e

CubeDLL Programmer's Guide

Version 1.02

Table of contents

1 Introduction

The CubeDLL (dynamic link library) provides an interface for communications with a Chembio Diagnostics GmbH Cube/MicroReader I device. This document describes the available functions and their usage as well as some general information.

The proper usage of this interface requires the installation of an FTDI driver, which should be supplied with the DLL file. For the latest driver version, please visit **https://ftdichip.com/drivers/**.

Code samples and function descriptions given in this document will be for Object Pascal and/or C/C++. The calling convention used for all functions is **stdcall**.

2 Preparational functions

This chapter describes functions used to set up the interface for proper functionality.

2.1 CL_PrepareInit

Description		Call first after loading library. Allocates memory for needed variables and initializes interface. **Not calling this function will lead to all other functions not working respectively returning FR_NOT_INITIALIZED.**
Definition	**Object Pascal**	function CL_PrepareInit(AllowBluetooth: UInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_PrepareInit(unsigned char AllowBluetooth);
Parameter 1	**Name**	AllowBluetooth
	Type	8bit unsigned int
	Description	Set to a value unequal 0 to allow usage of Bluetooth and BluetoothLE for device connection. Set to 0 if your devices do not support bluetooth connectivity or you are using cable connections exclusively.
Return		Will return 0 on success, for other values see **list of function return codes**.

2.2 CL_PrepareClose

Description		Always call before unloading library, frees all memory allocated by the interface. **Not calling this function before unloading the library will lead to memory leaks.**
Definition	**Object Pascal**	procedure CL_PrepareClose(); stdcall;
	C/C++	void __stdcall CL_PrepareClose();

2.3 CL_GetVersion

Description		Can be used to get version information about the interface.
Definition	**Object Pascal**	function CL_GetVersion(VersionMain: PUInt32; VersionSub: PUInt32; VersionRev: PUInt32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetVersion(unsigned int* VersionMain, unsigned int* VersionSub, unsigned int* VersionRev);
Parameter 1	**Name**	VersionMain
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the main version of the DLL. If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 2	**Name**	VersionSub
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the sub version of the DLL. If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 3	**Name**	VersionRev
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the version revision of the DLL. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

2.4 CL_SetMessageHandler

Description		Set a callback procedure for handling received messages (information, errors, progress, etc.). See **Messages** for further information.
Definition	**Object Pascal**	procedure CL_SetMessageHandler(Handler: TMessageHandler); stdcall;
	C/C++	void __stdcall CL_SetMessageHandler(Handler: tMessageHandler);
Parameter 1	**Name**	Handler
	Type	procedure TMessageHandler (MessageType: UInt32; MessageCode: UInt32; Data: UInt32); stdcall; void __stdcall TMessageHandler (unsigned int messageType; unsigned int messageCode; unsigned int data);
	Description	Callback-procedure to handle all kinds of messages from the interface (progress, errors, information). See **MessageHandler**.

2.5 CL_SetDateTimeHandler

Description		Set a Callback procedure for handling received date/time values (which will be sent after calling **CL_UpdateDateTime**, **CL_SetDateTime** and **CL_GetDateTime**).
Definition	**Object Pascal**	procedure CL_SetDateTimeHandler(Handler: TDateTimeHandler); stdcall;
	C/C++	void __stdcall CL_SetDateTimeHandler(Handler: tDateTimeHandler);
Parameter 1	**Name**	Handler
	Type	procedure (DateTime: Double); stdcall; void __stdcall (double dateTime);
	Description	Callback procedure to handle date/time messages from the interface.

2.6 CL_SetLicense

Description		Assign a license file to the interface. Needs to be called before a connection can be established. Without a valid license, the interface will not be able to establish a connection to a device. If no license was provided with the DLL or device, please contact Chembio Diagnostics GmbH.
Definition	**Object Pascal**	function CL_SetLicense(LicenseFileName: PAnsiChar; StrLength: UInt32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_SetLicense(char* LicenseFileName, unsigned int StrLength);
Parameter 1	**Name**	LicenseFileName
	Type	(Pointer) array of char
	Description	Location of the full (absolute) filepath- and name. If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 2	**Name**	StrLength
	Type	32bit unsigned int
	Description	Length of parameter 1.
Return		Will return 0 on success, for other values see **list of function return codes**.

2.7 CL_BTLEShowOnlyConnected

Description		Set up the BluetoothLE device detection to either detect all paired devices, or only those which have an active connection to the PC. Per default, the interface will detect all paired devices.
Definition	**Object Pascal**	function ML_BTLEShowOnlyConnected(ShowOnlyConnected: UInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall ML_BTLEShowOnlyConnected (char ShowOnlyConnected);
Parameter 1	**Name**	ShowOnlyConnected
	Type	8bit unsigned int
	Description	If set to 0, the device detection will detect all paired BluetoothLE devices, even if they are currently not connected. For any other value, only connected devices will be detected.
Return		Will return 0 on success, for other values see **list of function return codes**.

3 Detect devices

The interface can connect to devices through either DirectFTDI (special USB-cable with FTDI driver), classic Bluetooth (requires support by both the device and the PC) or BluetoothLE (requires support by the device and the PC as well as the 64bit version of the DLL).

Please be aware that for a device to be connectable via bluetooth, it must be paired first. Bluetooth is only supported for Windows 10, functionality on other versions of windows can not be guaranteed.

The interface will always search devices in all available channels at the same time (USB cable, classic Bluetooth and BluetoothLE), switching between different modi is not needed.

