CT400

All-Band Optical Component Tester

Programming Guide

CT400_PG_1.3v1.1

yenista OPTICS

FIBER OPTIC TEST & MEASUREMENT

Legal Notice

Copyright

Copyright © 2013–2015 by Yenista Optics. Published by Yenista Optics. All rights reserved.
This documentation is provided as a user manual to Yenista Optics' customers and potential customers only. The contents of this document may not be reproduced in any part or as a whole, transcribed, stored in a retrieval system, translated into any language, or transmitted in any form or by any means (electronic, mechanical, magnetic, optical, chemical, photocopying, manual, or otherwise) without the prior written permission of Yenista Optics.

Trademarks

"Yenista Optics", "Yenista" and "TUNICS" are registered trademarks of Yenista Optics. Other trademarks mentioned in this publication are used for identification purposes only.

Product Warranty and Limitation of Warranty

This Yenista Optics instrument product is warranted against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Yenista will repair products that prove to be defective. For warranty service or repair, such product must be returned to Yenista. Please contact Yenista support services for the appropriate procedure. Yenista does not accept any equipment shipped back without following the procedure.

Yenista Optics warrants that its software and firmware designated by Yenista for use with an instrument will execute its programming instructions when properly installed on that instrument. Yenista does not warrant that the operation of the instrument, software, or firmware will be uninterrupted or error-free.

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

Disclaimers

No Warranties: This documentation is provided "as is" without any ex-touch or implied warranty of any kind.
Limitation of Liability: Yenista Optics does not assume any liability arising out of the application or use of any products, or software described herein. In no event shall Yenista Optics or its affiliate companies be liable for any damages whatsoever (including, without limitation, consequential or incidental damages, damages for loss of profits, or otherwise) arising out of the use of the information provided in this documentation.

The contents of this publication have been carefully checked for accuracy. However, Yenista Optics makes no warranty as to the completeness, correctness, or accuracy of the information contained within this documentation. In the interest of continued product improvement, Yenista Optics further reserves the right to make changes to any products described herein without notice. This publication is subject to change without notice.

Contact Information

Headquarters

Yenista Optics
4 rue Louis de Broglie
22300 Lannion
FRANCE
Phone: +33 2 96 48 37 16
Fax: +33 2 96 48 73 04
Website: http://www.yenista.com

North American Office

Yenista Optics Inc.
2393 Teller Road, Suite 102
Newbury Park, CA 91320-6089
USA
Phone: +1 805 367 4075
Fax: +1 609 423 0891

Sales

Sales Americas
Email: sales-am@yenista.com
Phone: +1 805 367 4075

Sales Asia and Pacific
Email: sales-apac@yenista.com

Sales Europe, Middle East and Africa
Email: sales-emea@yenista.com
Phone: +33 296 483 716

Sales China
Email: sales-china@yenista.com

Customer Support and Repair Services

Americas
Email: support-am@yenista.com
Phone: +1 805 367 4075

Asia and Pacific
Email: support-apac@yenista.com

Europe, Middle East and Africa
Email: support-emea@yenista.com
Phone: +33 296 486 145

China
Email: support-china@yenista.com

About This Manual

Subject	This manual explains how to install the CT400 library on your computer, update from a preceding version to the new one, and describes the features of the CT400 DLL.
Application	Information in this document applies to the CT400 library v 1.3.x (with DSP version 1.09) and operating system from Windows XP to Windows 7.
Intended Readers	Users of this manual must be familiar with:

- The use of a C compiler or the use of the LabVIEW software
- The use of the CT400 product (see *CT400 User Manual*)

Date	19 May 2015
Manual Reference	CT400_PG_1.3v1.1

Typographical Conventions

bold	Identifies interface objects such as menu names, labels, buttons and icons.
italic	Identifies references to other sections or other guides.
`monospace`	Identifies portions of program codes, command lines, or messages displayed in command windows.
IMPORTANT	Identifies important information to which you must pay particular attention.

Table of Contents

1. Installing/Updating the CT400 Library

Subject   The CT400 library is provided with the CT400 software package.

When installed on your computer, the **Library x.xx** folder contains two versions of the library located in two different folders: one dedicated to 32-bit platforms and one dedicated to 64-bit platforms.

