Cirris Tester Access (CTA)

User Manual

CIRRIS®

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

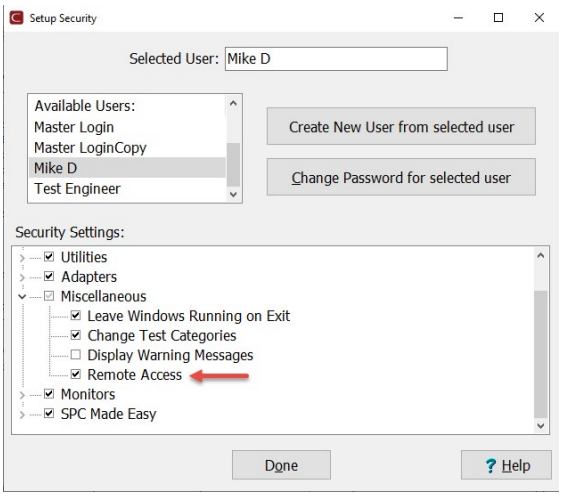
- **CTA Client configuration file (CirrisTesterAccessClient.ini)** - The key entry specifies the IP Address of the Server and the Port on which the Server is Listening: CTAClientConnectAddress=tcp://127.0.0.1:55555
- **CTA Server configuration file (CirrisTesterAccessServer.ini)** - The key entry specifies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

2. Getting Started

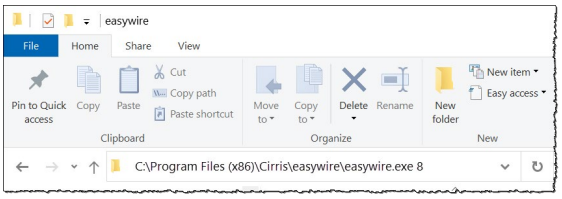
2.1 CTA Server & Client

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.

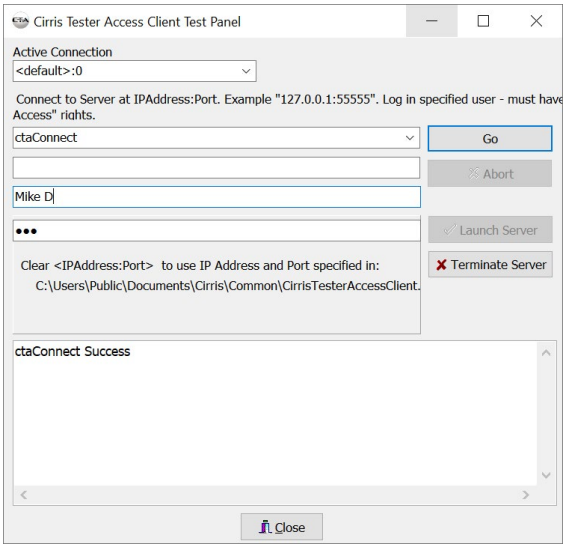


2. After the Easy-Wire software has been opened at least once with the tester attached to the Server PC, the CTA Server can be started from the **Cirris Tester Access Server** shortcut on the desktop or by opening Easy-Wire in the server mode by adding the command line parameter “8.”



3. 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.

4. To ensure the CTA Server and Client are installed and working properly, in the Test Panel, select **ctaConnect** from the **Function** drop down list.
   - Clear the **<IPAddress:Port>** field (this can be used later to specify a different IP address and Port for a Remote CTA Server).
   - 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 (see step 2 above).
5. When finished, always use the command **ctaDisconnect** before exiting the Test Panel.