3.1 CL_GetDeviceCount

Description		Get the number of available devices. The interface will detect devices through all available connection methods (cable, classic Bluetooth and Bluetooth LE).
Definition	**Object Pascal**	function CL_GetDeviceCount(Count: PUint32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDeviceCount(unsigned int* Count);
Parameter 1	**Name**	Count
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the number of available devices. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

3.2 CL_GetDeviceInfo

Description		Get info for the device on the specified index. Please be aware that, since Cube/MRI devices are connected through a special cable, this function will only return the description and serial of the cable, not of the device (FTDI devices only). For Bluetooth devices, the description parameter will contain the name of the device (as advertised through Bluetooth), while the serial will be left empty.
Definition	**Object Pascal**	function CL_GetDeviceInfo(Index: UInt32; Description: PAnsiChar; Serial: PAnsiChar): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDeviceInfo(unsigned int Index, char* Description, char* Serial);
Parameter 1	**Name**	Index
	Type	32bit unsigned int
	Description	The index of the required device.
Parameter 2	**Name**	Description
	Type	(Pointer) array of char with **fixed length 64**
	Description	Location to store the description/name of the device (leftover array indices will be set to 0). If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 3	**Name**	Serial
	Type	(Pointer) array of char with **fixed length 64**
	Description	Location to store the serial number of the device (leftover array indices will be set to 0). If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 3	**Name**	ConnectionType
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the connection type of the device. Connection type can be: 0 – FTDI (cable) 1 – Bluetooth LE 2 – Classic Bluetooth If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4 Device communication

This chapter describes functions used to connect/disconnect a device as well as functions concerning the general device behaviour and setup.

4.1 CL_ConnectAndValidate

Description		Initiate a connection to the device with the given index. The progress of establishing the connection will be reported through **Messages**. The message **IM_CONNECTION_ESTABLISHED** will signal the successfully established connection. Call **CL_SetLicense** before to set a valid license for the device you want to connect to.
Definition	**Object Pascal**	function CL_ConnectAndValidate(Index: UInt32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_ConnectAndValidate(unsigned int Index);
Parameter 1	**Name**	Index
	Type	32bit unsigned int
	Description	The index of the required device.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.2 CL_Disconnect

Description		Close a currently opened connection. Will have no effect if no connection is open. The message **IM_DEVICE_DISCONNECTED** will signal the finished disconnect.
Definition	**Object Pascal**	function CL_Disconnect(): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_Disconnect();
Return		Will return 0 on success, for other values see **list of function return codes**.

4.3 CL_DeviceConnected

Description		Used to check if there is an open connection from the interface.
Definition	**Object Pascal**	function CL_DeviceConnected(Connected: PUInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_DeviceConnected(unsigned char* Connected);
Parameter 1	**Name**	Connected
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the connection state (0 if there is no established connection, 1 if there is one). If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.4 CL_GetState

Description		Get the current state of the interface/device. Please be aware that every change or update of a state will be relayed by the interface per **MessageHandler**.
Definition	**Object Pascal**	function CL_GetState(State: PUInt32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetState(unsigned int* State);
Parameter 1	**Name**	State
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the current state of the interface/device (see **list of states**). If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.5 CL_UpdateDateTime

Description		Update date and time of a connected device to the current system date and time. The device will then send the set date and time back through the callback function set by **CL_SetDateTimeHandler**.
Definition	**Object Pascal**	function CL_UpdateDateTime(): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_UpdateDateTime();
Return		Will return 0 on success, for other values see **list of function return codes**.

4.6 CL_SetDateTime

Description		Update date and time of a connected device to the given date and time. The device will then send the set date and time back (see **CL_SetDateTimeHandler**).
Definition	**Object Pascal**	function CL_SetDateTime(DateTime: Double): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_SetDateTime(double DateTime);
Parameter 1	**Name**	DateTime
	Type	64bit floating point
	Description	The date and time value to be set to the connected device.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.7 CL_GetDateTime

Description		Request current date and time of the connected device. The device will send the date and time back (see **CL_SetDateTimeHandler**).
Definition	**Object Pascal**	function CL_GetDateTime(): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDateTime();
Return		Will return 0 on success, for other values see **list of function return codes**.

4.8 CL_GetDeviceEEPROMVersion

Description		Retrieve the EEPROM version of the currently connected device.
Definition	**Object Pascal**	function CL_GetDeviceEEPROMVersion(EEPROMVersion: PUInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDeviceEEPROMVersion(unsigned char* EEPROMVersion);
Parameter 1	**Name**	EEPROMVersion
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the EEPROM version of the connected device. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.9 CL_GetDeviceSerial

Description		Retrieve the serial number of the connected device as formatted string.
Definition	**Object Pascal**	function CL_GetDeviceSerial(Serial: PAnsiChar; StrLength: PUint32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDeviceSerial(char* Serial, unsigned int* StrLength);
Parameter 1	**Name**	Serial
	Type	(Pointer) array of char
	Description	Location to store the device serial (as formatted text). Set nil/NULL to retrieve string length only.
Parameter 2	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 1. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.10 CL_GetDeviceFirmware

Description		Retrieve the firmware version of the connected device as formatted string.
Definition	**Object Pascal**	function CL_GetDeviceFirmware(Firmware: PAnsiChar; StrLength: PUInt32);: Uint32; stdcall;
	C/C++	unsigned int __stdcall CL_GetDeviceFirmware(char* Firmware, unsigned int* StrLength);
Parameter 1	**Name**	Firmware
	Type	(Pointer) array of char
	Description	Location to store the firmware version (as formatted text). Set nil/NULL to retrieve string length only.
Parameter 2	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 1. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.11 CL_SetButtonLock

Description		Lock or release device control through the device button. Success will be reported through **MessageHandler** (Message **IM_BUTTON_SET**).
Definition	**Object Pascal**	function CL_SetButtonLock(LockButton: UInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_SetButtonLock(unsigned char LockButton);
Parameter 1	**Name**	LockButton
	Type	8bit unsigned int
	Description	Set to 1 to lock the device button (device will not react to physical usage of the button). Any other value will release the lock and make the button useable.
Return		Will return 0 on success, for other values see **list of function return codes**.

4.12 CL_FormatDeviceDB

Description		Initialize formatting of the database of the currently connected device. Will delete all saved measurement results on the device. Success will be reported through message **IM_DB_FORMATTED**.
Definition	**Object Pascal**	function CL_FormatDeviceDB(): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_Format DeviceDB();
Return		Will return 0 on success, for other values see **list of function return codes**.

