PROTOCOL SOLUTIONS GROUP
3385 SCOTT BLVD
SANTA CLARA, CA 95054

Conquest

Application Program Interface (API) Manual

Version 7.40

For Software Version 7.40

September 2008

Document Disclaimer

The information in this document has been carefully checked and is believed to be reliable. However, no responsibility can be assumed for inaccuracies that may not have been detected.

LeCroy reserves the right to revise the information in this document without notice or penalty.

Trademarks

LeCroy and Conquest are trademarks of LeCroy Corporation.

Universal Serial Bus and On-The-Go are registered trademarks of USB-IF.

Microsoft and Windows are registered trademarks, and Visual C++ and Visual Basic are trademarks, of Microsoft Corporation.

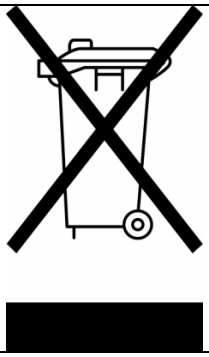
Borland and Delphi are trademarks of Borland Corporation.

All other trademarks and registered trademarks are property of their respective owners.

Copyright

Part number: 400-0066-001

WEEE Program



This electronic product is subject to disposal and recycling regulations that vary by country and region. Many countries prohibit the disposal of waste electronic equipment in standard waste receptacles.
For more information about proper disposal and recycling of your Catalyst product, please visit www.getcatalyst.com/recycle.

Introduction

The Conquest™ Application Program Interface (API) is a collection (library) of COM objects. The API enables programmers to use most of the Conquest Suite software functionality in their programs for the hardware, to execute a specific project with an event sequence and program selections. The software is for use with applications developed under COM Supported Platforms such as Microsoft® Visual C++™, Borland® Delphi™, and Microsoft Visual Basic™.

Using the Library

To use the library:

1. Copy the Conquest **SystemData** folder into the **system32** folder of the **Windows** directory of the PC.
2. Import **usb.tlb** into your software project.
3. Create a dispatch **USBGeneralService**s interface. See page 2.
4. Create a dispatch **USBEasyDataCapturePrj** interface. See page 5.
5. Use the API functions for required settings.

Note: You must run the Conquest Suite software before using the API.

Object Hierarchy

Objects are organized in a hierarchy of interfaces, with a set of methods for each.

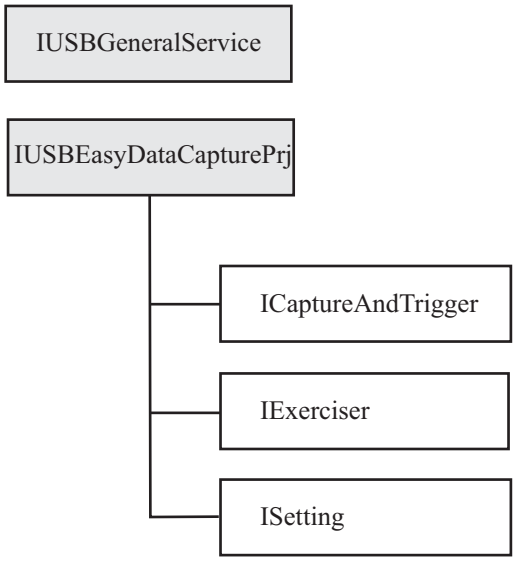


Figure 1 Object Hierarchy Definition

IUsbGeneral Services Interface

Method	Description
SetPort	Set hardware attached port.
IsHardwareStoped	Is hardware stopped?
SaveOutFiles	Save output file of last run project.
EnableProtocolError	Enable or disable protocol errors by Type.
IsEnableProtocolError	Is protocol error type enabled or not.
GetMaxMemorySize	Get the maximum size of memory.
AttachDevice	Attach device.
DetachDevice	Detach device.
SaveSmpAsText	Save captured data in a text file.

SetPort

Declaration: long SetPort(short sPortType)

Input(s): **sPortType** specifies hardware port type.
Valid port types are introduced in API Constants.

Return Value: Zero if the SetPort was successful. Otherwise, non-zero.
Each non-zero value indicates an error code.

Remark: This method sets the port type and connects the software to the hardware.

See Also API Errors

IsHardwareStoped

Declaration: long IsHardwareStoped(BOOL* bIsStoped)

Input(s): **bIsStoped** specifies hardware stop flag.

Return Value: Zero if the IsHardwareStoped was successful. Otherwise, non-zero.
Each non-zero value indicates an error code.

Remark: This function sets bIsStoped to TRUE if hardware is stopped.
Otherwise, sets to FALSE. Returns error code if an error occurs.

See Also API Errors

SaveOutFiles

Declaration: longSaveOutFiles()

Input(s): None

Return Value: Zero if the SaveOutFiles was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This method saves the output file for Protocol analyzer, Timing analyzer, or both.

See Also API Errors

EnableProtocolError

Declaration: long EnableProtocolError(short sProtocolErrorType, BOOL bEnable)

Input(s): **sProtocolErrorType** specifies type of protocol error. Valid types are introduced in API Constants.

bEnable specifies status of selected protocol error type.

Return Value: Zero if the EnableProtocolError was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: Enables or disables a protocol error type.

See Also API Errors

IsEnableProtocolError

Declaration: BOOL IsEnableProtocolError(short sProtocolErrorType)

Input(s) **sProtocolErrorType** specifies the type of protocol error. Valid types are introduced in API Constants.

Return Value: Zero if IsEnableProtocolError was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: Returns true if protocol error tspecified by sProtocolErrorType is enabled. Otherwise, it returns false.

See Also API Errors

GetMaxMemorySize

Declaration: long GetMaxMemorySize(long* lMaxSize)

Input(s): **lMaxSize** specifies maximum available memory size.

Return Value: Zero if GetMaxMemorySize was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This method sets lMaxSize to available memory size.

See Also API Errors

AttachDevice

Declaration: long AttachDevice()

Input(s): None

Return Value: Zero if AttachDevice was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function attaches the device to the analyzer.

See Also API Errors

AttachToDevice

Declaration: long AttachToDevice(LPCTSTR lpctstrMacAddress)

Input(s): **lpctstrMacAddress** specifies hardware MAC address.

Return Value: Zero if AttachToDevice was successful. Otherwise, non-zero. Each nonzero value indicates an error code.

Remark: This method attaches the software to the requested hardware specified by its MAC address.

See Also API Errors

GetDeviceIDList

Declaration: BSTR GetDeviceIDList(long nPortType)

Input(s): **nPortType** specifies hardware port type.
Valid port types are listed below.

Return Value: Each Device ID is separated with ',', and each Device ID number is in hexadecimal format, such as "0200000E850002D8,0400000E85000131".

Remark: This method enumerates all available devices on the selected port and returns available device IDs in string format.

CEIHAL_PORT_USB	0x02
CEIHAL_PORT_TCP	0x04
CEIHAL_PORT_EPP	0x08

DetachDevice

Declaration: long DetachDevice()

Input(s): None

Return Value: Zero if the DetachDevice was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function detaches the device from the analyzer.

