Cirris Tester Access (CTA)

User Manual

CIRRIS®

A KOMAX COMPANY

Table of Contents

1. Introduction

Cirris Tester Access (CTA) is an Application Programming Interface (API) that facilitates control of Cirris testers using external applications by providing the functions needed to customize and automate the test process in programming environments such as LabVIEW, C/C++, Python, and Delphi. CTA is compatible with 32-bit and 64-bit applications.

1.1 Components

CTA, consists of three main components.

- **CTA Client** - The CTA Client DLL provides functions that custom applications can use to communicate with the CTA Server and control Cirris testers. Both 32-bit and 64-bit versions of the DLL are provided. The name of the Cirris 64-bit DLL includes the designation "_x64" appended to the filename.

  The 32-bit DLL can be found under the path:
  C:\Program Files (x86)\Cirris\CTA\CirrisTesterAccess.dll

  The 32-bit DLL specifically for use with LabView can be found under the path:
  C:\Program Files (x86)\Cirris\CTA\LabView\CirrisTesterAccessLabView.dll

  The 64-bit DLL can be found under the path:
  C:\Program Files (x86)\Cirris\CTA\CTA x64\CirrisTesterAccess_x64.dll

  The 64-bit DLL specifically for use with LabView can be found under the path:
  C:\Program Files (x86)\Cirris\CTA\CTA x64\CirrisTesterAccessLabView_x64.dll

- **CTA Server (easywire.exe running in server mode)** - The CTA Server is the direct connection to Cirris Testers, providing a uniform interface for all of the functions defined in the Client DLL across all testers controlled by the Easy-Wire software. Easy-Wire can be started in CTA Server mode by using the command line parameter, "8."

  For example, if the CTA server is on the local drive: C:\Program Files (x86)\Cirris\easywire\easywire.exe 8

- **User-Created Custom Application** - Custom Applications can be written in any language that supports access to Windows DLLs. Examples include C/C++, Python, Delphi, and LabVIEW,

Custom Applications access functions contained in the CTA Client by using a "wrapper" that defines the functions and interface, for instance a C or C++ header file (*.h, *.hpp). This allows the Custom Application, to use the CTA DLL to send information to, request information from, and control the tester connected to the CTA Server. The CTA Client communicates with the CTA Server using a TCP/IP messaging protocol that allows for the CTA Server to be on the same, or a remote PC. Firewall, security, and anti-virus settings must be configured properly to allow remote CTA Server access.

1.2 Configuration Settings

The CTA Client and CTA Server use separate configuration files for the connection parameters. These files are located in the folder: C:\Users\Public\Documents\Cirris\Common

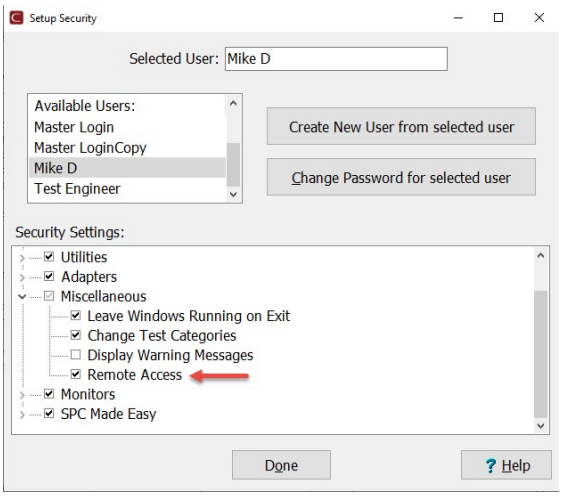
- **CTA Client configuration file (CirrisTesterAccessClient.ini)** - The key entry specifies the IP Address of the Server and the Port on which the Server is Listening: CTAClientConnectAddress=tcp://127.0.0.1:55555
- **CTA Server configuration file (CirrisTesterAccessServer.ini)** - The key entry specifies the port on which the server is listening for client connections: CTAServerBindAddress=tcp://*:55555

2. Getting Started

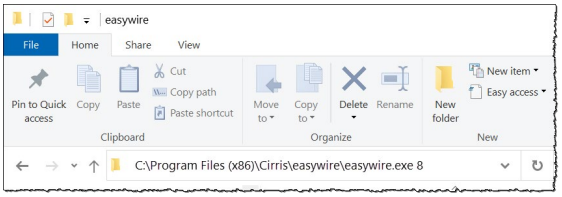
2.1 Starting Easy-Wire CTA Server

When the CTA Server and Client are located on the same PC, follow the steps below. When the CTA Server and Client are on separate PCs, also see the following section.

1. Before opening Easy-Wire in the CTA Server mode, open the Easy-Wire software using the desktop shortcut or from the Windows Start Menu > Cirris System Corporation > Easy-Wire. Create and/or configure the intended user with **Remote Access** rights in the Easy-Wire security settings accessed from the **Main Menu** > **Utilities** > **Setup Security** under the **Miscellaneous** section. See the Easy-Wire Help for more information on security settings.



2. After the Easy-Wire software has been opened at least once with the tester attached to the Server PC, the CTA Server can be started from the **Cirris Tester Access Server** shortcut on the desktop or by opening Easy-Wire in the server mode by adding the command line parameter “8.”



2.2 Client Test Panel

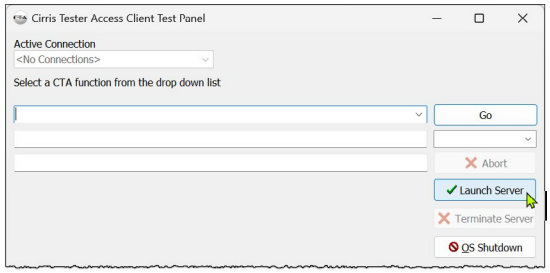
The Client Test Panel is a simple application, installed with Easy-Wire, that allows the user to establish a connection to the CTA Server and to experiment with functions individually, step-by-step. Functions are selected from the drop-down list and the interface will prompt the user to enter the variables required for the selected function. Select **Go** or press Enter to execute the selected function. The response will be displayed in the **Command Results** pane.

A complete list of the available functions begins on *page 14.*

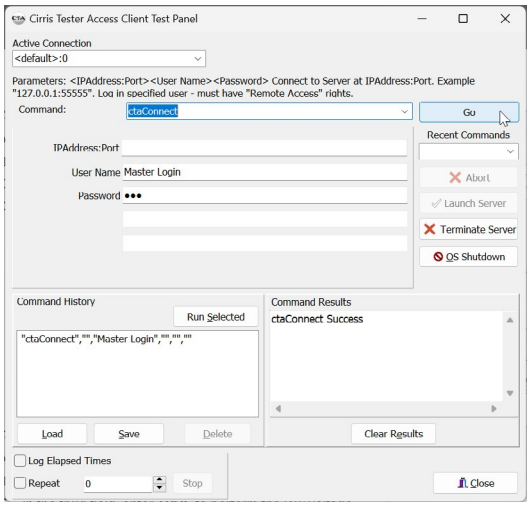
2.2.1 Connecting to the Server

1. Start **CTAClientTestPanel.exe.** The 32-bit version can be found under the path: C:\Program Files (x86)\Cirris\CTA\CTAClientTestPanel.exe. The 64-bit version can be found under the path: C:\Program Files (x86)\Cirris\CTA\CTA x64\CTAClientTestPanel_x64.exe.

