see "/user/cat/example/makefile"):

i.e.,

CFLAGS = -2 - fp3 - Wc, “ei” - wl - O - W3 - M$ (MODEL) - I$ (I)
LDFLAGS = -2 - M$ (MODEL) - I$ (P) - I$ (L) - I$ (M) - I$ (N)
        - I$(T) - I$(u)
The Custom Application Toolkit allows you to create a program that adjusts the set point of a station defined in the database. Using an equation based on the current value of related tags, CAT calculates the set point. This set point adjustment feature can be enabled and disabled using a standard graphic station faceplate.

The Custom Application Toolkit interfaces with the system and forwards requests to the INFI 90 OPEN system.

The user-written application program should be designed to adjust the set point periodically and the period calculated is faster than the computer timeout specification configured in the station block. The station outputs should be performed using the “pcvOutputToStn()” function.

Each time the application is run, it first calculates the required set point, then calls the “pcvOutputToStn()” function. If the station is in computer mode, the calculated set point value is sent. If the station is not in computer mode, the function sends a signal to the station indicating that a valid set point has been calculated and can be transmitted once the computer mode option is chosen.

If you want to control a number of stations but do not want to manually put each one into computer mode, the module logic shown in Figure 3-3 could be used to allow a single RCM to control the mode of several station blocks. No change to the application is required to operate in this manner.
## Figure 3-3. Module Logic

1. All stations in the group must be in 'AUTO' mode prior to allowing the operator to enable the 'computer' mode. (from N-2 output of station)

2. The 'computer' application (program) should run continuously and mainstream the 'computer ok' status. Even when the stations are not in 'computer' mode. 'Computer ok' is used as a permissive and allows the operator to select 'computer' mode. If the 'computer ok' watchdog times out on any station the 'computer' mode selection is reset (to local). (from N-5 output of station)

3. All stations in the group must be in 'computer' mode to allow the feedback on the 'computer' mode selector to indicate that 'computer' mode was achieved. (from N-3 output of station)

4. The 'computer' mode selector allows the operator to place all stations in the group into 'computer' mode.

5. The 'SCOMP' signal forces all stations in the group into 'computer' mode. (to S22 input of station)

6. The 'SLOCAL' signal forces all stations in the group into 'local' mode when 'computer' mode not selected. (to S21 input of station)
SECTION 4 - CAT LIBRARY REFERENCE

CAT LIBRARY REFERENCE

Each function is described in this section.

Table 4-1. Configuration Information Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcvGetTagIndex</td>
<td>Retrieves the tag index from the tag database. Note that the tag index is used to call most other library routines.</td>
</tr>
<tr>
<td>pcvGetTagType</td>
<td>Retrieves the tag type.</td>
</tr>
<tr>
<td>pcvGetTagName</td>
<td>Retrieves the tag name.</td>
</tr>
<tr>
<td>pcvGetTagDesc</td>
<td>Retrieves the tag descriptor.</td>
</tr>
<tr>
<td>pcvGetTagDefault</td>
<td>Retrieves the tag default value.</td>
</tr>
<tr>
<td>pcvGetTagAddr</td>
<td>Retrieves the tag address (loop, PCU, module, block) as defined in the tag database.</td>
</tr>
<tr>
<td>pcvGetTagEu</td>
<td>Retrieves the tag engineering units descriptor.</td>
</tr>
<tr>
<td>pcvGetTagLs</td>
<td>Retrieves the tag logic state descriptor (discrete type tags only).</td>
</tr>
<tr>
<td>pcvGetPriDisp</td>
<td>Retrieves the primary display name.</td>
</tr>
<tr>
<td>pcvGetStatics</td>
<td>Retrieves static information—includes tag name, tag descriptor, digital indicator, decimal precision.</td>
</tr>
<tr>
<td>pcvGetTagAlmval</td>
<td>Retrieves the alarm value.</td>
</tr>
<tr>
<td>pcvGetAlmDesc</td>
<td>Returns the high and low alarm comments of the specified tag.</td>
</tr>
<tr>
<td>pcvGetHaddr</td>
<td>Returns the tag index which matches the given hardware address.</td>
</tr>
<tr>
<td>pcvGetTndIndex</td>
<td>Gets the trend index for the given tag.</td>
</tr>
</tbody>
</table>

Table 4-2. Data Collection Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcvGetTagValue</td>
<td>Retrieves the tag value and status.</td>
</tr>
<tr>
<td>pcvGetRcmValues</td>
<td>Retrieves the tag values associated with Remote Control Memory (RCM) tags.</td>
</tr>
<tr>
<td>pcvGetDdValues</td>
<td>Retrieves the tag values associated with Device Driver type tags.</td>
</tr>
<tr>
<td>pcvGetMsdValues</td>
<td>Retrieves the tag values associated with Multistate Device Driver (MSDD) tags.</td>
</tr>
<tr>
<td>pcvGetStatModes</td>
<td>Retrieves the current mode of a station tag.</td>
</tr>
<tr>
<td>pcvGetVitals</td>
<td>Retrieves vital tag information—includes tag name, tag descriptor, value, alarm status, alarm value, engineering unit or logic state descriptor, decimal precision.</td>
</tr>
</tbody>
</table>
pcvGetAlmDesc()

Retrieves the high and low alarm comment for the given tag.

C Synopsis

```c
#include <pcvDServ.h>
int pcvGetAlmDesc(TagIndex index,
                   AlarmComment *cmmt)
```

Arguments

- **index**: Tag index.
- **cmmt**: Pointer to storage for AlarmComment structure.

Return Value

- **BCI_OK**: Success.
- **BCI_WARN**: Data has been returned but may be invalid, check errno.
- **BCI_FATAL**: An error has occurred, and errno has been set.

Description

pcvGetAlmDesc() returns both of the alarm comments corresponding to the given tag. If successful, “cmmt.High_alarm” will contain the high alarm comment string. “cmmt.Low_alarm” will contain the low alarm comment string.

Errors

- **ENODBA**: Invalid index.
- **EINVAL**: cmmt pointer is NULL.
- **ENOSERVER**: Database server is not running.
- **ENODAEMON**: Mail manager is not running.
pcvGetAlmcmmt()

Retrieves either the high or low alarm comment for the given tag.

C Synopsis

```c
#include <pcvDServ.h>
int pcvGetAlmcmmt(TagIndex index, TagStatus state, char *cmmt)
```

Arguments

- `index`: Tag index.
- `status`: Alarm status of comment to return.
- `cmmt`: Pointer to storage for comment string.