See Also API Errors

SaveSmpAsText

Declaration: long SaveSmpAsText(LPCTSTR strSmpFilePath, LPCTSTR strTextFilePath)

Input(s): **strSmpFilePath** specifies path of sample file (**.smp**).

strTextFilePath specifies path of out text file.

Return Value: Zero if the SaveSmpAsText was successful. Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function converts **smp** file to text file and saves it.

See Also: API Errors

IusbEasyDataCapturePrj Interface

This interface allows you to create an Easy Data Capture project or open and modify an existing Easy Data Project.

Method	Description
GetCaptureAndTrigger	Get capture and trigger interface.
GetExerciser	Get easy exerciser interface.
GetSetting	Get project setting interface.
Open	Open easy capture project (**.ecp**) file.
Save	Save easy capture project (**.ecp**) file.
Run	Run easy capture project.
Stop	Stop easy capture project.
SetAnalyzerType	Set analyzer type.
GetAnalyzerType	Get analyzer type.
SetPAOutFile	Set Protocol Analyzer output file name.
GetPAOutFile	Get Protocol Analyzer output file name.
SetTAOutFile	Set Timing Analyzer output file name.
GetTAOutFile	Get Timing Analyzer output file name.
SetPreTriggger	Set pre-trigger value.
GetPreTriggger	Get pre-trigger value.
SetEntireMemmory	Set use entire available memory.
IsSetEntireMemmory	Is use entire available memory set?
SetSamplesNo	Set capture samples number.
GetSamplesNo	Get capture samples number.
SetRunExerciser	Set run the easy exerciser.
IsSetRunExerciser	Is run the easy exerciser set?

GetCaptureAndTrigger

Declaration:	ICaptureAndTrigger GetCaptureAndTrigger()
Input(s):	None
Return Value:	Capture and Trigger interface.
Remark:	Returns interface of CaptureAndTrigger object. Do not use CreateDispatch to create CaptureAndTrigger interface.

GetExerciser

Declaration:	IExerciser GetExerciser()
Input(s):	None
Return Value:	Exerciser interface
Remark:	Returns interface of Exerciser object. Do not use CreateDispatch to create Exerciser interface.

GetSetting

Declaration:	ISetting GetSetting()
Input(s):	None.
Return Value	Setting interface
Remark:	Returns interface of Setting object. Do not use CreateDispatch to create Setting interface.

Open

Declaration:	long Open(LPCTSTR strFilePath)
Input(s):	**strFilePath** specifies path to a file.
Return Value:	Zero if the open was successful. Otherwise, non-zero. Each non-zero value indicates an error code.
Remark:	Opens easy capture project (**.ecp**) file.
See Also	API Errors

Save

Declaration:	long Save(LPCTSTR strFilePath)
Input(s):	**strFilePath** points to the fully qualified path to save the file.
Return Value:	Zero if the save was successful. Otherwise, non-zero. Each non-zero value indicates an error code.
Remark:	Saves current easy capture project (**.ecp**) file.
See Also	API Errors

Run

Declaration:	long Run()
Input(s):	None
Return Value	Zero if project can run. Otherwise, non-zero. Each non-zero value indicates an error code.
Remark:	Checks data entries before run and runs easy data capture project.
See Also	API Errors

Stop

Declaration:	long Stop()
Input(s):	None
Return Value:	Zero if project can stop. Otherwise, non-zero. Each non-zero value indicates an error code.
Remark:	Stops easy data capture project.
See Also	API Errors

SetAnalyzerType

Declaration:	long SetAnalyzerType(short sAnalyzerType)
Input(s):	**sAnalyzerType** specifies type of analyzer. Valid analyzer types are introduced in API Constants.
Return Value:	Error code or zero (no error)
Remark:	This attribute specifies type of analyzer.
See Also	API Errors

GetAnalyzerType

Declaration:	short GetAnalyzerType()
Input(s):	None
Return Value:	Returns selected Analyzer type. Valid analyzer types are introduced in API Constants.
Remark:	Identifies type of analyzer.

SetPAOutFile

Declaration:	long SetPAOutputFile(LPCTSTR strPAOutputFile)
Input(s):	**strPAOutputFile** specifies protocol analyzer output file name.
Return Value:	Error code or zero (no error)
Remark:	Specifies name of protocol analyzer output file.
See Also	API Errors

GetPAOutFile

Declaration:	CString GetPAOutputFile()
Input(s):	None
Return Value:	Protocol Analyzer output file name
Remark:	Specifies name of protocol analyzer output file.

SetTAOutFile

Declaration	long SetTAOutputFile(LPCTSTR strTAOutputFile)
Input(s):	**strTAOutputFile** specifies timing analyzer output file name.
Return Value	Error code or zero (no error)
Remark:	Specifies name of timing analyzer output file.
See Also	API Errors

GetTAOutFile

Declaration:	CString GetTAOutputFile()
Input(s):	None
Return Value:	Timing Analyzer output file name
Remark:	Specifies name of timing analyzer output file.

SetPreTrigger

Declaration:	long SetPreTrigger(short sPreTigger)
Input(s):	**sPreTigger** sets pre-trigger percent. It can be between 1 and 99.
Return Value:	Error code or zero (no error)
Remark:	Specifies pre-trigger percent. Value of this attribute can be between 1 and 99.
See Also	API Errors

GetPreTrigger

Declaration:	short GetPreTrigger()
Input(s):	None
Return Value:	Pre-trigger value. It is a short value between 1 and 99.
Remark:	Returns pre-trigger percent setting.

SetEntireMemory

Declaration:	void SetEntireMemory(BOOL bEntireMemory)
Input(s):	**bEntireMemory** sets entire the memory for capturing.
Output(s)	None
Return Value	Error code or zero (no error)
Remark:	If set to TRUE, the system uses entire installed memory. Otherwise, it uses number specified by SamplesNo.

IsSetEntireMemory

Declaration:	BOOL IsSetEntireMemory()
Input(s):	None
Return Value:	Use entire memory flag status.
Remark:	Reports if entire memory flag is set.

SetSamplesNo

Declaration:	long SetSamplesNo(long lSamplesNo)
Input(s):	**lSamplesNo** specifies the number of samples to capture.
Return Value	Error code or zero (no error)
Remark:	Specifies the number of samples to capture. Note that if EntireMemory property has been set to FALSE, the system uses this attribute.
See Also	API Errors

GetSamplesNo

Declaration:	long GetSamplesNo()
Input(s):	None
Output(s)	None
Return Value	Samples number
Remark:	Returns the number of samples used in the capture.

SetRunExerciser

Declaration:	void SetRunExerciser(BOOL bRun)
Input(s):	**bRun** sets the run exerciser flag.
Outputs(s):	None
Remark:	If set to TRUE, exerciser also runs. If set to FALSE, only analyzer runs.

IsSetRunExerciser

Declaration:	BOOL IsSetRunExerciser()
Input(s):	None
Return Value	Run exerciser flag status.
Remark:	If set to TRUE, exerciser is set to run. If it set to FALSE, only analyzer is set to run.