2. Click on **Launch Server** to open Easy-Wire in the CTA Server mode.



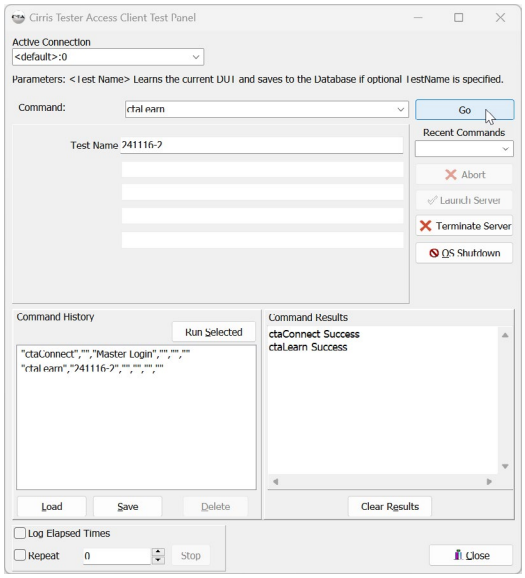
3. To ensure the CTA Server and Client are installed and working properly, in the Test Panel, select **ctaConnect** from the **Function** drop down list. Each required field will include an initial description.
   A. Clear the **<IPAddress:Port>** field (this can be used later to specify a different IP address and Port for a Remote CTA Server).
   B. Enter the Easy-Wire user credentials (username and password) and click **Go** or press Enter to connect to the server. Ensure the user credentials are set to allow Remote Access in the Easy-Wire security settings (*page 5*).
   C. The **Command Results** pane should display **ctaConnect Success**.



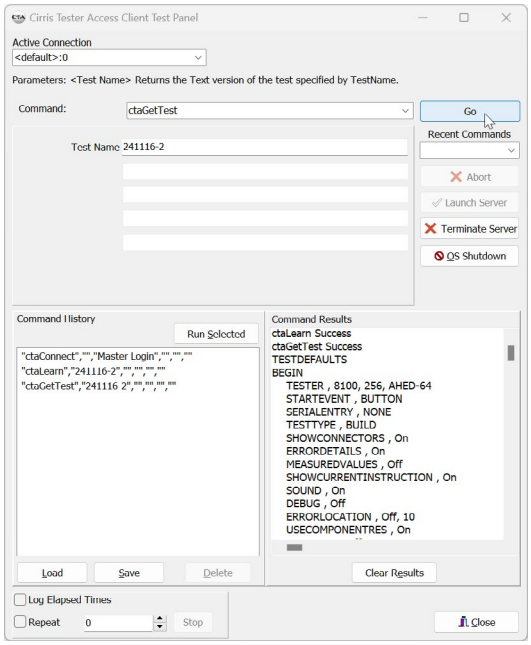
2.2.2 Client Test Panel Example

With the Client Test Panel connected, it is now possible to use other functions, step-by-step, to see how they work. As a very simple experiment, to learn and test a sample product:

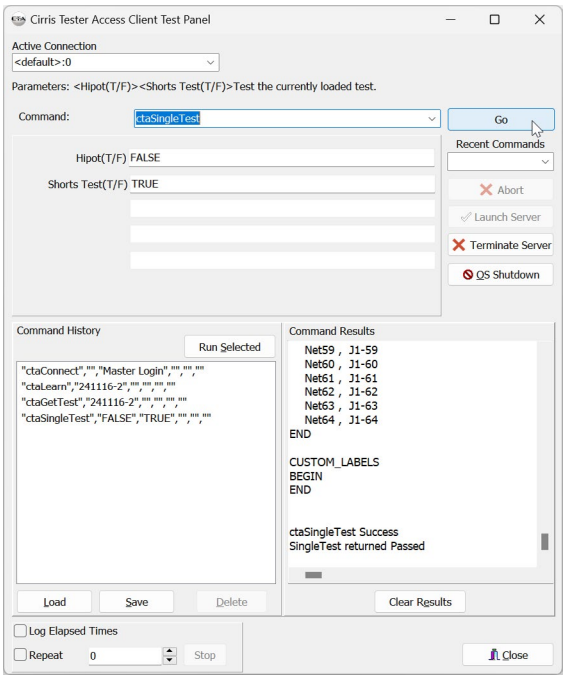
1. Connect a sample product to the tester.
2. To learn a test using the sample product:
   A. From the drop-down list, select **ctaLearn**,
   B. Enter a name for the new test in the second field.
   C. Click **Go**.
   D. The result should be **ctaLearnSuccess**.



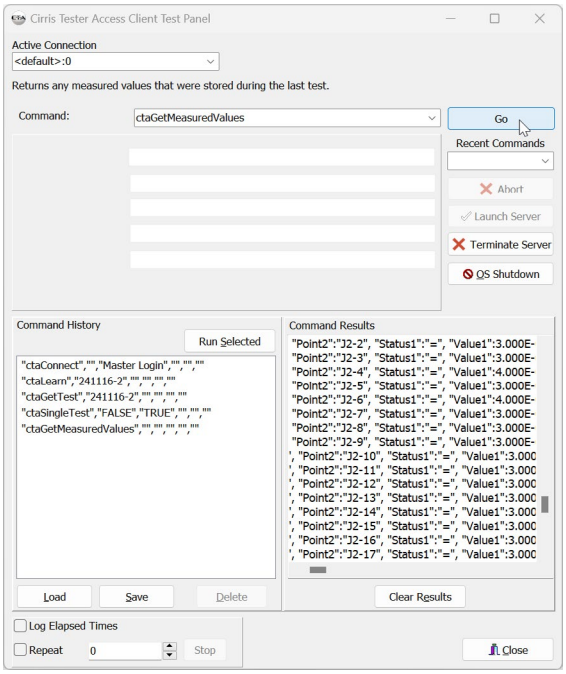
3. To view the learned test.
   A. From the drop-down list, select **ctaGetTest**,
   B. Enter a name of the learned test in the second field.
   C. Click **Go**.
   D. The result should be **ctaGetTest Success** and a text version of the test should be displayed. Scroll to view the entire test.



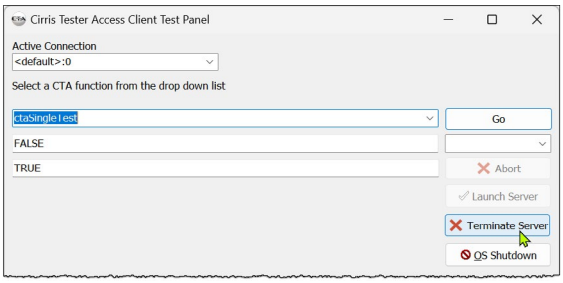
4. To execute the test:
   A. From the drop-down list, select **ctaSingleTest**,
   B. In the second Field, enter **FALSE** to skip Hipot testing.
   C. In the third field, enter **TRUE** to perform the low voltage shorts test.
   D. Click **Go**.
   E. The result should be **ctaSingleTest Success** and **SingleTest returned Passed**.



5. To view measured values from the test:
   A. From the drop-down list, select **ctaGetMeasuredValues**,
   B. Click **Go**.
   C. The result should be **ctaGetMeasuredValues Success** and list of the measured values. Scroll to view all the measurements.

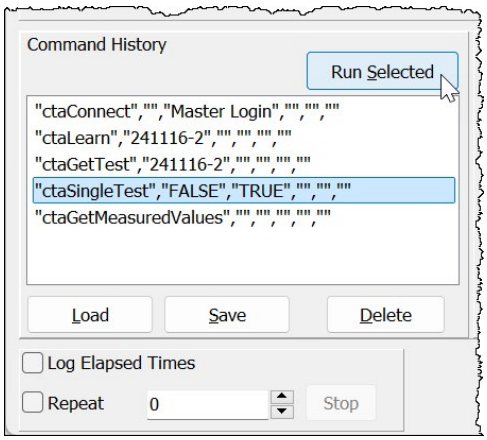


6. When finished, click **Terminate Server** or use the function **ctaDisconnect** before exiting the Test Panel.



2.2.3 Miscellaneous Commands

Additional miscellaneous commands are available on the Client Test Panel interface.



