BERTScope™ CR
BERTScope™ CR HS
BERTScope™ CRJ

Clock Recovery
Software Development Kit

Version 00.02

SYNTHESYS RESEARCH, INC.

3475-D Edison Way
Menlo Park, California U.S.A. 94025

Voice (650) 364-1853 Fax (650) 364-5716

Email tech_support@bertscope.com

http://www.bertscope.com/

This Software Development Kit is applicable to the following SyntheSys Research, Inc., products:

BERTScope™ CR 12500 Clock Recovery Instrument
BERTScope™ CR HS 12500 High Sensitivity Clock Recovery Instrument
BERTScope™ CRJ Clock Recovery Jitter Analyzer

BERTScope™ CR Software Development Kit
Part Number 00.02
15 June 2007

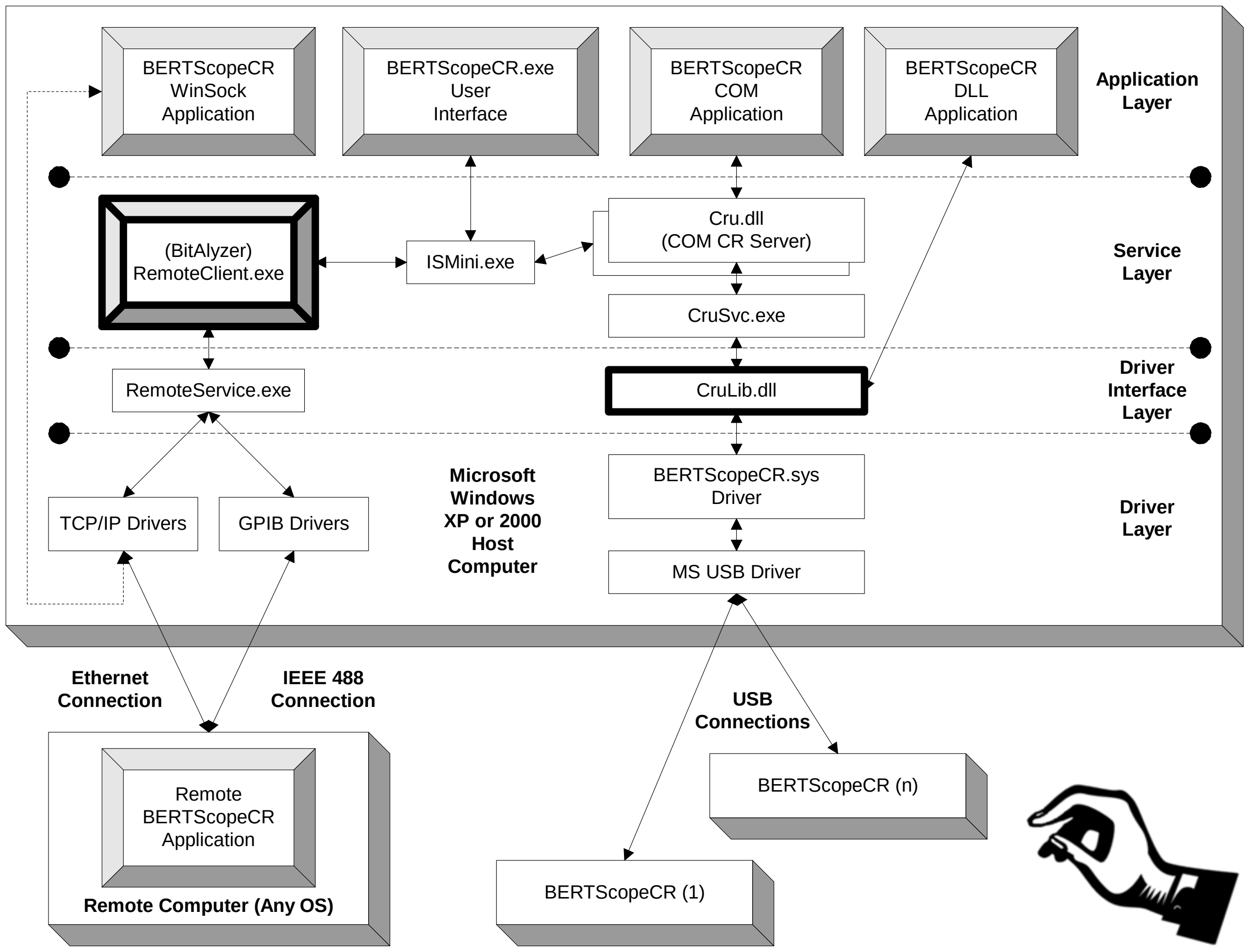
Table of Contents

BERTScope™ CR / CR HS / CRJ Software Development Kit

Overview

This document describes the various options available to develop programs for controlling and monitoring a BERTScope CR Clock Recovery instrument.

The diagram below depicts the BERTScope CR hardware and software components, and the communication links between them.



The 3-D boxes represent equipment. The diagram shows multiple BERTScope CRs connected to a Host PC, which is in turn connected to an optional Remote Computer. A Host PC is required for software control of a BERTScope CR.

A BERTScope CR is connected to a Host PC via a USB cable. The Host PC must therefore be located close to the BERTScope CR. The Host PC must be running Microsoft Windows 2000 or XP operating systems. Often, the ‘Host PC’ is actually a PC-based BERTScope instrument.

The Host PC is in turn connected to a Remote Computer via either an IEEE-488 (GPIB) or a TCP/IP connection. If TCP/IP is used, then the Remote Computer may be located a great distance away from the Host PC. The Remote Computer may run any OS.

The 2-D boxes represent software components running on the associated pieces of equipment. Framed boxes represent software components with a user interface. The 2 boxes with bold borders export the APIs (Application Programming Interfaces) available in this SDK for control software development.

Additionally, the hand in the diagram reminds us that a BERTScope CR may also be controlled manually -- applications should avoid caching the state of an instrument.

A control application may be one of the following types:

A local Windows application that links directly to the CruLib.dll low-level control interface.

A Microsoft COM-compatible application that uses the services supplied by the Cru.dll COM Clock Recovery Server. (The COM interface is not support in this SDK at this time. If you are interested in developing BERTScope CR applications for Microsoft's COM or .NET, please let us know).

A TCP/IP socket application using the text-based command set supplied by BitAlyzerRemoteClient.exe. The capabilities of this text-based API are roughly the same as the API exported by the CruLib.dll. A TCP/IP application might be written to run only locally, only remotely, or on either/both the Remote Computer and Host PC.

A remote IEEE-488 (GPIB) application (typically written with development tools from National Instruments). This type of program would also use the text-based command set supplied by BitAlyzerRemoteClient.exe.

A script or batch-file application that spawns RC.exe to execute each command. RC.exe is supplied with the SDK, and is a simple little BERTScope CR WinSock application that sends a single command or query to the BERTScope CR, then exits. It uses the services of BitAlyzerRemoteClient.exe. It is also handy for interactive control of the BERTScope CR from a command prompt.

The best application type to choose depends upon your situation.

If the Host PC is a BERTScope, you'll want to run your application on a remote computer, using the BitAlyzerRemoteClient.exe RC API for GPIB and TCP/IP. The RC API is described in a separate document -- the **BERTScope CR Remote Control Guide** (in the BERTScopeCRRemote.pdf file).

If the Host is a regular PC, is accessible, and is comfortable to work at, you might want to develop a local application using the Win32 API. The Win32 API is described in the last section of this document.

BERTScope CR SDK Installation Details

Where to Install this SDK

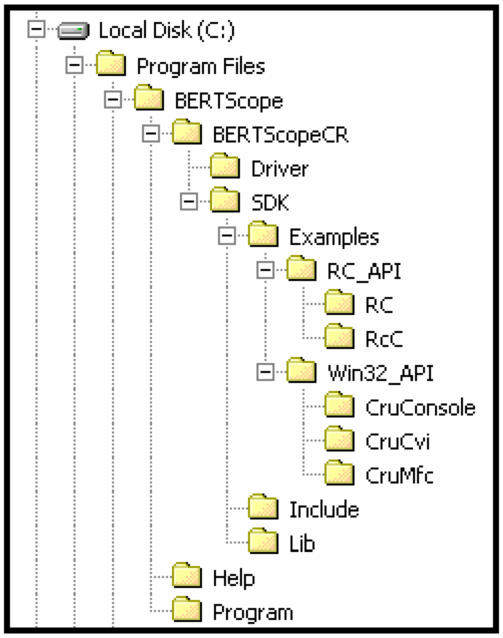
This SDK is primarily intended to support development of Win32 applications that link to our CruLib.dll. This SDK is distributed as part of the BERTScope CR software release. You should install this SDK on a Host Computer running Windows 2000 or Windows XP, linked by USB to a BERTScope CR. We DO NOT recommend that you install this SDK and your other development tools on a BERTScope, as this may degrade the BERTScope's performance.

This SDK presents a system overview and example applications that may be useful to those wishing to target the BitAlyzerRemoteClient (RC) API, as well. In this case, you might wish to install this SDK on a Remote (Microsoft Windows) computer, and talk with the Host computer (which might be a BERTScope) via TCP/IP or GPIB. See the **BERTScope CR Remote Control Guide** for the RC API.

New Directories and Files

The default installation location for the SDK is under "C:\Program Files\BERTScope".

The resulting BERTScope directory tree is shown below.



```
Local Disk (C:)
  Program Files
    BERTScope
      BERTScopeCR
        Driver
        SDK
          Examples
            RC_API
              RC
              RcC
            Win32_API
              CruConsole
              CruCvi
              CruMfc
          Include
          Lib
      Help
      Program
```

The BERTScope directory tree, and the software components in these folders, are shared by other members of the BERTScope product line. It is therefore best not to move or rename them, lest you accidentally break another application.

We will now discuss briefly the directories and files relevant to the BERTScope CR SDK:

The BERTScope Program Directory

The "C:\Program Files\BERTScope\Program" folder contains all the executable files for the BERTScope product line. These include the CruLib.dll, Cru.dll, RemoteClient.exe, and RC.exe files mentioned in the previous section, as well as a number of other files mentioned later.

Fans of the Tcl programming language may notice the TclCruLib.dll file in the Program folder. This is, indeed, a Tcl interface wrapper around the CruLib.dll. However, this DLL is maintained by our hardware and manufacturing engineers, and is subject to change at any time. We therefore discourage use of this DLL, and suggest you build your own wrapper around CruLib.dll.

The BERTScope Help Directory

The “C:\Program Files\BERTScope\Help” folder contains all the on-line and off-line Help files for the BERTScope product line. These include this document and the BERTScopeCRRemote.pdf file (which documents the TCP/IP and GPIB text-based API for the BERTScope CR.

The BERTScopeCR Driver Directory

The “C:\Program Files\BERTScope\BERTScopeCR\Driver” folder contains all the files used to install or uninstall the USB driver for the BERTScope CR. The BERTScopeCR.sys driver is installed automatically as part of the SDK installation. It may, however, be manually reinstalled with the INSTALL.CMD batch file, or uninstalled with the UNINSTALL.CMD file. Make sure to have the BERTScope CR plugged in and turned on before running the UNINSTALL.CMD file, or else the driver files and registry won’t be properly cleaned up by the operating system.

The SDK Include Directory

The “C:\Program Files\BERTScope\BERTScopeCR\SDK\Include” folder contains the files needed to define the BERTScope CR programming interfaces. The CruLib.h and CruTypes.h are used for projects that link to the CruLib.dll. Note that CruLib.h includes CruTypes.h, so your application need only include CruLib.h.

The SDK Lib Directory

The “C:\Program Files\BERTScope\BERTScopeCR\SDK\Lib” folder contains the files needed to resolve the BERTScope CR programming interfaces at link time. The CruLib.lib file is used for the CruLib.DLL API.

The SDK Examples Directories

The “C:\Program Files\BERTScope\BERTScopeCR\SDK\Examples” subdirectories contain example projects for the various types of applications you can develop with this SDK. We will discuss each of these examples later, in the appropriate section.

New COM Components Registered

Several COM Components are required for the operation of the BitAlyzerRemoteClient.exe program. These are described below.

Cru.dll

This in-process COM server implements the ICru and _ICruEvents automation-compatible interfaces with the Cru class. All COM-based BERTScope CR applications use these interfaces. Each COM application creates its own instance of this COM server, which then communicates with the CruSvc.exe COM singleton object, described next.

The installation program automatically registers Cru.dll. To manually register the component, type “`regsvr32 Cru.dll`” at a command prompt. To un-register the component, type “`regsvr32 /u Cru.dll`”.

CruSvc.exe

This out-of-process COM server is a singleton object that implements the ICruService and _ICruServiceEvents interfaces with the CruService class. This class can service multiple Cru clients, and can access multiple BERTScope CRs connected to the same Host computer. The ICruService interface should not be used directly by an application program – use the ICru interface instead. CruSvc.exe will be started automatically when it has clients, and stopped automatically when it has none.

The installation program automatically registers CruSvc.exe as a COM server. To manually register the component, type “`CruSvc -RegServer`” at a command prompt. To manually un-register the component, type “`CruSvc -UnregServer`”.