CaptureAndTrigger

This interface sets the Easy Capture Project parameters (for example, Trigger point and capture data). (This is equivalent to Data Capture And Trigger page in Easy capture project in the software).

When you instantiate the IEsbEasyDataCapturePrj interface, the ICaptureAndTrigger interface instantiates automatically in IEsbEasyDataCapturePrj.

You can get ICaptureAndTrigger interface from IEsbEasyDataCapturePrj by calling the IEsbEasyDataCapturePrj ::GetCaptureAndTrigger method.

Method	Description
SetTriggerType	Set trigger type.
GetTriggerType	Get trigger type.
SetManualTrig	Set trigger manually.
IsSetManualTrig	Is set to trigger manually?
SetTriggerDeviceAddress	Set trigger device address.
GetTriggerDeviceAddress	Get trigger device address.
SetTriggerEndPoint	Set trigger endpoint.
GetTriggerEndPoint	Get trigger endpoint.
SetTriggerDataPacket	Set trigger data packet type.
GetTriggerDataPacket	Get trigger data packet type.
SetDataCapture1	Set data capture item 1.
GetDataCapture1	Get data capture item 1.
SetDataCapture2	Set data capture item 2.
GetDataCapture2	Get data capture item 2.
SetDataCapture3	Set data capture item 3.
GetDataCapture3	Get data capture item 3.
ExcludeNakedTransactions	Exclude NAK’ed transactions from capturing.
IsExcludeNakTransactions	Are NAK’ed transactions excludedfrom capturing?
ExcludeNYETedTransactions	Exclude NYET’ed transactions from capturing.
IsExcludeNYETedTransactions	Are NYET’ed transactions excluded from capturing?
ExcludeKeepAliveEvents	Exclude Keep Alive events from capturing.
IsExcludeKeepAliveEvents	Is exclude Keep Alive events from capturing

ExcludeSOFPackets	Exclude SOF packets from capturing.
IsExcludeSOFPackets	Are Keep SOF packets excluded from capturing?
SetTimmingPattern	Set timing analyzer pattern.
GetTimmingPattern	Get timing analyzer pattern.

SetTriggerType

Declaration: long SetTriggerType(short sTriggerType)

Input(s): **sTriggerType** specifies type of USB object on which to trigger the hardware. Valid trigger types are in API Constants.

Return Value: Error code or zero (no error)

Remark: Sets trigger type. Trigger types are as same as Trigger combo contents on Easy page.

See Also API Errors

GetTriggerType

Declaration: short GetTriggerType()

Input(s) None

Return Value Type of USB object on which the hardware is set to trigger

Remark: Returns trigger type.

SetManualTrig

Declaration: void SetManualTrig(BOOL bManualTrig)

Input(s): **bManualTrig** sets use of manual trigger in place of trigger type.

Return Value None

Remark: Sets the hardware to respond to manual trigger.

IsSetManualTrig

Declaration: BOOL IsSetManualTrig()

Input(s) None

Return Value Status of manual trig flag

Remark: Reports if the hardware is set to trigger manually.

SetTriggerDeviceAddress

Declaration:	long SetTiggerDeviceAddress(LPCTSTR strDeviceAddress)
Input(s):	**strDeviceAddress** specifies device address of trigger item. Length of this parameter is 7 characters in binary mode, where each character is one of 0, 1, x, or X (as do not care).
Return Value	Error code or zero (no error)
Remark:	Sets the device address on which to trigger. Note that the passed device address must be exclusive with the trigger type that is specified with the SetTrigger method beforehand.
See Also	API Errors

GetTriggerDeviceAddress

Declaration:	CString GetTriggerDeviceAddress()
Input(s):	None
Return Value:	Device address of trigger item. Length of this parameter is 7 characters in binary mode, where each character is one of 0, 1, x, or X (as do not care).
Remark:	Returns device address of trigger item.

SetTriggerEndPoint

Declaration:	long SetTriggerEndPoint(LPCTSTR strEndPoint)
Input(s)	**strEndPoint** specifies end point of trigger item.
Return Value:	Error code or zero (no error)
Remark:	This method sets endpoint of trigger item.
See Also	API Errors

GetTriggerEndPoint

Declaration:	CString GetTriggerEndPoint()
Input(s):	None
Return Value:	Endpoint number of trigger item
Remark:	Returns endpoint of trigger item.
See Also	API Errors

SetTriggerDataPacket

Declaration: long SetTriggerDataPacket(short sDataPacketType, LPCTSTR strData, BOOL bIsDataBlock)

Input(s): **sDataPacketType** sets data packet type. Valid data packet types are presented in API Constants.

strData specifies data payload of data packet.

bIsDataBlock specifies strData is data block item name or not.

Return Value: Error code or zero (no error)

Remark: Sets type and data of data packet on which to trigger.

Note: To trigger on data packet, set trigger type to data packet by calling the SetTrigger function with the API_TRIG_DATA_PACKETS parameter.

See Also API Errors

GetTriggerDataPacket

Declaration: void GetTriggerDataPacket(short* sDataPacketType, BSTR* strData, BOOL* bIsDataBlock)

Input(s): **sDataPacketType** specifies data packet type.

strData specifies data of data packet.

bIsDataBlock specifies whether *strData* is Data Block item name.

Return Value: Error code or zero (no error)

Remark: Returns the type and data of data packet on which to trigger.

See Also API Errors

SetDataCapture1

Declaration: long SetDataCapture1(short sCaptureType, LPCTSTR strDeviceAddress, LPCTSTR strEndPoint)

Input(s): **sCaptureType** specifies type of capture item 1.
Valid capture item types are in API Constants.

strDeviceAddress specifies device address of capture item if capture type is one of transaction types.

strEndPoint specifies endpoint of capture item if capture type is one of transaction types.

Return Value Error code or zero (no error)

Remark: Identifies type of data, device address, and endpoint number of data on which to capture. Type of data can be a value specified in the Data Capture combo box in the Easy data capture page. If type of data is set to everything or any transaction, the given device address and endpoint number do not have any effect.

See Also API Errors

GetDataCapture1

Declaration: void GetDataCapture1(short* sCaptureType, BSTR* strDeviceAddress, BSTR* strEndPoint)

Input(s): **sCaptureType** specifies type of capture item 1.
Valid capture item types are in API Constants.

strDeviceAddress specifies device address of capture item if capture type is one of transaction types.

strEndPoint specifies endpoint of capture item if capture type is one of transaction types.

Return Value: None

Remark: Returns data capture type 1, its device address, and its endpoint number.

SetDataCapture2

Declaration: long SetDataCapture2(short sCaptureType, LPCTSTR strDeviceAddress, LPCTSTR strEndPoint)

Input(s): **sCaptureType** specifies type of capture item 2.
Valid capture item types are in API Constants.

strDeviceAddress specifies device address of capture item if capture type is one of transaction types.

strEndPoint specifies endpoint of capture item if capture type is one of transaction types.

