CTC 32-bit Data Communications
Functions Reference Guide
# Contents

## Notes to Readers

### 1 Using CTC's Data Communications Functions

- CTC Data Communications Functions                                                                                     1-2
  - Introduction                                                                                                         1-2
  - Installing the DLL Functions                                                                                         1-2
- Function Overview                                                                                                     1-3
- Using the Functions in a Visual Basic Program                                                                         1-5
- Using the Functions in a C++ Program                                                                               1-7

## 2 Function Descriptions

### Network Access Functions

- CtOpenConnection                                                                                                      2-2
- CtInitConnection                                                                                                      2-3
- CtCloseConnection                                                                                                     2-4
- CtSetConnectionTimeout                                                                                               2-5

### Register Functions

- CtGetRegister                                                                                                         2-6
- CtGetRegister16                                                                                                        2-7
- CtGetRegister50                                                                                                       2-8
- CtPutRegister                                                                                                         2-9
- CtGetNRegisters                                                                                                       2-9a

### Flag Functions

- CtGetFlag                                                                                                              2-10
- CtPutFlag                                                                                                              2-11

### Digital I/O Functions

- CtGetDigitalInput8                                                                                                     2-12
- CtGetDigitalOutput8                                                                                                   2-13
- CtGetDigitalInput128                                                                                                  2-14
- CtGetDigitalOutput128                                                                                                 2-15

### Analog I/O Functions

- CtGetAnalogInput                                                                                                      2-16
- CtGetAnalogOutput                                                                                                     2-17
- CtPutAnalogOutput                                                                                                     2-18
- CtGetAnalogInput32                                                                                                    2-19
- CtGetAnalogOutput32                                                                                                   2-20
Servo Functions ................................................................. 2-21
  CtGetServoPosition ........................................................... 2-21
  CtGetServoError .............................................................. 2-22
  CtGetServoInputs ............................................................ 2-23
Data Table Functions .......................................................... 2-24
  CtGetDataTableDim .......................................................... 2-24
  CtGetDataTableLoc ........................................................... 2-25
  CtPutDataTableLoc ........................................................... 2-26
  CtGetDataTableRow ......................................................... 2-27
  CtPutDataTableRow ........................................................... 2-28
Controller Status Functions .................................................. 2-29
  CtGetStatus ..................................................................... 2-29
  CtPutStatus ..................................................................... 2-30
  CtGetTaskBank .................................................................. 2-31
Controller Configuration Functions ...................................... 2-33
  CtGetModel ..................................................................... 2-33
  CtGetConfig .................................................................... 2-34
  CtPutConfig .................................................................... 2-35
  CtGetIoCount ................................................................... 2-36
  CtGetMiscIoCount ............................................................ 2-37
Programming Functions ...................................................... 2-38
  CtDownload ..................................................................... 2-38
  CtUpload .......................................................................... 2-39
Miscellaneous Functions ...................................................... 2-40
  CtGetConnectionInfo ........................................................ 2-40
  CtGetMessageInfo ............................................................ 2-41
  CtGetErrorInfo ................................................................ 2-42
Notes to Readers

The CTC 32-bit Data Communications Functions Reference Guide describes the CTC’s DLL functions used to communicate with a controller. These functions are available in a Dynamic-link Library (DLL) that can be used with 32-bit programs such as Visual Basic or C++ program.

Related Documents
The following documents contain additional information

- For information on the registers in your controller, refer to Register Reference Guide available on the Web site.
- For information on your controller and its modules, refer to the appropriate Installation and Applications Guide.
- For information on Microsoft Windows or your PC, refer to the manuals provided by the vendor.

Book Conventions
The following conventions are used in this book.

<table>
<thead>
<tr>
<th>ALL CAPS BOLDFACE</th>
<th>Identifies field lengths in the DLL functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boldface</td>
<td>Identifies DLL function names.</td>
</tr>
<tr>
<td><em>Italics</em> and <em>Boldface Italic</em></td>
<td>Indicates DLL parameters</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Indicates a word requiring an appropriate substitution. For example, replace <em>filename</em> with an actual file name.</td>
</tr>
<tr>
<td>Courier font</td>
<td>Identifies Visual Basic and C++ programming examples.</td>
</tr>
</tbody>
</table>

How to Contact Control Technology Corporation
Control Technology Corporation is located in Massachusetts, and we are open from 8:30 a.m. to 5:00 p.m. eastern time. Contact us at 1-508-435-9595 and 1-800-282-5008 or FAX 1-508-435-2373.


Your Comments
We welcome your suggestions and comments about this or any other Control Tech document. You can email comments about this manual or any other CTC document to techpubs@control.com.
Notes to Readers

Document Notice

The information in this document is subject to change without notice. The software described in this document is provided under license agreement and may be used or copied only in accordance with the terms of the license agreement.

The information, drawings, and illustrations contained herein are the property of Control Technology Corporation. No part of this manual may be reproduced or distributed by any means, electronic or mechanical, for any purpose other than the purchaser’s personal use, without the express written consent of Control Technology Corporation.

The following are trademarks of Control Technology Corporation:

• Quickstep
• CTC Monitor

MS-DOS and Windows are trademarks of Microsoft Corporation
Using CTC’s Data Communications Functions

Contents

CTC Data Communications Functions 1-2
Function Overview 1-3
Using the Functions in a Visual Basic Program 1-5
Using the Functions in a C++ Program 1-7
CTC Data Communications Functions

Introduction

Control Technology Corp. (CTC) provides two programs, CTC Monitor and CTC Utilities for Windows, for communicating with your controller. These programs are Windows applications and are suitable for most controller-computer communications. However, there are some applications where you may want to create your own program to communicate with the controller. For such cases, we have made available a group of communications functions in a Dynamic-link Library (DLL) that can be used with 32-bit programs such as Visual Basic or C++ program.

The CTC data communications functions are available from the customer section of the Control Technology web site. The self-extracting installation file, Ctccom32v2.exe, contains the following files:

- Ctccom32v2.dll: The DLL communications functions file
- Ctccom32v2.h: C header file containing the DLL function declarations
- Ctccom32v2.bas: Visual Basic 4.0/5.0/6.0 code module
- Ctccom32v2.lib: C library file
- Ctccom32v2.pdf: Ctccom32v2 function reference documentation file in Acrobat pdf format
- Readme.txt: Readme file

Installing the DLL Functions

To install the DLL functions,

1. Download the Ctccom32v2.exe to your computer from the web site.
2. Select Run from the Start menu on the Taskbar.
3. Enter the path location of the Ctccom32v2.exe file.
4. When the installation program prompts you, enter a location for the files.
5. Once you have run the installation program, make sure you place the Ctccom32v2.dll in the appropriate windows directory:
   - Windows 95 users copy the Ctccom32v2.dll file to your Windows\System directory
   - Windows NT the users copy Ctccom32v2.dll to your Windows\System32 directory
   - Windows 98 the users copy Ctccom32v2.dll to your Windows\System32 directory
## Function Overview

The following table lists each function and briefly describes it. For a complete description of each function, see chapter 2, *Function Descriptions.*

### CTC Communications Functions

#### Network Access Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtOpenConnection</td>
<td>Opens a serial, CTC ethernet, or CTC web connection to a controller</td>
</tr>
<tr>
<td>CtInitConnection</td>
<td>Initializes an open serial connection to a controller</td>
</tr>
<tr>
<td>CtCloseConnection</td>
<td>Closes a connection to a controller</td>
</tr>
<tr>
<td>CtSetConnectionTimeout</td>
<td>Sets a time-out value for computer-controller connection response time</td>
</tr>
</tbody>
</table>