Each folder contains the following files:

- **CT400_lib.dll** is the main DLL.
- **CT400_lib.h** is the main DLL header file.
- **CT400_Types.h** is a support header file ; it defines the integer types used.
  It is recommended to use the integer types defined in this file.
- **SiUSBXp.dll** is a DLL dependency (USB Driver).
- **Borland** and **MSVC** directories contains Lib files with the corresponding COFF format for both 32- and 64-bit MSCV; OMF format for Borland 32-bit: ELF format for Borland 64-bit.
- Examples (see section *Program Examples, p. 47*):
  - **CT400_testwrap.c** is an example of the DLL use in C language.
  - **CT400_testwrap.vi** is an example of the DLL use in LabVIEW version 9.
  - **CT400_testwrap.py** is an example of CT400 control file in Python language.

1.1   Installing the CT400 Library

Subject   To install the CT400 library, you must install the CT400 software package, as explained in the following procedure.

Procedure 1. Install the CT400 software package and activate the CT400 USB driver on your computer as described in *CT400 User Manual*, section *Installing the CT400 Software and USB Driver on Your Computer, p. 13*.

   The CT400 library is installed in one of the following folders (depending on your Windows version):
   - **C:\Program Files\Yenista Optics\CT400\Library x.xx**
   - **C:\Program Files (x86)\Yenista Optics\CT400\Library x.xx**

2. Start the CT400 GUI software and make sure you can operate the CT400 and laser through the CT400 graphical user interface.

   If the new version requires an update of the DSP code of the unit, you are prompted to upgrade the CT400 DSP. In this case, click **Yes** to update the DSP.

3. Create your own project with your preferred IDE and Compiler.

4. In this project, include all files contained in the **Win32** or **Win64** folder. All DLL files must be in the same folder as your final executable/DLL.

1.2 Updating the CT400 Library to v. 1.3.x

Subject   If you are using programs written with a previous library of the CT400 library, you must make them compatible with the new version, as explained in the following procedure:

Procedure   1. Install the last version of the CT400 GUI software on the PC as described in *CT400 User Manual*, section *Updating the CT400 System Version, p. 55*.

   The new version of the CT400 library is installed in one of the following folders (depending on your Windows version):
   - **C:\Program Files\Yenista Optics\CT400\Library x.xx**
   - **C:\Program Files\Yenista Optics (x86)\CT400\Library x.xx**

   2. Start the CT400 GUI software and make sure you can operate the CT400 and laser through the CT400 graphical user interface.

   If the new version requires an update of the DSP code of the unit, you are prompted to upgrade the CT400 DSP. In this case, click **Yes** to update the DSP.

   3. Replace the following existing files and directories by the new ones (located in the **Win32** or **Win64** folder, depending on your Windows platform):

   - **CT400_lib.h**, **CT400_Types.h** and **CT400_lib.dll** files.

   - **Borland** and **MSVC** directories.

   - **SiUSBXp.dll** file.

   4. Modify the updated functions:
      In the existing `CT400_SetLaser()` function, make sure the laser `Speed` argument is set (see section *CT400_SetLaser, p. 13*).

   - In the existing `CT400_SetSamplingResolution` function, make sure the data type of the `Resolution` parameter is set to unsigned int32.

   - Make sure you use the data types detailed in section *CT400 Data Types, p. 11*.

   - Make sure the data type of the `uiHandle` parameter is set to unsigned int64.

2. Description of Functions

Subject This section describes all functions of the CT400 library.

CT400 Data Types The following table describes all specific data types defined for the CT400 library.

Data Type	Description and Possible Values
rLaserSource	Type of laser: • LS_TunicsPlus • LS_TunicsPurity • LS_TunicsReference • LS_TunicsT100s (for TUNICS T100S and T100S-HP) • LS_TunicsT100r • LS_JdsuSws • LS_Agilent
rLaserInput	Laser input number on the CT400 front panel: • LI_1 • LI_2 • LI_3 • LI_4
rDetector	Detector number on the CT400 front and rear panels: • DE_1 • DE_2 • DE_3 • DE_4 • DE_5 (if the analog input of an external detector is connected to the BNC connector labeled "C" on the rear panel)
rEnable	State: • DISABLE • ENABLE
rUnit	Units: • Unit_mW • Unit_dBm

2.1 Initialization Functions

2.1.1 CT400_Init

Description   This function initializes the DLL for connection to a CT400 and returns a handle.