Return Value Error code or zero (no error)

Remark: Identifies the type of data, device address, and endpoint number of data on which to capture. Type of data can be a value specified in the Data Capture combo box on the Easy data capture page. If type of data is set to everything or any transaction, the given device address and endpoint number do have not any effect.

See Also API Errors

GetDataCapture2

Declaration: void GetDataCapture2(short* sCaptureType, BSTR* strDeviceAddress, BSTR* strEndPoint)

Input(s): **sCaptureType** specified type of capture item 2.
Valid capture item types are in API Constants.

strDeviceAddress specified device address of capture item if capture type is one of transaction types.

strEndPoint specified endpoint of capture item if capture type is one of transaction types.

Return Value None

Remark: Returns data capture type 2, its device address, and its endpoint number.

SetDataCapture3

Declaration: long SetDataCapture3(short sCaptureType, LPCTSTR strDeviceAddress, LPCTSTR strEndPoint)

Input(s): **sCaptureType** specifies type of capture item 3.
Valid capture item types are in API Constants.

strDeviceAddress specifies device address of capture item if capture type is one of transaction types.

strEndPoint specifies endpoint of capture item if capture type is one of transaction types.

Return Value Error code or zero (no error)

Remark: Sets the type of data, device address, and endpoint number of data on which to capture. The type of data can be a value specified in the Data Capture combo box on the Easy data capture page. If type of data is set to everything or any transaction, the given device address and endpoint number do have not any effect.

See Also API Errors

GetDataCapture3

Declaration: void GetDataCapture3(short* sCaptureType, BSTR* strDeviceAddress, BSTR* strEndPoint)

Input(s): **sCaptureType** specifies type of capture item 3.
Valid capture item types arein API Constants.

strDeviceAddress specifies device address of capture item if capture type is one of transaction types.

strEndPoint gets endpoint of capture item if capture type is one of transaction types.

Return Value: None

Remark: This method returns data capture type 3, its device address, and its endpoint number.

ExcludeNakedTransactions

Declaration: void ExcludeNAKedTransactions(BOOL bExclude)

Input(s): **bExclude** sets exclude NAK'ed transactions flag.

Return Value: None

Remark: Excludes NAK'ed transactions if TRUE.
Includes NAK'ed transactions if FALSE.

IsExcludeNakedTransactions

Declaration: BOOL IsExcludeNAKedTransactions()

Input(s): None

Return Value: Status of exclude NAK'ed transactions flag

Remark: Returns TRUE if exclude Nak transactions flag is selected.
Returns FALSE if not.

ExcludeNYETedTransactions

Declaration: void ExcludeNYETedTransactions(BOOL bExclude)

Input(s): **bExclude** sets exclude NYET'ed transactions flag.

Return Value None

Remark: Excludes NYET transactions if TRUE.
Includes NYET transactions if FALSE.

IsExcludeNYETedTransactions

Declaration: BOOL IsExcludeNYETedTransactions()

Input(s): None

Return Value: Status of exclude NYET'ed transactions flag.

Remark: Returns TRUE if exclude NYETed transactions flag is selected.
Returns FALSE if not.

ExcludeKeepAliveEvents

Declaration: void ExcludeKeepAliveEvents(BOOL bExclude)

Input(s) **bExclude** sets exclude KeepAlive events flag.

Return Value None

Remark: Excludes KeepAlive events if TRUE.
Includes KeepAlive events if FALSE.

IsExcludeKeepAliveEvents

Declaration: BOOL IsExcludeKeepAliveEvents()

Input(s): None

Return Value: Status of exclude KeepAlive Events flag

Remark: TRUE if exclude KeepAlive Events flag is set.
FALSE if not.

ExcludeSOFPackets

Declaration: void ExcludeSOFPackets(BOOL bExclude)

Input(s): **bExclude** sets exclude SOF packets flag.

Return Value: None

Remark: Excludes SOF packets if set to TRUE.
Includes SOF packets when set to FALSE.

IsExcludeSOFPackets

Declaration: BOOL IsExcludeSOFPackets()

Input(s): None

Return Value: Status of exclude SOF Packets flag

Remark: TRUE if exclude SOF Packets flag is set.
FALSE if not.

SetTimingPattern

Declaration: long SetTimingPattern(short sPatternType, LPCTSTR strPattern)

Input(s): **sPatternType** sets timing analyzer pattern type.
Valid types are in API Constants.

StrPattern sets timing analyzer pattern. Length of this pattern for full speed patterns is eight characters, and for high-speed patterns is four characters. Each character can be 0, 1, x, or X.

Return Value Error code or zero (no error)

Remark: Sets timing patterns for timing analyzer.

See Also API Errors

GetTimingPattern

Declaration: CString GetTimingPattern(short sPatternType)

Input(s): **sPatternType** sets timing analyzer pattern type.
Valid types are in API Constants.

Return Value: Pattern of specified pattern type

Remark: Returns pattern of specified timing analyzer pattern type.

IExerciser

This interface creates an exerciser program for an easy data capture project(This isequivalent to the Exerciser page in an Easy capture project in the software).

When you instantiate the IEsbEasyDataCapturePrj interface, the IExerciser interface instantiates automatically in IEsbEasyDataCapturePrj.

You can get the IExerciser interface from IEsbEasyDataCapturePrj by calling IEsbEasyDataCapturePrj ::GetExerciser method.

Method	Description
AddCommand	Add exerciser command.
AddUserDefineCommand	Add exerciser user-defined command.
InsertCommand	Insert exerciser command.
InsertUserDefineCommand	Insert exerciser user-defined command.
DeleteCommand	Delete exerciser command.
DeleteAllCommands	Delete allexerciser commands.
SetStartOfLoop	Set exerciser command to start of loop.
SetEndOfLoop	Set exerciser command to end of loop.
SetRepeatCommand	Set exerciser command to repeat.
FindDevice	Find attached devices.
GetFoundDevicesNo	Get found devices number.
GetDeviceEndPointsNo	Get endpoints number of a device.
GetEndPointType	Get type of endpoint.
GetEndPointDirection	Get direction of an endpoint.
GetEndPointNo	Get number of an endpoint.
IsDeviceHub	Is device HUB?
SetStartExerciserWith	Set exerciser start event type.

AddCommand

Declaration: long AddCommand(short sCommandType, short sDeviceIndex, const VARIANT& vtParam1, const VARIANT& vtParam2, const VARIANT& vtParam3)

Input(s): **sCommandType** selects the exerciser command type that you want to send to the device.
Valid command types are in API Constants.

sDeviceIndex sets the found device index.

vtParam1 sets the first parameter of exerciser command.

vtParam2 sets the second parameter of exerciser command.

vtParam3 sets the third parameter of exerciser command.

Return Value: Error code or zero (no error)

Remark: Adds a command to exerciser program. A command can use none, one, or more parameters defined in Exerciser Command Parameters.