Return Value

- `BCI_OK`: Success.
- `BCI_WARN`: Data has been returned but may be invalid, check errno.
- `BCI_FATAL`: An error has occurred, and errno has been set.

Description

pcvGetAlmcmmt() returns the alarm comment appropriate for the alarm status specified for the given tag. “cmmt” must point to a buffer at least 65 characters long.

Errors

- `ENODBA`: Invalid index.
- `EBADSTATUS`: Invalid alarm state.
- `EINVAL`: cmmt pointer is NULL.
- `ENOSERVER`: Database server is not running.
- `ENODAEMON`: Mail manager is not running.

pcvGetCiuData()

Retrieves dynamic database information for the given tag.

C Synopsis

```c
#include <pcvDServ.h>
int pcvGetCiuData(TagIndex index, CiuData *ciuData)
```

Arguments

- `index`: Tag index.
- `ciuData`: Pointer to storage for CiuData structure.

Return Value

- `BCI_OK`: Success.
- `BCI_WARN`: Data has been returned but may be invalid, check errno.
- `BCI_FATAL`: An error has occurred, and errno has been set.

Description

pcvGetCiuData() returns a copy of the dynamic database corresponding to the given tag.
pcvGetDdValues()

Retrieves process values of a device driver (DD) tag.

C Synopsis
#include <pcvDServ.h>
int pcvGetDdValues(TagIndex index, char *quality,
                  char *alarm, char *v, char *f2, char *f1,
                  char *fs, char *so, char *mode)

Arguments
index Tag index.
quality Quality indicator, 1 if bad quality else 0.
alarm 1 if in alarm, else 0.
v Output value of block, 1 or 0.
f2 1 if input 2 feedback state is 1, else 0.
f1 1 if input 1 feedback state is 1, else 0.
fs 1 if feedback status value bad is 1, else 0.
so 1 if override value is 1, else 0.
mode 0=auto, 1=remote, 2=manual, 3=unused.

Return Value
BCI_OK Success.
BCI_WARN Tag not established, invalid data may be returned.
BCI_FATAL An error has occurred, and errno has been set.

Description
pcvGetDdValues() returns process values from a device driver (DD) tag. The index passed must be the tag index of a DD and the tag must be established in the CIU in order for the function to succeed. You can use the pcvGetTagType() function to check the tag type of the index. If the function call is successful, the variables quality, alarm, v, f2, f1, fs, so, and mode contain the process values for the DD.

Errors
EDISEST Tag has not been established.
ECONFIG Tag at specified index is not a Device Driver.
ENODBA Invalid index.
ENOSERVER Database server is not running.
ENODAEMO Mail manager is not running.

tcvGetHaddr()

Retrieves tag index matching given hardware address.
C Synopsis

```c
#include <pcvDServe.h>
TagIndex pcvGetHaddr(u_int loop, u_int pcu,
             u_int mod, u_int block)
```

Arguments

- `loop` Tag loop address.
- `pcu` Tag pcu address.
- `mod` Tag module address.
- `block` Tag block address.

Return Value

- `>0` Database index of tag matching hardware address.
- `BCI_ERROR` Could not find requested tag address in database.
- `BCI_FATAL` An error has occurred, and errno has been set.

Description

`pcvGetHaddr()` returns the tag database index of the tag configured at the specified hardware address.

Errors

- `ENODBA` Unable to locate tag with matching address.
- `ENOSERVER` Database server is not running.
- `ENODAEMON` Mail manager is not running.

---

`pcvGetMsdValues()`

Retrieves process values from a multistate device driver (MSDD) tag.

C Synopsis

```c
#include <pcvDServe.h>
int pcvGetMsdValues(TagIndex index, char *quality,
             char *alarm, char *so, char *co, char *m,
             char *v, char *f1, char *f2, char *f3,  
             char *f4, char *gs, char *rs)
```

Arguments

- `index` Tag index.
- `quality` Quality indicator, 1 if bad quality, else 0.
- `alarm` 1 if alarm, else 0.
- `so` 1 if status override value is 1, else 0.
- `co` 1 if control override value is 1, else 0.
- `m` 1 if auto mode is 1, else manual 0.
- `v` 1 if in control output state 1, else 0.
- `f1` 1 if input 1 feedback state is 1, else 0.
- `f2` 1 if input 2 feedback state is 1, else 0.
- `f3` 1 if input 3 feedback state is 1, else 0.
- `gs` Good State table value (Table 4-4).
- `rs` Requested State table value (Table 4-5).
pcvGetPriDisp() retrieves the primary display name from the database for a given tag index.

**C Synopsis**
```
#include <pcvDServ.h>
int pcvGetPriDisp(TagIndex index, char *pdstring)
```

**Arguments**
- `index` Tag index.
- `pdstring` Pointer to character string storage for display name, 9 characters.

**Return Value**
- `BCI_OK` Success.

---

### Description
pcvGetMsdValues() returns process values from a multistate device driver (MSDD) tag. The index passed must be the tag index of an MSDD and the tag must be established in the CIU in order for the function to succeed. You can use the pcvGetTagType() function to check the tag type of the index. If the function call is successful, the variables `quality`, `alarm`, `so`, `co`, `v`, `f1`, `f2`, `f3`, `f4`, `gs`, and `rs` contain the process values for the MSDD.

### Errors
- **EDISEST** Tag has not been established.
- **ECONFIG** Tag at specified index is not a Multistate Device Driver.
- **ENODBA** Invalid index.
- **ENOSERVER** Database server is not running.
- **ENODAEMON** Mail manager is not running.

### Table 4-4. Good State Table Value

<table>
<thead>
<tr>
<th>Good State</th>
<th>State Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

### Table 4-5. Requested State Table Value

<table>
<thead>
<tr>
<th>Requested State</th>
<th>State Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Description
pcvGetPriDisp() returns the primary display name from the
database using the passed tag index. If the function call is
successful, pdstring will contain the file name of the primary
display for the tag.

Errors
ENODBA Invalid index.
EINVAL NULL pdsring.
ENOSERVER Database server is not running.
ENODAEMON Mail manager is not running.

pcvGetRcmValues()

Retrieves process values from a Remote Control Memory (RCM)
tag.