#### Register Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetRegister</td>
<td>Reads the current value of a register</td>
</tr>
<tr>
<td>CtGetRegister16</td>
<td>Reads the current values in a block of 16 contiguous registers.</td>
</tr>
<tr>
<td>CtGetRegister50</td>
<td>Reads the current values of a block of 50 contiguous registers</td>
</tr>
<tr>
<td>CtPutRegister</td>
<td>Writes a new value to a register</td>
</tr>
</tbody>
</table>

#### Flag Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetFlag</td>
<td>Reads the value of the flags (1 - 32) in the controller</td>
</tr>
<tr>
<td>CtPutFlag</td>
<td>Writes a new value to a flag</td>
</tr>
</tbody>
</table>

#### Digital I/O Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetDigitalInput8</td>
<td>Reads the values of a bank of eight digital inputs</td>
</tr>
<tr>
<td>CtGetDigitalOutput8</td>
<td>Reads the values of a bank of eight digital outputs</td>
</tr>
<tr>
<td>CtGetDigitalInput128</td>
<td>Reads the values of a bank of 128 digital inputs</td>
</tr>
<tr>
<td>CtGetDigitalOutput128</td>
<td>Reads the values of a bank of 128 digital outputs</td>
</tr>
</tbody>
</table>

#### Analog I/O Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetAnalogInput</td>
<td>Reads the value of an analog input</td>
</tr>
<tr>
<td>CtGetAnalogOutput</td>
<td>Reads the value of an analog output</td>
</tr>
<tr>
<td>CtPutAnalogOutput</td>
<td>Writes a value to an analog input</td>
</tr>
<tr>
<td>CtGetAnalogInput32</td>
<td>Reads the values of a bank of 32 analog inputs</td>
</tr>
<tr>
<td>CtGetAnalogOutput32</td>
<td>Reads the values of a bank of 32 analog outputs</td>
</tr>
</tbody>
</table>

#### Servo Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetServoPosition</td>
<td>Reads the current position of a servo</td>
</tr>
<tr>
<td>CtGetServoError</td>
<td>Reads the current error of a servo</td>
</tr>
<tr>
<td>CtGetServoInputs</td>
<td>Reads the value of servo inputs</td>
</tr>
</tbody>
</table>

#### Data Table Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetDataTableDim</td>
<td>Reads the dimensions of the data table</td>
</tr>
<tr>
<td>CtGetDataTableLoc</td>
<td>Reads a value at a location in the data table</td>
</tr>
<tr>
<td>CtPutDataTableLoc</td>
<td>Writes a value to a location in the data table</td>
</tr>
<tr>
<td>CtGetDataTableRow</td>
<td>Reads the values in a data table row</td>
</tr>
<tr>
<td>CtPutDataTableRow</td>
<td>Writes values to a row in a data table</td>
</tr>
</tbody>
</table>

#### Controller Status Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtGetStatus</td>
<td>Reads the controller status</td>
</tr>
<tr>
<td>CtPutStatus</td>
<td>Sets the controller status</td>
</tr>
<tr>
<td>CtGetTaskBank</td>
<td>Reads a program status task bank from the controller</td>
</tr>
</tbody>
</table>
Function Overview

CTC Communications Functions (Continued)

**Controller Configuration Functions**
- CtGetModel: Reads the controller model information
- CtGetConfig: Reads the controller configuration
- CtPutConfig: Sets the controller configuration
- CtGetIoCount: Reads the basic hardware configuration of a controller
- CtGetMiscIoCount: Reads the extended hardware configuration of a controller

**Data Table Functions**
- CtGetDataTableDim: Reads the dimensions of the data table
- CtGetDataTableLoc: Reads a value at a location in the data table
- CtPutDataTableLoc: Writes a value to a location in the data table
- CtGetDataTableRow: Reads the values in a data table row
- CtPutDataTableRow: Writes values to a row in a data table

**Controller Status Functions**
- CtGetStatus: Reads the controller status
- CtPutStatus: Sets the controller status
- CtGetTaskBank: Reads a program status task bank from the controller

**Controller Configuration Functions**
- CtGetModel: Reads the controller model information
- CtGetConfig: Reads the controller configuration
- CtPutConfig: Sets the controller configuration
- CtGetIoCount: Reads the basic hardware configuration of a controller
- CtGetMiscIoCount: Reads the extended hardware configuration of a controller

**Programming Functions**
- CtDownload: Downloads an object file into the controller
- CtUpload: Uploads the controller’s program to an object file

**Miscellaneous Functions**
- CtGetConnectionInfo: Returns information on the connection
- CtGetMessageInfo: Returns information on the last message sent and last response received for the connection
- CtGetErrorInfo: Returns information on the last error condition for the connection

**Programming Functions**
- CtDownload: Downloads an object file into the controller
- CtUpload: Uploads the controller’s program to an object file

**Miscellaneous Functions**
- CtGetConnectionInfo: Returns information on the connection
- CtGetMessageInfo: Returns information on the last message sent and last response received for the connection
- CtGetErrorInfo: Returns information on the last error condition for the connection
Using the Functions in a Visual Basic Program

When using the CTC communications functions in a Visual Basic program, you must insert the CtcCom32v2.bas file into your Visual Basic project. Once the CtcCom32v2.bas file has been inserted into your project, a number of useful constants are defined as well as the function definitions for the DLL. The following list is a subset of the function definitions in the CtcCom32v2.bas file. For a complete list of functions definitions, see Chapter 2, Function Descriptions.

' network access functions
Declare Function CtOpenConnection Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal CommPort As Long, ByVal Address As Long) As Long
Declare Function CtInitConnection Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal BaudRate As Long, ByVal Parity As Byte, ByVal DataBits As Byte, ByVal StopBits As Byte) As Long
Declare Function CtCloseConnection Lib "Ctccom32v2.dll" (ByVal ConnectID As Long) As Long
Declare Function CtSetConnectionTimeout Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal MilliSeconds As Long) As Long

' register functions
Declare Function CtGetRegister Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal RegAddress As Long, ByRef RegValue As Long) As Long
Declare Function CtGetRegister16 Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal RegAddress As Long, ByRef Reg16Val As Long) As Long
Declare Function CtGetRegister50 Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal RegBank As Long, ByRef Reg50Val As Long) As Long
Declare Function CtPutRegister Lib "Ctccom32v2.dll" (ByVal ConnectID As Long, ByVal RegAddress As Long, ByVal RegValue As Long) As Long

To use the functions, the user must first establish a connection to a CTC controller. The following code opens a serial connection on COM1:

Dim lType as long
Dim lLocal as long
Dim lRemote as long
Dim lConnectID as long
lType = SERIAL
lLocal = 1
lRemote = 0
lConnectID = CtOpenConnection(lType, lLocal, lRemote)

If the connection is opened, a non-zero number is returned. This number represents is connection identifier and will be used in all subsequent calls to the DLL functions. The following Visual Basic code sets the time-out in milliseconds for the connection:

Dim lTimeout as long
'set timeout
lTimeout = 500
If CtSetConnectionTimeout(lConnectID, lTimeout) = FAILURE Then
    MsgBox “Failed to set timeout.”, vbExclamation
End If
Once a connection has been opened you can call functions to read and write register data. In the examples shown here, the program includes a text box control named `Text1.Text` to display one of the register values obtained by the function. The `Format$` function converts a long integer to a string for the display.

```vba
Dim lValue as long
Dim lAddress as long
lAddress = 13002
If CtGetRegister(lConnectID, lAddress, lValue) <> SUCCESS Then
    MsgBox “Failed to get register.”, vbExclamation
Else
    ‘update display
    Text1.Text = Format$(lValue)
End If
```

```vba
Dim lValue(0 to 15) as long
lAddress = 13002
If CtGetRegister16(lConnectID, lAddress, lValue(0)) <> SUCCESS Then
    MsgBox “Failed to get 16 registers.”, vbExclamation
Else
    ‘update display
    Text1.Text = Format$(lValue(0))
End If
```

```vba
Dim lBank as long
Dim lValue(1 to 50) as long
lBank = 1
If CtGetRegister50(lConnectID, lBank, lValue(1)) <> SUCCESS Then
    MsgBox “Failed to get 50 registers.”, vbExclamation
Else
    ‘update display
    Text1.Text = Format$(lValue(1))
End If
```

```vba
lAddress = 1
lValue = 12345
‘put value
If CtPutRegister(lConnectID, lAddress, lValue) <> SUCCESS Then
    MsgBox “Failed to put register.”, vbExclamation
End If
```

Once finished with communications, you must close any open connection(s). The following code shows how to close a connection:

```vba
If CtCloseConnection(lConnectID) = 0 Then
    MsgBox “Failed to close connection”, vbExclamation
Else
    lConnectID = 0
End If
```

**NOTE:** CTC recommends that you check the return value from the function to determine success or failure. These examples use the Visual Basic `MsgBox` function to display a message in a dialog box in the event of a failure.
Using the Functions in a C++ Program

When using the CTC communications functions in a C+ program, you need to perform a series of declarations in your C program.

1. Declare data types for the functions in the DLL
2. Declare variables of the types defined previously
3. Call LoadLibrary to confirm the file exists and to load it
4. Call GetProcAddress to map the location of the functions in the DLL
5. Call the functions as needed
6. Close the connection(s) and free the library

The following program code shows how to declare data types for the functions in the DLL

```c
// ctccom32v2 functions
typedef UINT32 (*CTOPEN)( UINT32, UINT32, UINT32);
typedef UINT32 (*CTINIT)( UINT32, UINT32, UCHAR, UCHAR, UCHAR);
typedef UINT32 (*CTCLOSE)( UINT32);
typedef UINT32 (*CTSTIME)( UINT32, UINT32);
typedef UINT32 (*CTGETREG)( UINT32, UINT32, INT32 FAR*);
typedef UINT32 (*CTGETREG16)( UINT32, UINT32, REGS_16 FAR*);
typedef UINT32 (*CTGETREG50)( UINT32, UINT32, REGS_50 FAR*);
typedef UINT32 (*CTPUTREG)( UINT32, UINT32, INT32);
```

The following examples shows how to declare variables for the types defined above.

```c
INT32 CtLibInst = 0;
CTOPEN CtOpenConnection;
CTINIT CtInitConnection;
CTCLOSE CtCloseConnection;
CTSTIME CtSetConnectionTimeout;
CTGETREG CtGetRegister;
CTGETREG16 CtGetRegister16;
CTGETREG50 CtGetRegister50;
CTPUTREG CtPutRegister;
```

The following example calls LoadLibrary to confirm the file exists and to load it. It then calls GetProcAddress to map the location of the functions in the DLL.

```c
/* Get a handle to the DLL functions */
if ( CtLibInst == 0 ) {
    if ( ( CtLibInst = LoadLibrary("ctccom32v2.dll") ) != (INT32) NULL ) {
        CtOpenConnection = (CTOPEN)
        GetProcAddress(CtLibInst, "CtOpenConnection");
        CtInitConnection = (CTINIT)
        GetProcAddress(CtLibInst, "CtInitConnection");
        CtCloseConnection = (CTCLOSE)
        GetProcAddress(CtLibInst, "CtCloseConnection");
        CtSetConnectionTimeout = (CTSTIME)
        GetProcAddress(CtLibInst, "CtSetConnectionTimeout");
        CtGetRegister = (CTGETREG)
        GetProcAddress(CtLibInst, "CtGetRegister");
        CtGetRegister16 = (CTGETREG16)
    }
```

Function Descriptions

Control Technology Corporation proprietary, Reproduction or distribution forbidden.
The following are examples of function calls in a program.

```c
if (( conn = CtOpenConnection(CTNET, host, target)) != FAILURE) {
    // set connection timeout
    (void) CtSetConnectionTimeout(conn, timeout);
}

if ( CtGetRegister(conn, address, &result) == SUCCESS) {
    // Set the appropriate state and response buffer
    data = result;
}
```

When you are finished using the DLL functions, close the connection(s) and free the library.

```c
if ( CtCloseConnection( conn) != SUCCESS) {
    (void) CtGetErrorInfo( conn, &CtErr);
    sprintf (szAlm, "Couldn’t close connection, err %d.", CtErr.iErrCode);
}
```

// Free the DLL module
```c
if ( CtLibInst != 0 ) {
    if ( FreeLibrary(CtLibInst) == 0) {
        LastError = GetLastError();
        sprintf (szAlm, " Couldn’t unload CTCCOM32V2.DLL %d.", LastError);
    } else
        CtLibInst = 0;
```
# Function Descriptions

## Contents

<table>
<thead>
<tr>
<th>Function Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Access Functions</td>
<td>2-2</td>
</tr>
<tr>
<td>Register Functions</td>
<td>2-6</td>
</tr>
<tr>
<td>Flag Functions</td>
<td>2-10</td>
</tr>
<tr>
<td>Digital I/O Functions</td>
<td>2-12</td>
</tr>
<tr>
<td>Analog I/O Functions</td>
<td>2-16</td>
</tr>
<tr>
<td>Servo Functions</td>
<td>2-21</td>
</tr>
<tr>
<td>Data Table Functions</td>
<td>2-24</td>
</tr>
<tr>
<td>Controller Status Functions</td>
<td>2-29</td>
</tr>
<tr>
<td>Programming Functions</td>
<td>2-38</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>2-40</td>
</tr>
</tbody>
</table>
Network Access Functions

CtOpenConnection

CtOpenConnection opens a serial, CTC ethernet, or CTC web connection to a controller. Your program must call CtOpenConnection once to establish the connection. The connection, once established, remains in use until your program calls CtCloseConnection.

When making CTC ethernet connections, use a unique local node address for each connection that communicates with the same remote controller. When making CTC web connections, use a unique local port number. Port numbers from one to 1024 are reserved. Port numbers beginning at 4000 are suggested.

For CTC ethernet connections, the remote node address is the node number in register 20,000 of the controller.

The format for calling CtOpenConnection is as follows:

CtOpenNetwork(
    ULONG lConnectionType,
    ULONG lLocalAddress,
    ULONG lRemoteAddress
) ;

Input Parameters

lConnectionType

lConnectionType is a long integer that specifies the type of network connection to open. The allowable values are zero for no connection, one for serial connection, two for CTC ethernet connection, or 3 for a CTC web connection.

lLocalAddress

lLocalAddress is a long integer that specifies the address to use for the connection. The allowable values are a valid serial port number for a serial connection, a valid CTC network node number (1 to 32767) for an ethernet connection, or a valid port number for a CTC web connection.

lRemoteAddress

lRemoteAddress is a long integer that specifies the address of the controller you wish to connect to. The allowable values are a valid CTC network node number (1 to 32767) for an ethernet connection or an IP address for a CTC web connection. Specify zero (not used) when making a serial connection.

Success/Error Return Values

If the function succeeds, the return value is the connection number used with subsequent function calls.

Other Related Functions

CtInitConnection, CtCloseConnection, CtSetConnectionTimeout
CtInitConnection

*CtInitConnection* initializes an open serial connection to a controller. A program only needs to call *CtInitConnection* when you want to change the default serial communications values established at the time the connection is opened. The default serial port parameters are 9600 baud, no parity, eight data bits, and one stop bit.