See Also API Errors

AddUserDefineCommand

Declaration: long AddUserDefineCommand(short sSpeed, short sDeviceAddress, short sEndpointNo, short sEndpointType, short sCommandType, const VARIANT& vtParam1, const VARIANT& vtParam2, const VARIANT& vtParam3)

Input(s): **sSpeed** sets speed of device.
Valid speed types are in API Constants.

sDeviceAddress sets device address of the device.

sEndpointNo sets endpoint number.

sEndpointType sets type of the endpoint.
Valid endpoint type arein API Constants.

sCommandType sets the exerciser command type that you want to send to the device.
Valid command types are in API constants.

sDeviceIndex specifies device index.

vtParam1 specifies the first parameter of the exerciser command.

vtParam2 specifies the second parameter of the exerciser command.

vtParam3 specifies the third parameter of the exerciser command.

Return Value Error code or zero (no error).

Remark: Adds a user-defined command to the exerciser program. A command can use none, one, or more parameters defined in Exerciser Command Parameters

See Also API Errors

InsertCommand

Declaration: long InsertCommand(short sInsertAt, short sCommandType, short sDeviceIndex, const VARIANT& vtParam1, const VARIANT& vtParam2, const VARIANT& vtParam3)

Input(s): **sInsertAt** sets index of exerciser command for this command.

sCommandType sets exerciser command type that you want to send to the device. Valid types are defined in API Constants.

sDeviceIndex sets device index.

vtParam1 sets first parameter of the exerciser command.

vtParam2 sets second parameter of the exerciser command.

vtParam3 sets third parameter of the exerciser command.

Return Value Error code or zero (no error)

Remark: Adds a command to the exerciser program. A command can use none, one, or more parameters defined in Exerciser Command Parameters.

See Also API Errors

InsertUserDefineCommand

Declaration: long InsertUserDefineCommand(short sInsertAt, short sSpeed, short sDeviceAddress, short sEndpointNo, short sEndpointType, short sCommandType, const VARIANT& vtParam1, const VARIANT& vtParam2, const VARIANT& vtParam3)

Input(s) **sInsertAt** sets index of exerciser command fo which this command inserts a replacement.

sSpeed sets speed of device.
Valid speed types are in API Constants.

sDeviceAddress sets device address.

sEndpointNo sets endpoint number.

sEndpointType sets type of the endpoint.
Valid endpoint type are in API Constants.

sCommandType sets exerciser command type that you want to send to the device. Valid types are defined in API Constants.

sDeviceIndex sets device index.

vtParam1 sets first parameter of the exerciser command.

vtParam2 sets second parameter of the exerciser command.

vtParam3 sets third parameter of the exerciser command.

Return Value: Error code or zero (no error)

Remark: Adds a user-defined command to the exerciser program. A command can use none, one, or more parameters defined in Exerciser Command Parameters.

See Also API Errors

DeleteCommand

Declaration: long DeleteCommand(short sCommandIndex)

Input(s): **sCommandIndex** is index of exerciser command.

Return Value: Error code or zero (no error)

Remark: Deletes specified exerciser command (with index) from exerciser program.

See Also API Errors

DeleteAllCommands

Declaration: long DeleteAllCommands()

Input(s): None

Return Value: Error code or zero (no error)

Remark: Deletes all exerciser commands from exerciser program.

See Also API Errors

SetStartOfLoop

Declaration: long SetStartOfLoop(short sCommandIndex, long bSet)

Input(s): **sCommandIndex** sets index of exerciser command.

bSet sets status of start of loop flag for exerciser command.

Return Value: Error code or zero (no error)

Remark: Sets a specified command (with Index) as start of a loop.

See Also API Errors

SetEndOfLoop

Declaration: long SetEndOfLoop(short sCommandIndex, long bSet)

Input(s): **sCommandIndex** sets index of exerciser command.

bSet sets status of end of loop flag of exerciser command.

Return Value: Error code or zero (no error)

Remark: Sets a specified command (with Index) as end of a loop.

See Also: API Errors

SetRepeatCommand

Declaration: long SetRepeatCommand(short sCommandIndex, long bSet)

Input(s): **sCommandIndex** sets index of exerciser command.

bSet sets status of repeat flag of exerciser command.

Return Value: Error code or zero (no error)

Remark: Sets a specified command (with index) to repeat loop.

See Also: API Errors

FindDevice

Declaration: long FindDevice()

Inputs: None

Return Value: Error code or zero (no error)

Remark: This function finds any devices that are attached to the exerciser.

See Also: API Errors

GetFoundDevicesNo

Declaration: short GetFoundDevicesNo()

Input(s): None

Return Value: Number of devices found

Remark: This function returns the number of devices found.

Note: Use this function after a call to the FindDevice function.

GetDeviceEndPointsNo

Declaration: long GetDeviceEndPointsNo(short sDeviceIndex,
short* sEndPointsNo)

Input(s): **sDeviceIndex** sets index of device.

sEndPointsNo sets the number of endpoints of the device and fills in the function if the device index exists.

Return Value: Error code or zero (no error)

Remark: This function fills sEndPointsNo if the passed device index exists.

See Also API Errors

GetEndPointType

Declaration: long GetEndPointType(short sDeviceIndex, short sEndPointIndex,
short* sEndpointType)

Input(s): **sDeviceIndex** sets index of found device.

sEndPointIndex sets index of endpoint of a found device.

sEndpointType sets endpoint type of specified endpoint. This parameter fills if the device index and endpoint index exist.

Return Value: Error code or zero (no error)

Remark: This function fills sEndPointType if passed device index and endpoint index exist.

See Also: API Errors

GetEndPointDirection

Declaration: long GetEndPointDirection(short sDeviceIndex, short sEndPointIndex, short* sDirection)

Input(s): **sDeviceIndex** specifies the index of found device.

sEndPointIndex specifies the index of endpoint of a found device.

sDirection specifies direction of specified endpoint. This parameter fills if device index and endpoint index exist.

Return Value: Error code or zero (no error)

Remark: This function fills sDirection if passed device index and endpoint index exist.

See Also API Errors

GetEndPointNo

Declaration: long GetEndPointNo(short sDeviceIndex, short sEndPointIndex, short* sEndpointNo)

Input(s): **sDeviceIndex** specifies the index of found device.

sEndPointIndex specifies the index of endpoint of a found device.

sEndpointNo specifies the number of the specified endpoint. This parameter fills if device index and endpoint index exist.

Return Value: Error code or zero (no error)

Remark: This function fills sEndpointNo if passed device index and endpoint index exist.

See Also: API Errors

IsDeviceHub

Declaration: long IsDeviceHub(short sDeviceIndex, BOOL* bIsHub)

Input(s): **sDeviceIndex** specifies index of found device.

bIsHub specifies whether the device is a hub. This parameter fills if device index exists.

Return Value: Error code or zero (no error)

Remark: This function fills bIsHub if passed device index exists.

See Also: API Errors

SetStartExerciserWith

Declaration:	long SetStartExerciserWith(short sEventType)
Input(s):	**sEventType** specifies type of event with which the exerciser starts. Valid types are in API Constants.
Return Value:	Error code or zero (no error)
Remark:	This function sets the type of exerciser start event.
See Also:	API Errors

ISetting

This interface sets values of a project. (This is an equivalent to Setting page in Easy (or Advance) capture project in the software).