C Synopsis
#include <pcvDServ.h>
int pcvGetRcmValues(TagIndex index, char *quality,
char *alarm, char *ov, char *si, char *sp,
char *ri, char *fb, char *sc, char *rc)

Arguments
index Tag index.
quality Quality indicator, 1 if bad quality, else 0.
alarm 1 if alarm, else 0.
ov Output value of block, 1 or 0.
si 1 if logic set input is 1, else 0.
sp 1 if set permissive input is 1, else 0.
ri 1 if logic reset input value is 1, else 0.
fb 1 if feedback value is 1, else 0.
sc 1 if set command is 1, else 0.
rc 1 if reset command value is 1, else 0.

Return Value
BCI_OK Success.
BCI_WARN Tag not established, invalid data may be
returned.
BCI_FATAL An error has occurred, and errno has been
set.

Description
pcvGetRcmValues() returns process values from a Remote
Control Memory (RCM) tag. The index passed must be the tag
index of an RCM and the tag must be established in the CIU in
order for the function to succeed. You can use the pcvGetTag-
Type() function to check the tag type of the index. If the func-
tion call is successful, the variables quality, alarm, ov, si, sp,
ri, fb, sc, and rc contain the process values for the RCM.
pcvGetStatModes()

Retrieves the current station modes from a station tag.

C Synopsis
#include <pcvDserv.h>
int pcvGetStatModes(TagIndex index,
char *auto_man, char *cas_rat,
char *stn_lev, char *cmpt_stat,
char *stat_fault, char *op_trk,
char *man_lock, char *by_pass)

Arguments
index Tag index.

Table 4-6. Station Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Definition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>auto_man</td>
<td>Auto/Manual Mode Indicator</td>
<td>Auto</td>
</tr>
<tr>
<td>cas_rat</td>
<td>Cascade or Ratio/ Normal Indicator</td>
<td>Cascade or Ratio</td>
</tr>
<tr>
<td>stn_lev</td>
<td>Station Level</td>
<td>Computer</td>
</tr>
<tr>
<td>cmpt_stat</td>
<td>Computer OK Status</td>
<td>Computer OK</td>
</tr>
<tr>
<td>stat_fault</td>
<td>Station Output Status</td>
<td>Station Fail</td>
</tr>
<tr>
<td>op_trk</td>
<td>Output Tracking</td>
<td>Track</td>
</tr>
<tr>
<td>man_lock</td>
<td>Manual Interlock</td>
<td>Yes</td>
</tr>
<tr>
<td>by_pass</td>
<td>Station in Bypass Mode</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Return Value
- BCI_OK Success.
- BCI_WARN Tag not established, invalid data may be returned.
- BCI_FATAL An error has occurred, and errno has been set.

Description
pcvGetStatModes() returns the current station modes from the database. The tag must be established in the CIU in order for the function to succeed.

NOTE: This function does NOT check the tag type of the index you send it. Be sure to check the tag type yourself by using the pcvGetTagType() function.
If the function call is successful, the variables auto_man, cas_rat, stn_lev, cmpt_stat, stat_fault, op_trk, man_lock, and by_pass contain the modes of the station.

**Errors**
- **EDISEST** Tag has not been established.
- **ENOSERVER** Database server is not running.
- **ENODAEMON** Mail manager is not running.

---

**pcvGetStatics()**

Retrieves static tag information from the tag database using the passed tag index.

**C Synopsis**
```
#include <pcvDServ.h>
int pcvGetStatics(TagIndex index, char *name, 
char *desc, Bool *dig, u_char *dec)
```

**Arguments**
- **index** Tag index.
- **name** Pointer to character string storage for tag name; maximum 15 characters.
- **desc** Pointer to character string storage for tag descriptor; maximum 33 characters.
- **dig** Pointer to Boolean storage for digital indicator; 1 if digital.
- **dec** Pointer to unsigned character storage for number of decimals defined.

**Return Value**
- **BCI_OK** Success.
- **BCI_WARN** Tag not established, invalid data may be returned.
- **BCI_FATAL** An error has occurred, and errno has been set.

**Description**
pcvGetStatics() returns the static tag information from the tag database. If the function call is successful, the variables name, desc, dig, and dec contain the static configuration information for the tag.

**Errors**
- **ENOSERVER** Database server is not running.
- **ENODAEMON** Mail manager is not running.

---

**pcvGetTagAddr()**

Retrieves the tag location (loop, PCU, module, block) from the tag database using the passed tag index.

**C Synopsis**
```
#include <pcvDServ.h>
TagType pcvGetTagAddr(TagIndex index, u_int *loop, 
 u_int *pcu, u_int *mod, u_int *block)
```
Description
pcvGetTagAddr() will return the INFI 90 OPEN block location using the passed tag index parameter. The block location consists of a PCU, module, and block address as defined in the tag database.

Errors
- ENOSERVER: Database server is not running.
- ENODAEMON: Mail manager is not running.

**pcvGetTagAlmval()**

Retrieves the alarm value for an analog-type tag’s given alarm state.

**C Synopsis**

```c
#include <pcvDServ.h>
int pcvGetTagAlmval (TagIndex index, TagStatus state, float *value)
```

**Arguments**

- **index**: Tag index.
- **state**: Alarm state.
- **value**: Pointer to floating point storage for value.

**Table 4-7. Analog Alarm States**

<table>
<thead>
<tr>
<th>State</th>
<th>Alarm State</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>Low</td>
</tr>
<tr>
<td>6</td>
<td>High Deviation</td>
</tr>
<tr>
<td>7</td>
<td>Low Deviation</td>
</tr>
<tr>
<td>8</td>
<td>2 High</td>
</tr>
<tr>
<td>9</td>
<td>2 Low</td>
</tr>
<tr>
<td>10</td>
<td>3 High</td>
</tr>
<tr>
<td>11</td>
<td>3 Low</td>
</tr>
</tbody>
</table>
pcvGetTagAlmval() returns the alarm value for the given the alarm state.

NOTE: This function can only be used for analog, internal analog, analog report, RMSC, and station tags. You can check the tag type of the index by using the pcvGetTagType() function.

If the function call is successful, the variable value contains value for the given alarm state.

Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDISEST</td>
<td>Tag has not been established.</td>
</tr>
<tr>
<td>ENODBA</td>
<td>Invalid index.</td>
</tr>
<tr>
<td>EBADTYPE</td>
<td>Invalid status value.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Value pointer is NULL.</td>
</tr>
<tr>
<td>ENOSERVER</td>
<td>Database server is not running.</td>
</tr>
<tr>
<td>ENODAEMON</td>
<td>Mail manager is not running.</td>
</tr>
</tbody>
</table>

pcvGetTagDefault()

Retrieves the tag default value from the tag database using the passed tag index.