Declaration   `uint64_t CT400_Init(void);`

Parameters   None

Return Value   Handle used in subsequent functions.
- 0: the initialization failed.
- >0: the initialization succeeded.

Data type: unsigned int64

Example   `uint64_t uiHandle;`
`uiHandle = CT400_Init();`

2.1.2 CT400_CheckConnected

Description   This function verifies if the CT400 is connected to the computer.

Declaration   `int32_t CT400_CheckConnected(uint64_t uiHandle);`

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- 1: a CT400 is connected.
- 0: no CT400 is connected.

Data type: int32

Example   `int32_t isCT400_connected;`
`isCT400_connected = CT400_CheckConnected(uiHandle);`

2.1.3 CT400_Close

Description   This function closes the connection between your C application and the CT400.
It also releases all memory allocated by `CT400_Init`.

Declaration   `int32_t CT400_Close(uint64_t uiHandle);`

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- `0`: the closing operation succeeded.
- `-1`: the closing operation failed.

Data type: int32

Example   `CT400_Close(uiHandle);`

2.2   Configuration Functions

2.2.1   CT400_SetLaser

Description   This function configures the lasers connected to the CT400.
In the CT400 GUI, this function is identical to the **Sources Parameters** settings in the **Parameters** tab.

Declaration
```
int32_t CT400_SetLaser(uint64_t uiHandle,
                       rLaserInput eLaser,
                       rEnable eEnable,
                       int32_t iGPIBAdress,
                       rLaserSource eLaserType,
                       double dMinWavelength,
                       double dMaxWavelength,
                       int32_t Speed);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
eLaser	Laser input number. Possible values: see section *CT400 Data Types, p. 11*.	rLaserInput
eEnable	State of the laser input. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
iGPIBAdress	GPIB address of the laser. Possible values: 1 to 30	int32
eLaserType	Laser type. Possible values: see section *CT400 Data Types, p. 11*.	rLaserSource
dMinWavelength	Laser minimum wavelength in nm. Possible values: minimum of the wavelength range of the laser.	double
dMaxWavelength	Laser maximum wavelength in nm. Possible values: maximum of the wavelength range of the laser.	double
Speed	Speed of laser in nm/s. Possible values: from 10 to 100, limited to laser speed specifications.	int32

Return Value
- `0`: the laser setting operation succeeded.
- `-1`: the laser setting operation failed.

Data type: int32

Example
```
CT400_SetLaser(uiHandle,LI_1,ENABLE,12,LS_TunicsT100s,1480.0,
1600.0,100);
```

2.2.2　CT400_CmdLaser

Description　This function enables you to pilot a laser connected to the computer via GPIB. It sets the wavelength, power and state (enable/disable).

Declaration
```
int32_t CT400_CmdLaser(uint64_t uiHandle,
                       rLaserInput eLaser,
                       rEnable eEnable,
                       double dWavelength,
                       double dPower);
```

Parameter

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
eLaser	Laser input number. Possible values: see section *CT400 Data Types, p. 11*.	rLaserInput
eEnable	State of the laser input. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
dWavelength	Laser wavelength to set in nm. Possible values: depend on the laser's wavelength range.	double
dPower	Laser power to set in mW. Possible values: depend on the laser's power specifications.	double

Return Value
- 0: the operation succeeded.
- −1: the operation failed.

Data type: int32

Example　`CT400_CmdLaser(uiHandle,LI_1,ENABLE,1500.0, 1.0);`

2.2.3   CT400_SwitchInput

Description   This function selects a CT400 input port, which enables the use of the laser connected to this port.

Declaration
```
int32_t CT400_SwitchInput(uint64_t uiHandle,
                          rLaserInput eLaser);
```

Input Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
eLaser	CT400 input port to select. Possible values: section *CT400 Data Types, p. 11*	rLaserInput

Return value
• 0: the operation succeeded.
• -1: the operation failed.

Data type: int32

Example   `CT400_SwitchInput(uiHandle,1);`

2.2.4   CT400_SetSamplingResolution

Description   This function configures the CT400 sampling resolution.
In the CT400 GUI, this function is identical to the **Sampling Resolution** setting in the **Parameters** tab.

Declaration
```
int32_t CT400_SetSamplingResolution(uint64_t uiHandle,
                                    uint32_t uiResolution);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
Resolution	Resolution in pm. Possible values: 1 to 250	unsigned int32

Return Value
• 0: the operation succeeded.
• -1: the operation failed.