5 Measurement functions

All measurement results gotten from the device will be appended to a list handled by the interface. Functions that allow access to the entries of that list are described in chapters **Retrieving measurement information**, **Retrieving result information** and **Retrieving line information**.

The following functions allow the manipulation of that measurement list.

5.1 CL_StartMeasurement

Description		Initialize the execution of a measurement, either direct or timed. The test configuration will be read from an RFID tag. Progress will be reported through info messages (see **MessageHandler**). The measurement result will be added to the measure list handled by the interface, the interface will send a **IM_MEASUREMENT_DONE** message afterwards containing the index of the new measurement (for data retrieval).
Definition	**Object Pascal**	function CL_StartMeasurement(UseTimer: UInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_StartMeasurement(unsigned char UseTimer);
Parameter 1	**Name**	UseTimer
	Type	8bit unsigned int
	Description	If set to 1, a timed measurement will be started using the incubation time defined in the test configuration. In any other case, the incubation timer will be ignored and the measurement directly executed.
Return		Will return 0 on success, for other values see **list of function return codes**.

5.2 CL_StartMeasurementFromFile

Description		Initialize the execution of a measurement, either direct or timed. The test configuration will be read from the given file. Progress will be reported through info messages (see **MessageHandler**). The measurement result will be added to the measure list handled by the interface, which will send the message **IM_MEASUREMENT_DONE** afterwards containing the index of the new measurement (for data retrieval).
Definition	**Object Pascal**	function CL_StartMeasurementFromFile(UseTimer: UInt8; FilePathName: PAnsiChar; StrLength: UInt32): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_StartMeasurementFromFile(unsigned char UseTimer, char* FilePathName, unsigned int StrLength);
Parameter 1	**Name**	UseTimer
	Type	8bit unsigned int
	Description	If set to 1, a timed measurement will be started using the incubation time defined in the test configuration. In any other case, the incubation timer will be ignored and the measurement directly executed.
Parameter 2	**Name**	FilePathName
	Type	(Pointer) array of char
	Description	Location (absolute filepath) and name of the configuration file. If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 3	**Name**	StrLength
	Type	32bit unsigned int
	Description	Length of parameter 2.
Return		Will return 0 on success, for other values see **list of function return codes**.

5.3 CL_ReadDeviceDB

Description		Initialize reading of the device database (get all measurement results). Progress will be reported through progress messages (see **MessageHandler**). All received measurement results will be written to a list handled by the interface, from where the actual values can be retrieved through corresponding functions. Success will be signaled by message **IM_MEASURE_COUNT**, which also contains the amount of read measurements. Use **CL_GetMeasurementCount** afterwards to get the overall number of available measurements from the interface. Executing this function multiple times will not lead to measurements being present in the list more than one time. If a measurement is already in the list, it will be automatically skipped if read again.
Definition	**Object Pascal**	function CL_ReadDeviceDB(ClearMeasureList: UInt8): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_ReadDeviceDB(unsigned char ClearMeasureList);
Parameter 1	**Name**	ClearMeasureList
	Type	8bit unsigned int
	Description	If set to 1, the DLL-internal list of measurement results will be cleared before adding the newly read measurements.
Return		Will return 0 on success, for other values see **list of function return codes**.

5.4 CL_ClearMeasurementList

Description		Clear the list of measurements handled by the interface.
Definition	**Object Pascal**	function CL_ClearMeasurementList(): UInt32; stdcall;
	C/C++	unsigned int __stdcall CL_ClearMeasurementList();
Return		Will return 0 on success, for other values see **list of function return codes**.

6 Retrieving measurement information

Use the following functions to gain information on measurements. Using these functions requires a previous download of device data (through **CL_ReadDeviceDB**) or execution of one or more measurements through the interface (**CL_StartMeasurement** or **CL_StartMeasurementFromFile**).

6.1 CL_GetMeasurementCount

Description		Retrieve the amount of measurements currently available to the interface. Use **CL_ReadDeviceDB**, **CL_StartMeasurement**, **CL_StartMeasurementFromFile** and **CL_ClearMeasureList** to manipulate the list. Calling this function gives the range of indices available for all functions requiring a measurement index. When addressing these indices, please be aware that the first valid index is always 0.
Definition	**Delphi**	function CL_GetMeasurementCount(Count: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasureCount(unsigned int* Count);
Parameter 1	**Name**	Count
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the amount of measurement results currently available to the DLL. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.2 CL_GetMeasurementName

Description		Retrieve the name of the measurement at the given index.
Definition	**Delphi**	function CL_GetMeasurementName(MeasureIdx: UInt32; MeasureName: PAnsiChar; StrLength: PUInt32): UInt32); stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementName(unsigned int MeasureIdx, char* MeasureName, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	MeasureName
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the measurement name. Set nil/NULL to retrieve only the required length.
Parameter 3	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 2. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.3 CL_GetMeasurementID

Description		Retrieve the ID of the measurement at the given index.
Definition	**Delphi**	function CL_GetMeasurementID(MeasureIdx: UInt32; MeasureID: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementID(unsigned int MeasureIdx, char* MeasureID, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	MeasureID
	Type	(Pointer) array of char
	Description	Location to store the measurement ID (as formatted text). Set nil/NULL to retrieve only the required length.
Parameter 3	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 2. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.4 CL_GetMeasurementDateTime

Description		Retrieve the date and time of the measurement at the given index.
Definition	**Delphi**	function CL_GetMeasurementDateTime(MeasureIdx: UInt32; DateTime: PDouble): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementDateTime(unsigned int MeasureIdx, double* DateTime);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	DateTime
	Type	(Pointer) 64bit floating point
	Description	Location to store the date and time of the measurement execution on return. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.5 CL_GetMeasurementLotNr