2.2 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:

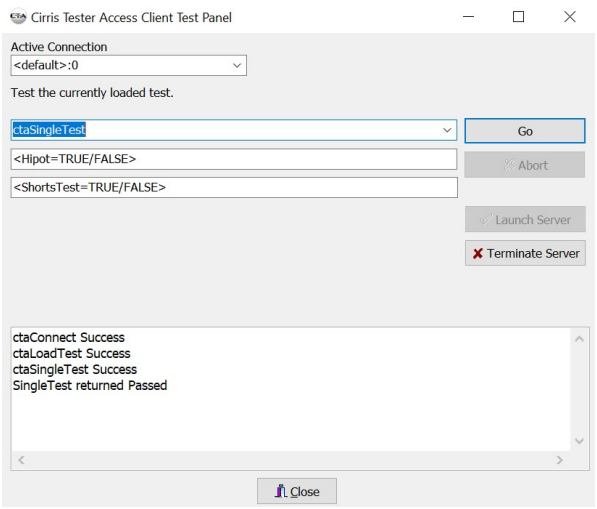
1. Copy the contents of the CTA folder (C:\Program Files (x86)\Cirris\CTA\) to the client computer
2. Place the **CirrisTesterAccessClient.ini** file in the directory: C:\Users\Public\Documents\Cirris\Common\
3. Edit the **CTAClientConnectAddress** in the **CirrisTesterAccessClient.ini** file to match 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.

2.3 CTA Client Test Panel

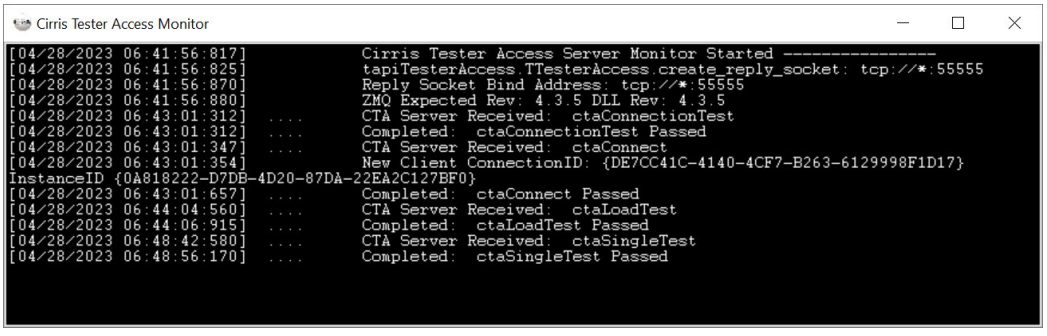
The **CTA Client Test Panel**, mentioned in Section 2.1, can be used to experiment with functions individually, step-by-step. Functions are selected from the drop-down list and the interface will prompt the user to enter the variables required for the selected function. Select **Go** or press Enter to execute the selected function. The response will be displayed in the status pane.

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



Tip: The function **ctaShowServerMonitor** (*page 19*) 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.



2.4 Examples

As shown in the following, simple example, three key steps are required in a custom application for a typical session, including:

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 9* and *Return Values* *page 21* for descriptions)
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