Data type: int32

Example   `CT400_SetSamplingResolution(uiHandle,1);`

2.2.5   **CT400_SetScan**

Description   This function sets the scan parameters.

In the CT400 GUI, this function is identical to the **Output Power** and **Scan Range** settings in the **Parameters** tab.

Declaration
```
int32_t CT400_SetScan(uint64_t uiHandle,
                       double dLaserPower,
                       double dMinWavelength,
                       double dMaxWavelength);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
dLaserPower	Laser power in mW, between 1 and 10 mW. Possible values: depend on the laser's power specifications.	double
dMinWavelength	Scan start wavelength. Possible values: minimum of the wavelength range of the laser.	double
dMaxWavelength	Scan stop wavelength. Possible values: maximum of the wavelength range of the laser.	double

Return Value
- 0: the operation succeeded.
- −1: the operation failed.

Data type: int32

Example   `CT400_SetScan(uiHandle,1.0,1500.0,1600.0);`

2.2.6   CT400_SetDetectorArray

Description   This function enables the CT400 detectors and BNC port labeled "C" so that data can be read from these ports.

Detector 1 is always enabled.

The number of enabled detectors impacts the number of points available for measurements (see functions *CT400_ScanGetWavelengthSyncArray, p. 24.CT400_ScanGetWavelengthResampledArray, p. 25*, *CT400_ScanGetPowerSyncArray, p. 26*, *CT400_ScanGetPowerResampledArray, p. 27*, *CT400_ScanGetDetectorArray, p. 28* and *CT400_ScanGetDetectorResampledArray, p. 29*).

In the CT400 GUI, this function is identical to the **Detector Array** selection setting in the **Parameters** tab.

Declaration
```
int32_t CT400_SetDetectorArray(uint64_t uiHandle,
                               rEnable eDect2,
                               rEnable eDect3,
                               rEnable eDect4,
                               rEnable eExt);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
eDect2	State of the detector 2. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
eDect3	State of the detector 3. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
eDect4	State of the detector 4. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
eExt	State of the BNC connector labeled "C" (located on the rear panel of the CT400). Possible values: see section *CT400 Data Types, p. 11*.	rEnable

Return Value   • `0`: the operation succeeded.
      • `−1`: the operation failed.

Data type: int32

Example   `CT400_SetDetectorArray(uiHandle,DISABLE,DISABLE,DISABLE,DISABLE);`

2.2.7   **CT400_SetBNC**

Description   This function configures the CT400 BNC analog port labeled "C" (located on the rear panel of the CT400) during the scan.

This function enables you to get the measured voltage converted in dB, according to the entered parameters.

If this function is not called, the "C" detector can still be read; in this case returned values represent the voltage at the port (see functions *CT400_ScanGetDetectorArray, p. 28* and *CT400_ScanGetDetectorResampledArray, p. 29*).

In the CT400 GUI, this function is identical to the **BNC input** settings in the **Control/ Calibration** tab.