Description		Retrieve the Lot-number of the measurement at the given index.
Definition	**Delphi**	function CL_GetMeasurementLotNr(MeasureIdx: UInt32; LotNr: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementLotNr(unsigned int MeasureIdx, char* LotNr, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LotNr
	Type	(Pointer) array of char
	Description	Location to store the Lot-number of the measurement. Set nil/NULL to retrieve only the required length.
Parameter 3	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 2. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.6 CL_GetMeasurementDistributorManufacturer

Description		Retrieve the distributor/manufacturer information of the measurement at the given index.
Definition	**Delphi**	function CL_GetMeasurementDistributorManufacturer(MeasureIdx: UInt32; DistrManu: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementDistributorManufacturer(unsigned int MeasureIdx, char* DistrManu, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	DistrManu
	Type	(Pointer) array of char
	Description	Location to store the distributor/manufacturer of the measurement. Set nil/NULL to retrieve only the required length.
Parameter 3	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 2. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.7 CL_GetMeasurementVoltage

Description		Retrieve the voltage of the device during the measurement.
Definition	**Delphi**	function CL_GetMeasurementVoltage(MeasureIdx: UInt32; Voltage: PUInt16): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementVoltage(unsigned int MeasureIdx, unsigned short* Voltage);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	Voltage
	Type	(Pointer) 16bit unsigned int
	Description	Location to store the voltage (in millivolt) of the measurement. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.8 CL_GetMeasurementTemperature

Description		Retrieve the temperature of the device during the measurement.
Definition	**Delphi**	function CL_GetMeasurementTemperature(MeasureIdx: UInt32; Temperature: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementTemperature(unsigned int MeasureIdx, float* Temperature);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	Temperature
	Type	(Pointer) 32bit floating point
	Description	Location to store the temperature (in °C). If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

6.9 CL_GetMeasurementImage

Description		Retrieve the pixel values for the measurement image. Not all devices are configured to save the measurement image. In those cases, the parameter ImageWidth will contain 0 as value for measurement result gotten through **CL_ReadDeviceDB**. Measurements executed directly through the interface will always contain image information (**CL_StartMeasurement** and **CL_StartMeasurementFromFile**).
Definition	**Delphi**	function CL_GetMeasurementImage(MeasureIdx: UInt32; ImageValues: PInt32; ImageWidth: PUInt16): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetMeasurementImage(unsigned int MeasureIdx, signed int* ImageValues, unsigned short* ImageWidth);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ImageValues
	Type	(Pointer) array of 32bit unsigned int
	Description	Location to store the actual image values. Set nil/NULL to retrieve only the required length of the array (width of the image).
Parameter 3	**Name**	ImageWidth
	Type	(Pointer) 16bit unsigned int
	Description	Location to store the width of the image (and therefore required length of parameter 1). If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

7 Retrieving result information

Retrieve information about the results in a measurement. For prerequisites, see chapter **Retrieving measurement information**.

7.1 CL_GetResultCount

Description		Retrieve the number of available results for the given measurement. Calling this function gives the range of indices available for all functions requiring a result index (for this measurement). When addressing these indices, please be aware that the first valid index is always 0.
Definition	**Delphi**	function CL_GetResultCount(MeasureIdx: UInt32; ResultCount: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetResultCount(unsigned int MeasureIdx, unsigned char* ResultCount);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultCount
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the amount of available results for the specified measurement. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

7.2 CL_GetResultName

Description		Retrieve the name of the specified result.
Definition	**Delphi**	function CL_GetResultName(MeasureIdx: UInt32; ResultIdx: UInt8; ResultName: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetResultName(unsigned int MeasureIdx, unsigned char ResultIdx, char* ResultName, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultIdx
	Type	8bit unsigned int
	Description	Index of the result (in the measurement).
Parameter 3	**Name**	ResultName
	Type	(Pointer) array of char
	Description	Location to store the name of the requested result. Set nil/NULL to retrieve only the required array length.
Parameter 4	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Will hold the required length for parameter 3. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

7.3 CL_GetResultValidity

Description		Retrieve the validity of the specified result.
Definition	**Delphi**	function CL_GetResultValidityMeasureIdx: UInt32; ResultIdx: UInt8; Validity: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetResultValidity(unsigned int MeasureIdx, unsigned char ResultIdx; unsigned char* Validity);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultIdx
	Type	8bit unsigned int
	Description	Index of the result (in the measurement).
Parameter 3	**Name**	Validity
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the validity of the result. If not assigned, function will return **FR_INVALID_POINTER**. Possible values: 0 - Result is valid 1 - Result is NaN (Not a Number) 2 - Result is infinite (either - or +) 3 - Result is invalid 4 - Device is expired
Return		Will return 0 on success, for other values see **list of function return codes**.

7.4 CL_GetResultValue

Description		Retrieve the value (as string) for the specified result.
Definition	**Delphi**	function CL_GetResultValue(MeasureIdx: UInt32; ResultIdx: UInt8; ResultValue: PAnsiChar; StrLength: PUint32; ResultValid: PUInt8); stdCall;
	C/C++	unsigned int __stdCall CL_GetResultValue(unsigned int MeasureIdx, unsigned char ResultIdx, char* ResultValue, unsigned int* StrLength, unsigned char* ResultValid);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultIdx
	Type	8bit unsigned int
	Description	Index of the result (in the measurement).
Parameter 3	**Name**	ResultValue
	Type	(Pointer) array of char
	Description	Location to store the value (as text) of the requested result. Set nil/NULL to only retrieve the required array length. If the value is an actual floating-point number, a "." will be used as decimal separator.
Parameter 4	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 3. If not assigned, function will return **FR_INVALID_POINTER**.
Parameter 5	**Name**	ResultValid
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the validity of the result. If not assigned, function will return **FR_INVALID_POINTER**. Possible values: 0 - Result is valid 1 - Result is NaN (Not a Number) 2 - Result is infinite (either positive or negative) 3 - Result is invalid 4 - Device is expired
Return		Will return 0 on success, for other values see **list of function return codes**.