- **Run Selected** - Performs the CTA command(s) highlighted in the **Command History** pane.
- **Load** - Opens a previously saved CTA (*.CTA) file.
- **Save** - Saves the command(s) in the **Command History** pane as a file with a .CTA filename extension.
- **Delete** - Deletes the command(s) highlighted in the **Command History** pane.
- **Log Elapsed Times** - Records and displays the elapsed time for each executed command in the **Command Results** pane.
- **Repeat** - Repeats the selected command(s) the specified number of times.
- **Stop** - Ends the execution of commands regardless of whether the specified number of repeat cycles has been reached.

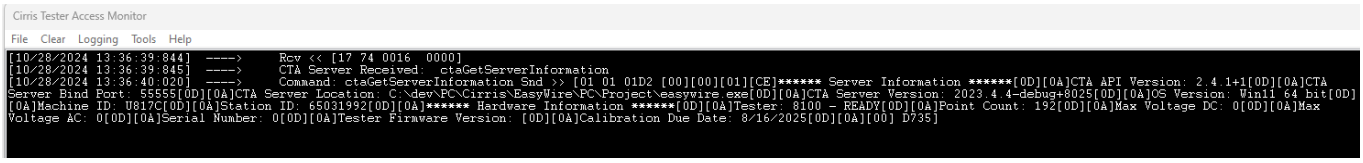
2.3 CTA Server & Client on separate PCs

When the CTA Server and Client are on separate PCs, the process is similar to that described above, except:

1. Copy the contents of the CTA folder (C:\Program Files (x86)\Cirris\CTA\) to the client computer
2. Place the **CirrisTesterAccessClient.ini** file in the directory: C:\Users\Public\Documents\Cirris\Common\
3. Edit the **CTAClientConnectAddress** in the **CirrisTesterAccessClient.ini** file to match the IP Address and Port of the Server PC (by default the server is listening for clients on port **55555**).
4. While it's not strictly required, it's recommended that both the **CirrisTesterAccess.dll** AND the **libzmq.dll** be placed within the system path so they can be accessed and can communicate with each other as needed.

2.4 Server Monitor

Tip: The function **ctaShowServerMonitor** (*page 29*) can be used to show or hide a status window to track the communication activity. The selected setting, show or hide, is saved in the CirrisTesterAccessServer.ini file and the state is maintained until changed. The monitor is not typically used except for troubleshooting or experimentation purposes.



2.5 Client Program Examples

As shown in the following, simple examples, three key steps are required in a typical custom application:

1. Connect with Username and Password (“JohnDoe” and “PW123!” in the example).
2. Use functions to control the tester and get results (see *Functions* *page 14* and *Return Values* *page 32)*.
3. Disconnect when finished.

Additional examples, including multiple wrapper files and program files, and the DLLs are located in the CTA folder in the directory: C:\Program Files (x86)\Cirris\CTA

***Note**: The LabView wrapper example uses the 32-bit DLL. If using the 64-bit version of LabView, an error will be encounter that states the DLL cannot be found. In that case, the project must use the 64-bit DLL which is located here: C:\Program Files (x86)\Cirris\CTA\CTA x64\CirrisTesterAccessLabView_x64.dll*

2.5.1 Python Program Example

```
// CTA Example...
// Connect() is used to establish a connection and log in to the server.
// connection_id is set by Connect() and used for subsequent commands
// throughout a session, until Disconnect() is called.
var
  word connection_id;
  word test_result;
  string test_errors;
if ctaConnect("127.0.0.12:55555", "JohnDoe", "PW123!", connection_id) = tacrSuccess
    if ctaLoadTest(connection_id, “Test123”) = tacrSuccess
      if ctaSingleTest(connection_id, 0, 1, test_result) = tacrSuccess
        if test_result = ctatrPassed then
          ShowMessage(“Test Passed”)
        else
          int error_length
          string errors;
          error_length = 5000
          SetLength(errors, error_length)
          if ctaGetTestErrors(connection_id, error_length, errrors) = tacrSuccess
            ShowMessage(“Test Failed with Errors:\n” + errors)
          else
            ShowMessage(“Unable to retrieve Test Errors”)
          endif
        endif
      else
        ShowMessage(“Single Test failed to execute”)
      endif
    else
      ShowMessage(“Unable to Load Test”)
    endif
    ctaDisconnect(connection_id) // 0=Leave Server Running 1=Shut Down Server
 else
   ShowMessage(“Unable to connect”)
 endif

```

2.5.2 C# Program Example

```
using CtaDotNet.Services;
using CtaDotNet.Models;
using CtaDotNet.Enums;

// const for connection parameters
const string serverIPandPort = "127.0.0.1:55555";
const string userName = "Master Login";
const string password = "";

// Create instance of CtaService to issue calls to api functions
CtaService service = new();

// Defining variables to receive command responses
CtaShortResult connection_result;
DllDualValueReturnObject<CtaTestErrorResult> test_errors_result;
CtaBoolResult selfTestResult;
CtaCommandResultCode enableJSONResult;
CtaCommandResultCode loadTestResult;
CtaStringResult loadedTestNameResult;
CtaStringResult getLastErrorResult;
CtaCommandResultCode startTestRunResult;
CtaTestResult test_result;
DllDualValueReturnObject<CtaMeasuredValuesResult> measured_values_result;
CtaCommandResultCode saveTestResult;

// variable to store connection id. This connection id will allow for one
// application to interact with multiple Cta servers.
short connection_id = -1;

// Example of Connect call. parameters are all strings, ip address and
// port of the server, username configured in EasyWire, password configurd in EasyWire.
connection_result = service.Connect(serverIPandPort, userName, password);

if (connection_result.DllCallResult == CtaCommandResultCode.Success)
{
    connection_id = connection_result.FunctionResult;
}
else
{
    // Handle failed connection logic here.
}

// Example of SelfTest
selfTestResult = service.SelfTest(connection_id);

// Example of EnableJSONResults in addition to connection_id one boolean parameter to indicate whether
// results will be returned in JSON format or not. Default behavior is to return JSON.
enableJSONResult = service.EnableJSONResults(connection_id, true);

// Example of Loadtest, one string parameter of the name of the test to load.
loadTestResult = service.LoadTest(connection_id, "CtaExample");

// Example of LoadedTestname
loadedTestNameResult = service.LoadedTestName(connection_id);

// Example of GetLastError
getLastErrorResult = service.GetLastError(connection_id);

// Example of StartTestRun, two string parameters in addition to connection_id.
// First is the name of the test to be used in the test run. Second is the LotId for the test run.
// The SaveTestResults command requires the StartTestRun prior to any test results you want saved.
startTestRunResult = service.StartTestRun(connection_id, "CtaExample", "1234");

// Example of SingleTest, two boolean parameters in addition to connection_id. First controls whether
// Hipot tests will be performed. Second controls whether the shorts test will be run. If both are
// false only the test instructions will be run.
test_result = service.SingleTest(connection_id, false, true);
if (test_result.TestResultCode == CtaTestResultCode.Passed)
{
    // Handle Test passed logic here
}
```

```
70          // Handle Test passed logic here
71      }
72      else
73      {
74          // Example of GetTestErrors
75          test_errors_result = service.GetTestErrors(connection_id);
76      }
77
78      // Example of GetMeasuredValues
79      measured_values_result = service.GetMeasuredValues(connection_id);
80
81      // Example of SaveTestResults, one additional string parameter of the serial number
82      saveTestResult = service.SaveTestResults(connection_id, "SN5678");
83
84      // Eample of EndTestRun
85      service.EndTestRun(connection_id);
86
87      // Example of Disconnect, one additional boolean parameter controlling whether the
88      // Cta server will terminate or stay running.
89      service.Disconnect(connection_id, false);
90
```

3. Specifying Test Points

Some CTA functions call for a measurement (test) between specified test points. Test points can be specified using typical Easy-Wire connector-pin designations (for example J1-1, J1-2 etc.). In Easy-Wire, connector designations are the references assigned to defined connectors/adapters under the **Define Connectors / Define Adapters** (Tab 1) of the Test Program Editor (for example J1, J2 etc.). Pin designators are assigned by connector type in the Connector Registry or on bench-top testers, the default pin designators of the installed adapters are assigned automatically. When specifying test points, the connector/adapter reference and the pin designators are separated by a dash (-). The connector-pin designations defer to point labels if they have been created and applied under the **Label Points** tab of the Test Program Editor.