Declaration
```
int32_t CT400_SetBNC(uint64_t uiHandle,
                     rEnable eEnable,
                     double dAlpha,
                     double dBeta,
                     rUnit eUnit);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
eEnable	State of the transfer function. Possible values: see section *CT400 Data Types, p. 11*.	rEnable
dAlpha	Alpha parameter in W/V or dBm/V (P = dBeta + dAlpha x V) Possible values: depend on the external power meter or detector used.	double
dBeta	Beta parameter in W or dBm (P = dBeta + dAlpha x V) Possible values: depend on the external power meter or detector used.	double
eUnit	Units. Possible values: see section *CT400 Data Types, p. 11*.	rUnit

Return Value   • `0`: the operation succeeded.
• `-1`: the operation failed.

Data type: int32

Example   `CT400_SetBNC(uiHandle,ENABLE,4.125E-03,0.1E-03,Unit_mW);`

2.2.8   CT400_SetExternalSynchronization

Description   This function configures the CT400 external synchronization output (BNC connector labeled "A" located on the rear panel of the CT400).

The CT400 generates a pulse each time a measurement occurs (see section *CT400_SetSamplingResolution, p. 15*).

In the CT400 GUI, this function is identical to the **External Synchronization** setting in the **Parameters** tab.

The following figure illustrates the hardware setup of the function:
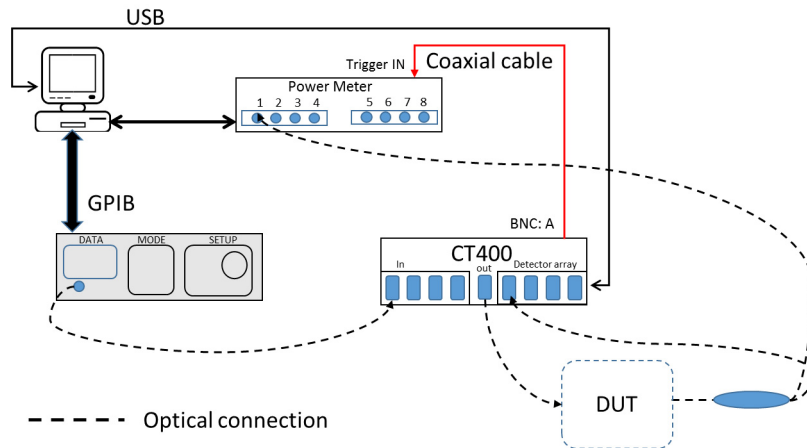
Figure 1: BNC A output enabled by the CT400_SetExternalSynchronization function

Declaration   ```
in32_t CT400_SetExternalSynchronization(uint64_t uiHandle,
                                        rEnable eEnable);
```

**Parameters**

| Input Parameter | Description | Data Type |
|---|---|---|
| uiHandle | Handle returned from CT400_Init | unsigned int64 |
| eEnable | State of the external synchronization. Possible values: see section *CT400 Data Types, p. 11*. | rEnable |