When you instantiate the IEsbEasyDataCapturePrj interface, the ISetting interface instantiates automatically in IEsbEasyDataCapturePrj .

You can get the ISetting interface from IEsbEasyDataCapturePrj by calling IEsbEasyDataCapturePrj::GetSetting method.

Method	Description
SetAnalyzerSpeed	Set analyzer speed.
GetAnalyzerSpeed	Get analyzer speed.
SetExerciserLoopCounter	Set exerciser loop counter.
GetExerciserLoopCounter	Get exerciser loop counter.
SetExternalTriggerType	Set external trigger type.
GetExternalTriggerType	Get external trigger type.
EnableNonStdClock	Enable non-standard clock.
IsEnableNonStdClock	Is non standard clock enabled?
SetNonStdClockRateType	Set non-standard clock rate type.
GetNonStdClockRateType	Get non-standard clock rate type.
EnableSecondPort	Enable second port.
IsEnableSecondPort	Is second port enabled?
SetSecondPortStore	Set second port store type.
GetSecondPortStore	Get second port store type.
EnableHighSpeedTestMode	Enable high-speed test mode.
IsEnableHighSpeedTestMode	Is high-speed test mode enabled?
SetInfiniteExerciserLoopCounter	Set exerciser loop counter to infinite.
IsInfiniteExerciserLoopCounter	Is exerciser loop counter set to infinite?

SetAnalyzerSpeed

Declaration: long SetAnalyzerSpeed(short sAnalyzerSpeed)

Input(s): **sAnalyzerSpeed** sets the speed of the analyzer.
Valid analyzer speed types are defined in API Constants.

Return Value: Zero if the SetAnalyzerSpeed was successful. Otherwise, non-zero.
Each non-zero value indicates an error code.

Remark: This function sets the speed of the analyzer.

See Also: API Errors

GetAnalyzerSpeed

Declaration: short GetAnalyzerSpeed()

Input(s): None

Return Value: Speed type of the analyzer.
Valid analyzer speed types are defined in API Constants.

Remark: This function returns the speed of the analyzer.

SetExerciserLoopCounter

Declaration: long SetExerciserLoopCounter(short sLoopCounter)

Input(s): **sLoopCounter** sets loop counter of exerciser. This value can be between 2 and 255.

Return Value: Zero if the SetExerciserLoopCounter was successful.
Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function sets the loop counter of the exerciser.

See Also API Errors

Get ExerciserLoopCounter

Declaration: short GetExerciserLoopCounter ()

Input(s) None

Return Value: Loop counter of the exerciser

Remark: Returns the loop counter of the exerciser.

See Also: API Errors

SetExternalTriggerType

Declaration: long SetExternalTriggerType(short sTriggerType)

Input(s): **sTriggerType** sets external trigger type.
Valid external trigger types are defined in API Constants.

Return Value: Zero if the SetExternalTriggerType was successful.
Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: Sets the external trigger type.

See Also: API Errors

GetExternalTriggerType

Declaration: short GetExternalTriggerType()

Input(s): **sTriggerType** is external trigger type.

Return Value: External trigger type

Remark: This function returns external trigger type.

See Also: API Errors

EnableNonStdClock

Declaration: void EnableNonStdClock(BOOL bEnable)

Input(s): **bEnable** sets status of non-standard clock flag.

Return Value: None

Remark: This function enables (disables) the non-standard clock.

IsEnableNonStdClock

Declaration: BOOL IsEnableNonStdClock()

Input(s): None

Return Value: Status of the non-standard clock flag

Remark: This function returns the non-standard clock status.

SetNonStdClockRateType

Declaration: long SetNonStdClockRateType(short sClockRateType)

Input(s): *s***ClockRateType** sets the non-standard clock rate.
Valid types are defined in API Constants.

Return Value: Zero if the SetNonStdClockRate was successful.
Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function sets rate of the non-standard clock.

See Also: API Errors

GetNonStdClockRateType

Declaration: short GetNonStdClockRateType()

Input(s): None

Return Value: Non-standard clock rate type.
Valid types are defined in API Constants.

Remark: This function returns the type of non-standard clock rate.

EnableSecondPort

Declaration: void EnableSecondPort(BOOL bEnable)

Input(s): **bEnable** set status of second port.

Return Value: None

Remark: This function enables (disables) second port.

IsEnableSecondPort

Declaration: BOOL IsEnableSecondPort()

Input(s): None

Return Value: TRUE if second port is enabled. Otherwise, returns FALSE.

Remark: This function returns status of second port.

SetSecondPortStore

Declaration: long SetSecondPortStore(short sStoreType)

Input(s): **sStoreType** sets the second port store type.
Valid types are defined in API Constants.

Return Value: Zero if the SetSecondPortStore was successful.
Otherwise, non-zero. Each non-zero value indicates an error code.

Remark: This function sets the second port store type.

See Also: API Errors

GetSecondPortStore

Declaration: short GetSecondPortStore()

Input(s): None

Return Value: Second port store type. Valid types are defined in API Constants.

Remark: This function returns the second port store type.

EnableHighSpeedTestMode

Declaration: void EnableHighSpeedTestMode(BOOL bEnable)

Input(s): **bEnable** sets status of the high-speed test mode flag.

Return Value: None

Remark: This function enables (disables) the high-speed test mode.

IsEnableHighSpeedTestMode

Declaration: BOOL IsEnableHighSpeedTestMode()

Input(s)*:* None

Return Value: Status of high-speed test mode flag

Remark: This function returns the status of the high-speed test mode flag.

SetInfiniteExerciserLoopCounter

Declaration: void SetInfiniteExerciserLoopCounter(BOOL bInfinite)

Input(s): **bInfinite** sets exerciser loop counter to infinite or finite value.

Return Value: None

Remark: This function sets the exerciser loop counter to infinite or finite count.

IsInfiniteExerciserLoopCounter

Declaration: BOOL IsInfiniteExerciserLoopCounter()

Input(s): None

Return Value: Status of exerciser loop counter

Remark: This function returns exerciser loop counter setting.

API Constants

Pre-defined API constants are in the **SBAEAPIConstants.h** file The following constants are used as parameters by some functions.