However, test points can also be specified using system points. When specifying system points using CTA:

- System point numbers are zero-based. Therefore, the first test point in the system is 0 and system points are counted forward sequentially until the last test point, which is the total number of installed points -1.
  For example, when using CTA, system points in a CH2 with 800 test points would be numbered 0 - 799.
- System point designations must start with an explanation point (!).
- Only numeric characters (0 - 9) may be used.
- The specified number must be within the tester's test point range (from 0 to the total number of installed test points -1).

Example:

ctaResistance("!12", "!23"); // measure resistance between system points 12 and 23

4. Integer Length

In the case of an integer marked as “var” for a length, the requested length of the associated character list can be specified in the call. If the length of the character list is not enough for the character list being pasted back from the method call, the value will be updated with the needed size of the character list.

5. Functions

ctaAbort

- Parameter: <AbortResponseLength>
- Return Variable: <AbortResponseLength>
- Aborts while waiting for the Tester to return results (Equivalent to clicking the Abort button in EW).
- function ctaAbort(wConnectionID: word; var iAbortResponseLength: integer; cpAbortResponse: ctaPChar): word;
- Prerequisite: None

ctaAdvancedInstruction

- Parameters: <Point1><Point2><InstructionName><InstructionParameters> <ResultLength>
- Return Variables: <Value1> <Value2> <Results> <ResultLength>
  - <Results> returns the value for the desired Advanced Instruction.
  - <Value1> returns a double based on the advanced instruction being called
  - <Value2> returns a double based on the advanced instruction being called
- Returns the result value for the desired Advanced Instruction.
- function ctaAdvancedInstruction(wConnectionID: word; cpPoint1: ctaPChar; cpPoint2: ctaPChar; cpInstructionName: ctaPChar; cpInstructionParameters: ctaPChar; out fValue1: double; out fValue2: double; var iResultLength: integer; cpResult: ctaPChar; out wResults: word): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest,)

ctaCablePresent

- Parameters: None
- Return Variable: <CablePresent>
  - Returns 1 if connections are detected, else 0.
- Detects whether a cable is present
- function ctaCablePresent(wConnectionID: word; out wCablePresent: word): word;
- Prerequisite: None

ctaCapacitance

- Parameters: <Point1><Point2>
- Return Variables: <Status> <MeasuredValue>
  - <Status> 0 is equal, 1 is under range, 2 is over range
  - <MeasuredValue> returns the measured capacitance value
- Returns the measured capacitance between specified points.
- function ctaCapacitance(wConnectionID: word; cpPoint1: PChar; cpPoint2: PChar; out bStatus: byte; out fMeasuredValue: double): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaChangeOutputs

- Parameters: <Change Mask><Change Values> 32 bit values. ChangeMask=(1 Bit=AllowChange) ChangeValues=(Bit Value to Update)
- Return Variables: None
- Change the built-In tester digital outputs, bits (set to a logic 1 state) in dwChangeMask to the bit values specified in dwChangeBits.
- function ctaChgOutputs(wConnectionID: word; dwChangeMask, dwChangeBits: cardinal): word;
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaChildSingleTestByIndex

- Parameters: Parameters: <Child Test Index><Hipot(T/F)><Shorts Test(T/F)>
- Return Variable: <Results>
    - Returns the results of the child test as indicated by the passed in index
- Test the child test by its index within the currently loaded parent test group.
- function ctaChildSingleTestByIndex(wConnectionID: word; iChildTestIndex: integer; bDoHipot, bDoShortsTest: boolean; out wResults: word): word;
- Prerequisite: Parent or Child Test in memory (prior call to Learn or LoadTest)

ctaChildSingleTestByName

- • Parameters: <Child Test Name><Hipot(T/F)><Shorts Test(T/F)>
- • Return Variable: <Results>
    - Returns the results of the child test as indicated by the passed in name
- Test the child test by its name within the currently loaded parent test group.
- function ctaChildSingleTestByName(wConnectionID: word; cpChildTestName: ctaPChar; bDoHipot, bDoShortsTest: boolean; out wResults: word): word;
- Prerequisite: Parent or Child Test in memory (prior call to Learn or LoadTest)

ctaClearOutputs

- Parameter: <Clear Bits>
- Return Variables: None
- 32 Bit bit value. 1 bit = CLEAR selected output.
- function ctaClearOutputs(wConnectionID: word; cOutputsState: Cardinal): word;
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaComplexLearn

- Parameters: <Test Name><Components>
- Return Variables: None
- ComplexLearn for the attached DUT and saves to the Database if optional TestName is specified.
    - <Components> parameter is a byte value
        - For a Signature Testers the Components byte is the Or'd value as follows:
        - Resistors = 1, Capacitors = 2, Diodes = 4, TwistedPairs = 8
        - 1 + 2 = 3 - Learn Resistors and Capacitors
        - 2 + 4 = 6 - Learn Capacitors and Diodes
        - 1 + 2 + 4 + 8 = 15 - Learn Resistors, Capacitors, Diodes, and Twisted Pairs
        - For LPCS testers any non-zero values = learn all components
- function ctaComplexLearn(wConnectionID: word; cpTestName: PChar; bComponents: byte): word;
- Prerequisite: None

ctaConnect

- Parameters: <IPAddress:Port><User Name><Password>
    - ServerConnetionIPAndPort can be left blank. If it is blank, the DLL will use the values from the CirrisTesterAccessClient.ini and connect to the server.
- Return Variable: <ConnectionID>
    - Returns a ConnectionID that is used with any subsequent commands that access the server.
- Connect to CTA server. Log in specified user3.
- function ctaConnect(cpServerConnectionIPAndPort: ctaPChar; cpUserName: PChar; cpPassword: PChar; word; out wConnectionID: word): word;
- Prerequisites:
    - Must be called before any other commands that access the Server
    - The returned ConnectionID is used with any subsequent commands that access the server.
    - User must have Remote Access rights which can be configured through Easy-Wire Security Settings

ctaConnectionTest

- Parameters: <ServerConnectionIPAndPort><TimeoutInMilliseconds>
- Return Variables: None
- Test ability to access a Cirris Tester Access Server.
- function ctaConnectionTest(cpServerConnectionIPAndPort: ctaPChar; iTimeoutInMilliseconds: integer): word;
- Prerequisite: None

ctaCTL (Only supported by signature tester types)

- Parameters: <Command><Parameters><AutoStat(T/F)> <StatResponseLength>
- Return Variables: <CTLResult> <StatResponseLength>
    - <CTLResult>returns 0 is Passed, 1 is Failed, 2 is Unknown
- Allows the user to send CTL commands to the attached Signature tester.
- function ctaCTL(wConnectionID: word; cpCTLCommand, cpCTLParameters: ctaPChar; out wCTLResult: word; bAutoStat: boolean; var iStatResponseLength: integer; cpStatResponse: ctaPChar): word;
- Prerequisite: Has to be a Signature Tester.