C Synopsis

```c
#include <pcvDServ.h>
int pcvGetTagDefault(TagIndex index, float *value)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Tag index.</td>
</tr>
<tr>
<td>value</td>
<td>Pointer to floating point storage for value.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCI_OK</td>
<td>Success.</td>
</tr>
<tr>
<td>BCI_WARN</td>
<td>Data has been returned but may be invalid, check errno.</td>
</tr>
<tr>
<td>BCI_ERROR</td>
<td>An expected error has occurred, check errno.</td>
</tr>
<tr>
<td>BCI_FATAL</td>
<td>An unexpected error has occurred, check errno.</td>
</tr>
</tbody>
</table>

Description

pcvGetTagDefault() returns the default tag value as defined in the tag database. If the function call is successful, the variable value contains the default value defined for the tag. This default value can be used as a substitute value if a piece of equipment is faulty or out of service.
pcvGetTagDesc()

Retrieves the tag description using the passed tag index.

C Synopsis
#include <pcvDServ.h>
int pcvGetTagDesc(TagIndex index, char *dsstring)

Arguments
- index: Tag index.
- dsstring: Pointer to character string storage for descriptor, maximum 33 characters.

Return Value
- BCI_OK: Success.
- BCI_WARN: Data has been returned but may be invalid, check errno.
- BCI_ERROR: An expected error has occurred, check errno.
- BCI_FATAL: An unexpected error has occurred, check errno.

Description
pcvGetTagDesc() will return the tag descriptor for the passed tag index. If the function call is successful, the variable dsstring contains the tag descriptor. The descriptor is configured in the tag database.

Errors
- ENODBA: Invalid index.
- EINVAL: Value pointer is NULL.
- ENOSERVER: Database server is not running.
- ENODAEMON: Mail server is not running.

pcvGetTagEu()

Retrieves the tag engineering units descriptor using the passed tag index.

C Synopsis
#include <pcvDServ.h>
int pcvGetTagEu(TagIndex index, char *eustring)

Arguments
- index: Tag index.
- eustring: Pointer to character string storage for engineering units, maximum seven characters.
Description
pcvGetTagEu() will return the tag engineering units (if applicable for the tag type) for the passed tag index. The engineering units are defined inside the INFI 90 OPEN Blockware (via index numbers), and reference the engineering units list defined in LAN-90 PCV. Internal analog and analog report tags have their engineering unit index configured inside the tag database only. If the function call is successful, the variable eustring contains the engineering units descriptor for the tag.

NOTE: This function applies to analog type tags only.

Errors
ENODBA Invalid index.
EINVAL eustring pointer is NULL.
ENOSERVER Database server is not running.
ENODAEMON Mail manager is not running.

pcvGetTagIndex()

Retrieves the tag index from the tag database.

C Synopsis
#include <pcvDServ.h>
TagIndex pcvGetTagIndex(char *name)

Arguments
name Pointer to an ASCII tag name.

Return Value
>0 Index of tag with matching tag name.
BCI_ERROR Tag name was blank or could not be found, check errno.
BCI_FATAL An unexpected error has occurred, check errno.

Description
pcvGetTagIndex() will return the tag index using the passed tag name parameter. If the passed tag name matches an entry in the tag configuration, then the corresponding index identifier is returned. The returned tag index can then be used to call several interface subroutines available.

Errors
ENODBA Tag name could not be found.
EINVAL Tag name pointer is NULL.
ENOSERVER Database server is not running.
ENODAEMON Mail manager is not running.
**pcvGetTagLs()**

Retrieves the logic state descriptor using the passed tag index and state indicator.

**C Synopsis**

```c
#include <pcvDServ.h>
int pcvGetTagLs(TagIndex index, int state, char *lsstring)
```

**Arguments**

- **index**: Tag index.
- **state**: 0 for zero state descriptor, 1 for one state descriptor.
- **lsstring**: Pointer to character string storage for logic state descriptor; maximum 7 characters.

**Return Value**

- **BCI_OK**: Success.
- **BCI_WARN**: Data has been returned but may be invalid, check errno.
- **BCI_ERROR**: An expected error has occurred, check errno.
- **BCI_FATAL**: An unexpected error has occurred, check errno.

**Description**

pcvGetTagLs() will return the tag logic state descriptor (if applicable for the tag type) for the passed tag index and state. The logic state descriptors are defined inside the tag database.

**Errors**

- **ENODBA**: Invalid index.
- **EBADSTATE**: Invalid state.
- **EINVAL**: lsstring pointer is NULL.
- **ENOSERVER**: Database server is not running.
- **ENODAEMON**: Mail manager is not running.

---

**pcvGetTagName()**

Retrieves the tag name.

**C Synopsis**

```c
#include <pcvDServ.h>
int pcvGetTagName(TagIndex index, char *nmstring)
```

**Arguments**

- **index**: Tag index.
- **nmstring**: Pointer to character string storage for name, maximum 15 characters.

**Return Value**

- **BCI_OK**: Success.
- **BCI_WARN**: Data has been returned but may be invalid, check errno.
- **BCI_ERROR**: An expected error has occurred, check errno.
- **BCI_FATAL**: An unexpected error has occurred, check errno.
**pcvGetTagName()** will return the tag name for the passed tag index. The descriptor is configured in the tag database.

**pcvGetTagType()**

Retrieves the tag type

C Synopsis

```
#include <pcvDServ.h>
TagType pcvGetTagType(TagIndex index)
```

**Arguments**

<table>
<thead>
<tr>
<th>index</th>
<th>Tag index.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>&gt;0</th>
<th>Returned tag type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCI_WARN</td>
<td>Data has been returned but may be invalid, check errno.</td>
</tr>
<tr>
<td>BCI_ERROR</td>
<td>An expected error has occurred, check errno.</td>
</tr>
<tr>
<td>BCI_FATAL</td>
<td>An unexpected error has occurred, check errno.</td>
</tr>
</tbody>
</table>

**Description**

pcvGetTagType() will return the tag type using the passed tag parameter. The returned type will match the entry as defined in the tag database. The following tables list the various return codes and matching tag type identifiers.

**Table 4-8. Database Tag Types**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Undefined.</td>
</tr>
<tr>
<td>5</td>
<td>Analog.</td>
</tr>
<tr>
<td>7</td>
<td>Digital.</td>
</tr>
<tr>
<td>12</td>
<td>Analog Report.</td>
</tr>
<tr>
<td>13</td>
<td>Digital Report.</td>
</tr>
<tr>
<td>14</td>
<td>Module Status.</td>
</tr>
<tr>
<td>15</td>
<td>Remote Control Memory (RCM).</td>
</tr>
<tr>
<td>17</td>
<td>Station.</td>
</tr>
<tr>
<td>18</td>
<td>Remote Manual Set Constant (RMSC).</td>
</tr>
<tr>
<td>19</td>
<td>Device Driver (DD).</td>
</tr>
<tr>
<td>22</td>
<td>Multistate Device Driver (MSDD).</td>
</tr>
<tr>
<td>23</td>
<td>Remote Motor Control Block (RMCB).</td>
</tr>
<tr>
<td>24</td>
<td>Digital Acquisition Analog (DAANALG).</td>
</tr>
<tr>
<td>28</td>
<td>Text.</td>
</tr>
<tr>
<td>31</td>
<td>ASCII Text Block.</td>
</tr>
</tbody>
</table>
Table 4-8. Database Tag Types (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Internal Analog.</td>
</tr>
<tr>
<td>33</td>
<td>Internal Digital.</td>
</tr>
</tbody>
</table>

Errors
- ENODBA: Invalid index.
- ENOSERVER: Database server is not running.
- ENODAEMON: Mail manager is not running.

*pcvGetTagValue()*

Retrieves the current tag value and status from the tag database using the passed tag index.

C Synopsis