**Return Value**
- `0`: the operation succeeded.
- `-1`: the operation failed.

Data type: int32

**Example**    ```
CT400_SetExternalSynchronization(uiHandle,ENABLE);
```

2.2.9   CT400_SetExternalSynchronizationIN

Description   This function configures the CT400 to run a scan triggered by the laser (BNC connector labeled "B" located on the rear panel of the CT400).

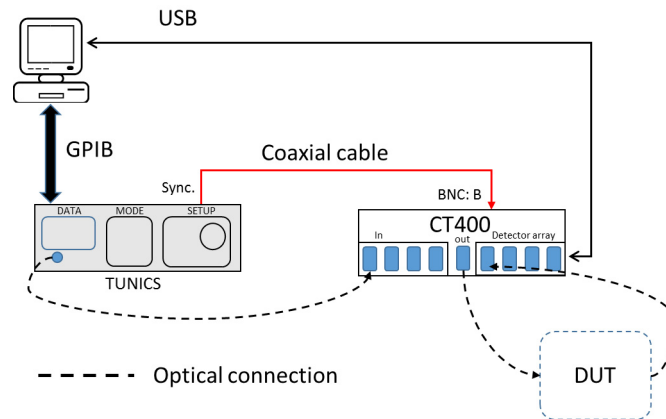The following figure illustrates the hardware setup of the function:

Figure 2: BNC B input enabled by the CT400_SetExternalSynchronizationIN function

Declaration
```
int32_t CT400_SetExternalSynchronizationIN(uint64_t uiHandle,
                                           rEnable eEnable);
```

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64
`eEnable`	State of the external synchronization input. Possible values: see section *CT400 Data Types, p. 11*. If Enable, the CT400 waits for an external trigger on the BNC port labeled "B" to start measurements.	rEnable

Return Value
- `0`: the operation succeeded.
- `-1`: the operation failed.

Data type: int32

Example   `CT400_SetExternalSynchronizationIN(uiHandle,ENABLE);`

2.2.10   CT400_UpdateCalibration

Description   This function calibrates a single CT400 detector. It must be called after a scan has been performed on the same detector.

Before sending the command, connect the CT400 output port directly to the selected detector with an SMF jumper.

In the CT400 GUI, this function is similar to the **Update Calibration** button of the **TF Calibration** area in the **Control/Calibration** tab; it updates the same calibration file as the one used by the GUI.

Declaration   `int32_t CT400_UpdateCalibration(uint64_t uiHandle,`
                                    `rDetector eDetector);`

Parameters

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64
`eDetector`	Detector number. Possible values: see section *CT400 Data Types, p. 11*.	rDetector

Return Value   • `0`: the operation succeeded.
                   • `-1`: the operation failed.

Data type: int32

Example   `CT400_UpdateCalibration(uiHandle,DE_1);`

2.2.11   CT400_ResetCalibration

Description   This function resets calibration of all CT400 detectors to default.

In the CT400 GUI, this function is identical to the **Reset Calibration** button in the **Control/Calibration** tab.

Declaration   `int32_t CT400_ResetCalibration (uint64_t uiHandle);`

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value   • `0`: the operation succeeded.
                   • `-1`: the operation failed.

Data type: int32

Example   `CT400_ResetCalibration(uiHandle);`

2.3   Measurement Control Functions

2.3.1   CT400_ScanStart

Description   This function starts a scan. If the `CT400_SetExternalSynchronizationIN` function is used (see *p. 20*), the CT400 waits for a trigger signal from the laser, on the BNC port labeled "B", to start a scan.
In the CT400 GUI, this function is identical to the green **Start** button.

Declaration   `int32_t CT400_ScanStart(uint64_t uiHandle);`

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- `0`: the operation succeeded.
- `-1`: the operation failed.

Data type: int32

Example   `CT400_ScanStart(uiHandle);`

2.3.2   CT400_ScanStop

Description   This function stops a scan.
In the CT400 GUI, this function is identical to the red **Stop** button when a start has already been performed.

Declaration   `int32_t CT400_ScanStop(uint64_t uiHandle);`

Parameter

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- `0`: the operation succeeded.
- `-1`: the operation failed.

Data type: int32

Example   `CT400_ScanStop(uiHandle);`

2.3.3   CT400_ScanWaitEnd

Description   This function waits for the scan to finish and returns errors or warnings. The list of these messages is available in *CT400 User Manual*.

Declaration
```
int32_t CT400_ScanWaitEnd(uint64_t uiHandle,
                          char tcError[1024]);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
tcError[1024]	String that describes the error.	char[1024]

Return Value
- >0: indicates the error/warning number.
- 0: the operation succeeded.

Data type: int32

Example
```
int32_t iErrorMsg;
char tcError[1024];
iErrorMsg = CT400_ScanWaitEnd(uiHandle, tcError);
if (iErrorMsg !=0)
printf("Warning: %s\n", tcError);
```

2.4 Data Handling Functions

2.4.1 CT400_ScanGetWavelengthSyncArray

Description   This function returns the total number of measured wavelength points and fills the input array with the measured wavelengths.

Declaration
```
int32_t CT400_ScanGetWavelengthSyncArray(uint64_t uiHandle,
                                         double *dArray,
                                         int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
dArray	Pointer over an initialized array.	double*
iArraySize	Size of the array. The number of detectors used (defined with the CT400_SetDetectorArray function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of measured points available in dArray.
Data type: int32

Example
```
int32_t iArraySize = CT400_GetNbDataPoints(uiHandle);
double
*dWavelengthSync=(double*)calloc(iArraySize,sizeof(double));
ScanGetWavelengthSyncArray(uiHandle,dWavelengthSync,iArraySize);
free(dWavelengthSync);
```

The result of the operation is the array dWavelengthSync [0 to PointsNumber-1] containing all the measured wavelength values (PointsNumber ≤ iArraySize).

2.4.2 CT400_ScanGetWavelengthResampledArray

Description   This function performs the following:
- Returns the total number of wavelength points measured by the CT400 after they have been re-sampled so that the step between points corresponds to the resolution set in the `CT400_SetSamplingResolution` function (*p. 15*)
- Fills the input array with the re-sampled wavelength in each point

Declaration
```
int32_t CT400_ScanGetWavelengthResampledArray
                            (uint64_t uiHandle,
                                double *dArray,
                                int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
dArray	Pointer over an initialized array.	double*
iArraySize	Size of the array. The number of detectors used (defined with the `CT400_SetDetectorArray` function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of resampled points available in `dArray`.
Data type: int32