ctaDeleteTest

- Parameter: <TestName>
- Return Variables: None
- Deletes the specified test program if it exists.
- function ctaDeleteTest(wConnectionID: word; cpTestName: PChar): word;
- Prerequisite: None

ctaDiode

- Parameters: <AnodePoint1><CathodePoint1>
- Return Variables: <VStatus> <ForwardVoltage> <RVStatus> <ReverseVoltage>
    - <VStatus> 0 is equal, 1 is under range, 2 is over range
    - <ForwardVoltage> returns the forward voltage drop value
    - <RVStatus> 0 is equal, 1 is under range, 2 is over range
    - <ReverseVoltage> returns the reverse voltage drop value
- Returns Forward Voltage Drop and Reverse Voltage Drop for a Diode.
- function ctaDiode(wConnectionID: word; cpAnode: PChar; cpCathode: PChar; out bFVStatus: byte; out fForwardVoltage: double; out bRVStatus: byte; out fReverseVoltage: double): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaDisconnect

- Parameter: <TerminateServer: True/False>
- Return Variables: None
- Log out user / disconnect from server - must be called before exiting client app.
- function ctaDisconnect(wConnectionID: word; bTerminateServer: boolean): word;
- Prerequisite: Prior call to ctaConnect

ctaDisplayMessage

- Parameters: <Caption> <Message> <ResponseRequired(T/F)> <TimeOutInSeconds> <ResponseLength>
- Return Variable: <ResponseLength>
- Displays message on the server, with optional or required user response. TimeOutInSeconds=0 will cause the dialog on the Server to wait indefinitely. The dialog on the server side will close after TimeOutInSeconds even if ResponseRequired is True and the user has not entered any response.
- function ctaDisplayMessage(wConnectionID: word; cpCaption: ctaPChar; cpMessage: ctaPChar; bResponseRequired: boolean; iTimeout: integer; var iResponseLength: integer; cpResponse: ctaPChar): word;
- Prerequisite: None

ctaEnableJSONResults

- Parameter: <Enable(T/F)>
- Return Variables: None
- Test errors will be returned as a JSON string when Enable=True, and normal text as seen in the EasyWire Test window when Enable=False. This setting is stored in the CirrisDataAccessServer.ini and will maintain it's state.
- function ctaEnableJSONResults(bEnable: boolean): word;
- Prerequisite: None
- See examples of JSON stings in Appendix (*page 35*).

ctaEndTestRun

- Parameters: None
- Return Variables: None
- Finalizes the Test Run for the Active Test used which includes Incrementing the run count and making sure all results have been saved.
- function ctaEndTestRun(wConnectionID: word): word;
- Prerequisite: Call ctaStartTestRun first.

ctaFastSingleTest

- Parameters: <Hipot(T/F)><Shorts Test(T/F)>
- Return Variable: <Results>
    - Returns the results of the test
- Fast Test the currently loaded test.
- function ctaFastSingleTest(wConnectionID: word; bDoHipot, bDoShortsTest: boolean; out wResults: word): word;
- Prerequisite: Must load a test before using this command.

ctaGetCategoryList

- Parameters: None
- Return Variables: None
- Returns a list of all categories with each category in the list tagged with whether it is used for Test Program, Connector Type, or used for both. Formatted in a JSON array with the following fields: CategoryName, CategoryType
- function ctaGetCategoryList(wConnectionID: word; var iCategoryListLength: integer; cpCategoryList: ctaPChar): word;
- Prerequisite: Must be connected to the server (ctaConnect)

ctaGetClientInformation

- Parameter: <InformationLength>
- Return Variable: <InformationLength>
- Returns Information about the Client dll.
- function ctaGetClientInformation(wConnectionID: word; var iInformationLength: integer; cpSystemInformation: ctaPChar): word;
- Prerequisite: None

ctaGetConnector

- Parameter: <ConnectorName> <ConnectorLength>
- Return Variable: <ConnectorLength>
- Returns the Text version of the connector specified by ConnectorName.
- function ctaGetConnector(wConnectionID: word; cpConnectorName: ctaPChar; var iConnectorLength: integer; cpConnectorText: ctaPChar): word; cdecl;
- Prerequisite: None

ctaGetHVSafetyStatus

- Parameters: None
- Return Variable: <HVSafetyStatus>
  - Returns 0=Unknown, 1=InterlockNotSupported, 2=HVEnabled, 3=HVDisabled, 4=InterlockInTransition
- Returns High Voltage Safety Status
- function ctaGetHVSafetyStatus(wConnectionID: word; out bHVSafetyStatus: byte): word;
- Prerequisite: None

ctaGetInput

- Parameter: <InputIndex>
- Return Variable: <InputState>
  - Returns the logic state of the Input specified by InputIndex
- Returns the logic state of the Input specified by InputIndex. InputIndex (0 to N-1, where N = Number of Inputs)
- function ctaGetInput(wConnectionID: word; iInputIndex: integer; out bInputState: boolean): word;
- Prerequisite: IO Available on attached tester
- Note: CTL references I/O from 1 to N. CTA references I/O from 0 to N-1 (like Easy-Wire).

ctaGetInputs

- Parameters: None
- Return Variable: <InputState>
  - Returns the current logic state of the available built-in tester inputs.
- function ctaGetInput(wConnectionID: word; iInputIndex: integer; out bInputState: boolean): word;
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaGetIOCount

- Parameters: None
- Return Variables: <InputCount> <OutputCount>
  - <InputCount> returns the count of available inputs
  - <OutputCount> returns the count of available outputs
- Returns the number of Inputs and Outputs available for the attached Tester.
- function ctaGetIOCount(wConnectionID: word; out iInputCount: integer; out iOutputCount: integer): word;
- Prerequisite: None - If I/O is not available on the attached tester, zero will be returned for both iInputCount and iOutputCount.
- Notes:
  - GetInput valid InputIndex Range is 0 to iInputCount - 1
  - SetOuput valid OutputIndex Range is 0 to iOutputCount – 1

ctaGetLastError

- Parameter: <ErrorBufferLength>
- Return Variable: <ErrorBufferLength>
- Get Last Errors as a string when a Command Failed.
- function ctaGetLastError(wConnectionID: word; var iErrorBufferLength: integer; cpErrorBuffer: PChar): word;
- Prerequisite: None

ctaGetMeasuredValues

- Parameter: <ValuesLength>
- Return Variable: <ValuesLength>
- Returns any measured values that were stored during the last test.
- function ctaGetMeasuredValues(wConnectionID: word; var iValuesLength: integer; cpValues: ctaPChar): word;
- Prerequisite: A prior call to ctaSingleTest

ctaGetParameter

- Parameters: <Parameter> <ParameterLength>
- Return Variable: <ParameterLength>
- Get the specified parameter from the current test.
- function ctaGetParameter(wConnectionID: word; cpParameter: ctaPChar; var iParameterLength: integer; cpParameterText: ctaPChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaGetParameters

- Parameter: <ParameterLength>
- Return Variable: <ParameterLength>
- Get the Test Parameters settings for the current test.
- function ctaGetParameters(wConnectionID: word; var iParameterLength: integer; cpParameterText: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaGetServerInformation

- Parameter: <InformationLength>
- Return Variable: <InformationLength>
- Returns server information for the currently connected Tester Access Server.
- function ctaGetServerInformation(wConnectionID: word; var iInformationLength: integer; cpSystemInformation: PChar): word;
- Prerequisite: None

ctaGetTemperatureAndHumidity

- Parameters: None
- Return Variables: <Temperature> <Humidity>
  - <Temperature> returns the temperature reading from the sensor on common hardware
  - <Humidity> returns the humidity reading from the sensor on common hardware
- Returns the temperature and humidity from sensor on common hardware.
- function ctaGetTemperatureAndHumidity(wConnectionID: word; out iTemperature, iHumidity: int32): word;
- Prerequisite: Common hardware attached.
- Note: If attached to a tester without the ability to send temp and humidity, an error will be thrown: 'Get temperature and humidity not supported'

ctaGetTest

- Parameters: <TestName> <TestLength>
- Return Variable: <TestLength>
- Returns the exported Text version of the test specified by TestName. TestName can be blank, to return the currently loaded test.
- function ctaGetTest(wConnectionID: word; cpTestName: PChar; var iTestLength: integer; cpTestText: PChar): word;
- Prerequisite: None
- Notes:
  - This command returns the text of the test in the same format as found in an Exported test from Easy-Wire.
  - When successfully executed, the specified test will also be "Active", or current test in memory.
  - If the command is executed without specifying a test name, it will return the test currently in memory. If the test in memory includes unsaved changes, it will differ from the test with the same name in the Easy-Wire database.
  - If the test name is specified, the test from the Easy-Wire database will be returned. This is true even if the same test is loaded in current memory in Easy-Wire.

ctaGetTestDefaults

- Parameter: <TestDefaultsLength>
- Return Variable: <TestDefaultsLength>
- Returns the current Test's settings and values.
- function ctaGetTestDefaults(wConnectionID: word; var iTestDefaultsLength: integer; cpTestDefaults: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaGetTesterParameters

- Parameter: <TesterParametersLength>
- Return Variable: <TesterParametersLength>
- Returns a JSON Parameter List of the parameters supported by the current tester.
- function ctaGetTesterParameters(wConnectionID: word; var iTesterParametersLength: integer; cpTesterParameters: PChar): word;
- Prerequisite: None
- See example of JSON return string in Appendix (*page 35*).

ctaGetTestErrors

- Parameter: <ErrorLength>
- Return Variable: <ErrorLength>
- Returns any errors that occurred during the last test.
- function ctaGetTestErrors(wConnectionID: word; var iErrorLength: integer; cpErrors: PChar): word;
- Prerequisite: None
- Notes:
    - See example of returned string when ctaEnableJSONResults has been called with False in Appendix (*page 35)*.
    - See example of returned JSON string when ctaEnableJSONResults has been called with True in Appendix (*page 36*).

ctaGetTestList

- Parameter: <TestListLength>
- Return Variable: <TestListLength>
- Returns the test program list, revision information and last motified date of the tests available for the attached tester.
- function ctaGetTestList(wConnectionID: word; var iTestListLength: integer; cpTestList: PChar): word;
- Prerequisite: None

ctaGetTestListByCategory

- Parameters: <Category> <TestListLength>
- Return Variable: <TestListLength>
- Returns a list of tests filtered by the specified parameter <Category>
- function ctaGetTestListByCategory(wConnectionID: word; cpCategory: ctaPChar; var iTestListLength: integer; cpTestList: ctaPChar): word;
- Prerequisite: None

ctaHelp

- Parameters: <Command> <HelpLength>
- Return Variable: <HelpLength>
- Returns Help on the specified command (can be blank), or entire command list if command is an empty string.
- function ctaHelp(wConnectionID: word; cpCommand: PChar; var iHelpLength: integer; cpHelp: PChar): word;
- Prerequisite: None

ctaHipotNet (Not supported by signature tester types)

- Parameter: <Point in Net1>
- Return Variable: <Results>
    - Returns a True or a False
- Performs a HiPot Test on the Net of the passed in Point.
- function ctaHipotNet(wConnectionID: word; cpPoint: PChar; out bResults: boolean): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaHipotTest

- Parameters: None
- Return Variable: <Results>
    - Returns a True or a False
- Performs a HiPot test on the currently loaded test program.
- function ctaHipotTest(wConnectionID: word; out bResults: boolean): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaLearn

- Parameter: <TestName>
- Return Variables: None
- Learns the current DUT and saves to the Database if optional TestName is specified.
- function ctaLearn(wConnectionID: word; cpTestName: PChar): word;
- Prerequisite: None

ctaLoadedTestName

- Parameter: <TestListLength>
- Return Variable: <TestListLength>
- Returns the name of the currently loaded test program. If a parent test has been loaded, it provides a list of the test names and their index within the test group.
- function ctaLoadedTestName(wConnectionID: word; var iTestListLength: integer; cpTestList: ctaPChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaLoadLuaScript

- Parameter: <ScriptFilePath>
- Return Variables: None
- Loads the specified script to the active test program. The server must have access to this file already.
- function ctaLoadLuaScript(wConnectionID: word; cpScriptFilePath: ctaPChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)
- Note: for ScriptFilePath, provide the path and full filename, including extension. ie. C:\Scripts\Container\MyScript.lua

ctaLoadTest

- Parameter: <TestName>
- Return Variables: None
- Loads the specified test into memory.
- function ctaLoadTest(wConnectionID: word; cpTestName: PChar): word;
- Prerequisite: None

ctaLuaComponent

- Parameters: <ComponentName><Parameters><DetailsLength>
- Return Variables: <Result><DetailsLength>
  - <Result> returns Lua command result. See Lua manual for details.
- Executes specified Lua Component and returns Result and Details.
- function ctaLuaComponent(wConnectionID: word; cpName: ctaPChar; cpParameters: ctaPChar; out bResult: byte; var iDetailsLength: integer; cpDetails: ctaPChar): word;
- Prerequisite: Test must be loaded and have a component script attached from the database or a prior call to LoadLuaScript or PutLuaScript.

ctaProbe

- Parameter: <ProbedPointLength>
- Return Variable: <ProbedPointLength>
- Returns Probed Point(s).
- function ctaProbe(wConnectionID: word; var iProbedPointLength: integer; cpProbedPoint: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaProbeResistance

- Parameter: <ProbedPointsAndValuesLength>
- Return Variable: <ProbedPointsAndValuesLength>
- Returns Probed Point(s) with Resistance Value(s) measured to Probe.
- function ctaProbeResistance(wConnectionID: word; var iProbedPointsAndValuesLength: integer; cpProbedPointsAndValues: ctaPChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaPutLuaScript

- Parameters: <ScriptFileName><ScriptText>
- Return Variables: None
- Saves a script file to the server computer using the provided name and text. The script name can be prefaced with a full path as well.
- function ctaPutLuaScript(wConnectionID: word; cpScriptFileName: ctaPChar; cpScriptText: ctaPChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)
- Note: for ScriptFileName, use the full filename and extension. Ie. MyScript.lua

ctaPutTest

- Parameters: <TestName><TestText><SetAsActiveTest(T/F)>
- Return Variables: None
- Saves specified text based version of the test to the TestName in the Easy-Wire database and optionally loads it into the Active Test for immediate use. If TestName is an empty string the test will be set as the active test without being saved to the database.
- function ctaPutTest(wConnectionID: word; cpTestName: PChar; cpTest: PChar; bSetAsActiveTest: boolean): word;
- Prerequisite: None
- Notes:
    - This uses the same format for the text of the test as Import does in Easy-Wire.
    - Signature Testers can use wirelist programs (*.WIR) or text programs (*.TXT).
    - Easy-Wire Testers (CR, CH2, etc.) require text programs (*.TXT).
    - The Tester Type that the file was created for must match the Tester that is attached to the server.

ctaPutTestAsChild

- Parameters: None
- Return Variable: <ChildTestText>
  - Returns integer of the index of the Child Test
- Adds and activates the sent test as a child test and returns the index of the child test in memory.
- function ctaPutTestAsChild(wConnectionID: word; cpTest: ctaPChar; out iChildTestIndex: integer): word;
- Prerequisites: None
- Notes:
  - The test sent is not saved to the database unless ctaSaveTest is called while it is the active test.
  - This uses the same format for the text of the test as Import does in Easy-Wire.
  - Signature Testers can use wirelist programs (*.WIR) or text programs (*.TXT).
  - Easy-Wire Testers (CR, CH2, etc.) require text programs (*.TXT).
  - The Tester Type that the file was created for must match the Tester that is attached to the server.