```c
#include <pcvDServ.h>
int pcvGetTagValue(TagIndex index, u_char parm,
                   float *value, TagStatus *status,
                   TagQuality *quality)
```

Arguments

- **index**: Tag index.
- **parm**: For station’s tag type, specify:
  - 1: PV (Process Variable).
  - 2: SP (Set Point).
  - 3: CO (Control Output).
  - 4: RI (Ratio Index).
- **value**: Pointer to floating point storage for value.
- **status**: Pointer to TagStatus storage for status of Value. The following Table shows all status Codes with descriptions:

Table 4-9. Tag Status Codes Set By pcvGetTagValue()

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Alarm.</td>
</tr>
<tr>
<td>1</td>
<td>Alarm (Boolean).</td>
</tr>
<tr>
<td>2</td>
<td>Bad Quality.</td>
</tr>
<tr>
<td>3</td>
<td>Out of Service.</td>
</tr>
<tr>
<td>4</td>
<td>High.</td>
</tr>
<tr>
<td>5</td>
<td>Low.</td>
</tr>
<tr>
<td>6</td>
<td>High Deviation.</td>
</tr>
<tr>
<td>7</td>
<td>Low Deviation.</td>
</tr>
</tbody>
</table>

- **quality**: Pointer to TagQuality storage for quality of Value. The following list shows all quality Codes with descriptions:
pcvGetTrdIndex()

Gets the index of the trend for the given tag name, trend type and interval.

C Synopsis

```
#include <pcvDServ.h>
int pcvGetTrdIndex(char *tagName, time_t interval,
                   char subType, char type, TrendIndex *index)
```

Arguments

tagName  Name of trended tag.
interval  Trend resolution.
subType   Trend subtype:
          TRDST_PV - Station Process Variable.
          TRDST_SP - Station Setpoint.
          TRDST_CO- Station Control Output.
          TRDST_RA- Station Ratio Index.

type     Trend type:
          tt_normal- “regular” trend.
          tt_tuning - tuning trend.
          tt_opassign - operator assignable trend.

index    Pointer to TrendIndex storage for trend index.

Table 4-10.  Tag Quality Codes Set
By pcvGetTagValue()

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Good.</td>
</tr>
<tr>
<td>1</td>
<td>Bad.</td>
</tr>
<tr>
<td>2</td>
<td>Disestablished.</td>
</tr>
<tr>
<td>3</td>
<td>Substituted.</td>
</tr>
<tr>
<td>4</td>
<td>Suspect.</td>
</tr>
<tr>
<td>5</td>
<td>Alarm Inhibited (auto).</td>
</tr>
<tr>
<td>6</td>
<td>Alarm Inhibited (group).</td>
</tr>
<tr>
<td>7</td>
<td>Alarm Inhibited (manual).</td>
</tr>
</tbody>
</table>

Return Value

BCI_OK      Success.
BCI_WARN    Data has been returned but may be invalid, check errno.
BCI_FATAL   An unexpected error has occurred, check errno.

Description

pcvGetTagValue() will return the current tag value, status and quality from the tag database.

Errors

EDISEST  Tag has not been established.
ENODBA   Invalid index.
ENOSERVER Database server is not running.
ENODAEMON Mail manager is not running.
pcvGetTrdIndex() searches the trend database for a trend which matches the characteristics specified. If an exact match for the trend resolution is not found, it will return the index of the trend with the closest resolution.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENODBA</td>
<td>Invalid index.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Data pointer is NULL.</td>
</tr>
<tr>
<td>ENOSERVER</td>
<td>Trend server is not running.</td>
</tr>
<tr>
<td>ENODAEMON</td>
<td>Mail manager is not running.</td>
</tr>
</tbody>
</table>

### C Synopsis