Example
```
int32_t iArraySize = CT400_GetNbDataPointsResampled(uiHandle);
double
*dWavelengthSync=(double*)calloc(iArraySize,sizeof(double));
ScanGetWavelengthResampledArray(uiHandle,dWavelengthSync,
iArraySize);
free(dWavelengthSync);
```

The result of the operation is the array `dWavelengthSync [0 to PointsNumber-1]` containing all the re-sampled wavelength values (`PointsNumber ≤ iArraySize`).

2.4.3 **CT400_ScanGetPowerSyncArray**

Description   This function returns the total number of output power values measured by the CT400 and fills the input array with the measured values.

Declaration
```
int32_t CT400_ScanGetPowerSyncArray (uint64_t uiHandle,
                                     double *dArray,
                                     int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
dArray	Pointer over an initialized array.	double*
iArraySize	Size of the array. The number of detectors used (defined with the CT400_SetDetectorArray function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of measured points available in dArray.
Data type: int32

Example
```
int32_t iArraySize = CT400_GetNbDataPoints(uiHandle);
double *dPowerSync=(double*)calloc(iArraySize,sizeof(double));
ScanGetPowerSyncArray(uiHandle,dPowerSync,iArraySize);
free(dPowerSync);
```

The result of the operation is the array dPowerSync [0 to PointsNumber-1] containing the measured output power values (PointsNumber ≤ iArraySize).

2.4.4   CT400_ScanGetPowerResampledArray

Description   This function performs the following:

- Returns the total number of output power values associated with the wavelength values measured by the CT400 after they have been re-sampled so that the step between points corresponds to the resolution set in the `CT400_SetSamplingResolution` function (*p. 15*)

- Fills the input array with the re-sampled output power in each point

Declaration
```
int32_t CT400_ScanGetPowerResampledArray (uint64_t uiHandle,
                                          double *dArray,
                                          int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
dArray	Pointer over an initialized array.	double*
iArraySize	Size of the array. The number of detectors used (defined with the `CT400_SetDetectorArray` function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of measured points available in `dArray`.
Data type: int32

Example
```
int32_t iArraySize = CT400_GetNbDataPointsResampled(uiHandle);
double *dPowerSync=(double*)calloc(iArraySize,sizeof(double));
ScanGetPowerResampledArray(uiHandle,dPowerSync,iArraySize);
free(dPowerSync);
```

The result of the operation is the array `dPowerSync [0 to PointsNumber-1]` containing the re-sampled output power values (`PointsNumber ≤ iArraySize`).

2.4.5   CT400_ScanGetDetectorArray

Description   This function gets explicitly the total number of measured transfer function points on a selected detector and implicitly the measured transfer function in each point.
This array is useful for direct use of recorded data.
This is the computed insertion loss, as displayed in the CT400 GUI under **Transfer Functions** tab, not the input power on the CT400 detectors.