ISMini.exe

This out-of-process COM server is a singleton object which partially implements the IISMain and _IISMainEvents interfaces with the ISMain class. It shares a GUID with ISMain.exe, a component used by the BERTScope software. ISMini exists to resolve references to IISMain when ISMain.exe is not installed (when the Host computer isn't a BERTScope). If you try to register both ISMini.exe and ISMain.exe, ISMain.exe wins. ISMini.exe will be started automatically when it has clients, and stopped automatically when it has none.

The installation program automatically registers ISMini.exe as a COM server. To manually register the component, type "`ISMini -RegServer`" at a command prompt. To manually un-register the component, type "`ISMini -UnregServer`".

IMPORTANT NOTE:

Don't try to un-register ISMini.exe if ISMain.exe exists, or you will disable the BERTScope software. If you suspect this has occurred, you may manually re-register ISMain.exe by typing "`ISMain –RegServer`" at a command prompt.

RemoteService.exe

This out-of-process COM server is a singleton object which implements the IRSServer and _IRSServerEvents interfaces with the RSServer class. It handles the IEEE 488 (GPIB) and TCP/IP interfaces into the Host Computer. The IRSServer interface should not be used directly by an application program.

RemoteService.exe will be started automatically when it has clients, and stopped automatically when it has none.

The installation program automatically registers RemoteService.exe as a COM server. To manually register the component, type "`RemoteService -RegServer`" at a command prompt. To manually un-register the component, type "`RemoteService -UnregServer`".

Other Additions to the Registry

Besides the changes resulting from installing the BERTScopeCR.sys USB driver and registering the COM components above, a few registry changes are made for use by BERTScope CR application programs:

A "HKLM\Software\SyntheSys Research\BERTScopeCR" key contains an InstallPath string.

The "HKLM\Software\Microsoft\Windows\Help" key contains a string entry for the CRUPDATE.HLP on-line help file path.

Additions to the PATH Environment Variable

The SDK installation program automatically adds "C:\Program Files\BERTScope\Program" to the end of the system PATH environment variable.

BERTScope CR Application Tips

No matter which API you choose, your BERTScope CR application will need to do the following:

Initialize

To initialize a BERTScope CR, you would discover it's name, open it, and set it into a known state.

Discover BERTScope CRs

Most applications will only ever encounter one BERTScope CR. JUST IN CASE, however, a customer wishes to connect more than one BERTScope CR to the Host computer, we have designed the APIs to support it.

You can discover all BERTScope CRs currently connected using the `CruGetNames` routine in the Win32 API.

If using the RC API, you would send an

```
ATTACHDEVNAMES?
```

or

```
NAMES?
```

query.

Open BERTScope CR

Before you can communicate with a BERTScope CR, you need to 'open' a connection to it.

If there is more than one BERTScope CR present, then you should prompt the user to select one. You would then open the device by name.

If there is only one BERTScope CR found, then in the Win32 API you would just open it without prompting, either by name, or by supplying an empty string for the name (in which case the software will choose the first, and only, one automatically).

If using the RC API, a solitary BERTScope CR is opened automatically for you.

In the Win32 API, you are returned a 'handle' upon opening a device, which you then use for all subsequent communication. The Win32 call looks like:

```
CruOpenDevice( szName, &hCru );
```

Using the RC API, you would send either

```
CRService:OPEN "szname"
```

or

```
CRS:OPEN "szname".
```

In the RC API, the BitAlyzerRemoteClient program remembers the handle for you. A side effect of this is that you can only have one BERTScope CR open at a time when using the RC API.

Restore Settings

You will most likely want to start your application by putting the BERTScope CR into a known state.

The BERTScope automatically restores the settings saved to the 'POWER_ON' setup slot.

In the Win32 API, you can choose a different set of settings with the `CruRecallSetup` routine.

In the RC API, you would send

```
CRControl:RCONFIGDEVICE
```

or

```
CRC:RDEV
```

In the RC API, you can also retrieve settings from the Host computer's disk.

To restore a saved set of BERTScope CR Control settings:

```
CRControl:RCONFIGDISK “pathname”
```

or

```
CRC:RDISK “pathname”
```

You can also load a pre-defined sub-set of control settings known as a ‘Standard’.

In the Win32 API, you would do this by calling the `CruSetStandardByName` routine.

In the RC API, you would send

```
CRControl:STANDARD
```

or

```
CRC:STANDARD
```

To restore a saved set of BERTScope CR SSC Waveform settings:

```
CRSSCWaveform:RCONFIGURATION “pathname”
```

or

```
SSCW:RCONFIG “pathname”
```

To restore a saved set of BERTScope CR Jitter Spectrum settings:

```
CRJitterSpectrum:RCONFIGURATION “pathname”
```

or

```
JS:RCONFIG “pathname”
```

To restore a saved set of ALL BERTScope CR settings:

```
RCONFIGURATION “pathname”
```

or

```
RCON “pathname”
```

Poll

The USB interface to the BERTScope CR is one-way only. The host computer must initiate all communication – messages are never sent from the BERTScope CR to the host computer. Consequently, in order to monitor the state of the BERTScope CR (which can be modified manually ‘behind your program’s back’ via the BERTScope CR’s front panel), the host computer must poll the BERTScope CR periodically. Once a second is about right. If your program is going to have an interactive user interface, then you should do the polling in a background thread.

IMPORTANT NOTE:

Only one client application can retrieve events, because they are cleared as soon as they are read. If you are using our BERTScope CR user interface program, either in stand-alone mode, or as part of our BERTScope GUI, then DO NOT monitor events. Instead, just poll all the properties every loop. The tips below assume your application has sole ownership of the BERTScope CR.

The polling loop should first check for events and alarms. If events indicate the BERTScope CR’s state has changed, retrieve the changes. You would then handle any errors encountered. Finally, update the user interface with the new settings and measurements.

Check Events and Alarms

In the Win32 API, you call `CruGetEventsAndAlarms` to retrieve the CRU_EVENT and CRU_ALARM bit-masks. Use of this command clears the events and alarms. You can check the alarms non-destructively using the `CruGetAlarms` routine.

There is no way to check events or alarms using the RC API.

Retrieve Changed Settings

Decode the CRU_EVENT bitmask (typically with a `switch` statement) to discover the setting or measurement groups that have changed. There isn't a separate bit for each setting, but 3 or 4 related settings grouped per bit, typically. You'll only need to check the states for members of groups marked as changed.

For example, if the CRU_EVENT_LOOP bit was set, you would need to call `CruGetBandwidthHZ`, `CruGetPeakingDB`, `CruGetEdgeDensityMode`, and `CruGetNominalEdgeDensityPCNT`.

Using the RC API, you would retrieve any dynamic settings you wish to monitor every loop, since events are not available.

Handle Errors

If any CRU_ALARMs are detected, you'll probably want to log them and/or send off a message to the UI. Similarly, you need to handle any other unexpected failures (if someone powers-off or disconnects the BERTScope CR, for example).

The Win32 API `CruMap_` routines convert alarm, error, and setting codes into character strings, for use in your error handling and debugging messages.

See "Command Status" section of the **BERTScope CR Remote Control Guide** for information about error-handling when using the RC API.

Update UI

This step varies considerably on your application type. It could be as simple as printing out a line of text, or as complicated as sending an event from your background thread to your UI's event handler.

Validate Settings

If the user is allowed change the settings of the BERTScope CR with your program, then you will need to validate the settings prior to changing them.

In the Win32 API, use the `CruGetDBaseFloat` routine to discover the valid ranges for the setting you want to validate. Note that a setting's limits may change depending upon the current data rate. Also note that some settings have a wider valid range than they do a calibrated range.

There is no way to pre-validate a setting using the RC API.

If you exceed the allowed range for any setting using either API, the passed in value is clipped. If you have a polling loop, you will discover the final value automatically. If you have no polling loop, then, when using the RC API, it would be a good idea to post-validate the setting.

Terminate

At quitting time, you should save your settings and close your connection to the BERTScope CR.

Save Settings

Once it is time to quit, you may want to save the state of the BERTScope CR.

If the 'Setup Auto-Save' flag is TRUE, then the CR Control settings are automatically saved at power-down to the POWER_ON setup slot (which is automatically restored).

Using the Win32 API, you can save the settings to another setup slot with the `CruSaveSetup` routine.

Using the RC API, you would save to a setup slot in the BERTScope CR using
`CRControl:SCONFIGDEVICE`
or
`CRC:``SDEV`

Using the RC API, you can also save the CR Control settings to the Host Computer's disk drive
`CRControl:SCONFIGDISK “pathname”`
or
`CRC:``SDISK “pathname”`

To save the BERTScope CR SSC Waveform settings:
`CRSSCWaveform:SCONFIGURATION “pathname”`
or
`SSCW:SCONFIG “pathname”`

To save the BERTScope CR Jitter Spectrum settings:
`CRJitterSpectrum:SCONFIGURATION “pathname”`
or
`JS:SCONFIG` “pathname”

To save ALL BERTScope CR settings:
`SCONFIGURATION “pathname”`
Or
`SCON “pathname”`

Close BERTScope CR

You must always remember to close any open connection to a BERTScope CR.
Under the Win32 UI, you pass your hCRU handle to the `CruCloseDevice` routine.
Using the RC API, you simply send `CRService:ClOSE` or `CRS:CLOSE`.

Example Projects

Microsoft Win32 Projects using the CruLib.Dll API

Three example projects are provided with this SDK to demonstrate use of the CruLib.dll Win32 API.

The default path to these samples is:

```
C:\Program Files\BERTScope\BERTScopeCR\SDK\Examples\Win32_API\
```

CruConsole – C++ Windows Command-Line Application Example

This example opens the first BERTScope CR it finds, uploads all the Standards stored in the device, dumps them to the command console, then closes the device and exits. It is a Microsoft VC++ version 6 application.

CruMfc – C++ Windows Dialog Box Application Example

This a Microsoft VC++ MFC dialog application utilizes the complete Win32 API supported in this SDK. It features background polling of the BERTScope CR state. Try starting multiple instances of this application, then watch them interact with each other and the BERTScope CR front panel.

CruCvi – National Instruments CVI Application Example

This example is functionally equivalent to CruConsole, but uses NI CVI rather than MS VC++.

Please note that this SDK does NOT include the CVI runtime environment required to execute this example.

Remote Control Projects Using the RC API

Two example projects are provided with this SDK to demonstrate use of the RC API.

The default path to these samples is:

```
C:\Program Files\BERTScope\BERTScopeCR\SDK\Examples\RC_API\
```

These examples may be run either on the Host computer, or a remote (Windows) computer.

Remember that the BitAlyzerRemoteClient.exe application must be running on the Host computer, and that the protocol selection must match that for the example program.

RC – Microsoft WinSock Command-Line Application in C

This example is a simple little Windows socket application that allows you to enter a single RC API command or query at the command line. Typing ‘RC’ displays the following help:

```
>rc
Expecting '[/host=ip_address] feature:operation number_param'
or        '[/host=ip_address] feature:operation \"string_param\"'
or        '[/host=ip_address] feature:operation?'
```

It defaults to ‘localhost’ for the IP address, if omitted.

Note that you must escape (`\"`) any quotes required for string parameters.

Sample commands:

```
>rc crs:names?
CRU0003,
>rc crs:open?
NONE
>rc crs:open \"CRU0003\"

>rc crs:open?
CRU0003
```

You can also put a number of commands into a Windows batch file.

Validating a query in a batch file is tricky, but it can be done:

```
@echo off
rc crs:open? > rctmp.txt & set /p rc_result= < rctmp.txt
if %rc_result%==NONE goto :Error
type rctmp.txt
goto :Done

:Error
echo Clock Recovery not open

:Done
```

RCC – National Instruments CVI Application Example

This example is a NI CVI version of RCC that works over a IEEE 488 (GPIB) connections as well as over TCP/IP. Please note that this SDK does NOT include the CVI runtime environment required to execute this example.

The CruLib.Dll Win32 API

This Appendix is based on the definitions found in the CruLib.h and CruTypes.h include files. If there is an inconsistency between the include files and this document, then the include files win.

Some routines in CruLib.h, however, are either obsolete or are still not implemented. Other routines are simply not useful for CR application programming. These will be marked as such in the Command Summary, below, and omitted entirely from the Command Reference.