The format for calling *CtInitConnection* is as follows:

```c
ULONG CtInitConnection (  
    ULONG lConnectID  
    DWORD dwBaudRate  
    BYTE bParity      
    BYTE bDataBits    
    BYTE bStopBits    
);  
```

**Input Parameters**

*CtConnectID*

*CtConnectID* is a long integer value that specifies the connection to initialize.

*dwBaudRate*

*dwBaudRate* is a double word value that specifies the baud rate of the serial connection. The default value for CTC controllers is 9600; other allowable values are controller dependent.

*bParity*

*bParity* is a byte value that specifies the parity used. The only allowable value is zero (no parity).

*bDataBits*

*bDataBits* is a byte value that specifies the number of data bits used. The only allowable value is eight (eight data bits used).

*bStopBits*

*bStopBits* is a byte value that specifies the number of stop bits used. The only allowable value is zero (one stop bit used).

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtOpenConnection*, *CtCloseConnection*, *CtSetConnectionTimeout*
Network Access Functions

CtCloseConnection

*CtCloseConnection* closes a connection to a controller. This command needs to be issued once for each connection.

The format for calling *CtCloseConnection* is as follows:

```c
ULONG CtCloseConnection ( 
    ULONG    lConnectID );
```

**Input Parameters**

*lConnectID*

*lConnectID* is a long integer value that specifies the connection to close.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtOpenConnection, CtInitConnection, CtSetConnectionTimeout*
**CtSetConnectionTimeout**

The **CtSetConnectionTimeout** function is used by all functions in communication with the controller. If a response is not obtained within the time specified by the time-out value, the function reports an error.

The format for calling **CtSetConnectionTimeout** is as follows:

```c
ULONG CtSetConnectionTimeout(
    ULONG lConnectD,
    ULONG lMilliseconds);
```

**Input Parameters**

- **lConnectD**
  - `lConnectD` is a long integer that specifies the connection to change.

- **lMilliseconds**
  - `lMilliseconds` is a long integer that specifies the number of milliseconds to wait for a response before timing out. The default value is 250 milliseconds.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the **CtGetErrorInfo** function.

**Other Related Functions**

- CtOpenConnection, CtInitConnection, CtCloseConnection
Register Functions

CtGetRegister

CtGetRegister reads the current value of a register.

The format for calling CtGetRegister is as follows:

```c
ULONG CtGetRegister(
    ULONG lConnectID,
    ULONG lRegister,
    LPLONG pValue);
```

Input Parameters

- **lConnectID**
  - lConnectID is a long integer that specifies the connection to use.

- **lRegister**
  - lRegister is a long integer that specifies the register address.

Output Parameters

- **pValue**
  - pValue points to a long integer that receives the value of the register.

The register value returned to the pValue pointer is a number from $-2,147,483,647$ to $2,147,483,647$.

Success/Error Return Values

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

- CtGetRegister16, CtGetRegister50, CtPutRegister
CtGetRegister16

CtGetRegister16 reads the current values in a bank of 16 contiguous registers. You must specify the number of the first register in the block.

The format for calling CtGetRegister16 is as follows:

```c
ULONG CtGetRegister16 (  
    ULONG lConnectID
    ULONG lRegister
    REGS_16 * pValues
    );
```

**Typedef** ULONG REGS_16[16];

**Input Parameters**

- `lConnectID`  
  `lConnectID` is a long integer that specifies the connection to use.

- `lRegister`  
  `lRegister` is a long integer that specifies the starting register address.

**Output Parameters**

- `pValues`  
  `pValues` points to a 16-element array of long integers that receives the register values.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtGetRegister, CtGetRegister50, CtPutRegister
Register Functions

CtGetRegister50

CtGetRegister50 reads the current values of a bank of 50 contiguous registers. Using this function you can read registers one to 1000, by specifying the bank number (1 to 20) of the registers. Bank one contains registers one - 50, bank two contains registers 51 - 100, up to bank 20, which is for registers 951 to 1000.

The format for calling CtGetRegister50 is as follows:

```
ULONG CtGetRegister50 ( 
   ULONG  lConnectID 
   ULONG  lBank 
   REGS_50 *  pValues 
);
```

Typedef ULONG REGS_50[50];

Input Parameters

`lConnectID`

`lConnectID` is a long integer that specifies the connection to use.

`lBank`

`lBank` is a long integer that specifies the bank number.

Output Parameters

`pValues`

`pValues` points to a 50-element array of long integers that receives the register values.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetRegister, CtGetRegister16, CtPutRegister
**CtPutRegister**

CtPutRegister writes a new value to a register. Values can range from –2,147,483,647 to 2,147,483,647.

The format for calling CtPutRegister is as follows:

```c
ULONG CtPutRegister ( 
    ULONG       lConnectID
    ULONG       lRegister
    LONG        lValue   );
```

**Input Parameters**

- **lConnectID**
  lConnectID is a long integer that specifies the connection to use.

- **lRegister**
  lRegister is a long integer that specifies the register address.

**Output Parameters**

- **lValue**
  lValue is a long integer that specifies the new register value to write.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtGetRegister, CtGetRegister16, CtGetRegister50
CtGetNRegisters

CtGetNRegisters reads non-contiguous registers from the controller. The user can specify a non-contiguous set of registers in a single read transaction. The controller will return the content of these registers in the order they were requested.

Input Parameters:
- lConnectID
  lConnectID is a long integer that specified the connection to use.
- lRegisterCount
  lRegisterCount is a long integer to define the number of registers to read (in a long array), 50 max.
- pRegisterList
  pRegisterList - Pointer to an array of unsigned longs which contains a list of each register to read, in the order listed

Output Parameters:
- pValues - Pointer to an array of unsigned longs where to store the 32 bit returned values of each requested register.

Success/Error Return Values
If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions:
CTGetRegister, CTGetRegister16, CTGetRegister50
**VB6 Example:**

Declare Function CtGetNRegisters Lib "Ctccom32v2.dll" _
    (ByVal ConnectID As Long, ByVal RegCount As Long, ByRef RegList As Long, ByRef Values As Long) As Long

Dim Values(1 To 50) As Long
Dim RegList(1 To 50) As Long
...
    RegList(1) = 11
    RegList(2) = 2
    RegList(3) = 34
    RegList(4) = 13002
    RegList(5) = 100

' Read the above 5 registers
If CtGetNRegisters(ConnectID, 5, RegList(1), Values(1)) = SUCCESS Then
    ' Success
    ' Values array contains register contents
Else
    ' Failed
End If
VB.NET Example:

```vbnet
Dim results As Integer
Dim Rregs As Ctccom32v2.CT_RANDOM_REGS
Dim RregVals As Ctccom32v2.CT_RANDOM_REGVALS
Dim i As Integer

ReDim RregVals.lRegVals(50)
For i = 1 To 50
  RregVals.lRegVals(i) = 0
Next

'Load the registers of interest
ReDim Rregs.lRegList(50)
Rregs.lRegList(0) = 1
Rregs.lRegList(1) = 13002
Rregs.lRegList(2) = 501
Rregs.lRegList(3) = 32001
Rregs.lRegList(4) = 2
Rregs.lRegList(5) = 87
Rregs.lRegList(6) = 501
Rregs.lRegList(7) = 32401
Rregs.lRegList(8) = 567
Rregs.lRegList(9) = 130
Rregs.lRegList(10) = 5
Rregs.lRegList(11) = 3
Rregs.lRegList(12) = 456
Rregs.lRegList(13) = 13
Rregs.lRegList(14) = 531
Rregs.lRegList(15) = 320
Rregs.lRegList(16) = 345
Rregs.lRegList(17) = 130
Rregs.lRegList(18) = 545
Rregs.lRegList(19) = 32001
Rregs.lRegList(20) = 563
Rregs.lRegList(21) = 102
Rregs.lRegList(22) = 531
Rregs.lRegList(23) = 2001

'Call the function to get 50 registers
results = Ctccom32v2.CtGetNRegisters(CTconnection, 50, Rregs, RregVals)

If results = SUCCESS Then
  writeStatus.Text = "SUCCESS"
  writeStatus.Text = results
Else
  writeStatus.Text = "ERROR"
End If
```
Flag Functions

CtGetFlag

CtGetFlag reads the value of the flags (1 - 32) in the controller.