```c
#include <pcvDServ.h>

int pcvGetValues(TagIndex index, float *value, TagStatus *status, float *alarm, char *euls, Bool *dig, u_char *dec, TagQuality *quality)
```

### Arguments

- **index**: Tag index.
- **value**: Pointer to floating point storage for value.
- **status**: Pointer to TagStatus storage for status of value.

### Table 4-11. Tag Status Codes Set By pcvGetValues()

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Alarm.</td>
</tr>
<tr>
<td>1</td>
<td>Alarm (Boolean).</td>
</tr>
<tr>
<td>2</td>
<td>Bad Quality.</td>
</tr>
<tr>
<td>3</td>
<td>Out of Service.</td>
</tr>
<tr>
<td>4</td>
<td>High.</td>
</tr>
<tr>
<td>5</td>
<td>Low.</td>
</tr>
<tr>
<td>6</td>
<td>High Deviation.</td>
</tr>
<tr>
<td>7</td>
<td>Low Deviation.</td>
</tr>
</tbody>
</table>
pcvGetValues() will return tag values including: value, alarm status, alarm value, engineering unit or logic state descriptor, discrete-state tag indicator, and decimal precision.

alarm Pointer to floating point storage for alarm value.
euls Pointer to character storage for engineering units (analog type tags), or logic state descriptor (discrete-state tags).
dig Pointer to Boolean storage for the digital type flag. Set to TRUE if tag is a digital point.
dec Pointer to unsigned character storage for the decimal precision.
quality Pointer to TagQuality storage for quality of value. The following Table shows all quality codes with descriptions:

Table 4-12. Tag Quality Codes Set By pcvGetValues()

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Good.</td>
</tr>
<tr>
<td>1</td>
<td>Bad.</td>
</tr>
<tr>
<td>2</td>
<td>Disestablished.</td>
</tr>
<tr>
<td>3</td>
<td>Substituted.</td>
</tr>
<tr>
<td>4</td>
<td>Suspect.</td>
</tr>
<tr>
<td>5</td>
<td>Alarm Inhibited (auto).</td>
</tr>
<tr>
<td>6</td>
<td>Alarm Inhibited (group).</td>
</tr>
<tr>
<td>7</td>
<td>Alarm Inhibited (manual).</td>
</tr>
</tbody>
</table>

Return Value

BCI_OK Success.
BCI_WARN Point not established, invalid data might be returned.
BCI_FATAL An unexpected error has occurred, check errno.

Description

pcvGetValues() will return tag values including: value, alarm status, alarm value, engineering unit or logic state descriptor, discrete-state tag indicator, and decimal precision.

Errors

EDISEST Tag has not been established.
ENODBA Invalid index.
ENOSERVER Database server is not running.
ENODAEMON Mail manager is not running.
pcvGetVitals()

Retrieves the vital statistics for a given tag.

C Synopsis
#include <pcvDServ.h>
int pcvGetVitals(TagIndex index, char *tag,
    char *desc, float *value, TagStatus *status,
    float *alarm, char *euls, Bool *dig,
    u_char *dec, TagQuality *quality)

Arguments
- index: Tag index to get stats for.
- tag: Pointer to character string storage for tag name, maximum 15 characters.
- desc: Pointer to character string storage for tag descriptor, maximum 33 characters.
- value: Pointer to floating point storage for value status.
- status: Pointer to TagStatus storage for status of value. The following Table lists all status codes with descriptions.
- alarm: Pointer to floating point storage for alarm value (if in alarm).
- euls: Pointer to character string storage for tag engineering units or logic state descriptor buffer.
- dig: Pointer to Boolean storage for digital indicator; TRUE if digital.
- dec: Pointer to unsigned character storage for number of decimals defined.
- quality: Pointer to TagQuality storage for quality of value. The Table 4-14 shows all quality codes with descriptions.

Table 4-13. Tag Status Codes Set By pcvGetVitals()

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Alarm.</td>
</tr>
<tr>
<td>1</td>
<td>Alarm (Boolean).</td>
</tr>
<tr>
<td>2</td>
<td>Bad Quality.</td>
</tr>
<tr>
<td>3</td>
<td>Out of Service.</td>
</tr>
<tr>
<td>4</td>
<td>High.</td>
</tr>
<tr>
<td>5</td>
<td>Low.</td>
</tr>
<tr>
<td>6</td>
<td>High Deviation.</td>
</tr>
<tr>
<td>7</td>
<td>Low Deviation.</td>
</tr>
</tbody>
</table>

pcvGetVitals()
**pcvOutputToStn()**

Function to allow applications to set values for station type tags. This function will check the computer mode of the station before any values are sent. Note that the value will only be sent if the station is in computer mode.

**C Synopsis**

```
#include <pcvDServ.h>
int pcvOutputToStn(TagIndex index, float value, u_char parm)
```

**Arguments**

- `index`: Tag index.
- `value`: Floating point target value.
- `parm`: 1 to output set point, 2 to output ratio index, 3 to output control output.

**Return Value**

- `BCI_OK`: Success.
- `BCI_WARN`: Operation was performed but may be suspect, check errno.
- `BCI_ERROR`: An expected error has occurred, check errno.

---

**Table 4-14. Tag Quality Codes Set By pcvGetVitals()**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Good.</td>
</tr>
<tr>
<td>1</td>
<td>Bad.</td>
</tr>
<tr>
<td>2</td>
<td>Disestablished.</td>
</tr>
<tr>
<td>3</td>
<td>Substituted.</td>
</tr>
<tr>
<td>4</td>
<td>Suspect.</td>
</tr>
<tr>
<td>5</td>
<td>Alarm Inhibited (auto).</td>
</tr>
<tr>
<td>6</td>
<td>Alarm Inhibited (group).</td>
</tr>
<tr>
<td>7</td>
<td>Alarm Inhibited (manual).</td>
</tr>
</tbody>
</table>

**Errors**

- `EDISEST`: Tag has not been established.
- `ENODBA`: Invalid index.
- `ENOSERVER`: Database server is not running.
- `ENODAEMON`: Mail manager is not running.

---

**pcvGetVitals()**

`pcvGetVitals()` will return the vital tag information from the tag database.
pcvOutputToStn() sends the given value to the station tag if the station is in computer mode. The parm argument specifies whether the value is sent to the station's set point, ratio index, or control output.

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENODBA</td>
<td>Invalid index.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid argument.</td>
</tr>
<tr>
<td>EDISEST</td>
<td>Tag has not been established.</td>
</tr>
<tr>
<td>ENOSERVER</td>
<td>CIU SCAN server is not running.</td>
</tr>
<tr>
<td>ENODAEMON</td>
<td>Mail manager is not running.</td>
</tr>
</tbody>
</table>

**pcvReadTrendData()**

Retrieves selected amounts of unsigned collected trend data for a given trend index.

**C Synopsis**

```c
#include <pcvDServ.h>
int pcvReadTrendData(TrendIndex tindex, 
                     time_t time, time_t res, unsigned number, 
                     float *data)
```

**Arguments**

- `tindex`: Trend index number (1-n).
- `time`: Starting date/time (linear date format).
- `res`: Sample interval in seconds.
- `number`: Number of samples to return.
- `data`: Storage for floating point trend values.

**Return Value**

- `BCI_OK`: Success.
- `BCI_FATAL`: An unexpected error has occurred, check errno.

**Description**

pcvReadTrendData() will return selected amounts of trend data for a trend index.

**Errors**

- `ENODBA`: Invalid index.
- `EINVAL`: Data pointer is NULL.
- `ENOSERVER`: Trend server is not running.
- `ENODAEMON`: Mail manager is not running.

**pcvSendStaMode()**

Sets the station mode using the passed and mode parameter.

**C Synopsis**