ctaResetToDefaultTest

- Parameters: None
- Return Variables: None
- Unloads all tests in memory and loads the default test.
- function ctaResetToDefaultTest(wConnectionID: word): word;
- Prerequisites: None

ctaResistance

- Parameters: <Point1><Point2>
  - <Point1> is the Source Point
  - <Point2> is the Sink Point
- Return Variable: <Status>
  - Returns 0 is equal, 1 is under range, 2 is over range
- Returns the resistance measured between the specified points.
- function ctaResistance(wConnectionID: word; cpPoint1: PChar; cpPoint2: PChar; out bStatus: byte; out fMeasuredValue: double): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest, or PutTest)

ctaResistance4W

- Parameters: <Point1><Point2>
  - <Point1> is the Source 4 Wire Point
  - <Point2> is the Sink 4 Wire Point
- Return Variables: <Status><MeasuredValue>
  - <Status> returns 0 is equal, 1 is under range, 2 is over range
  - <MeasuredValue> returns the measured resistance value.
- Returns the 4-wire resistance measured between the specified points, which must be wired as 4W points.
- function ctaResistance4W(wConnectionID: word; cpPoint1: PChar; cpPoint2: PChar; out bStatus: byte; out fMeasuredValue: double): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest). Points passed into the function are defined as 4W point in the test.

ctaResistance4WEx

- Parameters: <SourcePoint><SourceMatePoint><SinkPoint><SinkMatePoint><MaxCurrent>
- Return Variables: <Status><MeasuredValue>
  - <Status> returns 0 is equal, 1 is under range, 2 is over range
  - <MeasuredValue> returns the measured resistance value.
- Returns 4-Wire resistance measured between the specified points using a current limit. A Max Current of “0” uses the attached tester defaults.
- function ctaResistance4WEx(wConnectionID: word; cpSourcePoint: ctaPChar; cpSourceMatePoint: ctaPChar; cpSinkPoint: ctaPChar; cpSinkMatePoint: ctaPChar; fMaxCurrent: double; out bStatus: byte; out fMeasuredValue: double): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest). Points passed into the function are defined as 4W point in the test.

ctaSaveTest

- Parameter: <TestName>
- Return Variables: None
- Saves the test currently in memory to TestName.
- function ctaSaveTest(wConnectionID: word; cpTestName: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaSaveTestResults

- Parameter: <SerialNumber>
- Return Variables: None
- Increments the test counts and saves the results of the last test.
- function ctaSaveTestResults(wConnectionID: word; cpSerialNumber: ctaPChar): word;
- Prerequisite: call ctaStartTestRun and perform at least a ctaSingleTest so there are results to save.

Warning *- The saved, overall Pass / Fail status of a test is determined by the results of programmed operations only. A Pass result does NOT indicate that a complete and thorough test was performed.*

ctaSelfTest

- Parameters: None
- Return Variable: <SelfTestPassed>
    - Returns True=Pass or False=Fail
- Performs a SelfTest on the attached tester.
- function ctaSelfTest(wConnectionID: word; out bSelfTestPassed: boolean): word;
- Prerequisite: None

ctaServerReady

- Parameters: None
- Return Variable: <ServerReady>
    - Returns True=Ready or False=NotReady.
- Checks if the CTA server is ready to receive commands.
- function ctaServerReady(wConnectionID: word; out bServerReady: boolean): word;
- Prerequisite: ctaConnect - need to have an active connection

ctaSetChildTestActivebyIndex

- Parameters: <ChildIndex>
- Return Variables: None
- This sets the specified test active and runs Initialize Test to make it ready to run SingleTest or FastSingleTest, etc.
- function ctaSetChildTestActivebyIndex(wConnectionID: word; iChildIndex: integer): word;
- Prerequisite: Parent Child test in memory

ctaSetChildTestActivebyName

- Parameters: <ChildTestName>
- Return Variables: None
- This sets the specified test active and runs Initialize Test to make it ready to run SingleTest or FastSingleTest, etc.
- function ctaSetChildTestActivebyName(wConnectionID: word; sChildTestName: string): word;
- Prerequisite: Parent Child test in memory

ctaSetOutput

- Parameters: <OutputIndex> <OutputState (0 or 1)>
- Return Variables: None
- Sets the logic state of OutputIndex to OutputState. OutputIndex (0 to N-1, where N = Number of Outputs)
- function ctaSetOutput(wConnectionID: word; iOutputIndex: integer; bOutputState: boolean): word;
- Prerequisite: IO Available on attached tester
- Note: CTA references I/O from 0 to N-1 (like Easy-Wire).

ctaSetOutputs

- Parameters: <SetMask>
- Return Variables: None
- Sets the built-in tester outputs (set to a logic 1 state) for the bits specified (set to a logic 1 state) in dwSetMask.
- function ctaSetOutputs(wConnectionID: word; dwSetMask: cardinal): word;
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaSetParameters

- Parameters: <TestParameters>
- Return Variables: None
- Sets the current test''s parameter values to those specified.
- function ctaSetParameters(wConnectionID: word; cpParameters: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaSetServerConnection

- Parameters: <IPAddress:PortNumber>
- Return Variables: None
- Specifies the IP Address CirrisTesterServer and the port number that the server is listening on. Example: 176.14.25.6:5660
- function ctaSetServerConnection(cpServerConnectionIPAndPort: ctaPChar): word;
- Prerequiste: Must be Disconnected from server (prior to Connect or after calling Disconnect).

ctaSetTestDefaults

- Parameters: <TestDefaults>
- Return Variables: None
- Sets the current Test's settings to values specified.
- function ctaSetTestDefaults(wConnectionID: word; cpTestDefaults: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaShowServerMonitor

- Parameters: <Show(T/F)>
- Return Variables: None
- Causes the CTA Server to show/hide a status window. This setting is storred in the CirrisTesterAccessServer.ini and will maintain state.
- function ctaShowServerMonitor(bShow: boolean): word;
- Prerequisite: None

ctaSingleTest

- Parameters: <Hipot(T/F)><Shorts Test(T/F)>
- Return Variable: <Results>
    - Returns the result of the selected test that was executed.
- Test the currently loaded test.
- function ctaSingleTest(wConnectionID: word; bDoHipot, bDoShortsTest: boolean; out wResults: word): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaStartTestRun

- Parameters: <TestName><LotID>
- Return Variables: None
- Starts a Test Run with the provided test name so Results can be saved for reporting.
- function ctaStartTestRun(wConnectionID: word; cpTestName: ctaPChar; cpLotID: ctaPChar): word;
- Prerequisite: None

ctaSwitchToEasyWire

- Parameters: <Mode>
- Return Variables: None
- Switches from CTA Server mode over to Easy-Wire in the specified Mode, defined by an enumeration. The available modes are:
    - 0 = Initial
    - 1 = Edit
    - 2 = Test
    - 3 = Connector Registry
    - 4 = Utilities
    - 5 = Revision History
    - 6 = Test Program Report
    - 7 = Assembly Wiring Report
    - 8 = Print Results
- function ctaSwitchToEasyWire(wConnectionID: word; wMode: word): word;
- Prerequisite: ctaConnect - Need to have an active connection

ctaTwistedPair

- Parameters: <Point1><Point2>
- Return Variable: <Twisted>
    - Returns true or false
- Returns True if the pair is twisted, False if they are not.
- function ctaTwistedPair(wConnectionID: word; cpPoint1: PChar; cpPoint2: PChar; out bTwisted: boolean): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

This page intentionally left blank.