The format for calling CtGetFlag is as follows:

```c
ULONG CtGetFlag ( 
    ULONG lConnect,
    ULONG lFlag,
    LPLONG pValue );
```

**Input Parameters**

- **lConnectID**
  
  lConnectID is a long integer that specifies the connection to use.

- **lFlag**
  
  lFlag is a long integer that specifies the flag number.

**Output Parameters**

- **pValue**
  
  pValue points to a long integer that receives the value of the flag.

  If a flag is clear, the returned value is zero, and if set, the returned value is one.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtPutFlag
CtPutFlag

The CtPutFlag function writes a new value to a flag. You can set or clear any flag (1 to 32). To set a flag, specify any positive number, and to clear the flag, enter any negative value or zero.

The format for calling CtPutFlag is as follows:

```c
ULONG CtPutFlag(
    ULONG IConnect,
    ULONG lFlag,
    ULONG lValue
);
```

**Input Parameters**

- **IConnect**
  - `IConnect` is a long integer that specifies the connection to use.

- **lFlag**
  - `lFlag` is a long integer that specifies the flag number to update.

- **lValue**
  - `lValue` is a long integer that specifies the new flag value to write.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtGetFlag
Digital I/O Functions

CtGetDigitalInput8

The CtGetDigitalInput8 function reads the values of a bank of eight digital inputs. Using this function you can read digital inputs in groups of eight by specifying the bank number. Bank number one is for inputs one to 8; bank two for inputs nine to 16, and so on.

The format for calling CtGetDigitalInput8 is as follows:

```
ULONG CtGetDigitalInput8 (?
    UULONG  lConnectID
    ULONG  lBank
    LPLONG  pValues
);?
```

Input Parameters

- **lConnectID**
  *lConnectID* is a long integer that specifies the connection to use.

- **lBank**
  *lBank* is a long integer that specifies the bank number of the block of eight digital inputs.

Output Parameters

- **pValues**
  *pValues* points to a long integer that receives the value of the eight digital inputs.

  The return value for the inputs is the bit mask of the block of eight inputs. The lower eight bits of the long integer contains the block of eight digital inputs.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetDigitalOutput8, CtGetDigitalInput128, CtGetDigitalOutput128
CtGetDigitalOutput8

CtGetDigitalOutput8 reads the value of a block of eight digital outputs. Using this function you can read digital outputs by specifying the bank number. Bank number one is for outputs one to 8; bank two for outputs nine to 16, and so on.

The format for calling CtGetDigitalOutput8 is as follows:

```
ULONG CtGetDigitalOutput8 (  
   UULONG lConnectID  
   ULONG lBank        
   LPLONG pValues     
 );
```

Input Parameters

- **lConnectID**
  - lConnectID is a long integer that specifies the connection to use.

- **lBank**
  - lBank is a long integer that specifies the bank number of the block of eight digital outputs.

Output Parameters

- **pValues**
  - pValues points to a long integer that receives the value of the eight digital outputs.

  The return value for the outputs is the bit mask of the block of eight outputs. The lower eight bits of the long integer contains the block of eight digital outputs.

Success/Error Return Values

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetDigitalInput8, CtGetDigitalInput128, CtGetDigitalOutput128
**Digital I/O Functions**

**CtGetDigitalInput128**

*CtGetDigitalInput128* reads the values of a block of 128 digital inputs. Using this function you can read digital inputs in groups of 128 by specifying the bank number. Bank number one is for inputs one to 128; bank two is for inputs 129 to 256, and so on.

The format for calling *CtGetDigitalInput128* is as follows:

```
ULONG CtGetDigitalInput128 (  
    ULONG  lConnectID  
    ULONG  lBank  
    DIGS_128*  pValues  
);  
```

Typedef ULONG DIGS_128[4];

**Input Parameters**

- **lConnectID**
  
  *lConnectID* is a long integer that specifies the connection to use.

- **lBank**
  
  *lBank* is a long integer that specifies the bank number of the block of 128 digital inputs.

**Output Parameters**

- **pValues**
  
  *pValues* points to a four-element array of long integers that receives the values of the 128 digital inputs.

  The values are returned in a bit mask. If there are less than 128 digital inputs in a block, zero is returned in each bit position for the undefined resources.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtGetDigitalInput8, CtGetDigitalOutput8, CtGetDigitalOutput128*
**CtGetDigitalOutput128**

*CtGetDigitalOutput128* reads the values of a block of 128 digital outputs. Using this function you can read digital outputs in groups of 128 by specifying the bank number. Bank number one is for outputs one to 128; bank two is for outputs 129 to 256, and so on.

The format for calling *CtGetDigitalOutput128* is as follows:

```c
ULONG CtGetDigitalOutput128 ( 
    ULONG lConnectID, 
    ULONG lBank, 
    DIGS_128* pValues );
```

**Typedef** `ULONG DIGS_128[4];`

**Input Parameters**

`lConnectID`

`lConnectID` is a long integer that specifies the connection to use.

`lBank`

`lBank` is a long integer that specifies the bank number of the block of 128 digital outputs.

**Output Parameters**

`pValues`

`pValues` points to a four-element array of long integers that receives the values of 128 digital outputs.

The values are returned in a bit mask If there are less than 128 digital outputs in a block, zero is returned in each bit position for the undefined resources.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtGetDigitalInput8, CtGetDigitalOutput8, CtGetDigitalInput128*
**Analog I/O Functions**

**CtGetAnalogInput**

*CtGetAnalogInput* reads the value of an analog input.

The format for calling *CtGetAnalogInput* is as follows:

```
ULONG CtGetAnalogInput ( 
    ULONG    lConnectID 
    ULONG    lAnalogIn 
    LPLONG   pValue   
); 
```

**Input Parameters**

- **lConnectID**
  - *lConnectID* is a long integer that specifies the connection to use.

- **lAnalogIn**
  - *lAnalogIn* is a long integer that specifies the number of the analog input.

**Output Parameters**

- **pValue**
  - *pValue* points to a long integer that receives the value of the analog input.

  The analog input value returned is a number from zero to 10,000.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

- CtGetAnalogOutput, CtPutAnalogOutput, CtGetAnalogInput32, CtGetAnalogOutput32
**CtGetAnalogOutput**

*CtGetAnalogOutput* reads the value of an analog output.