7.5 CL_GetResultClass

Description		Retrieve the class of the specified result.
Definition	**Delphi**	function CL_GetResultClass(MeasureIdx: UInt32; ResultIdx: UInt8; ResultClass: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetResultClass(unsigned int MeasureIdx, unsigned char ResultIdx, char* ResultClass, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultIdx
	Type	8bit unsigned int
	Description	Index of the result (in the measurement).
Parameter 3	**Name**	ResultClass
	Type	(Pointer) array of char
	Description	Location to store the class of the requested result. Set nil/NULL to retrieve the required array length.
Parameter 4	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 3. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

7.6 CL_GetResultUnit

Description		Retrieve the unit of the specified result.
Definition	**Delphi**	function CL_GetResultUnit(MeasureIdx: UInt32; ResultIdx: UInt8; ResultUnit: PAnsiChar; StrLength: PUInt32): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetResultUnit(unsigned int MeasureIdx, unsigned char ResultIdx, char* ResultUnit, unsigned int* StrLength);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	ResultIdx
	Type	8bit unsigned int
	Description	Index of the result (in the measurement).
Parameter 3	**Name**	ResultUnit
	Type	(Pointer) array of char
	Description	Location to store the unit of the requested result. Set nil/NULL to retrieve the required array length.
Parameter 4	**Name**	StrLength
	Type	(Pointer) 32bit unsigned int
	Description	Location to store the required length for parameter 3. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8 Retrieving line information

Retrieve information about the test lines in a measurement. For prerequisites, see chapter **Retrieving measurement information**.

Please be aware that not every device is saving the line information. Therefore, measurement results gotten through **CL_ReadDeviceDB** will not necessarily contain line information.

Measurements executed directly through the interface (**CL_StartMeasurement** and **CL_StartMeasurementFromFile**) will always contain line information.

8.1 CL_GetLineCount

Description		Retrieve the number of available lines for the specified measurement. Calling this function gives the range of indices available for all functions requiring a line index (for this measurement). When addressing these indices, please be aware that the first valid index is always 0.
Definition	**Delphi**	function CL_GetLineCount(MeasureIdx: UInt32; LineCount: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineCount(unsigned int MeasureIdx, unsigned char* LineCount);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineCount
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the number of available lines for the measurement. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.2 CL_GetLineMean

Description		Retrieve the mean value for the specified line.
Definition	**Delphi**	function CL_GetLineMean(MeasureIdx: UInt32; LineIdx: UInt8; LineMean: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineMean(unsigned int MeasureIdx, unsigned char LineIdx, float* LineMean);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineMean
	Type	(Pointer) 32bit floating point
	Description	Location to store the mean value for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.3 CL_GetLineMin

Description		Retrieve the minimum value for the specified line.
Definition	**Delphi**	function CL_GetLineMin(MeasureIdx: UInt32; LineIdx: UInt8; LineMin: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineMin(unsigned int MeasureIdx, unsigned char LineIdx, float* LineMin);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineMin
	Type	(Pointer) 32bit floating point
	Description	Location to store the minimum value for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.4 CL_GetLineMax

Description		Retrieve the maximum value for the specified line.
Definition	**Delphi**	function CL_GetLineMax(MeasureIdx: UInt32; LineIdx: UInt8; LineMax: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineMax(unsigned int MeasureIdx, unsigned char LineIdx, float* LineMax);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineMax
	Type	(Pointer) 32bit floating point
	Description	Location to store the maximum value for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.5 CL_GetLineLeftMean

Description		Retrieve the mean value of the left half for the specified line.
Definition	**Delphi**	function CL_GetLineLeftMean(MeasureIdx: UInt32; LineIdx: UInt8; LineLeftMean: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineLeftMean(unsigned int MeasureIdx, unsigned char LineIdx, float* LineLeftMean);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineLeftMean
	Type	(Pointer) 32bit floating point
	Description	Location to store the mean value of the left half for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.6 CL_GetLineRightMean

Description		Retrieve the mean value of the right half for the specified line.
Definition	**Delphi**	function CL_GetLineRightMean(MeasureIdx: UInt32; LineIdx: UInt8; LineRightMean: PSingle): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineRightMean(unsigned int MeasureIdx, unsigned char LineIdx, float* LineRightMean);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineRightMean
	Type	(Pointer) 32bit floating point
	Description	Location to store the mean value of the right half for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.7 CL_GetLineMaxPosAbs

Description		Retrieve the absolute position of the maximum value for the specified line.
Definition	**Delphi**	function CL_GetLineMaxPosAbs(MeasureIdx: UInt32; LineIdx: UInt8; MaxPosAbs: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineMaxPosAbs(unsigned int MeasureIdx, unsigned char LineIdx, unsigned char* MaxPosAbs);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	MaxPosAbs
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the position of the maximum value for the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.8 CL_GetLineCenterPosAbs

Description		Retrieve the absolute position of the center (middle) of the line.
Definition	**Delphi**	function CL_GetLineCenterPosAbs(MeasureIdx: UInt32; LineIdx: UInt8; CenterPosAbs: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetCenterPosAbs(unsigned int MeasureIdx, unsigned char LineIdx, unsigned char* CenterPosAbs);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	CenterPosAbs
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the center position of the specified line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.9 CL_GetLineUsedPixel

Description		Retrieve the number of pixels in the line that contributed to the evaluation.
Definition	**Delphi**	function CL_GetLineUsedPixel(MeasureIdx: UInt32; LineIdx: UInt8; LineUsedPixel: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineUsedPixel(unsigned int MeasureIdx, unsigned char LineIdx, unsigned char* LineUsedPixel);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineUsedPixel
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the number of pixels in the line that contributed to the line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

8.10 CL_GetLineWidth

Description		Retrieve the line width.
Definition	**Delphi**	function CL_GetLineWidth(MeasureIdx: UInt32; LineIdx: UInt8; LineWidth: PUInt8): UInt32; stdCall;
	C/C++	unsigned int __stdCall CL_GetLineWidth(unsigned int MeasureIdx, unsigned char LineIdx, unsigned char* LineWidth);
Parameter 1	**Name**	MeasureIdx
	Type	32bit unsigned int
	Description	Index of the measurement.
Parameter 2	**Name**	LineIdx
	Type	8bit unsigned int
	Description	Index of the line (in the measurement).
Parameter 3	**Name**	LineWidth
	Type	(Pointer) 8bit unsigned int
	Description	Location to store the width of the line. If not assigned, function will return **FR_INVALID_POINTER**.
Return		Will return 0 on success, for other values see **list of function return codes**.