Command Summary (Organized Functionally)

Device Access and Identification Routines

`CruGetDeviceNames` – Gets USB driver-name list of CR devices
`CruGetNames` – Gets user-defined unit-name list of CR devices
`CruOpenDevice` – Requests non-exclusive access to a CR device
`CruCloseDevice` – Terminates connection to previously opened CR device
`CruGetDeviceType` – Lets you check if CR device is real or simulated
`CruGetDeviceName` – Gets USB driver's name for specified CR device
`CruGetDeviceSerialNumber` – Gets CR device’s serial number
`CruGetDeviceModel` – Gets model string for specified CR device
`CruSetUnitName` – Sets CR device’s user-defined unit-name
`CruGetUnitName` – Gets CR device’s user-defined unit-name
`CruGetCapabilities` – Gets CR device’s capabilities
`CruGetDeviceRev` – Gets SW, FW, HW, FPGA, and Exp revision strings
`CruHog` – Sets a mutex-protected flag that can be used by applications to reserve CRU access
`CruShare` – Clears a mutex-protected flag that can be used by applications to reserve CRU access

Device Status Routines

`CruConnected` – Checks if CR device is still connected and powered-on
`CruNotBusy` – Checks a mutex-protected flag to see if a CR is available for shared access
`CruOperationInProgress` – Please use `CruOperationComplete`, below
`CruOperationComplete` – Checks if a CR device state change has stabilized
`CruGetLockState` – Gets the current CR lock state
`CruMapLockState` – Converts a lock state value into a string, for use in diagnostic messages
`CruGetAlarms` – Checks for CR alarms without resetting internal CR event buffer
`CruGetEventsAndAlarms` – Checks for CR events and alarms, then resets event buffer
`CruMapEvent` – Converts an event code into a string, for use in diagnostic messages
`CruMapAlarm` – Converts an alarm code into a string, for use in diagnostic messages
`CruMapStatus` – Converts a `CRUSTAT` return code into a string, for use in diagnostic messages

Clock Recovery Measurements

`CruGetMeasuredDataRateHZ` – Get current measured data rate, in Hertz
`CruGetMeasuredPhaseErrorPk2PkPCNT` – Get current peak-to-peak phase error, in %UI
`CruGetMeasuredPhaseErrorRmsPCNT` – Get current RMS phase error, in %UI
`CruGetMeasuredEdgeDensityPCNT` – Get current measured data edge density, in %

Clock Recovery Control Settings

`CruSetLockMode` / `CruGetLockMode` – The desired locking mode
`CruMapLockMode` – Converts lock mode code into a string, for use in diagnostic messages
`CruAcquireLock` – Initiate a search for the clock frequency
`CruGetLockCount` / `CruResetLockCount` – Times lock has been acquired
`CruSetFrequencyHZ` / `CruGetFrequencyHZ` – Nominal clock frequency
`CruSetLockRangeHZ` / `CruGetLockRangeHZ` – The +/- freq. range about nominal
`CruSetBandwidthHZ` / `CruGetBandwidthHZ` – The -3dB loop response frequency
`CruSetPeakingDB` / `CruGetPeakingDB` – The loop response peak setting
`CruSetPhaseErrorLimitPCNT` / `CruGetPhaseErrorLimitPCNT` – Max phase error
`CruSetClockEnable` / `CruGetClockEnable` – Enable / Disable Clock output
`CruSetClockAmplitudeMV` / `CruGetClockAmplitudeMV` – Clock output amplitude.
`CruSetSubrateClockEnable` / `CruGetSubrateClockEnable` – Enable / Disable Sub-rate clock
`CruSetSubrateClockAmplitudeMV` / `CruGetSubrateClockAmplitudeMV` – Sub-rate ampl.
`CruSetSubrateDivisor` / `CruGetSubrateDivisor` – Sub-rate clock divisor setting
`CruFetchSubrateTable` – Get list of valid sub-rate clock divisors
`CruSetEdgeDensityMode` / `CruGetEdgeDensityMode` – How edge density is computed
`CruMapEdgeDensityMode` – Converts edge density mode into a string, for diagnostic messages
`CruSetNominalEdgeDensityPCNT` / `CruGetNominalEdgeDensityPCNT` – Nom. ED

Obsolete Clock Recovery Control Settings

`CruSetClockDutyCyclePCNT`/ `CruGetClockDutyCyclePCNT`
`CruSetSubrateClockDutyCyclePCNT`/ `CruGetSubrateClockDutyCyclePCNT`
`CruSetInputThreshold` / `CruGetInputThreshold`

Unimplemented Clock Recovery Control Settings

`CruSetTriggerMode` / `CruGetTriggerMode` / `CruMapTriggerMode`
`CruSetTriggerThresholdMV` / `CruGetTriggerThresholdMV`
`CruSetTriggerTerminationMV` / `CruGetTriggerTerminationMV`
`CruSetTriggerHysteresisMV` / `CruSetTriggerHysteresisMV`

Clock Recovery Setting Limits (From Internal Database)

`CruGetDBaseString` – **Unimplemented**
`CruGetDBaseLong` – Get a limit of type `long,` given an enumeration
`CruGetDBaseLongByName` – Get a limit of type `long,` given setting name string
`CruGetDBaseFloat` – Get a limit of type `float,` given an enumeration
`CruGetDBaseFloatByName` – Get a limit of type `float,` given setting name string
`CruMapDBase` – Turn an enumerated value into a setting name string

Clock Recovery Standard Routines

`CruGetStandardTable` – Get array of Standards stored in BERTScope CR
`CruGetStandardByName` – Get name of current Standard
`CruSetStandardByName` – Set current Standard (changing settings above)
`CruAddModifyStandard` – Save the current settings as a new Standard
`CruDeleteStandard` – Delete a Standard from the BERTScope CR

A CR Standard consists of:
`Name` – Up to 12 characters, case sensitive, spaces allowed
`FrequencyHZ` – Nominal Frequency, in Hertz
`LockRangeHZ` – The +/– frequency range about nominal to search for lock
`BandwidthHZ` – The –3 dB loop bandwidth frequency
`PeakingDB` – Peak frequency of loop response curve
`NominalEdgeDensityPCNT` – Nominal edge density of data
`SSC` – Spread spectrum clocking flag (0 = off, 1 = on)

Device Configuration Routines

`CruSaveSetup` – Save the current settings to a setup slot
`CruRecallSetup` – Restore a set of settings, given a setup slot name
`CruSetSetupAutoSave` / `CruGetSetupAutoSave` – Settings saved to `POWER_ON` slot

Jitter Spectrum Routines (Model CRJ only)

`CruStartJitSpecSession` – Starts continuous collection of jitter spectrum data
`CruGetJitSpecStatus` – Retrieves data acquisition progress information
`CruPauseJitSpecSession` – Pauses a jitter spectrum session
`CruResumeJitSpecSession` – Resumes a paused a jitter spectrum session
`CruStopJitSpecSession` – Aborts a jitter spectrum session
`CruGetJitSpecResults` – Get the current results for the jitter spectrum session
`CruGetJitSpecMeasurement` – Retrieves integrated jitter measurement
`CruGetJitSpecPeak` – Retrieves Searches for peak within the defined window
`CruClearJitSpecSessionData` – Frees data from the previous jitter spectrum session
`CruExportJitSpecCsvFile` – Saves jitter spectrum data to a CSV formatted file on disk
`CruImportJitSpecCsvFile` – Loads jitter spectrum data from a CSV formatted file on disk

SSC Waveform Routines

`CruStartSscWaveformSession` – Starts continuous collection of SSC Waveform data
`CruGetSscWaveformStatus` – Retrieves data acquisition progress information
`CruPauseSscWaveformSession` – Pauses an SSC Waveform session
`CruResumeSscWaveformSession` – Resumes a paused SSC Waveform session
`CruStopSscWaveformSession` – Aborts an SSC Waveform session
`CruGetSscWaveformResults` – Get the current results for the SSC Waveform session
`CruGetSscWaveformMeasurement` – Retrieves specified SSC Waveform measurement.
`CruClearSscWaveformSessionData` – Frees data from the previous SSC Waveform session
`CruExportSscWaveformCsvFile` – Saves SSC Waveform data to a CSV formatted file on disk
`CruImportSscWaveformCsvFile` – Loads SSC Waveform data from a CSV formatted file on disk

Low-Level USB Interface Routines (Not for use by CR Applications)

```
CruUsbEnum
CruSimplyOpen
CruSendVendorOrClassRequest
CruPoke
CruPeek
CruBulkRead
CruBulkWrite
CruSetInterface
CruApplicationRunning
CruFastUsb
```

Calibration and Diagnostic Routines (Not for use by CR Applications)

```
CruSetInputThresholdMV
CruGetStatisticsPacket
CruSetDDSMode / CruGetDDSMode
CruMapDDSMode
CruSetDDSFrequencyHZ / CruGetDDSFrequencyHZ
CruSetDDSAmplitudePCNT / CruGetDDSAmplitudePCNT
CruSetDDSWaveform / CruGetDDSWaveform
CruMapDDSWaveform
CruSetRamSource / CruGetRamSource
CruMapRamSource
CruSetRamSampleLen / CruGetRamSampleLen
CruSetRamDecimation / CruGetRamDecimation
CruStartRamCapture
CruAbortRamCapture
CruGetRamState
CruMapRamState
CruGetRamSamples
CruGetCombinedRamSamples
CruGetCombinedRamSamplesAndRawPhases
CruScanInput
CruGetScanInputState
CruGetInputHistogram
CruGetInputStat
CruGetMeasuredDutyCycleDistortionPCNT
CruToggleSimFlag
```

`CRUSTAT CruAcquireLock( hCRU h );`

Initiates a search for the clock frequency, and lock onto it once found.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

This routine merely initiates a search for lock and returns immediately. Achieving lock may take some time. Achieving lock is not guaranteed and should not be assumed. The application should call `CruGetLockState` to determine the actual lock status.

SEE ALSO:

```
CruGetLockState
CruGetLockCount, CruResetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockState, CruMapLockMode
```

```
CRUSTAT CruAddModifyStandard( hCRU h,
                              CRUSTANDARD *pVal );
```

Adds or modifies (if it already exists) the Clock Recovery Standard specified.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to name, frequency, lock range, bandwidth, peaking, edge density, and SSC specs.

REMARKS:

The CRUSTANDARD structure is defined as:

```
typedef struct _CRUSTANDARD
{
   char Name[12];
   double FrequencyHZ;
   double LockRangeHZ;
   double BandwidthHZ;
   double PeakingDB;
   double NominalEdgeDensityPCNT;
   char SSC; // Spread Spectrum Clocking Flag ( 0 = off, 1 = on )
} CRUSTANDARD, *PCRUSTANDARD ;
```

If a Standard already exists with the same `Name`, the new settings replace the old.

SEE ALSO:

```
CruGetStandardTable
CruSetStandardByName, CruGetStandardByName
CruDeleteStandard
```

`CRUSTAT CruClearJitSpecSessionData( hCRU h );`

Clears any and all data from any previous Jitter Spectrum data acquisition session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

This routine frees all memory associated with a previous session, and resets the scan counter.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

`CRUSTAT CruClearSscWaveformSessionData( hCRU h );`

Clears any and all data from any previous SSC Waveform data acquisition session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

This routine frees all memory associated with a previous session, and resets the scan counter.

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

`CRUSTAT CruCloseDevice( hCRU h );`

Closes previously opened BERTScope CR device, freeing all allocated memory.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

This operation changes the state of a global table of device handles, which is protected by a mutex.

SEE ALSO:

`CruOpenDevice`
`CruGetNames`

`CRUSTAT CruConnected( hCRU h );`

Tries to talk to the specified BERTScope CR device to check the USB connection.

RETURNS:

CRU_OK, if still connected
CRU_NO_LONGER_CONNECTED, if disconnected
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

This routine simply verifies the connection to the USB chip in a previously opened BERTScope CR.
It is independent of any BERTScope CR firmware, or device state
(other than that the chip is connected and has power).

Note that the software remembers the serial number of every BERTScope CR it encounters.
If a device is disconnected and then reconnected (via either the cable or a power-cycle),
then 'h', the device handle, is reactivated automatically – there is no need to re-open the device.

SEE ALSO:

`CruNotBusy`
`CruOperationComplete`

```
CRUSTAT CruDeleteStandard( hCRU h,
                           char * StdName );
```

Removes the Clock Recovery Standard specified from the BERTScope CR.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `StdName` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`StdName` = case-sensitive zero-terminated char string that identifies Standard

REMARKS:

If you delete the currently selected Standard, the selection will change to "- - -" (None).

SEE ALSO:

`CruGetStandardTable`
`CruSetStandardByName, CruGetStandardByName`
`CruAddModifyStandard`

```
CRUSTAT CruExportJitSpecCsvFile( hCRU h,
                                 char * pszPath );
```

Saves the complete set of Jitter Spectrum data to a comma-separated-variable format ASCII file

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pszPath` = zero-terminated string designating disk path and filename

REMARKS:

This routine may block on a mutex if the data buffers are being updated.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruImportJitSpecCsvFile
```

```
CRUSTAT CruExportSscWaveformCsvFile( hCRU h,
                                     char * pszPath );
```

Saves the complete set of SSC Waveform data to a comma-separated-variable format ASCII file

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pszPath` = zero-terminated string designating disk path and filename

REMARKS:

This routine may block on a mutex if the data buffers are being updated.

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruClearSscWqaveformSessionData,
CruImportSscWaveformCsvFile
```

```
CRUSTAT CruFetchSubrateTable( hCRU h,
                              CRUSUBRATES * pVal );
```

Retrieves the valid sub-rate clock divisors.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pVal = pointer to structure containing array of valid sub-rate divisors.

REMARKS:

The CRUSUBRATES structure is defined as:

```
typedef struct _CRUSUBRATES
{
    long count;
    unsigned short divisors[ CRU_CONST_MAX_DIVISORS ];
} CRUSUBRATES, *PCRUSUBRATES ;
```

The `count` structure member indicates the number of valid `divisor[]` array elements.