The format for calling *CtGetAnalogOutput* is as follows:

```c
ULONG CtGetAnalogOutput ( 
    ULONG lConnectID 
    ULONG lAnalogOut 
    LPLONG pValue
);```

**Input Parameters**

- `lConnectID` is a long integer that specifies the connection to use.
- `lAnalogOut` is a long integer that specifies the number of the analog output.

**Output Parameters**

- `pValue` points to a long integer that receives the value of the analog output.

The analog output value returned is a number from zero to 10,000.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtGetAnalogInput*, *CtPutAnalogOutput*, *CtGetAnalogInput32*, *CtGetAnalogOutput32*
Analog I/O Functions

CtPutAnalogOutput

CtPutAnalogOutput writes a value to an analog input. You can specify a value from zero to 10,000.

The format for calling CtPutAnalogOutput is as follows:

ULONGLONG CtPutAnalogOutput ( 
    ULONGLONG lConnectID
    ULONGLONG lAnalogOut
    ULONGLONG lValue
);

Input Parameters

lConnectID
lConnectID is a long integer that specifies the connection to use.

lAnalogOut
lAnalogOut is a long integer that specifies the number of the analog output.

lValue
lValue is a long integer that specifies the value to write to the analog output.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetAnalogInput, CtGetAnalogOutput, CtGetAnalogInput32, CtGetAnalogOutput32
**CtGetAnalogInput32**

*CtGetAnalogInput32* reads the values of a contiguous block of 32 analog inputs. Using this function you can read analog inputs in groups of 32 by specifying the bank number. Bank number one is for inputs one to 32; bank two for inputs 33 to 64, and so on.

The format for calling *CtGetAnalogInput32* is as follows:

```c
ULONG CtGetAnalogInput32 ( 
    ULONG  lConnectID, 
    ULONG  lBank, 
    ANGS_32* pValues );
```

**Typedef**

```
typedef ULONG ANGS_32[32];
```

**Input Parameters**

- `lConnectID`
  - is a long integer that specifies the connection to use.

- `lBank`
  - is a long integer that specifies the number of the analog input bank.

**Output Parameters**

- `pValues`
  - points to a 32-element array of long integers that receives the values of the block of 32 analog inputs.

  If there are less than 32 analog inputs in a block, zero is returned for the undefined resources.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

- CtGetAnalogInput, CtPutAnalogOutput, CtPuttAnalogOutput,
- CtGetAnalogOutput32
Analog I/O Functions

CtGetAnalogOutput32

CtGetAnalogOutput32 reads the values of a contiguous block of 32 analog outputs. Using this function you can read analog outputs in groups of 32 by specifying the bank number. Bank number one is for outputs one to 32; bank two for outputs 33 to 64, and so on.

The format for calling CtGetAnalogOutput32 is as follows:

ULONG CtGetAnalogOutput32 (  
    ULONG lConnectID  
    ULONG lBank  
    ANGS_32* pValues  
);

Typedef ULONG ANGS_32[32];

Input Parameters

lConnectID  
lConnectID is a long integer that specifies the connection to use.

lBank  
lBank is a long integer that specifies the number of the analog output bank.

Output Parameters

pValues  
pValues points to a 32-element array of long integers that receives the values of the block of 32 analog outputs.

If there are less than 32 analog outputs in a block, zero is returned for the undefined resources.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetAnalogInput, CtPutAnalogOutput, CtPuttAnalogOutput,  
CtGetAnalogInput32
Servo Functions

CtGetServoPosition

**CtGetServoPosition** reads the current position of a servo. You can specify a servo number from one to 16.

The format for calling **CtGetServoPosition** is as follows:

```c
ULONG CtGetServoPosition ( 
    ULONG   lConnectID
    ULONG   lServo
    LPLONG  pValue  );
```

**Input Parameters**

- **lConnectID**
  - *lConnectID* is a long integer that specifies the connection to use.

- **lServo**
  - *lServo* is a long integer that specifies the servo number.

**Output Parameters**

- **pValue**
  - *pValue* points to a long integer that receives the value of the servo position.
  
  The servo position value can range from –2,147,483,647 to 2,147,483,647.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the **CtGetErrorInfo** function.

**Other Related Functions**

- CtGetServoError, CtGetServoInputs
Servo Functions

CtGetServoError

CtGetServoError reads the current error of a servo. You can specify a servo number from one to 16.

The format for calling CtGetServoError is as follows:

```
ULONG CtGetServoError ( 
    ULONG lConnectID 
    ULONG lServo 
    LPLONG pValue 
); 
```

**Input Parameters**

- **lConnectID**
  
  lConnectID is a long integer that specifies the connection to use.

- **lServo**
  
  lServo is a long integer that specifies the servo number.

**Output Parameters**

- **pValue**
  
  pValue points to a long integer that receives the value of the servo error.

  The servo error value can range from –2,147,483,647 to 2,147,483,647.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

- CtGetServoPosition, CtGetServoInputs
CtGetServoInputs

CtGetServoInputs reads the value of servo inputs. You can specify a servo number from one to 16.

The format for calling CtGetServoInputs is as follows:

```c
ULONG CtGetServoInputs( 
    ULONG lConnectID, 
    ULONG lServo, 
    LPLONG pValue);
```

**Input Parameters**

- `lConnectID`
  `lConnectID` is a long integer that specifies the connection to use.

- `lServo`
  `lServo` is a long integer that specifies the servo number.

**Output Parameters**

- `pValue`
  `pValue` points to long integer that receives the value of the servo inputs.

  The returned value is a bit mask of the inputs of the servo where:

  - Bit 0 = Not used
  - Bit 1 = Home
  - Bit 2 = Start
  - Bit 3 = Kill Command
  - Bit 4 = Reverse Limit
  - Bit 5 = Forward Limit
  - Bit 6 = Index
  - Bits 7 to 31 = Not used/reserved for future use

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtGetServoPosition, CtGetServoError
Data Table Functions

CtGetDataTableDim

CtGetDataTableDim reads the dimensions of the data table. If no data table exists, the return values are zero.

The format for calling CtGetDataTableDim is as follows:

ULONG CtGetDataTableDim ( 
    ULONG  lConnectID  
    LPLONG pRows   
    LPLONG pCols );

Input Parameters

lConnectID

lConnectID is a long integer that specifies the connection to use.

Output Parameters

pRows

pRows points to a long integer that receives the number of rows in the data table.

pCols

pCols points to a long integer that receives the number of columns in the data table.

Value Returned

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetDataTableLoc, CtPutDataTableLoc, CtGetDataTableRow, CtPutDataTableRow
CtGetDataTableLoc

CtGetDataTableLoc reads a value at a location in the data table.

The format for calling CtGetDataTableLoc is as follows:

ULONG CtGetDataTableLoc ( 
    ULONG  lConnectID
    ULONG  lRows
    ULONG  lCols
    LPLONG pValue
);

Input Parameters

lConnectID
lConnectID is a long integer that specifies the connection to use.

lRows
lRows is a long integer that specifies the row location in the data table.

lCols
lCols is a long integer that specifies the column location in the data table.

Output Parameters

pValue
pValue points to a long integer that receives the value from the specified location in the data table.

The returned value is a number from 0 to 65535.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetDataTableDim, CtPutDataTableLoc, CtGetDataTableRow, CtPutDataTableRow
Data Table Functions

CtPutDataTableLoc

CtPutDataTableLoc writes a value to a location in the data table.

The format for calling CtPutDataTableLoc is as follows:

`ULONG CtPutDataTableLoc (``
  ULONG lConnectID``
  ULONG lRows``
  ULONG lCols``
  ULONG lValue``
);``

Input Parameters

- `lConnectID`
  `lConnectID` is a long integer that specifies the connection to use.

- `lRows`
  `lRows` is a long integer that specifies the row location in the data table.

- `lCols`
  `lCols` is a long integer that specifies the column location in the data table.

- `lValue`
  `lValue` is a long integer that specifies the value in the data table.

  Allowable values for `lValue` are numbers from 0 to 65535.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the `CtGetErrorInfo` function.

Other Related Functions

- CtGetDataTableDim, CtGetDataTableLoc, CtGetDataTableRow, CtPutDataTableRow
CtGetDataTableRow

CtGetDataTableRow reads the values in a data table row. You must specify a starting location in the data table.

The format for calling CtGetDataTableRow is as follows:

```c
ULONG CtGetDataTableRow (  
  ULONG   lConnectID  
  ULONG   lRow        
  ULONG   lCol        
  ULONG   lNumCols    
  CT_DT_ROW* pValues  
); 
```

Typedef struct {
  ULONG   lRow;
  ULONG   lColumns[255];
} CT_DT_ROW;

Input Parameters

lConnectID
lConnectID is a long integer that specifies the connection to use.

lRow
lRows is a long integer that specifies the row location in the data table.

lCol
lCols is a long integer that specifies the column location in the data table.

lNumCols
lNumCols is a long integer that specifies the number of columns of data to read from the data table. It can be any number between one and 255.

Output Parameters

pValues
pValues points to a structure that holds the data read from the data table.

The structure contains the data returned from the lRow row. Only the data beginning with the lcol columns and extending from the lNumCols columns (or until the last column in the data table) are updated in the structure.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetDataTableDim, CtGetDataTableLoc, CtPutDataTableLoc, CtPutDataTableRow
Data Table Functions

CtPutDataTableRow

CtPutDataTableRow writes values to a row in a controller’s data table. You must specify a starting location within the bounds of the data table.

The format for calling CtPutDataTableRow is as follows:

```
ULONG CtPutDataTableRow(
    ULONG lConnectID,
    ULONG lRow,
    ULONG lCol,
    ULONG lNumCols,
    CT_DT_ROW * pValues
);
```

```
typedef struct {
    ULONG lRow;
    ULONG lColumns[255];
} CT_DT_ROW;
```

Input Parameters

- **lConnectID**
  - lConnectID is a long integer that specifies the connection to use.

- **lRow**
  - lRow is a long integer that specifies the row location in the data table.

- **lCol**
  - lCol is a long integer that specifies the column location in the data table.

- **lNumCols**
  - lNumCols is a long integer that specifies the number of columns of data to write to the data table. It can be any number between one and 255.

- **pValues**
  - pValues points to a structure that holds the data to write to the data table. The structure contains the data to write in the lRow row. Only the data beginning with lCol column and extending for lNumCols columns (or until the last column in the data table) are written.

Success/Error Return Values

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

- CtGetDataTableDim, CtGetDataTableLoc, CtPutDataTableLoc, CtGetDataTableRow
Controller Status Functions

**CtGetStatus**

*CtGetStatus* reads the controller status.

The format for calling *CtGetStatus* is as follows:

```c
ULONG CtGetStatus (  
    ULONG   lConnectID  
    LPLONG  pValue      );
```

**Input Parameters**

*lConnectID*

*lConnectID* is a long integer value that specifies the connection to use.

**Output Parameters**

*pValue*

*pValue* points to a long integer that receives the value of the controller status.

The returned value is a bit mask of the controller status where:

- Bit 0 = 0 if running, 1 if stopped
- Bit 1 = 0 if normal mode, 1 if programming mode
- Bit 2 = 0 if status OK., 1 if software fault
- Bit 3 = 0 if mid-program, 1 if fresh reset
- Bit 4 to 31 are reserved for future use.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

*CtPutStatus, CtGetTaskBank*
**Controller Status Functions**

**CtPutStatus**

CtPutStatus sets the controller status.

The format for calling CtPutStatus is as follows:

```c
ULONG CtPutStatus ( 
    ULONG       lConnectID 
    ULONG       lValue    );
```

**Input Parameters**

- **lConnectID**
  
  lConnectID is a long integer value that specifies the connection to use.

- **lValue**
  
  lValue is a long integer that specifies the value of the controller status.

  Specify the status value as a bit mask where:

  - Bit 0 = 1 to start controller, 0 to stop controller
  - Bit 1 = 0 (required)
  - Bit 2 = 0 (required)
  - Bit 3 = 1 to reset controller, otherwise 0
  - Bit 4 to 31 = 0 (required)

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

**Other Related Functions**

CtGetStatus, CtGetTaskBank
**CtGetTaskBank**

*CtGetTaskBank* reads a program status task bank from the controller. Depending on the model, CTC controllers can have 16, 32, or 64 tasks running in a program. *CtGetTaskBank* can access the status of each of these tasks by reading task banks of eight tasks each.

The format for calling is **CtGetTaskBank** as follows:

```
ULONG CtGetTaskBank ( 
    ULONG   lConnectID
    ULONG   lBank
    CT_TASK_BANK  * pValues );
```

**Typedef struct {**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULONG</td>
<td>1Stopped;</td>
</tr>
<tr>
<td>ULONG</td>
<td>lFaultType;</td>
</tr>
<tr>
<td>ULONG</td>
<td>lFaultStep;</td>
</tr>
<tr>
<td>ULONG</td>
<td>lFaultData;</td>
</tr>
<tr>
<td>ULONG</td>
<td>lTaskStep[8];</td>
</tr>
<tr>
<td>ULONG</td>
<td>lTaskMask1[8];</td>
</tr>
<tr>
<td>ULONG</td>
<td>lTaskMask2[8];</td>
</tr>
</tbody>
</table>

} CT_TASK_BANK;**

**Input Parameters**

*iConnectID*  
*iConnectID* is a long integer value that specifies the connection to use.

*iBank*  
*iBank* is a long integer that specifies the program task bank.

**Output Parameters**

*pValues*  
*pValues* points to a task bank structure that receives the returned values.

**Remarks**

*iBank Value*  
The *iBank* value can be any number between one and four, where one returns the data for tasks one to 8; two returns the data for tasks nine to 16; etc. For CTC controllers equipped with a 2701E CPU, the *iBank* value can be any number between 65 and 72.

**CT_TASK_BANK Structure**  
The **CT_TASK_BANK** structure returned contains the following runtime status information:

- Controller state, one if stopped, zero otherwise
- Controller fault code, if any
- Step where the fault has occurred, if any
- Data associated with the fault condition, if any
- Step number that each task is executing
- Step data for each task

The step data returned is a bit mask value of any other tasks that this task is waiting on before resuming execution. A bit value of one means that the task is waiting on the task associated with the bit number, i.e., waiting on task seven if bit seven is set.
Controller Status Functions

Success/Error Return Values
If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions
CtGetStatus, CtPutStatus
Controller Configuration Functions

CtGetModel

CtGetModel reads the controller’s configuration.

The format for calling CtGetModel is as follows:

ULONG CtGetModel (  
    ULONG lConnectID  
    LPLONG pModel  
    BOOL * pIsEA );

Input Parameters

lConnectID

lConnectID is a long integer value that specifies the connection to use.

Output Parameters

pModel

pModel points to a long integer that receives the controller model.

pIsEA

pIsEA points to a boolean that receives the controller architecture type.

Remarks

The model number returned is a value specific to the controller model. The architecture type returned is false for non-EA controllers, or true for EA controllers. EA controllers have additional internal resources and command sets not available in non-EA models.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetConfig, CtPutConfig, CtGetIoCount, CtGetMiscIoCount
Controller Configuration Functions

CtGetConfig

CtGetConfig reads the controller configuration.

The format for calling CtGetConfig is as follows:

ULONG CtGetConfig ( 
    ULONG lConnectID 
    LPLONG pValue 
); 

Input Parameters

lConnectID
lConnectID is a long integer value that specifies the connection to use.

Output Parameters

pValue
pValue points to long integer that receives the configuration value.

The returned value is a bit mask of the controller configuration where:

- Bit 0 = 1 if using input 1 for START function, 0 otherwise
- Bit 1 = 1 if using input 2 for STOP function, 0 otherwise
- Bit 2 = 1 if using input 3 for RESET function, 0 otherwise
- Bit 3 = 1 if using input 4 for STEP function, 0 otherwise
- Bit 4 to 31 = 0

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetModel, CtPutConfig, CtGetIoCount, CtGetMiscIoCount
CtPutConfig

**CtPutConfig** sets the controller configuration.

The format for calling **CtPutConfig** is as follows:

```c
ULONG CtPutConfig(
    ULONG lConnectID
    ULONG lValue
);```

**Input Parameters**

- **lConnectID**
  A long integer value that specifies the connection to use.