9 Return codes

The following table gives an overview of all codes the interface functions can return. Please be aware that many functions, especially those leading to direct communication with the device, only initialize that communication und therefore will report errors mainly through the **MessageHandler**.

Name	Value	Description
FR_OK	0x00	OK, no error(s)
FR_NOT_INITIALIZED	0x01	Interface has not been initialized - call prepareInit-Function.
FR_NO_VALID_INTERFACE	0x02	Device detection failed because no valid interface is available.
FR_INVALID_INDEX	0x03	The specified index is out of range of available indices.
FR_COULD_NOT_OPEN_PORT	0x04	The port could not be opened (invalid device?).
FR_FILE_NOT_FOUND	0x05	The specified filename & -path could not be found.
FR_DEVICE_NOT_READY	0x06	The device/DLL is busy with another operation.
FR_LICENSE_INVALID	0x07	The specified license is not valid.
FR_NO_LICENSE_SET	0x08	No license has been set.
FR_ALREADY_INITIALIZED	0x09	The DLL initialization has already been executed.
FR_DEVICE_NOT_AVAILABLE	0x0A	The FTDI device is not available (connected to another software?).
FR_CONNECTION_ALREADY_OPEN	0x0B	There is already an active connection, please close it before trying to open a new one.
FR_NO_DEVICE_CONNECTED	0x0D	There is no device connected.
FR_CONFIG_INVALID	0x0E	The specified configuration is not valid.
FR_INVALID_POINTER	0x10	A required pointer is invalid (nil/NULL).

10 Messages

10.1 MessageHandler

To be able to receive and handle any messages, a callback procedure needs to be set through **CL_SetMessageHandler**. The callback-procedure has the following definition:

Description		Callback procedure for message handling.
Definition	**Object Pascal**	procedure (MessageType: UInt32; MessageCode: UInt32; Data: UInt32); stdcall;
	C/C++	void __stdcall (unsigned int messageType; unsigned int messageCode; unsigned int data);
Parameter 1	**Name**	MessageType
	Type	32bit unsigned int
	Description	See Table **MessageType**.
Parameter 2	**Name**	MessageCode
	Type	32bit unsigned int
	Description	Interpretation depending on parameter MessageType.
Parameter 3	**Name**	Data
	Type	32bit unsigned int
	Description	Interpretation depending on parameters MessageType and MessageCode.

10.2 Message types

The following table describes the possible values of the parameter **MessageType**:

Message type	Value	Message code	Description
MT_INFO	0x00	See **list of InfoMessages**	Info message
MT_ERROR	0x01	See **list of ErrorMessages**	Error message
MT_PROGRESS	0x02	**Message code** will contain the current progress (as absolute number), while **Data** represents the maximum progress value.	Progress update (e.g., when reading device database). Should **Data** (maximum progress) have the value 0, this message is to be interpreted as a busy signal from the interface.

10.3 MT_INFO - Message codes

The following table describes the potential values fort he parameter **MessageCode** in case of **MessageType** being **MT_INFO**:

InfoMessage (MessageType = mtInfo)	Code	Data	Description
IM_PLACE_WHITE	0x00		Put the white cassette (undeveloped test) in the device.
IM_PLACE_TEST	0x01		Put the test cassette (developed test) in the device.
IM_TIMER_RUNNING	0x02	Seconds left until the execution of the measurement	Measurement timer is running (will be sent approximately once per second as long as the timer of a measurement is running).
IM_EVALUATION_RUNNING	0x03		Measurement evaluation is running (will be sent once the actual measurement starts).
IM_MEASUREMENT_DONE	0x04	Index of new measurement in measure list of the DLL	Will be sent whenever a measurement has been executed successfully.
IM_DEVICE_BUSY	0x05	The estimated busy time	Device is busy. This message is also fired if a measurement has been started.
IM_CONNECTION_ESTABLISHED	0x06		A device has been connected.
IM_DEVICE_DISCONNECTED	0x07		A device has been disconnected.
IM_STATE_SET	0x08	The current state	Device/Interface changed state. See **list of states**.
IM_MEASURE_COUNT	0x09	The amount of measurements read from the device	The database has been read.
IM_BUTTON_SET	0x0A		The device button has been enabled/disabled.
IM_DB_FORMATTED	0x0C		Database successfully formatted.

10.4 MT_ERROR - Message codes

The following table describes the potential values for the parameter **MessageCode** in case of **MessageType** being **MT_ERROR**:

ErrorMessage (MessageType = mtError)	Code	Data	Description
EM_COMMAND_NOT_SUPPORTED	0x00		The sent command is not supported by the device.
EM_DEVICE_ERROR	0x01	See **list of device errors**	A device error occurred (see list of device errors).
EM_COMMUNICATION_ERROR	0x02	See **list of communication errors**	A communication error occured (see list of communication errors).
EM_DATE_TIME_INVALID	0x03		The date/time value received from the device is invalid.
EM_TIME_OUT	0x04	0 - device has been successfully reacknowledged after timeout 1 - device has been disconnected	Time out on device communication.
EM_LICENSE_MISMATCH	0x05		The license does not match the device.
EM_TEST_CONFIG_ERROR	0x06	See **list of test config errors**	The test configuration produced an error.