6. Return Values

All functions return a word (UInt16) value as a result.

0=Success

Any other value indicates an error. An error string associated with the last error can be retrieved by calling: **ctaGetLastError**

6.1 Definitions

tacrSuccess = 0

tacrBufferTooSmall = 1

tacrUnknownError = 2

tacrSendCommandToServerFailed = 3

tacrServerReturnedInvalidData = 4

tacrConnectionFailed = 5

tacrUserNameRequired = 6

tacrConnectFailed = 7

tacrValidFileNameRequired = 8

tacrValidImportTextRequired = 9

tacrLoadTestFailed = 10

tacrInvalidParameters = 11

tacrValidTestNameRequired = 12

tacrInvalidServerIPAndPort = 13

tacrDisconnectToChangeServerAddress = 14

tacrCommandAborted = 15

tacrNotConnectedToServer = 16

tacrInvalidConnectionID = 17

tacrInvalidUserName = 18

tacrInvalidPassword = 19

tacrRemoteUserNotSet = 20

tacrSecurityUnassigned = 21

tacrVersionMismatch = 22

6.2 Test Result Values

Test functions also return result values.

ctaSingleTest, ctaChildSingleTestByIndex, ctaChildSingleTestByName:

ctatrIncomplet = 0,

ctatrFailed = 1,

ctatrPassed = 2

ctaAdvancedInstruction:

ctaaiPassed = 0,

ctaaiFailed = 1,

ctaaiBusy = 2,

ctaaiNotDone = 3,

ctaaiInternalError = 4

6.3 Test Measurement Values

The Measurement commands, such as ctaResistance, also return status bytes for the measurements taken.

These status bytes are used to identify offscale (out of range) values. for example tmsUnderRange means the measured result is below the minimum measurement range of the attached tester for that component measurement.

TesterMeasurementStatus:

tmsEqual = 0,

tmsUnderRange = 1,

tmsOverRange = 2,

tmsVoltageOff = 3,

tmsNoMeasurement = 4

7. Help / Support

Additional Documentation

- Tester User Manuals are available for download from each tester's product page on the Cirris web site.
- Contextual Help is available from each window in the Easy-Wire application.

Support

- In the United States contact Technical Support at TechSupport@cirris.com or by telephone at 1-801-973-4600.
- Outside the United States, visit the Contact page of the Cirris web site at cirris.com to find the nearest sales and support office.
- Visit www.cirris.com/learning-center to access articles on Cirris products and other testing subjects.

8. Appendix

8.1 Examples of Return Strings

8.1.1 ctaGetTesterParameters JSON String

The example below show a JSON return string for the **ctaGetTesterParameters** function:


```
{
“ParameterList”:
   [
      {“Name”:”WIRERES”, “ID”:”17000”, “Min”:”1.00E-01”, “Max”:”100”, “Units”:”Ohms”},
      {“Name”:”RESISTORRES”, “ID”:”17001”, “Min”:”1.00E-01”, “Max”:”1.00E+06”, “Units”:”Ohms”},
      {“Name”:”RESISTORTOL”, “ID”:”17002”, “Min”:”2”, “Max”:”100”, “Units”:”Percent”},
      {“Name”:”SHORTSRES”, “ID”:”17003”, “Min”:”1”, “Max”:”1.00E+06”, “Units”:”Ohms”},
      {“Name”:”FORWARDV”, “ID”:”17004”, “Min”:”1.00E-02”, “Max”:”6”, “Units”:”Volts”},
      {“Name”:”REVERSEV”, “ID”:”17005”, “Min”:”1.00E-02”, “Max”:”6”, “Units”:”Volts”},
      {“Name”:”FORWARDTOL”, “ID”:”17006”, “Min”:”5”, “Max”:”100”, “Units”:”Percent”},
      {“Name”:”REVERSETOL”, “ID”:”17007”, “Min”:”5”, “Max”:”100”, “Units”:”Percent”},
      {“Name”:”4W_WIRERES”, “ID”:”17022”, “Min”:”5.00E-03”, “Max”:”100”, “Units”:”Ohms”},
      {“Name”:”4W_RESISTORRES”, “ID”:”17023”, “Min”:”5.00E-03”, “Max”:”1.00E+06”, “Units”:”Ohms”},
      {“Name”:”4W_RESISTORTOL”, “ID”:”17024”, “Min”:”1”, “Max”:”100”, “Units”:”Percent”},
      {“Name”:”CAPACITANCE”, “ID”:”17025”, “Min”:”1.00E-10”, “Max”:”1.00E-03”, “Units”:”Farads”},
      {“Name”:”CAPTOLERANCE”, “ID”:”17026”, “Min”:”10”, “Max”:”99”, “Units”:”Percent”},
      {“Name”:”4W_WIREMIN”, “ID”:”17027”, “Min”:”0”, “Max”:”100”, “Units”:”Ohms”},
      {“Name”:”DELAYRESISTANCERES”, “ID”:”17028”, “Min”:”1.00E-01”, “Max”:”5.00E+03”, “Units”:”Ohms”},
      {“Name”:”DELAYRESISTANCETOL”, “ID”:”17029”, “Min”:”1”, “Max”:”75”, “Units”:”Percent”},
      {“Name”:”COMMENTDELAY”, “ID”:”17030”, “Min”:”1”, “Max”:”3.60E+03”, “Units”:”Seconds”},
      {“Name”:”4W_WIRERESTARE”, “ID”:”17034”, “Min”:”0”, “Max”:”10”, “Units”:”Ohms”},
      {“Name”:”WIRERESTARE”, “ID”:”17035”, “Min”:”0”, “Max”:”5”, “Units”:”Ohms”},
      {“Name”:”COMPONENT_RESISTANCE”, “ID”:”17041”, “Min”:”0”, “Max”:”1.00E+07”, “Units”:”Ohms”},
      {“Name”:”INDUCTANCE”, “ID”:”17065”, “Min”:”1.00E-09”, “Max”:”1”, “Units”:”Farads”},
      {“Name”:”INDUCTANCE_TOL”, “ID”:”17066”, “Min”:”5”, “Max”:”100”, “Units”:”Percent”}
   ]
}
```


8.1.2 ctaGetTestErrors JSON String False

The example below shows an example of the return sting when **ctaEnableJSONResults** function has been set to **False**. The same error with the JSON string set to **True** is shown in Section 7.1.3 below (*page 36*).

Net Net1: OPEN J1B001 to J1B003
Net NC: SHORT J1B007 to J1B009

8.1.3 ctaGetTestErrors JSON String True

The example below shows an example of return sting when **ctaEnableJSONResults** function has been set to **True**. The same error with the JSON string set to False is shown in Section 7.1.2 above (*page 35*).


```
{
  “ErrorList”:
  [
    {“ErrorID”:”155”,”Point1”:”J1B001”,”Point2”:”J1B003”,”ValueStatus1”:”0”,”Value1”:”0”,”ValueStatus2”:”0”,
“Value2”:”0”,”Message”:”Net Net1: OPEN J1B001 to J1B003”},
    {“ErrorID”:”156”,”Point1”:”J1B007”,”Point2”:”J1B009”,”ValueStatus1”:”0”,”Value1”:”0”,”ValueStatus2”:”0”,
“Value2”:”0”,”Message”:”Net NC: SHORT J1B007 to J1B009”}
  ]
}
```

This page intentionally left blank.

Cirris Tester Access (CTA) User Manual
Version 2024.4.2

401 North 5600 West
Salt Lake City, Utah 84116
USA
www.cirris.com

CIRRIS®
A KOMAX COMPANY