SEE ALSO:

`CruSetSubrateDivisor, CruGetSubrateDivisor`

```
CRUSTAT CruGetAlarms( hCRU h,
                      unsigned long * pAlarms );
```

Retrieves CRU_ALARM bit-mask without clearing the BERTScope CR's internal event buffer.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pAlarms` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pAlarms` = pointer to bitmask containing one or more alarms (really of type enum `CRU_ALARM`)

REMARKS:

The `CRU_ALARM` codes currently defined are:

`CRU_ALARM_FLASH_ERROR :`
Corrupted flash records found.

`CRU_ALARM_NOM_ED_MISMATCH :`
Nominal Edge Density doesn't match Measured Edge Density.

`CRU_ALARM_LAST_LOCK_ED_MISMATCH :`
On-Lock Edge Density doesn't match Measured Edge Density.

`CRU_ALARM_BANDWIDTH_UNCAL :`
Loop Bandwidth set outside calibrated range for Nominal Frequency.

`CRU_ALARM_PEAKING_UNCAL :`
Peaking set outside calibrated range for Nominal Frequency and Loop Bandwidth.

SEE ALSO:

`CruGetEventsAndAlarms`
`CruMapEvent, CruMapAlarm`

```
CRUSTAT CruGetBandwidthHZ( hCRU h,
                           double * pVal );
```

Retrieves the current frequency setting for the -3 dB point of the clock recovery loop response, in Hertz.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the current loop bandwidth setting

REMARKS:

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

`CRU_DBASE_MIN_BW_HZ`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MINCAL_BW_HZ`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAXCAL_BW_HZ`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAX_BW_HZ`	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

SEE ALSO:

`CruSetBandwidthHZ`
`CruGetDBaseFloat`

```
CRUSTAT CruGetCapabilities( hCRU h,
                            unsigned long * pCapabilities );
```

Retrieves a bitmask that indicates various capabilities of the BERTScope CR.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pCapabilities` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pCapabilities` = pointer to (`CRU_CAPABILITY_BIT` enum) bitmask

REMARKS:

The only `CRU_CAPABILITY_BIT` codes normally used are:

`CRU_CAPABILITY_BIT_CRJ == 1`	if is a BERTScope CRJ
`else CRU_CAPABILITY_BIT_HS == 1`	if is a BERTScope CR HS
`else`	is a BERTScope CR

SEE ALSO:

```
CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetClockAmplitudeMV( hCRU h,
                                double * pVal );
```

Retrieves the current clock output amplitude setting, in milliVolts.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the current clock amplitude setting, in milliVolts.

REMARKS:

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

`CRU_DBASE_MIN_CLK_AMP_V`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MINCAL_CLK_AMP_V`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAXCAL_CLK_AMP_V`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAX_CLK_AMP_V`	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

Note: The limits are returned in Volts, not milliVolts!

SEE ALSO:

`CruSetClockAmplitudeMV`
`CruSetSubrateClockAmplitudeMV, CruGetSubrateClockAmplitudeMV`
`CruGetDBaseFloat`

```
CRUSTAT CruGetClockEnable( hCRU h,
                           BOOL * pVal );
```

Checks whether or not the clock output is enabled.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to Enabled flag.

REMARKS:

`*pVal == TRUE`	if clock output is currently enabled
`*pVal == FALSE`	if clock output is currently disabled

SEE ALSO:

`CruSetClockEnable`
`CruSetSubrateClockEnable, CruGetSubrateClockEnable`

```
CRUSTAT CruGetDBaseFloat( hCRU h,
                          CRU_DBASE code,
                          double * pVal );
```

Retrieves a floating-point setting limit, given the `CRU_DBASE` code.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`code` = setting limit identification code
`pVal` = pointer to the setting limit to be retrieved

REMARKS:

CRU_DBASE codes that are valid for this routine are:

```
CRU_DBASE_MIN_FREQ_HZ
CRU_DBASE_MINCAL_FREQ_HZ
CRU_DBASE_MAXCAL_FREQ_HZ
CRU_DBASE_MAX_FREQ_HZ

CRU_DBASE_MIN_LOCKRANGE_HZ
CRU_DBASE_MAX_LOCKRANGE_HZ

CRU_DBASE_MIN_BW_HZ
CRU_DBASE_MINCAL_BW_HZ
CRU_DBASE_MAXCAL_BW_HZ
CRU_DBASE_MAX_BW_HZ

CRU_DBASE_MIN_PEAKING_DB
CRU_DBASE_MINCAL_PEAKING_DB
CRU_DBASE_MAXCAL_PEAKING_DB
CRU_DBASE_MAX_PEAKING_DB

CRU_DBASE_MIN_NOM_EDGE_DENSITY_PCNT
CRU_DBASE_MAX_NOM_EDGE_DENSITY_PCNT

CRU_DBASE_MIN_PHASE_ERROR_LIMIT_PCNT
CRU_DBASE_MAX_PHASE_ERROR_LIMIT_PCNT

CRU_DBASE_MIN_CLK_AMP_V
CRU_DBASE_MINCAL_CLK_AMP_V
CRU_DBASE_MAXCAL_CLK_AMP_V
CRU_DBASE_MAX_CLK_AMP_V

CRU_DBASE_MIN_SUBCLK_AMP_V
CRU_DBASE_MINCAL_SUBCLK_AMP_V
CRU_DBASE_MAXCAL_SUBCLK_AMP_V
CRU_DBASE_MAX_SUBCLK_AMP_V
```

SEE ALSO:

`CruGetDBaseFloatByName`
`CruGetDBaseLong, CruGetDBaseLongByName`
`CruMapDBase`

```
CRUSTAT CruGetDBaseFloatByName( hCRU h,
                                char * Name,
                                double * pVal );
```

Retrieves a floating-point setting limit, given the setting Name.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `Name == NULL` or `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`Name` = setting limit identification string
`pVal` = pointer to the setting limit to be retrieved

REMARKS:

Setting `Name` strings that are valid for this routine are:

"MIN_FREQ"
"MINCAL_FREQ"
"MAXCAL_FREQ"
"MAX_FREQ"

"MIN_LOCKRANGE_HZ"
"MAX_LOCKRANGE_HZ"

"MIN_BW"
"MINCAL_BW"
"MAXCAL_BW"
"MAX_BW"

"MIN_PEAKING_DB"
"MINCAL_PEAKING_DB"
"MAXCAL_PEAKING_DB"
"MAX_PEAKING_DB"

"MIN_NOM_EDGE_DENSITY_PCNT"
"MAX_NOM_EDGE_DENSITY_PCNT"

"MIN_PHASE_ERROR_LIMIT_PCNT"
"MAX_PHASE_ERROR_LIMIT_PCNT"

"MIN_CLK_AMP_V"
"MINCAL_CLK_AMP_V"
"MAXCAL_CLK_AMP_V"
"MAX_CLK_AMP_V"

"MIN_SUBCLK_AMP_V"
"MAX_SUBCLK_AMP_V"
"MINCAL_SUBCLK_AMP_V"
“MAXCAL_SUBCLK_AMP_V"

SEE ALSO:

```
CruGetDBaseFloat
CruGetDBaseLong, CruGetDBaseLongByName
CruMapDBase
```

```
CRUSTAT CruGetDBaseLong( hCRU h,
                         CRU_DBASE code,
                         long * pVal );
```

Retrieves an integer setting limit, given the `CRU_DBASE` code.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`code` = setting limit identification code
`pVal` = pointer to the setting limit to be retrieved

REMARKS:

There are no `CRU_DBASE` codes currently used with this routine.

SEE ALSO:

`CruGetDBaseLongByName`
`CruGetDBaseFloat, CruGetDBaseFloatByName`
`CruMapDBase`

```
CRUSTAT CruGetDBaseLongByName( hCRU h,
                               char * Name,
                               long * pVal );
```

Retrieves an integer setting limit, given the setting `Name`.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`Name` = setting limit identification string
`pVal` = pointer to the setting limit to be retrieved

REMARKS:

There are no setting `Name` strings currently used with this routine.

SEE ALSO:

```
CruGetDBaseLong
CruGetDBaseFloat, CruGetDBaseFloatByName
CruMapDBase
```

```
CRUSTAT CruGetDeviceModel( hCRU h,
                           char * pBuf );
```

Retrieves the BERTScope CR model string.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pBuf == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pBuf = pointer to a char buffer of at least CRU_CONST_MAX_INFO_LEN.

REMARKS:

"BERTScope CR" or “BERTScope CR HS”

SEE ALSO:

```
CruGetCapabilities
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetDeviceName( hCRU h,
                          char * pBuf );
```

Gets driver's name for device:

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pBuf == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pBuf` = zero-terminated string containing driver's name for the BERTScope CR

REMARKS:

"BERTScopeCR-0", etc.
Assumes `*pBuf` is at least CRU_CONST_MAX_USB_DEVICE_NAME bytes.

**The device name is assigned by the BERTScopeCR USB driver,
and may change if the device is power-cycled or momentarily disconnected.**

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetDeviceNames( char * pszUsbDeviceNames,
                           short * pNumDevices );
```

Retrieves count and list of valid BERTScopeCR driver devices

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pszUsbDeviceNames == NULL` or `pNumDevices == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

`pszUsbDeviceNames` = zero-terminated string containing space separated list of driver devices
`pNumDevices` = pointer to number of devices in the list

REMARKS:

"BERTScopeCR-0 BERTScopeCR-1 BERTScopeCR-5 ", etc.
Assumes `*pszUsbDeviceNames` is at least CRU_CONST_MAX_USB_DEVICE_CHARS bytes.

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetDeviceRev( hCRU h,
                         CRUREV * pStruct );
```

Retrieves a structure containing revision strings for the various components of the BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pStruct == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pStruct = pointer to CRUREV struct containing revision strings

REMARKS:

CRUREV is defined as:

```
typedef struct _CRUREV
{
    char DllRev[CRU_CONST_MAX_INFO_LEN];  // CruLib.dll rev         “2.4”
    char FWRev[CRU_CONST_MAX_INFO_LEN];   // Firmware rev         “1.0.1”
    char HWRev[CRU_CONST_MAX_INFO_LEN];   // Hardware rev   “0130-201-04”
    char FPGARev[CRU_CONST_MAX_INFO_LEN]; // FPGA rev               “2.8”
    char EXPRev[CRU_CONST_MAX_INFO_LEN];  // Expansion board rev   “XXXX”
} CRUREV, *PCRUREV;
```

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetDeviceSerialNumber( hCRU h,
                                  char * pBuf );
```

Retrieves the BERTScope CR serial number string

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pBuf == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pStruct = pointer to a char buffer of at least CRU_CONST_MAX_INFO_LEN.

REMARKS:

This is the primary identifier for a BERTScope CR.
It is of the format “17nnnn”, where n is a digit.

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev
CruGetDeviceType
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetDeviceType( hCRU h,
                          CRU_TYPE * pType );
```

Retrieves the device type (real or simulated)

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pType == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pType` = pointer to the CRU_TYPE device type code

REMARKS:

The following CRU_TYPE device types are defined:

```
CRU_TYPE_SIM  // No USB chip, simulated BERTScope CR
CRU_TYPE_USB  // Real USB chip, but simulated BERTScope CR
CRU_TYPE_CRU  // Real USB chip, Real BERTScope CR
```

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruSetUnitName, CruGetUnitName
```

```
CRUSTAT CruGetEdgeDensityMode( hCRU h,
                               CRU_EDGE_DENSITY_MODE * pVal );
```

Retrieves the current edge density mode setting

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to edge density mode variable

REMARKS:

The edge density modes currently defined are:

CRU_EDGE_DENSITY_MODE_NOM :
Use the nominal edge density set with `CruSetNominalEdgeDensityPCNT`.

CRU_EDGE_DENSITY_MODE_ON_LOCK :
Use the edge density that was measured at moment lock was achieved.

SEE ALSO:

`CruSetEdgeDensityMode`
`CruSetNominalEdgeDensityPCNT`, `CruGetNominalEdgeDensityPCNT`
`CruGetMeasuredEdgeDensityPCNT`