10.5 Communication errors

List of **communication errors**:

Error	Value	Description
DC_ERR_NO_ERROR	0x00	no error (placeholder, won't result in an actual message)
DC_ERR_CRC_MISMATCH	0x01	CRC check failed (device received a data package with invalid CRC)
DC_ERR_UNKNOWN_COMMAND	0x02	the command is not supported by this device
DC_ERR_RECEIVE_BUFFER_OVERFLOW	0x04	the received package is to big to fit in the buffer
DC_ERR_NOT_ENOUGH_MEMORY	0x05	the device does not have enough free memory to handle the received package
DC_ERR_COMMAND_ERROR	0x06	placeholder - occurs if a command is not yet completely implemented (was used in development phase)
DC_ERR_INVALID_FLASH_START_ADDRESS	0x10	flash start address is invalid
DC_ERR_INVALID_FLASH_END_ADDRESS	0x11	flash end address is invalid
DC_ERR_INVALID_EEPROM_START_ADDRESS	0x12	eeprom start address is invalid
DC_ERR_INVALID_EEPROM_END_ADDRESS	0x13	eeprom end address is invalid
DC_ERR_PAGE_WRITE_ERROR	0x14	error while writing in flash memory
DC_ERR_UNKNOWN_SUB_COMMAND	0x21	sub command is not implemented
DC_ERR_NOT_POWERED	0x24	device is not powered
DC_ERR_FLASH_IS_WRITE_PROTECTED	0x25	flash is write protected
DC_ERR_FLASH_WRITE_FAILED	0x26	write check failed (memory needs to be erased before being written)
DC_ERR_IMAGE_NO_DATA	0x30	there is no available image data for transmission
DC_ERR_IMAGE_INVALID_FLAGS	0x31	the command flags for the image data request are invalid
DC_ERR_WRITE_BIG_BUFFER_OVERFLOW	0x40	the package size is to big to fit in the buffer
DC_ERR_WRITE_BIG_BUFFER_INVALID_ADDRESS	0x41	the target address for the write operation is invalid (no memory assigned)

Error	Value	Description
DC_ERR_READ_BIG_BUFFER_NO_FKT	0x42	the target address for the write operation is invalid (no write function for this memory location)
DC_ERR_READ_BIG_BUFFER_NO_DATA	0x43	attempted to read more data than is available (with auto size)
DC_ERR_WRITE_BIG_MEMORY_OVERFLOW	0x44	attempted to write more data than the memory can contain
DC_ERR_WRITE_BIG_FLASH_FAILED	0x45	writing in the flash memory failed
DC_ERR_READ_BIG_MEMORY_OVERFLOW	0x46	attempted to read more data than is available at the memory location
DC_ERR_CUBE_NOT_VALIDATED	0x50	the command is not supported without validation

10.6 Device errors

List of **device errors**:

Error	Value	Description
NO_ERROR	0x00	placeholder, won't result in an actual message
EVALUATION_EXPIRY_DATE_EXCEEDED	0x10	the expiry date of the test configuration has been exceeded
EVALUATION_LOW_BATTERY	0x11	voltage sinking below minimum value during measurement
EVALUATION_CAPTURE_BLACK_ERROR_HIGH	0x12	errors on image capturing
EVALUATION_CAPTURE_BLACK_ERROR_LOW	0x13	
EVALUATION_CAPTURE_LIGHT_ERROR_HIGH	0x14	
EVALUATION_CAPTURE_LIGHT_ERROR_LOW	0x15	
EVALUATION_NOMINAL_POS_OUT_OF_RANGE	0x16	nominal position of a testline is out of range
EVALUATION_SEARCHW_SMALLER_LINEW	0x17	search width of a test line is smaller than its actual width
EVALUATION_INVALID_CONFIGURATION	0x1A	loaded test configuration is invalid
EVALUATION_INVALID_VERSION	0x1C	loaded test configuration has an invalid/unsupported version
EVALUATION_USER_ABORTED	0x1D	evaluation has been aborted by the user
EVALUATION_QC_TEST_NEEDED	0x1E	a QC test is needed before another measurement can be done
RES_ASSIGN_ILLEGAL_INDEX	0x20	Result assignment: illegal index
RES_ASSIGN_ILLEGAL_CHARACTER	0x21	Result assignment: illegal character
RES_ASSIGN_ILLEGAL_JUMP	0x22	Result assignment: illegal jump address
RES_ASSIGN_ILLEGAL_CHAR_IN_NUMBER	0x23	Result assignment: expected number but found char
RES_ASSIGN_ILLEGAL_INDEX_2	0x24	Result assignment: illegal index
RES_ASSIGN_ILLEGAL_INDEX_3	0x25	Result assignment: illegal index
RES_ASSIGN_INDEX_OUT_OF_RANGE	0x26	Result assignment: index out of range
RES_ASSIGN_VALUE_LOW	0x28	Result assignment: value out of range (below range)
RES_ASSIGN_VALUE_HIGH	0x29	Result assignment: value out of range (above range)
RES_ASSIGN_EXPECTED_OPERAND	0x2A	Result assignment: expected operand, received something else
RES_ASSIGN_JUMP_ADDRESS_OUT_OF_RANGE	0x2B	Result assignment: jump address out of range