Analyzer Type definitions

API_MAX_ANLYZER_TYPES	3
API_ANLYZER_TYPE_PA	0 //Protocol Analyzer
API_ANLYZER_TYPE_TA	1 //Timing Analyzer
API_ANLYZER_TYPE_BOTH	2

Analyzer Speed definitions

API_MAX_ANLYZER_SPEEDS	3
API_ANLYZER_SPEED_HIGH	0
API_ANLYZER_TYPE_FULL_LOW	1
API_ANLYZER_TYPE_AUTO	2

Trigger Items definitions

API_MAX_TRIG_ITEMS	26
API_TRIG_SNAP_SHOT	0
API_TRIG_ANY_TRANS	1
API_TRIG_DATA_PACKETS	2
API_TRIG_BUS_RESET	3
API_TRIG_RESUME	4
API_TRIG_SUSPEND	5
API_TRIG_CHIRP	6
API_TRIG_HNP	7
API_TRIG_SRP	8
API_TRIG_SESSION	9
API_TRIG_PE	10 //Protocol Error
API_TRIG_SETUP_TRANS	11
API_TRIG_IN_TRANS	12
API_TRIG_OUT_TRANS	13
API_TRIG_PING_TRANS	14
API_TRIG_SETUP_SPLIT_TRANS	15
API_TRIG_BULK_IN_SPLIT_TRANS	16
API_TRIG_BULK_OUT_SPLIT_TRANS	17
API_TRIG_INT_IN_SPLIT_TRANS	18
API_TRIG_INT_OUT_SPLIT_TRANS	19
API_TRIG_ISO_IN_SPLIT_TRANS	20
API_TRIG_ISO_OUT_SPLIT_TRANS	21
API_TRIG_SETUP_PREAMBLE_TRANS	22
API_TRIG_ASYNC_IN_PREAMBLE_TRANS	23
API_TRIG_ASYNC_OUT_PREAMBLE_TRANS	24
API_TRIG_TA_PATTERNS	25

Data Packet Type definitions

API_MAX_TRIG_DATA_PACKET_TYPES	5
API_DATA_PACKET_TYPE_ANY	0
API_DATA_PACKET_TYPE_DATA0	1
API_DATA_PACKET_TYPE_DATA1	2
API_DATA_PACKET_TYPE_DATA2	3
API_DATA_PACKET_TYPE_MDATA	4

Timing Analyzer Pattern Type definitions

API_MAX_TA_PATTERN_TYPES	5
API_TA_FLS_TRIG_PATTERN_DP	0 //Full/Low Speed D+ Pattern
API_TA_FLS_TRIG_PATTERN_DM	1 //Full/Low Speed D- Pattern
API_TA_FLS_TRIG_PATTERN_RCV	2 //Full/Low Speed RCV Pattern
API_TA_HS_TRIG_PATTERN_DP	3 //High Speed D+ Pattern
API_TA_HS_TRIG_PATTERN_DM	4 //High Speed D- Pattern

Capturing Items definitions

API_MAX_CAPTURE_ITEMS	17
API_CAP_NO_CAPTURE	0 //None
API_CAP_ANY	1 //Everything
API_CAP_ANY_TRANS	2 //Any Transactions
API_CAP_SETUP_TRANS	3
API_CAP_ASYNC_IN_TRANS	4
API_CAP_ASYNC_OUT_TRANS	5
API_CAP_PING_TRANS	6
API_CAP_SETUP_SPLIT_TRANS	7
API_CAP_BULK_IN_SPLIT_TRANS	8
API_CAP_BULK_OUT_SPLIT_TRANS	9
API_CAP_INT_IN_SPLIT_TRANS	10
API_CAP_INT_OUT_SPLIT_TRANS	11
API_CAP_ISO_IN_SPLIT_TRANS	12
API_CAP_ISO_OUT_SPLIT_TRANS	13
API_CAP_SETUP_PREAMBLE_TRANS	14
API_CAP_ASYNC_IN_PREAMBLE_TRANS	15
API_CAP_ASYNC_OUT_PREAMBLE_TRANS	16

Exerciser Command Type definitions

API_MAX_EXR_COMMAND_TYPES	18
API_EXR_COMMAND_GET_CONFIGURATION	0
API_EXR_COMMAND_GET_DESCRIPTO	1
API_EXR_COMMAND_GET_INTERFACE	2
API_EXR_COMMAND_GET_DEVICE_STATUS	3
API_EXR_COMMAND_SET_ADDRESS	4
API_EXR_COMMAND_SET_CONFIGURATION	5
API_EXR_COMMAND_CLEAR_HUB_FEATURE	6
API_EXR_COMMAND_CLEARE_HUB_PORT_EATURE	7
API_EXR_COMMAND_GET_HUB_CLASS_DESCRIPTOR	8
API_EXR_COMMAND_GET_HUB_STATUS	9
API_EXR_COMMAND_SET_HUB_FEATURE	10
API_EXR_COMMAND_SET_HUB_PORT_FEATURE	11
API_EXR_COMMAND_CLEAR_HUB_TT_BUFFER	12
API_EXR_COMMAND_REASET_HUB_TT	13
API_EXR_COMMAND_GET_HUB_TT_STATUS	14
API_EXR_COMMAND_STOP_HUB_TT	15
API_EXR_COMMAND_DATA_IN	16
API_EXR_COMMAND_DATA_OUT	17

HUB Features definitions

API_MAX_HUB_FEATURES	2
API_HUB_FEATURE_LOCAL_POWER	0
API_HUB_FEATURE_OVER_CURRENT	1

HUB Port Features definitions

API_MAX_HUB_PORT_FEATURES	9
API_HUB_FEATURE_CONNECTION	0 // port connection
API_HUB_FEATURE_ENABLE	1 // port enable
API_HUB_FEATURE_SUSPEND	2 // port suspend
API_HUB_FEATURE_OVER_CURRENT	3 // port over current
API_HUB_FEATURE_RESET	4 // port reset
API_HUB_FEATURE_POWER	5 // port power
API_HUB_FEATURE_LS	6 // port low speed
API_HUB_FEATURE_TEST	7 // port test
API_HUB_FEATURE_INDICATOR	8 // port indicator

Port definitions

API_MAX_SBAE_PORTS	5
API_SBAE_PORT_ALL	0
API_SBAE_PORT_LPT1	1
API_SBAE_PORT_LPT2	2
API_SBAE_PORT_LPT3	3
API_SBAE_PORT_USB	4

Protocol Error Type definitions

API_MAX_PROTOCOL_ERROR_TYPES	13
API_PE_BIT_STUFF	0
API_PE_PID_CHECK	1
API_PE_PID_UNKNOWN	2
API_PE_SYNCH	3
API_PE_CRC5	4
API_PE_CRC16	5
API_PE_FRAME_LEN	6
API_PE_BABBLE	7
API_PE_DATA_TOGGLE	8
API_PE_FALSE_EOP	9
API_PE_ACT_LOSS	10 // loss of activity
API_PE_TIME_OUT	11
API_PE_BUS_ERR	12 //bus error

External Trigger Type definitions

API_MAX_EXTERNAL_TRIGGER_TYPE	5
API_EXT_TRG_MASK	0 // No external trigger
API_EXT_TRG_LEVEL_1	1 // Trigger on level one of signal
API_EXT_TRG_LEVEL_0	2 // Trigger on level zero of signal
API_EXT_TRG_EDGE_P	3 // Trigger on positive edge of signal
API_EXT_TRG_EDGE_N	4 // Trigger on negative edge of signal

Non Standard Clock Rate Type definitions

API_MAX_NON_STD_CLOCK_RATE_TYPES	5
API_NON_STD_CLOCK_RATE _125K	0
API_NON_STD_CLOCK_RATE _250K	1
API_NON_STD_CLOCK_RATE _500K	2
API_NON_STD_CLOCK_RATE _1M	3
API_NON_STD_CLOCK_RATE _EXTERNAL	4