```
CRUSTAT CruGetEventsAndAlarms( hCRU h,
                               unsigned long * pEvents,
                               unsigned long * pAlarms );
```

Retrieves CRU_EVENT and CRU_ALARM bitmasks, clearing the BERTScope CR's internal event buffer

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pEvents == NULL or pAlarms == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pEvents` = pointer to bitmask containing one or more eventss (really of type enum `CRU_EVENT`)
`pAlarms` = pointer to bitmask containing one or more alarms (really of type enum `CRU_ALARM`)

REMARKS:

This routine should be called periodically to check for BERTScope CR state changes.
Only the settings that have changed then need to be retrieved.

The `CRU_EVENT` codes currently defined are:

`CRU_EVENT_FREQ :`
Nominal Frequency setting changed

`CRU_EVENT_LOOP :`
Bandwidth, Peaking, Edge Density Mode, or Nominal Edge Density changed

`CRU_EVENT_LOCKPARAM :`
Lock Range, Lock Mode, or Phase Error Limit changed

`CRU_EVENT_CLKOUT :`
Clock Enable or Clock Amplitude changed

`CRU_EVENT_SUBOUT :`
Sub-rate Clock Enable, Sub-rate Clock Amplitude, Sub-rate, or Sub-rate Divisor changed

`CRU_EVENT_STANDARD :`
Current Standard, or the Standard list changed

`CRU_EVENT_LOCK :`
Lock Count changed

`CRU_EVENT_SETUP :`
Current Setup or, Setup Auto-Save flag changed

`CRU_EVENT_ALARM :`
Alarm bit-mask has changed

The `CRU_ALARM` codes currently defined are:

`CRU_ALARM_FLASH_ERROR :`
Corrupted flash records found.

`CRU_ALARM_NOM_ED_MISMATCH :`
Nominal Edge Density doesn't match Measured Edge Density.

`CRU_ALARM_LAST_LOCK_ED_MISMATCH :`
On-Lock Edge Density doesn't match Measured Edge Density.

`CRU_ALARM_BANDWIDTH_UNCAL :`
Loop Bandwidth set outside calibrated range for Nominal Frequency.

`CRU_ALARM_PEAKING_UNCAL :`
Peaking set outside calibrated range for Nominal Frequency and Loop Bandwidth.

SEE ALSO:
`CruGetAlarms`
`CruMapEvent, CruMapAlarm`

```
CRUSTAT CruGetFrequencyHZ( hCRU h,
                           double * pVal );
```

Retrieves the nominal data frequency setting, in Hertz

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to nominal frequency variable

REMARKS:

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

`CRU_DBASE_MIN_FREQ_HZ`	Minimum allowed setting
`CRU_DBASE_MINCAL_FREQ_HZ`	Minimum calibrated setting
`CRU_DBASE_MAXCAL_FREQ_HZ`	Minimum calibrated setting
`CRU_DBASE_MAX_FREQ_HZ`	Maximum allowed setting

Where `MIN <= MINCAL < MAXCAL <= MAX`

SEE ALSO:

```
CruSetFrequencyHz
CruGetMeasuredDataRateHz
CruSetLockRangeHZ, CruGetLockRangeHZ
CruGetDBaseFloat
```

```
CRUSTAT CruGetJitSpecMeasurement( hCRU h,
                                  CRU_MEAS_TYPE MeasType,
                                  double MinHz,
                                  double MaxHz,
                                  BOOL IsPs,
                                  double * pIntJitMeas );
```

Computes the integrated jitter measurement for the specified modulation frequency range

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`MeasType` = CRU_MEAS_TYPE_JS_INT_PP (Peak-to-Peak integrated jitter)
or
CRU_MEAS_TYPE_JS_INT_RMS (RMS integrated jitter)
`MinHz` = min modulation frequency limit
`MaxHz` = max modulation frequency limit
`IsPs` = TRUE if need to return values in picosecs, else %UI
`pIntJitMeas` = pointer to the integrated jitter measurement

REMARKS:

`MinHz` and `MaxHz` must lie within `*pMinDataHz` and `*pMaxDataHz` returned at session start.

SEE ALSO:

`CruStartJitSpecSession, CruGetJitSpecStatus,`
`CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,`
`CruGetJitSpecResults, CruGetJitSpecPeak,`
`CruClearJitSpecSessionData,`
`CruExportJitSpecCsvFile, CruImportJitSpecCsvFile`

```
CRUSTAT CruGetJitSpecPeak( hCRU h,
                           double MinFreq,
                           double MaxFreq,
                           double MinAmpl,
                           double MaxAmpl,
                           BOOL IsPs,
                           double *pPeakFreq,
                           double *pPeakAmpl,
                           BOOL * pFoundOne );
```

Searches for a jitter amplitude peak within the defined limits

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`MinFreq` = min modulation frequency limit, in Hertz
`MaxFreq` = max modulation frequency limit, in Hertz
`MinAmpl` = min amplitude limit
`MaxAmpl` = max amplitude limit
`IsPs` = TRUE if amplitudes are sent and to be returned as picoseconds, else %UI
`pPeakFreq` = pointer to the modulation frequency of the detected peak, in Hertz
`pPeakAmpl` = pointer to the amplitude of the detected peak
`pFoundOne` = TRUE if a peak was detected within the specified limits

REMARKS:

`MinFreq` and `MaxFreq` must lie within `*pMinDataHz` and `*pMaxDataHz` returned at session start.

Keep the search window small -- this routine isn't very smart.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

```
CRUSTAT CruGetJitSpecResults( hCRU h,
                              double MinPlotHz,
                              double MaxPlotHz,
                              long Bins,
                              BOOL IsLogHz,
                              BOOL IsLogAmpl,
                              BOOL IsPs,
                              long * pNumScans,
                              double * pHzPerBin,
                              double * pMaxAmplPts,
                              double * pAvgAmplPts,
                              double * pMinAmplPts );
```

Gets the current results for the Jitter Spectrum session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`MinPlotHz` = min modulation frequency to be plotted
`MaxPlotHz` = max modulation frequency to be plotted
`Bins` = num plot area pixels in horizontal dir.
`IsLogHz` = TRUE if need to bin in Log10 mode
`IsLogAmpl` = TRUE if need to take Log10 of amplitude
IsPs = TRUE if need to return values in picosecs, else %UI
`pNumScans` = pointer to the total number of scans since session start
`pHzPerBin` = pointer to the Hz-per-Bin
`pMaxAmplPts` = pointer to an array of max amplitudes encountered in entire session
`pAvgAmplPts` = pointer to an array of avg amplitudes
`pMinAmplPts` = pointer to an array of min amplitudes encountered in entire session

REMARKS:

This routine may block on a mutex if the data buffers are being updated.

The raw spectrum results are decimated to the requested number of `Bins`,
within the `MinPlotHz` and `MaxPlotHz` range requested (may be a subset of total data range).

The complete dataset can be retrieved if `Bins == NumPts` specified at session start,
and `MinPlotHz == *pMinDataHz, MaxPlotHz == *pMaxDataHz` returned at session start.

`MinPlotHz` and `MaxPlotHz` will be clipped if they define a range outside of
`*pMinDataHz` and `*pMaxDataHz.`

`pMaxAmplPts[]` and `pMinAmplPts[]` describe the total envelope of the session data.
`pAvgAmplPts[bin]` is actually the maximum value in the bin for the averaged data set.

SEE ALSO:

`CruStartJitSpecSession, CruGetJitSpecStatus,`
`CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,`
`CruGetJitSpecMeasurement, CruGetJitSpecPeak,`
`CruClearJitSpecSessionData,`
`CruExportJitSpecCsvFile, CruImportJitSpecCsvFile`

```
CRUSTAT CruGetJitSpecStatus( hCRU h,
                             long * pPcntComplete );
```

Retrieves data acquisition progress information

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pPcntComplete` = pointer to scan completion percentage (returns 0 to 100)

REMARKS:

The client should poll this until 100% is reached, then call `CruGetJitSpecResults()`
If the session is paused or stopped, this routine returns `*pPcntComplete == -1`.

SEE ALSO:

```
      CruStartJitSpecSession,
      CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
      CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
      CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

```
CRUSTAT CruGetLockCount( hCRU h,
                         long * pVal );
```

Retrieves the number of times lock has been achieved since last reset of lock counter

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to lock count variable

REMARKS:

Use the lock count to monitor the stability of a clock recovery setup over long periods.

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruResetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockState, CruMapLockMode
```

```
CRUSTAT CruGetLockMode( hCRU h,
                        CRU_LOCK_MODE * pVal );
```

Retrieves the current lock mode setting from the BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a lock mode variable

REMARKS:

The lock modes available are:

`CRU_LOCK_MODE_MANUAL`	does not attempt to relock to data input until ordered to
`CRU_LOCK_MODE_AUTO`	tries to relock automatically anytime lock is lost
`CRU_LOCK_MODE_NARROW`	like auto mode, but limits search to a narrow freq. range

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruGetLockCount, CruResetLockCount
CruSetLockMode
CruMapLockState, CruMapLockMode
```

```
CRUSTAT CruGetLockRangeHZ( hCRU h,
                           double * pVal );
```

Retrieves the current lock range setting, in Hertz

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a lock range variable

REMARKS:

The lock range is the +/- frequency about the nominal data frequency to search for lock

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

`CRU_DBASE_MIN_LOCKRANGE_HZ`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MAX_LOCKRANGE_HZ`	Maximum allowed setting at the current nominal frequency

Where `MIN < MAX`

SEE ALSO:

```
CruSetFrequencyHz, CruGetFrequencyHz
CruGetMeasuredDataRateHz
CruSetLockRangeHZ
CruGetDBaseFloat
```

```
CRUSTAT CruGetLockState( hCRU h,
                         CRU_LOCK_STATE * pVal );
```

Retrieves the lock state of the BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a lock state variable

REMARKS:

The lock state will be one of the following:

CRU_LOCK_STATE_UNLOCKED	unlocked, and inactive
CRU_LOCK_STATE_ACQUIRING_LOCK	unlocked, but actively attempting to lock
CRU_LOCK_STATE_LOCKED	locked
CRU_LOCK_STATE_LOCKED_HIGH_JITTER	locked, but lock state is unstable (may be false lock)

SEE ALSO:

```
CruAcquireLock
CruGetLockCount, CruResetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockState, CruMapLockMode
```

```
CRUSTAT CruGetMeasuredDataRateHZ( hCRU h,
                                  double * pVal );
```

Retrieves the measured data frequency, in Hertz

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to measured data rate variable

REMARKS:

The measured data rate should be close to the nominal frequency setting, if the BERTScope CR has achieved true lock to the data input.

SEE ALSO:

```
CruSetFrequencyHz, CruGetFrequencyHz
CruSetLockRangeHZ, CruGetLockRangeHZ
CruGetDBaseFloat
```

```
CRUSTAT CruGetMeasuredEdgeDensityPCNT( hCRU h,
                                       double * pVal );
```

Measures the edge density of the incoming data, in percent.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the edge density measurement

REMARKS:

The measured edge density should be close to the nominal edge density,
if the BERTScope CR has achieved true lock to the data input.

The BERTScope CR will set an alarm flag if this is not the case:

CRU_ALARM_NOM_ED_MISMATCH
if nominal edge density was set with `CruSetNominalEdgeDensityPCNT`

CRU_ALARM_LAST_LOCK_ED_MISMATCH
if nominal edge density was set on lock

SEE ALSO:

```
CruMapEdgeDensityMode
CruSetEdgeDensityMode, CruGetEdgeDensityMode
CruSetNominalEdgeDensityPCNT, CruGetNominalEdgeDensityPCNT
```

```
CRUSTAT CruGetMeasuredPhaseErrorPk2PkPCNT( hCRU h,
                                           double * pVal );
```

Retrieves the measured peak-to-peak phase error, in percent of unit interval

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the peak-to-peak phase error measurement

REMARKS:

SEE ALSO:

CruGetMeasuredPhaseErrorRmsPCNT
CruSetPhaseErrorLimitPCNT, CruGetPhaseErrorLimitPCNT

```
CRUSTAT CruGetMeasuredPhaseErrorRmsPCNT( hCRU h,
                                         double * pVal );
```

Retrieves the measured RMS phase error, in percent of unit interval

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the RMS phase error measurement

REMARKS:

SEE ALSO:

CruGetMeasuredPhaseErrorPk2PkPCNT
CruSetPhaseErrorLimitPCNT, CruGetPhaseErrorLimitPCNT

```
CRUSTAT CruGetNames( CRUNAMES * pNames,
                     short * pNumDevices );
```

Retrieves count of BERTScope CR's connected, and the identification information for each

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pNames == NULL or pNumDevices == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

`pNames` = pointer to an array of CRUNAMES structures
`pNumDevices` = pointer to an device count variable

REMARKS:

Assumes Names array is at least CRU_CONST_MAX_USB_DEVICE_NUMBER elements.

The CRUNAMES structure is defined as:

```
typedef struct _CRUNAMES
{
    USHORT idVendor;
    USHORT idProduct;
    CRU_TYPE Type;
    char szSerialNumber[CRU_CONST_MAX_INFO_LEN];
    char szUnitName[CRU_CONST_MAX_INFO_LEN];
} CRUNAMES, *PCRUNAMES;
```

This routine is usually called prior to calling `CruOpenDevice.`

SEE ALSO:

`CruOpenDevice, CruCloseDevice`

```
CRUSTAT CruGetNominalEdgeDensityPCNT( hCRU h,
                                      double * pVal );
```

Retrieves the nominal (expected) edge density setting for the input data, in percent

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer the nominal edge density setting

REMARKS:

This setting is only used if the edge density mode is set to CRU_EDGE_DENSITY_MODE_NOM.