Error	Value	Description
RES_ASSIGN_ILLEGAL_ASSIGNMENT	0x2C	Result assignment: assignment not valid (can not reassign constant values)
RES_ASSIGN_OPERAND_IS_NAN	0x2D	Result assignment: assigned constant value is invalid (not a number)
RES_ASSIGN_INFINITE_LOOP	0x2E	Result assignment: potential infinite loop
RFID_TIME_OUT	0x30	time out on receiving data from RFID
RFID_STATUS_ERROR	0x31	RFID status error
RFID_RECEIVE_LENGTH_OVERFLOW	0x32	buffer to small for received data
RFID_COLLISION_DETECTED	0x33	communication collision detected (more than one tag in range)
RFID_MIFARE_NAK	0x34	MIFARE-tag: acknowledge has not been received
RFID_RECEIVE_CRC_INVALIDE	0x35	the CRC for the received data is invalid
RFID_BUFFER_TO_SMALL	0x36	buffer to small for requested operation
RFID_ATQA_ANSWER_INVALID	0x37	ATQA-command failed
RFID_NOTHING_RECEIVED	0x38	timeout on receiving RFID data (no tag in range)
RFID_STATUS_INVALID	0x39	RFID: status invalid
RFID_INTERNAL_ERROR	0x3A	RFID: internal error
RFID_MEMORY_ADDRESS_OUT_OF_RANGE	0x3B	the requested address is not available in the RFID memory (out of range)
RFID_MEMORY_AUTHENTIFICATION_FAILED	0x3C	the authentification of a RFID memory block failed
RFID_TRANCEIVE_ERROR	0x3D	error while transceiving data
RFID_READ_BUFFER_TO_SMALL	0x3E	the read buffer is to small for requested operation
RFID_NO_VALID_TAG	0x3F	no RFID tag detected or initialization failed
RFID_MAIN_TIMEOUT	0x40	timeout on RFID communication
RFID_MEMORY_ADDRESS_OUT_OF_RANGE2	0x41	the requested address is not available in the RFID memory (out of range)

Error	Value	Description
TEST_CONFIGURATION_GETCHAR_TIMEOUT	0x45	timeout during read operation
TEST_CONFIGURATION_GETCHAR_EOF	0x46	end of file reached (more data requested than available)
TEST_CONFIGURATION_CFG_TO_BIG	0x47	test configuration exceeds maximum size
TEST_CONFIGURATION_UNKNOWN_CFG	0x48	test configuration unknown (version not supported)
TEST_CONFIGURATION_EMPTY	0x49	test configuration empty
TEST_CONFIGURATION_FLASH_FULL	0x4A	(saving results) test configuration does not fit in remaining memory
TEST_CONFIGURATION_CHECKSUMFAIL	0x4B	test configuration: checksum mismatch (test configuration corrupted)
TEST_NO_CONFIG_FOUND	0x4C	no test configuration found
DB_LOAD_REQUESTED_RESULT_CFG_NOTFOUND	0x50	loading result: corresponding configuration not found
DB_LOAD_RESULT_EMPTY	0x51	loading result: result empty
DB_LOAD_RESULT_DELETED	0x52	loading result: result deleted
INTFLASH_ERASE_FAIL	0x63	erasing/nulling of internal flash failed
RFID_WRITE_ENCRYPT_FIRST_MUST_START_AT_0	0x70	writing encrypted tag for the first time: start at address 0
RFID_UNENCRYPTED_TAG_READ_NOT_ALLOWED	0x71	reading of unencrypted RFID tags is not allowed
RFID_CONFIG_NO_CREDIT	0x72	the credit counter on the RFID tag has reached 0, the tag is not usable anymore
RFID_NOT_SUPPORTED	0x73	the RFID tag type is not supported
CRYPT_ALU_NOT_ENOUGH_DATA	0x80	validation errors
CRYPT_ALU_CALCULATION_PROGRAMMED_BREAK	0x81	
CRYPT_ALU_INTERNAL_PROGRAM_EMPTY	0x82	
CRYPT_RANDOM_PATTERN_INVALID	0x83	
TEST_CONFIGURATION_noValidCustomer	0x91	the test configuration is not meant for the connected device
SAVE_TEST_NO_RESULT_FOUND_WITH_THIS_ID	0x9E	the given ID does not match any saved result

11 List of device/interface states

Name	Value	Description
STATE_DISCONNECTED	0x00	Communication port is disabled
STATE_IDLE	0x01	Software is in idle state, cube is connected
STATE_BUSY	0x02	Cube is busy, unable to receive/handle new commands
STATE_ERROR	0x03	Currently in error state (normally while message is displayed)
STATE_ERROR_ACK	0x04	GetAcknowledge command issued when an error occured
STATE_CONNECT	0x10	Software is settting up a connection to the cube
STATE_READ	0x20	Software is reading data from the cube
STATE_WRITE	0x30	Software is writing data to the cube
STATE_CAPTURE	0x40	Cube is capturing image(s)
STATE_EVALUATE	0x50	Software and cube are in the process of a measurement
STATE_FORMATTING	0x60	Device is formatting its database
STATE_DATE_TIME	0x70	Setting/getting device date/time
STATE_SET_BUTTON_CTRL	0x72	Setting button control (lock/unlock button)
STATE_LOCK	0x73	Locking device (removing validation and access to certain commands)

12 List of test configuration errors

Name	Value	Description
CONFIG_EMPTY	0x01	The configuration is empty.
CONFIG_WRONG_VERSION	0x02	The version of the configuration is not supported or incompatible to the device.
CONFIG_INVALID_CHECKSUM	0x03	The configuration has an invalid checksum (data is corrupted).
CONFIG_PARSE_ERROR	0x04	An error occurred while the configuration has been parsed (not enough or invalid data).
CONFIG_INVALID_FILE	0x05	The given filename for the configuration does not exist.

13 Version History

Version	Changes	Date
1.00	Initial release based on Cube DLL Version 1.5.152	June 2021
1.01	Updated for DLL V1.5.153: SetBaudrate and GetBaudrate removed for the time being (not supported by current cube firmware)	June 2021
1.02	Updated for DLL V1.5.165 - Added support for BluetoothLE (64bit version only). - Added function ML_BTLEShowOnlyConnected - Changed device detection behaviour, devices in all channels (FTDI; BT; BTLE) will be detected simultaneously without a need to actively change the mode. - Removed CL_StartBluetooth, CL_GetBluetoothDeviceCount, CL_GetBluetoothDeviceInfo and CL_GetConnectionMode - Renamed CL_GetFTDIDeviceCount to CL_GetDeviceCount - Renamed CL_GetFTDIDeviceInfo to CL_GetDeviceInfo - Renamed FR_FTDI_INVALID to FR_NO_VALID_INTERFACE to reflect its updated meaning - Removed unused function return codes and states referring to the superfluous separate Bluetooth mode - Added Parameter ConnectionType to CL_GetDeviceInfo	August 2021