- **lValue**
  A long integer that sets the value of the controller configuration.

The configuration value is a bit mask where:

- Bit 0 = 1 to using input 1 for START function, 0 otherwise
- Bit 1 = 1 to using input 2 for STOP function, 0 otherwise
- Bit 2 = 1 to using input 3 for RESET function, 0 otherwise
- Bit 3 = 1 to using input 4 for STEP function, 0 otherwise
- Bit 4 to 31 = 0

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the **CtGetErrorInfo** function.

**Other Related Functions**

- CtGetModel
- CtGetConfig
- CtGetIoCount
- CtGetMiscIoCount
Controller Configuration Functions

CtGetIoCount

CtGetIoCount reads the controller's basic hardware configuration. The function returns values listing the number of flags, digital inputs, digital outputs, stepping motors, servo motors, analog inputs, and analog outputs installed in the controller.

The format for calling CtGetIoCount is as follows:

```c
ULONG CtGetIoCount (  
    ULONG lConnectID  
    CT_IO_COUNT * pValue  
);  
```

```c
typedef struct {  
    ULONG lFlags;  
    ULONG lDigitalInputs;  
    ULONG lDigitalOutputs;  
    ULONG lSteppingMotors;  
    ULONG lServoMotors;  
    ULONG lAnalogInputs;  
    ULONG lAnalogOutputs;  
} CT_IO_COUNT;  
```

Input Parameters

`lConnectID`
`lConnectID` is a long integer value that specifies the connection to use.

Output Parameters

`pValue`
`pValue` points to a structure that receives the basic hardware configuration information.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetModel, CtGetConfig, CtPutConfig, CtGetMiscIoCount
CtGetMiscIoCount

CtGetMiscIoCount reads the controllers extended hardware configuration. The function returns values listing the number of prototype boards, high speed counters, four-digit thumbwheels, and four-digit numeric displays installed in the controller.

The format for calling is as follows:

```c
ULONG CtGetMiscIoCount ( 
    ULONG CtlConnectID 
    CT_MISC_IO_COUNT * pValue 
); 
```

Typedef struct {
    ULONG lPrototypes;
    ULONG lHighSpeedCounters;
    ULONG lThumbwheels;
    ULONG lDisplays;
} CT_MISC_IO_COUNT;

Input Parameters

`CtlConnectID`

`CtlConnectID` is a long integer value that specifies the connection to use.

Output Parameters

`pValue`

`pValue` points to a structure that receives the extended hardware configuration information.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtGetModel, CtGetConfig, CtPutConfig, CtGetIoCount
Programming Functions

CtDownload

CtDownload downloads an object file into the controller. The function returns values listing the number of steps, bytes, errors, and warnings in the object file downloaded to the controller.

The format for calling CtDownload is as follows:

```
ULONG CtDownload (
    ULONG lConnectID,
    CONST CHAR FAR * pObjFile,
    CT_PROG_COUNT * pCounts);
```

Typedef struct {
    ULONG lSteps;
    ULONG lBytes;
    ULONG lErrors;
    ULONG lWarnings;
} CT_PROG_COUNT;

Input Parameters

lConnectID
lConnectID is a long integer value that specifies the connection to use.

pObjFile
pObjFile points to the object file to be downloaded.

Output Parameters

pCounts
pCounts points to a structure that receives the object file information.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtUpload
CtUpload uploads the controllers program to an object file. The function returns values listing the number of steps, bytes, errors, and warnings in the object file downloaded to the controller.

The format for calling CtUpload is as follows:

```c
Ulong CtGetMiscIoCount (Ulong lConnectID, const char far * pObjFile, CT_PROG_COUNT * pCounts);
```

Typedef struct {
  ULONG lSteps;
  ULONG lBytes;
  ULONG lErrors;
  ULONG lWarnings;
} CT_PROG_COUNT;

Input Parameters

- **lConnectID**
  - `lConnectID` is a long integer value that specifies the connection to use.

- **pObjFile**
  - `pObjFile` points to the object file to be uploaded.

Output Parameters

- **pCounts**
  - `pCounts` points to a structure that receives the object file information.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CtGetErrorInfo function.

Other Related Functions

CtDownload
Miscellaneous Functions

CtGetConnectionInfo

*CtGetConnectionInfo* returns information on the connection. It lists the connection type, the connection time out value, and ?????? *What do some of the other Typedef parameters mean? I could use the info here.*

The format for calling *CtGetConnectionInfo* is as follows:

```c
ULONG CtGetConnectionInfo ( 
    EULONG     lConnectID
    CT_CONN_INFO * pConnInfo
);
```

```
Typedef struct {
    ULONG     lConnType;
    ULONG     lSrcAddr;
    ULONG     lDestAddr;
    ULONG     lTimeout;
    BOOL      fConnected;
    BYTE      srcEAddr[6];
    BYTE      destEAddr[6]
} CT_CONN_INFO;
```

Input Parameters

*lConnectID*

*lConnectID* is a long integer value that specifies the connection to use.

Output Parameters

*pConnInfo*

*pConnInfo* points to a structure that receives the connection information.

Success/Error Return Values

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

Other Related Functions

*CtGetMessageInfo, CtGetErrorInfo*
**CtGetMessageInfo**

*CtGetMessageInfo* returns information on the last message sent and last response received for the connection. The messages retrieved can have a length of 256 characters.

The format for calling *CtGetMessageInfo* is as follows:

```c
ULONG CtGetMessageInfo ( 
    ULONG lConnectID
    CT_MSG_INFO * pMsgInfo
);
```

**Typedef struct {**

```c
    ULONG lMsglen;
    BYTE sMsgbuf[256];
    ULONG lRsplen;
    BYTE sRspbuf[256];
} CT_MSG_INFO;
```

**Input Parameters**

*lConnectID*

*lConnectID* is a long integer value that specifies the connection to use.

**Output Parameters**

*pMsgInfo*

*pMsgInfo* points to a structure that receives the message information.

**Value Returned**

The *CT_MSG_INFO* structure returned contains the message data for the connection.

**Success/Error Return Values**

If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the *CtGetErrorInfo* function.

**Other Related Functions**

CtGetConnectionInfo, CtGetErrorInfo
CTGetErrorInfo

CTGetErrorInfo returns information on the last error condition for the connection. The error message returned can have a length of 256 characters.

The format for calling CTGetErrorInfo is as follows:

```c
ULONG CtGetErrorInfo ( 
    ULONG lConnectID, 
    CT_ERR_INFO * pError );
```

Typedef struct {
    ULONG lCode;
    CHAR bMessage[256];
} CT_ERR_INFO;

**Input Parameters**

- **lConnectID**
  lConnectID is a long integer value that specifies the connection to use.

**Output Parameters**

- **pError**
  pError points to a structure that receives the error information.

**Success/Error Return Values**

- If the function succeeds, the return value is one. If the function fails, the return value is zero. Extended error information can be acquired by calling the CTGetErrorInfo function.

**Other Related Functions**

CtGetConnectionInfo, CtGetMessageInfo