SEE ALSO:

```
CruMapEdgeDensityMode
CruSetEdgeDensityMode, CruGetEdgeDensityMode
CruSetNominalEdgeDensityPCNT
CruGetMeasuredEdgeDensityPCNT
```

```
CRUSTAT CruGetPeakingDB( hCRU h,
                         double * pVal );
```
Retrieves the loop response peaking setting, in dB

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a peaking varaible

REMARKS:

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

`CRU_DBASE_MIN_PEAKING_DB`	Minimum allowed at current nominal frequency
`CRU_DBASE_MINCAL_PEAKING_DB`	Minimum calibrated at current nominal frequency
`CRU_DBASE_MAXCAL_PEAKING_DB`	Minimum calibrated at current nominal frequency
`CRU_DBASE_MAX_PEAKING_DB`	Maximum allowed at current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

SEE ALSO:

`CruSetPeakingDB`
`CruGetDBaseFloat`

```
CRUSTAT CruGetPhaseErrorLimitPCNT( hCRU h,
                                   double * pVal );
```

Retrieves the phase error limit setting, in percent of unit-interval

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a phase error limit variable

REMARKS:

SEE ALSO:

CruGetMeasuredPhaseErrorPk2PkPCNT, CruGetMeasuredPhaseErrorRmsPCNT
CruSetPhaseErrorLimitPCNT

```
CRUSTAT CruGetSetupAutoSave( hCRU h,
                             BOOL * pVal );
```
Retrieve the state of the setup auto-save flag

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to a setup auto-save variable

REMARKS:

If the setup-auto-save flag is TRUE,
then the BERTScope CR will save its state to the setup `POWER_ON` slot automatically at power-down.

SEE ALSO:

`CruSaveSetup, CruRecallSetup`
`CruSetSetupAutoSave`

```
CRUSTAT CruGetSscWaveformMeasurement( hCRU h
                                      CRU_MEAS_TYPE MeasType,
                                      long AverageOver,
                                      double * pMeas );
```

Computes measurement averaged over last `AverageOver` histograms

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`MeasType` = type of SSC waveform measurement to make
`AverageOver` = requested number of histograms to average over (clipped by `n_histograms`)
`pMeas` = pointer to the requested measurement

REMARKS:

Valid SSC Waveform measurement types are:

`CRU_MEAS_TYPE_SSC_DEVMIN_PPM`
Average of Minimum Frequency relative to CR Nominal Frequency (ppm)

`CRU_MEAS_TYPE_SSC_DEVMAX_PPM`
Average of Maximum Frequency relative to CR Nominal Frequency (ppm)

`CRU_MEAS_TYPE_SSC_DEVMIN_HZ`
Average of Minimum Frequency relative to CR Nominal Frequency (Hertz)

`CRU_MEAS_TYPE_SSC_DEVMAX_HZ`
Average of Maximum Frequency relative to CR Nominal Frequency (Hertz)

`CRU_MEAS_TYPE_SSC_MODFREQ`
Modulation frequency, in Hertz

`CRU_MEAS_TYPE_SSC_NOMFREQ`
Clock frequency, in Hertz

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformResults,
CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

```
CRUSTAT CruGetSscWaveformResults( hCRU h,
                                  long Bins,
                                  long Shift,
                                  CRU_SSC_UNITS Units,
                                  long AverageOver,
                                  double * pHistogram,
                                  double * pNomFreq,
                                  long * pAveragedOver,
                                  long * pNumScans );
```

Gets the current results for the SSC Waveform session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`Bins` = num plot area pixels in horiz. dir. Must be the same as histogram_points on session start
`Shift` = circle-shift the result histogram by this amount (0 and `Bins`-1)
`Units` = CRU_SSC_UNITS_PPM or CRU_SSC_UNITS_HZ
`AverageOver` = requested number of histogram to average over (clipped by n_histograms)
`pHistogram` = pointer to the histogram data
`pNomFreq` = pointer to nominal frequency averaged over the requested number of histograms
`pAveragedOver` = pointer to the number of histograms averaged
`pNumScans` = pointer to the total scans since session start

REMARKS:

This routine frees all memory associated with a previous session, and resets the scan counter.

The resulting histogram is an average over last `AverageOver` histograms
(or less, if the acquisition has just started - the actual number is returned in `pAveragedOver`).

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformMeasurement,
CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

```
CRUSTAT CruGetSscWaveformStatus( hCRU h,
                                 long * pPcntComplete,
                                 int * n_zeros,
                                 int * n_corrected_zeros );
```

Retrieves data acquisition progress information

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pPcntComplete` = pointer to scan completion percentage (returns 0 to 100)
`n_zeros` = NULL (was used for debugging during development)
`n_corrected_zeros` = NULL (was used for debugging during development)

REMARKS:

The client should poll this until 100% is reached, then call `CruGetSscWaveformResults()`
If the session is paused or stopped, this routine returns `*pPcntComplete == -1`.
Parameters `n_zeros` and `n_corrected_zeros` are not currently used; pass NULL for both.

SEE ALSO:

`CruStartSscWaveformSession`,
`CruPauseSscWaveformSession`, `CruResumeSscWaveformSession`,
`CruStopSscWaveformSession`,
`CruGetSscWaveformResults`, `CruGetSscWaveformMeasurement`,
`CruClearSscWaveformSessionData`,
`CruExportSscWaveformCsvFile`, `CruImportSscWaveformCsvFile`

```
CRUSTAT CruGetStandardByName( hCRU h,
                              char * StdName );
```

Retrieves the Name of the currently selected Standard (or empty string if none selected)

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `StdName == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`StdName` = zero-terminated, case-sensitive `char` string

REMARKS:

StdName is set to `Name` string if current settings match a Standard

StdName is set to the empty string if current settings don't match a Standard

The CRUSTANDARD structure is defined as:

```
typedef struct _CRUSTANDARD
{
   char Name[12];
   double FrequencyHZ;
   double LockRangeHZ;
   double BandwidthHZ;
   double PeakingDB;
   double NominalEdgeDensityPCNT;
   char SSC; // Spread Spectrum Clocking Flag ( 0 = off, 1 = on )
} CRUSTANDARD, *PCRUSTANDARD ;
```

SEE ALSO:

```
CruGetStandardTable
CruSetStandardByName
CruAddModifyStandard, CruDeleteStandard
```

```
CRUSTAT CruGetStandardTable( hCRU h,
                             long * pCount,
                             CRUSTANDARD * pVal );
```

Retrieves the array of Standards stored in the BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pCount == NULL` or `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pCount` = pointer to Count variable
`pVal` = Pointer to array of Standards, with CRU_CONST_MAX_STANDARDS elements

REMARKS:

The CRUSTANDARD structure is defined as:

```
typedef struct _CRUSTANDARD
{
   char Name[12];
   double FrequencyHZ;
   double LockRangeHZ;
   double BandwidthHZ;
   double PeakingDB;
   double NominalEdgeDensityPCNT;
   char SSC; // Spread Spectrum Clocking Flag ( 0 = off, 1 = on )
} CRUSTANDARD, *PCRUSTANDARD ;
```

SEE ALSO:

```
CruSetStandardByName, CruGetStandardByName
CruAddModifyStandard, CruDeleteStandard
```

```
CRUSTAT CruGetSubrateClockAmplitudeMV( hCRU h,
                                       double * pVal );
```

Retrieves the current sub-rate clock output amplitude setting, in milliVolts.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to the current sub-rate clock amplitude setting, in milliVolts.

REMARKS:

Use `CruGetDBaseFloat` to check whether setting is within calibrated range:

CRU_DBASE_MIN_SUBCLK_AMP_V	Minimum allowed setting at the current nominal frequency
CRU_DBASE_MINCAL_SUBCLK_AMP_V	Minimum calibrated setting at the current nominal frequency
CRU_DBASE_MAXCAL_SUBCLK_AMP_V	Minimum calibrated setting at the current nominal frequency
CRU_DBASE_MAX_SUBCLK_AMP_V	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

Note: The limits are returned in Volts, not milliVolts!

SEE ALSO:

`CruSetSubrateClockAmplitudeMV`
`CruSetClockAmplitudeMV, CruGetClockAmplitudeMV`
`CruGetDBaseFloat`

```
CRUSTAT CruGetSubrateClockEnable( hCRU h,
                                  BOOL * pVal );
```

Checks whether or not the sub-rate clock output is enabled.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to Enabled flag.

REMARKS:

`*pVal == TRUE`	if sub-rate clock output is currently enabled
`*pVal == FALSE`	if sub-rate clock output is currently disabled

SEE ALSO:

`CruSetSubrateClockEnable`
`CruSetClockEnable, CruGetClockEnable`

```
CRUSTAT CruGetSubrateDivisor( hCRU h,
                              long * pVal );
```

Retrieves the current sub-rate clock divisor setting

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pVal == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pVal` = pointer to sub-rate divisor

REMARKS:

*pVal should be one of the valid divisors retrieved via `CruFetchSubrateTable`

SEE ALSO:

`CruFetchSubrateTable`
`CruSetSubrateDivisor`

```
CRUSTAT CruGetUnitName( hCRU h,
                        char * pBuf );
```

Retrieves the user-assignable name string for a BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pBuf` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pBuf = pointer to char buffer of at least CRU_CONST_MAX_INFO_LEN.

REMARKS:

This is the secondary identifier for a BERTScope CR.

If the BETScope CR's UnitName is blank,
then `GetUnitName` will return a default of "CR_17nnnn", where the "17nnnn" is the serial number.

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruSetUnitName
```

```
CRUSTAT CruHog( hCRU h,
                int ProcId );
```

Tries to acquire exclusive access to a BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_IS_BUSY, if another process has already claimed the BERTScope CR

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`ProcId` = system process ID

REMARKS:

Waits on a mutex for access to the BERTScope CR's 'Busy' flag.
If BERTScope CR is available, then reserves it with the ProcId passed in.
Finally, releases the mutex.

SEE ALSO:

`CruShare`
`CruNotBusy`

```
CRUSTAT CruImportJitSpecCsvFile( hCRU h,
                                 char * pszPath );
```

Loads a complete set of Jitter Spectrum data from a comma-separated-variable format ASCII file

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pszPath` = zero-terminated string designating disk path and filename

REMARKS:

This routine should not be called if a jitter spectrum session is running.
Any session data previously in memory is deleted.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile
```

```
CRUSTAT CruImportSscWaveformCsvFile( hCRU h,
                                     char * pszPath );
```

Loads a complete set of SSC Waveform data from a comma-separated-variable format ASCII file

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`pszPath` = zero-terminated string designating disk path and filename

REMARKS:

This routine should not be called if an SSC Waveform session is running.
Any session data previously in memory is deleted.

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruClearSscWqaveformSessionData,
CruExportSscWaveformCsvFile
```

`char * CruMapAlarm( unsigned long x );`

Converts CRU_ALARM codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

x = CRU_ALARM code to convert

REMARKS:

Performs the following conversions:

`CRU_ALARM_FLASH_ERROR`	to	"CRU_ALARM_FLASH_ERROR”
`CRU_ALARM_NOM_ED_MISMATCH`	to	"CRU_ALARM_NOM_ED_MISMATCH"
`CRU_ALARM_LAST_LOCK_ED_MISMATCH`	to	"CRU_ALARM_LAST_LOCK_ED_MISMATCH"
`CRU_ALARM_BANDWIDTH_UNCAL`	to	"CRU_ALARM_BANDWIDTH_UNCAL"
`CRU_ALARM_PEAKING_UNCAL`	to	“CRU_ALARM_PEAKING_UNCAL"

SEE ALSO:

`CruGetAlarms`
`CruGetEventsAndAlarms`
`CruMapEvent`

`char * CruMapDBase( CRU_DBASE code );`

Converts CRU_DBASE codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

`code` = CRU_DBASE code to convert

REMARKS:

Performs the following conversions:

CRU_DBASE_MIN_FREQ_HZ	to	"MIN_FREQ"
CRU_DBASE_MINCAL_FREQ_HZ	to	"MINCAL_FREQ"
CRU_DBASE_MAXCAL_FREQ_HZ	to	"MAXCAL_FREQ"
CRU_DBASE_MAX_FREQ_HZ	to	"MAX_FREQ"
CRU_DBASE_MIN_LOCKRANGE_HZ	to	"MIN_LOCKRANGE_HZ"
CRU_DBASE_MAX_LOCKRANGE_HZ	to	"MAX_LOCKRANGE_HZ"
CRU_DBASE_MIN_BW_HZ	to	"MIN_BW"
CRU_DBASE_MINCAL_BW_HZ	to	"MINCAL_BW"
CRU_DBASE_MAXCAL_BW_HZ	to	"MAXCAL_BW"
CRU_DBASE_MAX_BW_HZ	to	"MAX_BW"
CRU_DBASE_MIN_PEAKING_DB	to	"MIN_PEAKING_DB"
CRU_DBASE_MINCAL_PEAKING_DB	to	"MINCAL_PEAKING_DB"
CRU_DBASE_MAXCAL_PEAKING_DB	to	"MAXCAL_PEAKING_DB"
CRU_DBASE_MAX_PEAKING_DB	to	"MAX_PEAKING_DB"
CRU_DBASE_MIN_NOM_EDGE_DENSITY_PCNT	to	"MIN_NOM_EDGE_DENSITY_PCNT"
CRU_DBASE_MAX_NOM_EDGE_DENSITY_PCNT	to	"MAX_NOM_EDGE_DENSITY_PCNT"
CRU_DBASE_MIN_PHASE_ERROR_LIMIT_PCNT	to	"MIN_PHASE_ERROR_LIMIT_PCNT"
CRU_DBASE_MAX_PHASE_ERROR_LIMIT_PCNT	to	"MAX_PHASE_ERROR_LIMIT_PCNT"
CRU_DBASE_MIN_CLK_AMP_V	to	"MIN_CLK_AMP_V"
CRU_DBASE_MINCAL_CLK_AMP_V	to	"MINCAL_CLK_AMP_V"
CRU_DBASE_MAXCAL_CLK_AMP_V	to	"MAXCAL_CLK_AMP_V"
CRU_DBASE_MAX_CLK_AMP_V	to	"MAX_CLK_AMP_V"
CRU_DBASE_MIN_SUBCLK_AMP_V	to	"MIN_SUBCLK_AMP_V"
CRU_DBASE_MINCAL_SUBCLK_AMP_V	to	"MINCAL_SUBCLK_AMP_V"
CRU_DBASE_MAXCAL_SUBCLK_AMP_V	to	"MAXCAL_SUBCLK_AMP_V"
CRU_DBASE_MAX_SUBCLK_AMP_V	to	"MAX_SUBCLK_AMP_V"

SEE ALSO:

`CruGetDBaseLong, CruGetDBaseLongByName`
`CruGetDBaseFloat, CruGetDBaseFloatByName`

`char * CruMapEdgeDensityMode( CRU_EDGE_DENSITY_MODE x );`

Converts CRU_EDGE_DENSITY_MODE codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

`x` = CRU_EDGE_DENSITY_MODE code to convert

REMARKS:

Performs the following conversions:

`CRU_EDGE_DENSITY_MODE_NOM`	to	"NOMINAL"
`CRU_EDGE_DENSITY_MODE_ON_LOCK`	to	"ON_LOCK"

SEE ALSO:

```
CruSetEdgeDensityMode, CruGetEdgeDensityMode
CruSetNominalEdgeDensityPCNT, CruGetNominalEdgeDensityPCNT
CruGetMeasuredEdgeDensityPCNT
```

`char * CruMapEvent( unsigned long x );`

Converts CRU_EVENT codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

`x` = CRU_EVENT code to convert

REMARKS:

Performs the following conversions:

`CRU_EVENT_FREQ`	to	"CRU_EVENT_FREQ"
`CRU_EVENT_LOOP`	to	"CRU_EVENT_LOOP"
`CRU_EVENT_LOCKPARAM`	to	"CRU_EVENT_LOCKPARAM"
`CRU_EVENT_CLKOUT`	to	"CRU_EVENT_CLKOUT"
`CRU_EVENT_SUBOUT`	to	"CRU_EVENT_SUBOUT"
`CRU_EVENT_STANDARD`	to	"CRU_EVENT_STANDARD"
`CRU_EVENT_LOCK`	to	"CRU_EVENT_LOCK"
`CRU_EVENT_SETUP`	to	"CRU_EVENT_SETUP"
`CRU_EVENT_ALARM`	to	"CRU_EVENT_ALARM"

SEE ALSO:

`CruGetAlarms`
`CruGetEventsAndAlarms`
`CruMapAlarm`

`char * CruMapLockMode( CRU_LOCK_MODE x );`

Converts CRU_LOCK_MODE codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

`x` = CRU_LOCK_MODE code to convert

REMARKS:

Performs the following conversions:

`CRU_LOCK_MODE_MANUAL`	to	"MANUAL"
`CRU_LOCK_MODE_AUTO`	to	"AUTO"
`CRU_LOCK_MODE_NARROW`	to	"NARROW"

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruGetLockCount, CruResetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockState
```

`char * CruMapLockState( CRU_LOCK_STATE x );`

Converts CRU_LOCK_STATE codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

`x` = CRU_LOCK_STATE code to convert

REMARKS:

Performs the following conversions:

`CRU_LOCK_STATE_UNLOCKED`	to	"UNLOCKED"
`CRU_LOCK_STATE_ACQUIRING_LOCK`	to	"ACQUIRING_LOCK"
`CRU_LOCK_STATE_LOCKED`	to	"LOCKED"
`CRU_LOCK_STATE_LOCKED_HIGH_JITTER`	to	"LOCKED_HIGH_JITTER"

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruGetLockCount, CruResetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockMode
```

```
char * CruMapStatus( hCRU h,
                     CRUSTAT x );
```
Converts CRUSTAT codes into strings, for debugging messages

RETURNS:

Statically allocated, zero-terminated string

PARAMETERS:

h = handle to a BERTScope CR
x = CRUSTAT code to convert

REMARKS:

Performs the following conversions:

Defaults	to	"CRUSTAT_UNKNOWN"
CRU_OK	to	"CRU_OK"
CRU_BAD_HANDLE	to	"CRU_BAD_HANDLE"
CRU_NULL_POINTER	to	"CRU_NULL_POINTER"
CRU_INVALID_PARAMETER	to	"CRU_INVALID_PARAMETER"
CRU_NOT_IMPLEMENTED	to	"CRU_NOT_IMPLEMENTED"
CRU_NO_MEMORY	to	"CRU_NO_MEMORY"
CRU_UNEXPECTED	to	"CRU_UNEXPECTED"
CRU_ABORT	to	"CRU_ABORT"
CRU_FAIL	to	"CRU_FAIL"
CRU_ACCESSDENIED	to	"CRU_ACCESSDENIED"
CRU_DEVICE_NOT_FOUND	to	"CRU_DEVICE_NOT_FOUND"
CRU_NO_LONGER_CONNECTED	to	"CRU_NO_LONGER_CONNECTED"
CRU_TOO_MANY_HANDLES	to	"CRU_TOO_MANY_HANDLES"
CRU_UNSTABLE_VAL	to	"CRU_UNSTABLE_VAL"
CRU_NOT_LOCKED	to	"CRU_NOT_LOCKED"
CRU_IS_BUSY	to	"CRU_IS_BUSY"
CRU_CLIPPED_INPUT	to	"CRU_CLIPPED_INPUT"
CRU_FW_TOO_OLD	to	"CRU_FW_TOO_OLD"
CRU_UNABLE_TO_MEASURE	to	"CRU_UNABLE_TO_MEASURE"
CRU_CALIBRATION_MISSING	to	"CRU_CALIBRATION_MISSING"
CRU_CALIBRATION_CORRUPT	to	“"CRU_CALIBRATION_CORRUPT"
CRU_FILE_NOT_FOUND	to	"CRU_FILE_NOT_FOUND"
CRU_FILE_FORMAT_ERROR	to	"CRU_FILE_FORMAT_ERROR"

Special case for `CRU_USB_ERROR` :

If hCRU is not NULL,
    maps to a string any System (using FormatMessage) or USB driver errors
else
    converts to "CRU_USB_ERROR"

SEE ALSO:

`FormatMessage`
USBD State and Status codes in Microsoft’s usbdi.h

`CRUSTAT CruNotBusy( hCRU h );`

Checks a BERTScope CR's 'Busy' flag to see if it can be shared.

RETURNS:

CRU_OK, if no error and it is OK to access BERTScope CR
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_IS_BUSY, if BERTscope CR has been reserved by another process

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

All this does is check a flag protected by a mutex.
A cleared flag does not guarantee exclusive access -- that is up to the applications sharing it!

SEE ALSO:

```
CruHog, CruShare
CruConnected
CruOperationComplete
```

```
CRUSTAT CruOpenDevice( char * szName,
                       hCRU * phCru );
```

Requests non-exclusive access to a BERTScope CR device with name of szName

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `phCru == NULL`
CRU_TOO_MANY_HANDLES, if the CruLib.dll's hCRU table is used up (highly unlikely!)
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

`szName` = zero-terminated string, or NULL
`phCru` = pointer to a handle to a BERTScopeCR

REMARKS:

This operation changes the state of a global table of device handles, which is protected by a mutex.

The `szName` may be on of the following:

serial number
user assigned unit name
NULL or empty string

If NULL or empty string, the first device found is opened.

`*phCru` is set to NULL if the open is unsuccessful.

You must call `CruCloseDevice` before your program exits to avoid a resource leak.

SEE ALSO:

`CruCloseDevice`
`CruGetNames`

```
CRUSTAT CruOperationComplete( hCRU h,
                              unsigned long * pOPC );
```

Checks if a state changing operation has completed

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pOPC == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pOPC = pointer to operation state flag

REMARKS:

*pOPC = 0 if operation is in progress, else 1

SEE ALSO:

`CruConnected`
`CruNotBusy`

`CRUSTAT CruPauseJitSpecSession( hCRU h );`

Pauses a running Jitter Spectrum session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are frozen.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

`CRUSTAT CruPauseSscWaveformSession( hCRU h );`

Pauses a running SSC Waveform session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are frozen.

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruResumeSscWaveformSession, CruStopSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

```
CRUSTAT CruRecallSetup( hCRU h,
                        CRU_SETUP newVal );
```

Restore a set of settings, given a setup code

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = setup code to restore

REMARKS:

A BERTScope CR can remember up to six sets of settings (5 are user-configurable), identified by the following setup codes:

`POWER_ON`	Settings restored at power-on
`SETUP_1`	User slot 1
`SETUP_2`	User slot 2
`SETUP_3`	User slot 3
`SETUP_4`	User slot 4
`FACTORY`	Restores factory default settings

The POWER_ON setup set is restored whenever the BERTScope CR is powered up.

If the setup-auto-save flag is TRUE,
then the BERTScope CR will save its state to the setup POWER_ON slot automatically at power-down.

SEE ALSO:

`CruSaveSetup`
`CruSetSetupAutoSave, CruGetSetupAutoSave`

`CRUSTAT CruResetLockCount( hCRU h );`

Resets the BERTScope CR's internal lock counter to zero

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

Reset lock counter prior to a lock stability test

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruGetLockCount
CruSetLockMode, CruGetLockMode
CruMapLockState, CruMapLockMode
```

`CRUSTAT CruResumeJitSpecSession( hCRU h );`

Resumes a paused Jitter Spectrum session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are resumed.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

`CRUSTAT CruResumeSscWaveformSession( hCRU h );`

Resumes a paused SSC Waveform session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are resumed.

SEE ALSO:

```
      CruStartSscWaveformSession, CruGetSscWaveformStatus,
      CruPauseSscWaveformSession, CruStopSscWaveformSession,
      CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
      CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

```
CRUSTAT CruSaveSetup( hCRU h,
                      CRU_SETUP newVal );
```

Save the current BERTScope CR settings to a setup ‘slot’

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = setup ‘slot’ code

REMARKS:

A BERTScope CR can remember 5 user-configurable sets of settings, identified by the following setup codes:

`POWER_ON`	Settings restored at power-on
`SETUP_1`	User slot 1
`SETUP_2`	User slot 2
`SETUP_3`	User slot 3
`SETUP_4`	User slot 4

SEE ALSO:

`CruRecallSetup`
`CruSetSetupAutoSave, CruGetSetupAutoSave`

```
CRUSTAT CruSetBandwidthHZ( hCRU h,
                           double newVal );
```

Sets the -3 dB point of the clock recovery loop response, in Hertz.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = new bandwidth setting, in Hertz

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_BW_HZ`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MINCAL_BW_HZ`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAXCAL_BW_HZ`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAX_BW_HZ`	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

If a setting is outside the allowed range, it is automatically clipped.

SEE ALSO:

`CruGetBandwidthHZ`
`CruGetDBaseFloat`

```
CRUSTAT CruSetClockAmplitudeMV( hCRU h,
                                double newVal );
```

Sets the clock output amplitude, in milliVolts.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` =

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

CRU_DBASE_MIN_CLK_AMP_V	Minimum allowed setting at the current nominal frequency
CRU_DBASE_MINCAL_CLK_AMP_V	Minimum calibrated setting at the current nominal frequency
CRU_DBASE_MAXCAL_CLK_AMP_V	Minimum calibrated setting at the nominal frequency
CRU_DBASE_MAX_CLK_AMP_V	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

If a setting is outside the allowed range, it is automatically clipped.

Note: The limits are returned in Volts, not milliVolts!

SEE ALSO:

`CruGetClockAmplitudeMV`
`CruSetSubrateClockAmplitudeMV, CruGetSubrateClockAmplitudeMV`
`CruGetDBaseFloat`

```
CRUSTAT CruSetClockEnable( hCRU h,
                           BOOL newVal );
```

Sets the clock output state.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = TRUE to enable clock output, FALSE to disable it

REMARKS:

SEE ALSO:

`CruGetClockEnable`
`CruSetSubrateClockEnable, CruGetSubrateClockEnable`

```
CRUSTAT CruSetEdgeDensityMode( hCRU h,
                               CRU_EDGE_DENSITY_MODE newVal );
```

Determines How the BERTScope CR chooses an nominal value for the edge density of the incoming data

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = the desired edge density mode setting

REMARKS:

The edge density modes currently defined are:

CRU_EDGE_DENSITY_MODE_NOM :
Use the nominal edge density set with `CruSetNominalEdgeDensityPCNT.`

CRU_EDGE_DENSITY_MODE_ON_LOCK :
Use the edge density that was measured at moment lock was achieved.

SEE ALSO:

```
CruMapEdgeDensityMode
CruGetEdgeDensityMode
CruSetNominalEdgeDensityPCNT, CruGetNominalEdgeDensityPCNT
CruGetMeasuredEdgeDensityPCNT
```

```
CRUSTAT CruSetFrequencyHZ( hCRU h,
                           double newVal );
```

Sets the nominal frequency of the data input stream, in Hertz

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = nominal data frequency, in Hertz

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_FREQ_HZ`	Minimum allowed setting
`CRU_DBASE_MINCAL_FREQ_HZ`	Minimum calibrated setting
`CRU_DBASE_MAXCAL_FREQ_HZ`	Minimum calibrated setting
`CRU_DBASE_MAX_FREQ_HZ`	Maximum allowed setting

Where `MIN <= MINCAL < MAXCAL <= MAX`

If a setting is outside the allowed range, it is automatically clipped.

SEE ALSO:

```
CruSetFrequencyHz
CruGetMeasuredDataRateHz
CruGetDBaseFloat
```

```
CRUSTAT CruSetLockMode( hCRU h,
                        CRU_LOCK_MODE newVal );
```

Set the lock mode

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = new CRU_LOCK_MODE

REMARKS:

The lock modes available are:

`CRU_LOCK_MODE_MANUAL`	does not attempt to relock to data input until ordered to
`CRU_LOCK_MODE_AUTO`	tries to relock automatically anytime lock is lost
`CRU_LOCK_MODE_NARROW`	like auto mode, but limits search to a narrow freq. range

SEE ALSO:

```
CruAcquireLock, CruGetLockState
CruGetLockCount, CruResetLockCount
CruGetLockMode
CruMapLockState, CruMapLockMode
```

```
CRUSTAT CruSetLockRangeHZ( hCRU h,
                           double newVal );
```

Sets the lock range, in Hertz

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = lock range setting, in Hertz

REMARKS:

The lock range is the +/- frequency about the nominal data frequency to search for lock

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_LOCKRANGE_HZ`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MIN_LOCKRANGE_HZ`	Maximum allowed setting at the current nominal frequency

Where `MIN < MAX`

If a setting is outside the allowed range, it is automatically clipped.

SEE ALSO:

```
CruGetLockRangeHZ
CruSetFrequencyHz, CruGetFrequencyHz
CruGetMeasuredDataRateHz
CruGetDBaseFloat
```

```
CRUSTAT CruSetNominalEdgeDensityPCNT( hCRU h,
                                      double newVal );
```

Set the nominal (expected) edge density setting for the input data, in percent

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = the nominal edge density for the input data stream, in percent

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_NOM_EDGE_DENSITY_PCNT`	Min allowed at the current nominal frequency
`CRU_DBASE_MAX_NOM_EDGE_DENSITY_PCNT`	Max allowed at the current nominal frequency

If a setting is outside the allowed range, it is automatically clipped.

This setting is only used if the edge density mode is set to CRU_EDGE_DENSITY_MODE_NOM.

SEE ALSO:

```
CruSetEdgeDensityMode , CruGetEdgeDensityMode
CruGetNominalEdgeDensityPCNT
CruGetMeasuredEdgeDensityPCNT
```

```
CRUSTAT CruSetPeakingDB( hCRU h,
                         double newVal );
```

Sets the loop peaking, in DB

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = loop peaking setting, in dB

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_PEAKING_DB`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MINCAL_PEAKING_DB`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAXCAL_PEAKING_DB`	Minimum calibrated setting at the nominal frequency
`CRU_DBASE_MAX_PEAKING_DB`	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

If a setting is outside the allowed range, it is automatically clipped.

SEE ALSO:

`CruGetPeakingDB`
`CruGetDBaseFloat`

```
CRUSTAT CruSetPhaseErrorLimitPCNT( hCRU h,
                                   double newVal );
```

Sets the maximum phase error allowed by the lock search algorithm, in percent of unit-interval

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = phase error limit setting, in percent of unit-interval

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_PHASE_ERROR_LIMIT_PCNT`	Minimum allowed setting
`CRU_DBASE_MAX_PHASE_ERROR_LIMIT_PCNT`	Maximum allowed setting

Where `MIN < MAX`

If a setting is outside the allowed range, it is automatically clipped.

SEE ALSO:

`CruGetMeasuredPhaseErrorPk2PkPCNT, CruGetMeasuredPhaseErrorRmsPCNT`
`CruGetPhaseErrorLimitPCNT`

```
CRUSTAT CruSetSetupAutoSave( hCRU h,
                             BOOL newVal );
```

Set the setup auto-save flag

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = TRUE to auto-save setup at power-down, else FALSE

REMARKS:

If the setup auto-save flag is TRUE,
then the BERTScope CR will save its state to the setup `POWER_ON` slot automatically at power-down.

SEE ALSO:

`CruSaveSetup`, `CruRecallSetup`
`CruGetSetupAutoSave`

```
CRUSTAT CruSetStandardByName( hCRU h,
                              char * StdName );
```

Selects one of the Standards stored in the BERTScope CR, restoring Standard settings

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `StdName == NULL`
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`StdName` = zero-terminated, case-sensitive, `char` string that identifies a Standard

REMARKS:

The CRUSTANDARD structure is defined as:

```
typedef struct _CRUSTANDARD
{
   char Name[12];
   double FrequencyHZ;
   double LockRangeHZ;
   double BandwidthHZ;
   double PeakingDB;
   double NominalEdgeDensityPCNT;
   char SSC; // Spread Spectrum Clocking Flag ( 0 = off, 1 = on )
} CRUSTANDARD, *PCRUSTANDARD ;
```

SEE ALSO:

```
CruGetStandardTable
CruGetStandardByName
CruAddModifyStandard, CruDeleteStandard
```

```
CRUSTAT CruSetSubrateClockAmplitudeMV( hCRU h,
                                       double newVal );
```

Sets the sub-rate clock output amplitude, in milliVolts.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = new amplitude, in milliVolts

REMARKS:

Use `CruGetDBaseFloat` to validate setting:

`CRU_DBASE_MIN_SUBCLK_AMP_V`	Minimum allowed setting at the current nominal frequency
`CRU_DBASE_MINCAL_SUBCLK_AMP_V`	Minimum calibrated setting at the current nominal frequency
`CRU_DBASE_MAXCAL_SUBCLK_AMP_V`	Minimum calibrated setting at the nominal frequency
`CRU_DBASE_MAX_SUBCLK_AMP_V`	Maximum allowed setting at the current nominal frequency

Where `MIN <= MINCAL < MAXCAL <= MAX`

If a setting is outside the allowed range, it is automatically clipped.

Note: The limits are returned in Volts, not milliVolts!

SEE ALSO:

`CruGetSubrateClockAmplitudeMV`
`CruSetClockAmplitudeMV, CruGetClockAmplitudeMV`
`CruGetDBaseFloat`

```
CRUSTAT CruSetSubrateClockEnable( hCRU h,
                                  BOOL newVal );
```

Sets the sub-rate clock output state.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = TRUE to enable sub-rate clock output, FALSE to disable it.

REMARKS:

SEE ALSO:

`CruGetSubrateClockEnable`
`CruSetClockEnable, CruGetClockEnable`

```
CRUSTAT CruSetSubrateDivisor( hCRU h,
                              long newVal );
```

Sets the current sub-rate clock divisor.

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_INVALID_PARAMETER, if the sub-rate divisor `newVal` is invalid
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`newVal` = a valid divisor

REMARKS:

Call `CruFetchSubrateTable` to retrieve a list of valid divisors.

SEE ALSO:

`CruFetchSubrateTable`
`CruGetSubrateDivisor`

```
CRUSTAT CruSetUnitName( hCRU h,
                        char * pBuf );
```

Sets the user-assignable name string for a BERTScope CR

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid
CRU_NULL_POINTER, if `pBuf` == NULL
Any other CRUSTAT returned by BERTScope CR firmware

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
pBuf = pointer to zero-terminated char string of max length CRU_CONST_MAX_INFO_LEN-1.

REMARKS:

This is the secondary identifier for a BERTScope CR.

If a blank or empty-string is sent,
then `GetUnitName` will return a default of "CR_17nnnn", where the "17nnnn" is the serial number.

SEE ALSO:

```
CruGetCapabilities, CruGetDeviceModel
CruGetDeviceName, CruGetDeviceNames
CruGetDeviceRev, CruGetDeviceSerialNumber
CruGetDeviceType
CruGetUnitName
```

```
CRUSTAT CruShare( hCRU h,
                  int ProcId );
```

Clears a BERTScope CR's 'Busy' flag, releasing it from exclusive ownership by a process

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`ProcId` = system process ID

REMARKS:

This routine never actually uses ProcId, so ANY process with a valid handle may clear the flag. This allows recovery from a dead-lock, but also permits abuse. BE CAREFUL!

Waits on a mutex for access to the BERTScope CR's 'Busy' flag.
Clears the BERTScope CR's 'Busy' flag
Releases the mutex

SEE ALSO:

`CruHog`
`CruNotBusy`

```
CRUSTAT CruStartJitSpecSession( hCRU h,
                                long NumAvgs,
                                CRU_JS_SCANRES_MODE ScanResMode,
                                CRU_JS_CRINIT_MODE CRInitMode,
                                double * pMinDataHz,
                                double * pMaxDataHz );
```

Starts continuous collection of jitter spectrum data

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
`NumAvgs` = number of scans averaged together to create spectra
`ScanResMode` = determines the number of data points collected per scan
`CRInitMode` = determines the CR control settings used during the session
`pMinDataHz` = pointer returning minimum modulation frequency for session, in Hertz
`pMaxDataHz` = pointer returning maximum modulation frequency for session, in Hertz

REMARKS:

This routine spawns some background threads to handle the jitter spectrum data acquisition, then immediately returns.

This routine sets the BUSY Flag, disabling the CR front panel.

While scanning continues until it is stopped, only the last `NumAvgs` scans are used.
Setting either 0 or 1 for `NumAvgs` has the same affect – a single pass of data is used.

Valid CRU_JS_SCANRES_MODE settings:

`CRU_JS_SCANRES_NORMAL`
Suitable resolution for peak detection but not integrated jitter measurements.

`CRU_JS_SCANRES_HIGHRES`
4x the `NORMAL` resolution (and scan time). Use for accurate integrated jitter measurements.

`CRU_JS_SCANRES_AUTO`
Automatically uses `NORMAL` resolution for USB 1.x connections, and `HIGHRES` for USB 2.0.

The frequency limits returned are derived from the `ScanResMode` specified.

Valid CRU_JS_CRINIT_MODE settings:

`CRU_JS_CRINIT_NOSSC`
Selects a Clock Recovery Control settings suitable for most input data signals.

`CRU_JS_CRINIT_SSC`
Selects a Clock Recovery Control settings suitable for input data signals with SSC.

`CRU_JS_CRINIT_AUTO`
Automatically tries `NOSSC` settings, then tries `SSC` capable settings if lock is not achieved.

SEE ALSO:

```
CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession, CruStopJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

```
CRUSTAT CruStartSscWaveformSession( hCRU h,
                                    long acq_samples,
                                    long avg_count,
                                    long n_histograms,
                                    long histogram_points );
```

Starts continuous collection of SSC Waveform data

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR
acq_samples = number of frequency samples to be acquired per scan (normally 131070 or 65535)
avg_count = averaging ratio for internal algorithm (normally 16)
n_histograms = number of histograms averaged together (normally 20)
histogram_points = number of points in the averaged histogram (normally 256)

REMARKS:

This routine spawns some background threads to handle the SSC Waveform data acquisition, then immediately returns. Acquisition continues until it is stopped.

This routine sets the BUSY Flag, disabling the CR front panel.

SEE ALSO:

```
CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruStopSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

`CRUSTAT CruStopJitSpecSession( hCRU h );`

Stops a running Jitter Spectrum session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are signaled to terminate themselves. This routine does not return until the termination is complete.

SEE ALSO:

```
CruStartJitSpecSession, CruGetJitSpecStatus,
CruPauseJitSpecSession, CruResumeJitSpecSession,
CruGetJitSpecResults, CruGetJitSpecMeasurement, CruGetJitSpecPeak,
CruClearJitSpecSessionData,
CruExportJitSpecCsvFile, CruImportJitSpecCsvFile
```

`CRUSTAT CruStopSscWaveformSession( hCRU h );`

Stops a running SSC Waveform session

RETURNS:

CRU_OK, if no error
CRU_BAD_HANDLE, if handle to the BERTScope CR, h, is invalid

If CRUSTAT != CRU_OK, then use `CruMapStatus` to retrieve a diagnostic string.

PARAMETERS:

h = valid handle to a BERTScope CR

REMARKS:

The background data collection and processing threads are signaled to terminate themselves. This routine does not return until the termination is complete.

SEE ALSO:

```
CruStartSscWaveformSession, CruGetSscWaveformStatus,
CruPauseSscWaveformSession, CruResumeSscWaveformSession,
CruGetSscWaveformResults, CruGetSscWaveformMeasurement,
CruClearSscWaveformSessionData,
CruExportSscWaveformCsvFile, CruImportSscWaveformCsvFile
```

Index