Second Port Capturing Items definitions

API_MAX_SP_CAP_ITEMS	9
API_SP_CAP_ALL	0
API_SP_CAP_ALL_PACKETS	1
API_SP_CAP_NONE	2
API_SP_CAP_IDLE	3
API_SP_CAP_RESUME	4
API_SP_CAP_RESET	5
API_SP_CAP_DISCONNECT	6
API_SP_CAP_KEEPALIVE	7
API_SP_CAP_SUSPEND	8

Endpoint Type definitions

API_MAX_ENDPOINT_TYPES	4
API_ENDPOINT_TYPE_CONTROL	0
API_ENDPOINT_TYPE_ISO	1 // Isochronous
API_ENDPOINT_TYPE_BULK	2
API_ENDPOINT_TYPE_INT	3 // interrupt

Endpoint Direction Definitions

API_MAX_ENDPOINT_DIRECTIONS	3
API_DATA_DIRECTION_NONE	0
API_DATA_DIRECTION_IN	1
API_DATA_DIRECTION_OUT	2

Exerciser Start With event definition

API_MAX_EXRCISER_START_WITH_EVENTS	2
API_EXR_START_WITH_RESUME	0
API_EXR_START_WITH_RESET	1

API Errors

Error codes are in the **SBAEAPIErrors.h** file. They are returned by some functions upon an error condition.

IDE_NO_ERROR	0
IDE_INVALID_PORT_NO	1
IDE_ERROR_IN_DETECTING_HW	2
IDE_ERROR_HW_NOT_FOUND	3
IDE_CAN_NOT_OPEN_FILE	4
IDE_CAN_NOT_SAVE_FILE	5
IDE_TA_HS_TRIG_ON_SE1	6
IDE_HW_IS_BUSY	7
IDE_EXERCISER_PROGRAM_ERROR	8
IDE_SYSTEM_DATA_IS_CORRUPTED	9
IDE_CANNOT_RUN_PROJECT	10
IDE_NOT_AUTHORIZED_TA_SPEED	11
IDE_IS_NOT_ABLE_TO_SAVE	12
IDE_HW_IS_NOT_STOPPED	13
IDE_BAD_DATA_ENTRY	14
IDE_HIGH_SPEED_TEST_MODE_IN_FULL_LOW	15
IDE_NON_STD_CLOCK_IN_NO_FULL_LOW_SPEED	16
IDE_TIMMING_TRIG_WHEN_TA_ISNOT_ENABLE	17
IDE_NO_MATCH_TRIG_ITEM_WITH_ANALYZER_SPEED	18
IDE_NO_MATCH_CAP_ITEM_WITH_ANALYZER_SPEED	19
IDE_NO_MATCH_TRIG_CAPTURE_ITEMS_SPEED	20
IDE_NO_MATCH_EXCLUDE_NYET_WITH_ANALYZER_SPEED	21
IDE_NO_MATCH_EXCLUDE_KEEPALIVE_WITH_ANALYZER_SPEED	22
DE_NO_MATCH_EXCLUDE_KEEPALIVE_WITH_TRIG_OR_CAP_ITEMS_SPEED	23
IDE_NO_MATCH_EXCLUDE_NYET_WITH_TRIG_OR_CAP_ITEMS_SPEED	24
IDE_NO_MATCH_EXCLUDE_NAK_WITH_TRIG_ITEMS	25
IDE_NO_MATCH_EXCLUDE_NYET_WITH_TRIG_ITEMS	26
IDE_NO_MATCH_EXCLUDE_SOF_OR_KEEP_ALIVE_WITH_CAP_ITEMS	27
IDE_NO_MATCH_CAPTURE_ITEM0_WITH_OTHERS	28
IDE_CANNOT_FILL_CAPTURE_ITEM3_WHEN_CAPTURE_ITEM2_IS_NONE	29
IDE_CANNOT_SELECT_MORE_THAN_TWO_SPLIT	30
IDE_CANNOT_FILL_CAPTURE_ITEM3_WHEN_SELECT_SPLIT	31
IDE_FULL_LOW_SPEED_NOT_AUTHORIZED	32
IDE_HIGH_SPEED_NOT_AUTHORIZED	33
IDE_TA_FULL_LOW_SPEED_NOT_AUTHORIZED	34
IDE_TA_HIGH_SPEED_NOT_AUTHORIZED	35
IDE_SECOND_PORT_NOT_AUTHORIZED	36
IDE_EXTERNAL_CLOCK_NOT_AUTHORIZED	37
IDE_CANNOT_FIND_DEVICE	38
IDE_CANNOT_READ_DESCRIPTORS	39
IDE_ISNOT_MEMORY_ENOUGH	40
IDE_IS_SET_PORT_BEFORE	41
IDE_EXR_USE_HUB_COMMAND_FOR_NON_HUB_DEVICE	42

Exerciser Command Parameters

The following exerciser commands require no parameters. These commands do not use vtParam1, vtParam2, and vtParam3.

- API_EXR_COMMAND_GET_CONFIGURATION
- API_EXR_COMMAND_GET_INTERFACE
- API_EXR_COMMAND_GET_HUB_CLASS_DESCRIPTOR
- API_EXR_COMMAND_GET_HUB_STATUS
- API_EXR_COMMAND_CLEAR_HUB_TT_BUFFER
- API_EXR_COMMAND_GET_HUB_TT_STATUS
- API_EXR_COMMAND_STOP_HUB_TT

The following exerciser commands use vtParam1:

API_EXR_COMMAND_GET_DESCRIPTOR - vtParam1 should be a short variable that identifies descriptor type. Valid descriptor types are Device Descriptor and Configuration Descriptor.

API_EXR_COMMAND_SET_ADDRESS - vtParam1 should be a short variable that identifies new device address.

API_EXR_COMMAND_SET_CONFIGURATION - vtParam1 should be a short variable that identifies configuration value.

API_EXR_COMMAND_CLEAR_HUB_FEATURE - vtParam1 should be a short variable that identifies selector type. (Valid selectors are in API Constants.)

API_EXR_COMMAND_SET_HUB_ FEATURE - vtParam1 should be a short variable that identifies selector type. (Valid selectors are in API Constants.)

API_EXR_COMMAND_CLEAR_HUB_PORT_FEATURE - vtParam1 should be a short variable that identifues selector type. (Valid selectors are in API Constants.) vtParam2 should be a short variable that identifies port number.

API_EXR_COMMAND_SET_HUB_ PORT_FEATURE - vtParam1 should be a short variable that identifies selector type. (Valid selectors are in API Constants.)
vtParam2 should be a short variable that identifies port number.

API_EXR_COMMAND_DATA_OUT - vtParam1 should be a short variable that identifies endpoint index of selected device index. vtParam2 should be a string variable that identifes data to be output. This data is in hexadecimal base, and if it is a data block, should be the data block name. vtParam3 should be a boolean variable that identifies whether data is a data block item.

API_EXR_COMMAND_DATA_IN - vtParam1 should be a short variable that identifies endpoint index of selected device index.