```c
#include <pcvDServ.h>
int pcvSendStaMode(TagIndex index, TagMode mode)
```
pcvSendTagValue() will set the station mode using the passed tag index and mode parameter. (Note: This function should be used normally.) See the pcvOutputToStn() function and USING STATION TAGS WITH CAT in Section 3, for details.

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>local manual.</td>
</tr>
<tr>
<td>1</td>
<td>local auto.</td>
</tr>
<tr>
<td>2</td>
<td>local cascade/ratio.</td>
</tr>
<tr>
<td>3</td>
<td>computer manual.</td>
</tr>
<tr>
<td>4</td>
<td>computer auto.</td>
</tr>
<tr>
<td>5</td>
<td>computer cascade/ratio.</td>
</tr>
<tr>
<td>6</td>
<td>local level.</td>
</tr>
<tr>
<td>7</td>
<td>computer level.</td>
</tr>
<tr>
<td>8</td>
<td>computer backup state.</td>
</tr>
<tr>
<td>9</td>
<td>computer OK.</td>
</tr>
<tr>
<td>10</td>
<td>previous state.</td>
</tr>
</tbody>
</table>

pcvSendTagValue()

Sets the tag value, state, or mode.

C Synopsis

```c
#include <pcvDServ.h>
int pcvSendTagValue(TagIndex index, float value, TagState state)
```

Arguments

- **index**: Tag index.
- **value**: The target value.
- **state**: The target state or mode.

Station:
1. setpoint value
2. ratio index value
3. control output value

Return Value

- BCI_OK: Success.
- BCI_FATAL: An unexpected error has occurred, check errno.

Errors

- ENODBA: Invalid index.
- ENOSERVER: CIU SCAN server is not running.
- ENODAEMON: Mail manager is not running.
pcvSendTagValue() will set the tag value using the passed tag index, value, and state.

**RCM:**
1  sustain reset state
2  sustain set state
5  pulse reset state
6  pulse set state

**DD:**
8  automatic mode
4  manual mode
2  set state
1  reset state

**MSDD:**
8  automatic mode
4  manual mode
3  3 state
2  2 state
1  1 state
0  0 state

**Return Value**
- **BCI_OK** Success.
- **BCI_FATAL** An unexpected error has occurred, check errno.

**Description**
pcvSendTagValue() will set the tag value using the passed tag index, value, and state.

**Errors**
- **ENODBA** Invalid index.
- **ENOSERVER** CIU SCAN server is not running.
- **ENODAEMON** Mail manager is not running.

**CAUTION**
This function should not be used with station type tags. See the pcvOutputToStn() function. Also see USING STATION TAGS WITH CAT in Section 3.
SECTION 5 - PORTING CAT APPLICATIONS

PORTING FROM QNX2 TO QNX4

Porting any user applications, whether they use the Custom Application Toolkit (CAT) or not, require changing from the Computer Innovations C86® compiler to the WATCOM C compiler. Since these are both “C”-language compilers, porting simple applications that don't use CAT could be as straightforward as recompiling using the new compiler. Applications that make use of some of the more advanced language tools, such as terminal I/O, message passing, etc., will require the person doing the port to become familiar with the POSIX syntax supported by the WATCOM compiler.

The QNX Migration Guide - Moving from QNX 2.1 to QNX 4.0, has been included with the Custom Application Toolkit to aid in the port to QNX4. It contains an in depth discussion of language and operating system porting issues.

Translating CAT Header Files and Functions

The CAT package uses different header files and function names from previous releases. Below are tables which allow translation from the old names to the new.

<table>
<thead>
<tr>
<th>Old Header Name</th>
<th>New Header Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>almcmmt.h</td>
<td>pcvAlmCmmt.h</td>
</tr>
<tr>
<td>c_serv.h</td>
<td>pcvCiServ.h</td>
</tr>
<tr>
<td>d_serv.h</td>
<td>pcvDServ.h</td>
</tr>
<tr>
<td>d_size.h</td>
<td>pcvDSize.h</td>
</tr>
<tr>
<td>ibm.h</td>
<td>ibm.h</td>
</tr>
<tr>
<td>lazy_types.h</td>
<td>bciTypes.h</td>
</tr>
<tr>
<td>t_serv.h</td>
<td>pcvTrdServ.h</td>
</tr>
<tr>
<td>t_t_size.h</td>
<td>pcvTSize.h</td>
</tr>
<tr>
<td>tag_type.h</td>
<td>pcvTagType.h</td>
</tr>
<tr>
<td>trd_scan.h</td>
<td>pcvTrdScan.h</td>
</tr>
<tr>
<td>trend.h</td>
<td>pcvTrend.h</td>
</tr>
</tbody>
</table>
Porting Utility

To assist in porting applications from QNX2 to QNX4, a porting function, `portfix`, is supplied with the CAT package. To run this program, type:

```
/usr/bin/portfix <filename>
```

where filename is the name of your old “C” application program file. `portfix` will modify your file, where possible, to conform to QNX4 and the new CAT function names. Your old file is renamed, with a “-” character at the end of the filename.

For example, if you have copied your old application programs to “/usr/cat/mydir”, and your program name is “myprog.c”, type:

```
cd /usr/cat/mydir
/usr/bin/portfix myprog.c
```
This will copy “myprog.c” to “myprog.c-”, and create a new version of “myprog.c” which has been updated to conform to QNX4 and CAT.

`portfix` is provided only as a “first step” in the porting process. It will not change any software logic or make any additions or deletions. `portfix` will only translate old function and file names into new ones.

**CAUTION**
Always check the source code for your program after running `portfix`, to make sure no incorrect translations have taken place. You may still have to make additional changes to your software before it will compile and run properly.
## Index

| A | Adding Software ......................................................... 3-4  
   | auto_man ........................................................................ 4-8  |
|---|---|
| B | by_pass ........................................................................... 4-8  |
| C | cas_rat ............................................................................ 4-8  
   | CAT Library .................................................................... 4-1  
   | Catman .......................................................................... 3-2  
   | cmpt_stat ....................................................................... 4-8  
   | Compilers ........................................................................ 3-1  
   | Configuration Information .............................................  
     | pcvGetAlmDesc ................................................................ 4-1  
     | pcvGetHaddr ..................................................................... 4-1  
     | pcvGetStatics .................................................................. 4-1  
     | pcvGetTagAddr ................................................................... 4-1  
     | pcvGetTagAlmval ............................................................ 4-1  
     | pcvGetTagDefault .......................................................... 4-1  
     | pcvGetTagDesc ................................................................... 4-1  
     | pcvGetTagDisp .................................................................. 4-1  
     | pcvGetTagEu ....................................................................... 4-1  
     | pcvGetTagIndex .................................................................. 4-1  
     | pcvGetTagLs ........................................................................ 4-1  
     | pcvGetTagName ................................................................... 4-1  
     | pcvGetTagType .................................................................... 4-1  
     | pcvGetTrdIndex .................................................................... 4-1  
   | Custom Application Toolkit (CAT) .....................................  
     | Introduction ...................................................................... 1-1  
     | Using ............................................................................... 3-1  
   | Custom Software Programs ............................................... 3-1  |
| D | Data Collection .................................................................. 4-1  
   | pcvGetAlmcmmt ............................................................... 4-2  
   | pcvGetCiudata ................................................................. 4-2  
   | pcvGetDdValues .............................................................. 4-1  
   | pcvGetMsdValues ............................................................. 4-1  
   | pcvGetRcmValues ............................................................ 4-1  
   | pcvGetStatModes .............................................................. 4-1  
   | pcvGetTagValue .............................................................. 4-1  
   | pcvGetValues ..................................................................... 4-2  
   | pcvGetVitals ....................................................................... 4-1  
   | pcvReadTrendData ........................................................... 4-2  
   | Data Sending ....................................................................... 4-2  
   | pcvOutputToStn .................................................................... 4-2  
   | pcvSendStaMode ............................................................... 4-2  |
| E | pcvSendTagValue ................................................................ 4-2  |
| F | Examples ............................................................................ 3-2  
   | Faceplates ......................................................................... 3-5  
   | File Utilities ...................................................................... 3-1  
   | Function Names ..................................................................  
     | Translating ....................................................................... 5-2  
   | Functions ..........................................................................  
     | pcvOutputToStn() ............................................................. 3-5  |
| H | Header Files ......................................................................  
   | Translating ....................................................................... 5-1  |
| I | Installation ....................................................................... 2-1  |
| L | Library Routines ................................................................ 3-1, 3-2  
   | Loading CAT ...................................................................... 2-2  
   | Is ................................................................................... 3-1  |
| M | man_lock .......................................................................... 4-8  |
| O | On-Line Documentation ..................................................... 3-2  
   | op_trk .............................................................................. 4-8  |
| P | pax .................................................................................. 3-1  
   | pcvGetAlmcmmt() ............................................................ 4-3  
   | pcvGetAlmDesc() ............................................................ 4-2  
   | pcvGetCiudata() ............................................................. 4-3  
   | pcvGetDdValues() ............................................................ 4-4  
   | pcvGetHaddr() ................................................................... 4-4  
   | pcvGetMsdValues() ........................................................... 4-5  
   | pcvGetPriDisp() .............................................................. 4-6  
   | pcvGetRcmValues() .......................................................... 4-7  
   | pcvGetStatics() ............................................................... 4-9  
   | pcvGetStatModes() ........................................................... 4-8  
   | pcvGetTagAddr() ............................................................. 4-9  
<p>| pcvGetTagAlmval() .......................................................... 4-10  |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcvGetTagDefault()</td>
<td>4-11</td>
</tr>
<tr>
<td>pcvGetTagDesc()</td>
<td>4-12</td>
</tr>
<tr>
<td>pcvGetTagEu()</td>
<td>4-13</td>
</tr>
<tr>
<td>pcvGetTagIndex()</td>
<td>4-14</td>
</tr>
<tr>
<td>pcvGetTagName()</td>
<td>4-14</td>
</tr>
<tr>
<td>pcvGetTagType()</td>
<td>4-15</td>
</tr>
<tr>
<td>pcvGetTagValue()</td>
<td>4-16</td>
</tr>
<tr>
<td>pcvGetTrdIndex()</td>
<td>4-17</td>
</tr>
<tr>
<td>pcvGetValues()</td>
<td>4-18</td>
</tr>
<tr>
<td>pcvGetVitals()</td>
<td>4-19</td>
</tr>
<tr>
<td>pcvOutputToStn()</td>
<td>4-21</td>
</tr>
<tr>
<td>pcvReadTrendData()</td>
<td>4-22</td>
</tr>
<tr>
<td>pcvSendStaMode()</td>
<td>4-22</td>
</tr>
<tr>
<td>pcvSendTagValue()</td>
<td>4-23</td>
</tr>
<tr>
<td>Porting</td>
<td>5-1</td>
</tr>
<tr>
<td>CAT Header Files and Functions</td>
<td>5-1</td>
</tr>
<tr>
<td>From QNX2 to QNX4</td>
<td>5-1</td>
</tr>
<tr>
<td>Porting Utility</td>
<td>5-2</td>
</tr>
<tr>
<td>portfix</td>
<td>5-2</td>
</tr>
<tr>
<td>Program</td>
<td>3-4</td>
</tr>
<tr>
<td>Creating</td>
<td>3-4</td>
</tr>
<tr>
<td>Program Source Code</td>
<td>3-1</td>
</tr>
<tr>
<td>Programming</td>
<td>3-1</td>
</tr>
<tr>
<td>Set Point</td>
<td>3-5</td>
</tr>
<tr>
<td>Setup Utility</td>
<td>2-1</td>
</tr>
<tr>
<td>size</td>
<td>3-1</td>
</tr>
<tr>
<td>stat_fault</td>
<td>4-8</td>
</tr>
<tr>
<td>Station Tags</td>
<td>3-5</td>
</tr>
<tr>
<td>stn_lev</td>
<td>4-8</td>
</tr>
<tr>
<td>Storing Files</td>
<td>3-3</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Text Editors</td>
<td>3-1</td>
</tr>
<tr>
<td>Vedit</td>
<td>3-1</td>
</tr>
<tr>
<td>Translating Function Names</td>
<td>5-2</td>
</tr>
<tr>
<td>Translating Header Files</td>
<td>5-1</td>
</tr>
<tr>
<td>Tutorial</td>
<td>3-2</td>
</tr>
<tr>
<td>Main Display</td>
<td>3-3</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>Utilities</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>3-1</td>
</tr>
<tr>
<td>ls</td>
<td>3-1</td>
</tr>
<tr>
<td>pax</td>
<td>3-1</td>
</tr>
<tr>
<td>Porting</td>
<td>5-2</td>
</tr>
<tr>
<td>size</td>
<td>3-1</td>
</tr>
<tr>
<td>vol</td>
<td>3-1</td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Vedit</td>
<td>3-1</td>
</tr>
<tr>
<td>vol</td>
<td>3-1</td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>WATCOM C &quot;Compiler&quot;</td>
<td>1-1, 2-1, 3-1</td>
</tr>
</tbody>
</table>
Our worldwide staff of professionals is ready to meet your needs for process automation. For the location nearest you, please contact the appropriate regional office.