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

```

3. Specifying Test Points

Many 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. Functions

ctaAbort

- Parameters: <ConnectionID><out AbortResponse>
- 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: <ConnectionID><Point1><Point2><InstructionName><InstructionParameters>
- 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; cdecl;
- Prerequisite: Test in memory (prior call to Learn or LoadTest,)

ctaCablePresent

- Parameters: <ConnectionID><out CablePresent>
- Returns True if connections are detected, else False.
- function ctaCablePresent(wConnectionID: word; out wCablePresent: word): word;
- Prerequisite: None

ctaChangeOutputs

- Parameters: <ConnectionID><ChangeMask><ChangeBits>
- function ctaChgOutputs(wConnectionID: word; dwChangeMask, dwChangeBits: cardinal): word
- Change the built-In tester digital outputs, bits (set to a logic 1 state) in dwChangeMask to the bit values specified in dwChangeBits.
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaCapacitance

- Parameters: <ConnectionID><Point1><Point2>
- 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)

ctaChildSingleTestByIndex

- Parameters: <ConnectionID><ChildTestIndex><Hipot: True/False><Shorts Test: True/False><out Results>
- 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: <ConnectionID><ChildTestName><Hipot: True/False><Shorts Test: True/False><out Results>
- 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)

ctaClrOuputs

- Parameters: <ConnectionID><ClearMask>
- function ctaClrOutputs(wConnectionID: word; dwClearMask: cardinal): word;
- Clear the testers built in digital outputs (set to logic 0 state) for the bits set high (logic 1 state) in dwClearMask
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaComplexLearn

- Parameters: <ConnectionID><TestName><Components>
- ComplexLearn for the attached DUT and saves to the Database if optional TestName is specified.
    - <Components> parameter is a byte value
        - For 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 Pair
            - This uses the same format for the text of the test as Import does in Easy-Wire.
        - For LPCS testers any non-zero values = learn all components
- function ctaComplexLearn(wConnectionID: word; cpTestName: PChar; bComponents: byte): word
- Prequisite: None

ctaConnect

- Parameters: <ServerConnectionIPAndPort><User Name><Password><out ConnectionID>
    - ServerConnetionIPAndPort can be left blank. If it is blank, the DLL will use the values from the CirrisTesterAccessClient.ini and connect to 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>
- Test ability to access a Cirris Tester Access Server.
- function ctaConnectionTest(cpServerConnectionIPAndPort: ctaPChar; iTimeoutInMilliseconds: integer): word;
- Prerequisite: None

ctaDeleteTest

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

ctaDiode

- Parameters: <ConnectionID><AnodePoint><CathodePoint>
- Returns Forward Voltage Drop and Reverse Voltage Drop.
- 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

- Parameters:<ConnectionID><TerminateServer: True/False>
- 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: <ConnectionID><Caption> <Message> <ResponseRequired> <TimeOutInSeconds>
- 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

- Parameters: <ConnectionID><Enable>
- 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 its state.
- function ctaEnableJSONResults(bEnable: boolean): word;
- Prerequisite: None

ctaEndTestRun

- Parameters: <ConnectionID>
- For use with Easy-Wire reporting - 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.

ctaGetClientInformation

- Parameters:<ConnectionID><out SystemInformation>
- Returns Information about the Client dll.
- function ctaGetClientInformation(wConnectionID: word; var iInformationLength: integer; cpSystemInformation: ctaPChar): word;
- Prerequisite None

ctaGetInput

- Parameters: <ConnectionID><InputIndex><out InputState>
- 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: <ConnectionID>
- Returns the current logic state of the available built-in tester inputs.
- Prerequisites: IO Available on attached tester, and tester supports bulk I/O changes.

ctaGetIOCount

- Parameters: <ConnectionID><out InputCount><out OutputCount>
- 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.
- Note:
    - GetInput valid InputIndex Range is 0 to iInputCount - 1
    - SetOuput valid OutputIndex Range is 0 to iOutputCount - 1

ctaGetLastError

- Parameters: <ConnectionID><out ErrorBuffer>
- Get Last Errors as a string when a Command Failed.
- function ctaGetLastError(wConnectionID: word; var iErrorBufferLength: integer; cpErrorBuffer: PChar): word;
- Prerequisite: None

ctaGetMeasuredValues

- Parameters: <ConnectionID><out MeasuredValues>
- 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

ctaGetParameters

- Parameters: <ConnectionID><out ParameterText>
- 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

- Parameters: <ConnectionID><out SystemInformation>
- Returns server information for the currently connected Tester Access Server.
- function ctaGetServerInformation(wConnectionID: word; var iInformationLength: integer; cpSystemInformation: PChar): word;
- Prerequisite: None

ctaGetTest

- Parameters: <ConnectionID><TestName> <out TestText>
- 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 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.

ctaGetTestDefaults

- Parameters: <ConnectionID><out TestDefaults>
- 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

- Parameters: <ConnectionID><out TesterParameters>
- 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
- Example of JSON return string:


```
{
“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”}
]
}

ctaGetTestErrors

- Parameters: <ConnectionID><Errors>
- Returns any errors that occurred during the last test.
- function ctaGetTestErrors(wConnectionID: word; var iErrorLength: integer; cpErrors: PChar): word;
- Prerequisite: None
- Example of return string when ctaEnableJSONResults has been called with False :

  Net Net1: OPEN J1B001 to J1B003

  Net NC: SHORT J1B007 to J1B009
- Example of returned JSON string when ctaEnableJSONResults has been called with True:

{
“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”}
]
}

ctaGetTestList

- Parameters:<ConnectionID><out TestList>
- 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

ctaHelp

- Parameters: <Command> <out Help>
- 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

- Parameters: <ConnectionID><Point in Net> <out Results>
- 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: <ConnectionID><out Results>
- 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

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

ctaLoadedTestName

- Parameters:<ConnectionID><out TestList>
- 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

- Parameters: <ConnectionID><Path\To\Script.extension>
- 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)

ctaLoadTest

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

ctaLuaComponent

- Parameters: <ConnectionID><ComponentName><Parameters> <out Result><out 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

- Parameters:<ConnectionID><out ProbedPoint>
- Returns Probed Point(s).
- function ctaProbe(wConnectionID: word; var iProbedPointLength: integer; cpProbedPoint: PChar): word;
- Prerequisite: Test in memory (prior call to Learn or LoadTest)

ctaPutLuaScript

- Parameters: <ConnectionID><ScriptName.extension><ScriptText>
- 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)

ctaPutTest

- Parameters: <ConnectionID><TestName><TestText><SetAsActiveTest>
- 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 is 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.

ctaResistance

- Parameters: <ConnectionID><SourcePoint><SinkPoint> <out Status><out MeasuredValue>
- 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: <ConnectionID><Source4WPoint><Sink4WPoint> <out Status><out MeasuredValue>
- 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; cdecl;
- Prerequisite: Test in memory (prior call to Learn or LoadTest). Points passed into the function are defined as 4W point in the test.

ctaSaveTest

- Parameters: <ConnectionID><TestName>
- 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

- Parameters: <ConnectionID><SerialNumber>
- For use with Easy-Wire reporting - 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.

ctaSelfTest

- Parameters: <ConnectionID><out SelfTestPassed>
- function ctaSelfTest(wConnectionID: word; out bSelfTestPassed: boolean): word;
- Prerequisite: None

ctaSetChildTestActivebyIndex

- Parameters: <ConnectionID><ChildIndex>
- function ctaSetChildTestActivebyIndex(wConnectionID: word; iChildIndex: integer): word;
- Prerequisite: Parent Child test in memory
- Notes: This sets the specified test active and runs Initialize Test to make it ready to run SingleTest or FastSingleTest, etc.

ctaSetChildTestActivebyName

- Parameters: <ConnectionID><ChildTestName>
- function ctaSetChildTestActivebyName(wConnectionID: word; sChildTestName: string): word;
- Prerequisite: Parent Child test in memory
- Notes: This sets the specified test active and runs Initialize Test to make it ready to run SingleTest or FastSingleTest, etc.

ctaSetOutput

- Parameters: <ConnectionID><OutputIndex> <OutputState 0 or 1>
- 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: <ConnectionID><SetMask>
- function ctaSetOutputs(wConnectionID: word; dwSetMask: cardinal): word;
- Sets the built-in tester outputs (set to a logic 1 state) for the bits specified (set to a logic 1 state) in dwSetMask.
- Prerequisite: IO Available on attached tester and tester supports bulk I/O changes.

ctaSetParameters

- Parameters: <ConnectionID><TestParameters>
- 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>
- 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: <ConnectionID><TestDefaults>
- 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>
- Causes the CTA Server to show/hide a status window. This setting is stored in the CirrisTesterAccessServer.ini and will maintain state.
- function ctaShowServerMonitor(bShow: boolean): word;
- Prerequisite: None

ctaSingleTest

- Parameters: <ConnectionID><1=Hipot 0=No Hipot><1=Shorts Test 0=No Shorts Test><out Results>
- 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: <ConnectionID><TestName><LotID>
- For Easy-Wire reporting - Starts a Test Run using the specified test name so Results can be saved for reporting.
- function ctaStartTestRun(wConnectionID: word; cpTestName: ctaPChar; cpLotID: ctaPChar): word;
- Prerequisite: None

ctaTwistedPair

- Parameters: <ConnectionID><Point1><Point2><out Twisted>
- 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)

5. 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**

5.1 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

5.2 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

5.3 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

6. 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.

Cirris Tester Access (CTA) User Manual
Version 2023.2.0

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