Declaration
```
int32_t CT400_ScanGetDetectorArray(uint64_t uiHandle,
                                   rDetector eDetector,
                                   double *dArray,
                                   int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
eDetector	Detector number. Possible values: see section *CT400 Data Types, p. 11*.	rDetector
dArray	Pointer over an initialized array	double*
iArraySize	Size of the array. The number of detectors used (defined with the CT400_SetDetectorArray function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of measured points available in dArray.
Data type: int32
If the eDetector parameter is set to DE_5 and the CT400_SetBNC function (*p. 18*) is set to DISABLE, the value returned by the CT400_ScanGetDetectorArray function is the voltage measured at the BNC port labeled "C".

Example
```
int32_t iArraySize = CT400_GetNbDataPoints(uiHandle);
double *dPowerdet =(double*)calloc(iArraySize,sizeof(double));
ScanGetDetectorArray(uiHandle,DE_1,dPowerdet,iArraySize);
free(dPowerdet);
```

The result of the operation is the array dPowerdet [0 to PointsNumber-1] containing all the measured transfer function values (PointsNumber ≤ iArraySize).

2.4.6   CT400_ScanGetDetectorResampledArray

Description   This function performs the following:

- Returns the transfer function points on a selected detector associated with the wavelength points measured by the CT400 after they have been re-sampled, so that the step between points corresponds to the resolution set in the `CT400_SetSamplingResolution` function (*p. 15*)

- Fills the input array with the re-sampled transfer function in each point

Declaration
```
int32_t CT400_ScanGetDetectorResampledArray(uint64_t uiHandle,
                                rDetector eDetector,
                                double *dArray,
                                int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
eDetector	Detector number. Possible values: see section *CT400 Data Types, p. 11*.	rDetector
dArray	Pointer over an initialized array	double*
iArraySize	Size of the array. The number of detectors used (defined with the `CT400_SetDetectorArray` function (*p. 17*)) limits the available number of points. Recommended values are: • For 1 detector used: up to 333333 • For 2 detectors used: up to 250000 • For 3 detectors used: up to 200000 • For 4 detectors used: up to 166666 • For 5 detectors used: up to 142857	int32

Return Value   Number of measured points available in `dArray`.

Data type: int32

If the `eDetector` parameter is set to `DE_5` and the `CT400_SetBNC` function (*p. 18*) is set to `DISABLE`, the value returned by the `CT400_ScanGetDetectorResampledArray` function is the voltage measured at the BNC port labeled "C".

Example
```
int32_t iArraySize = CT400_GetNbDataPointsResampled(uiHandle);
double *dPowerdet=(double*)calloc(iArraySize,sizeof(double));
ScanGetDetectorResampledArray(uiHandle,DE_1,dPowerdet,
iArraySize);
free(dPowerdet);
```

The result of the operation is the array `dPowerdet [0 to PointsNumber-1]` containing all the re-sampled transfer function values (`PointsNumber ≤ iArraySize`).

2.4.7   CT400_ScanSaveWavelengthSyncFile

Description   This function saves the wavelength points measured by the CT400 in a text file.

In the CT400 GUI, this function is identical to the **Save Wavelength Sync File** button in the **Parameters** tab.

Declaration
```
int32_t CT400_ScanSaveWavelengthSyncFile(uint64_t uiHandle,
                                    char *pcPath);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
pcPath	Path to the file to write (absolute or relative path), Possible file extension: txt	char *

Return Value
- 0: the operation succeeded.
- −1: the operation failed.

Data type: int32

Example
```
CT400_ScanSaveWavelengthSyncFile(uiHandle,"D:\\measurements\\wave
length.txt");
```

2.4.8   CT400_ScanSaveWavelengthResampledFile

Description   This function saves in a text file the wavelength values measured by the CT400 after they have been re-sampled so that the step between values corresponds to the resolution set in the CT400_SetSamplingResolution function (*p. 15*).

In the CT400 GUI, these data is available using the **Save** button in the **Transfer Functions** tab (the difference is that the file only contains the wavelength points).

Declaration
```
int32_t CT400_ScanSaveWavelengthResampledFile(
                    uint64_t uiHandle,
                    char *pcPath);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
pcPath	Path to the file to write (absolute or relative path), Possible file extension: txt	char *

Return Value
- 0: the operation succeeded.
- −1: the operation failed.

Data type: int32

Example
```
CT400_ScanSaveWavelengthSyncFileInterp(uiHandle,"D:\\measurements
\\wavelength_resampled.txt");
```

2.4.9   CT400_ScanSavePowerSyncFile

Description   This function saves in a text file the output power and transfer function values on the enabled detector(s) associated with the recorded pulse number. Recorded values are those measured by the CT400 (not the re-sampled ones).

In the CT400 GUI, this function is identical to the **Save Power Sync File** button in the **Parameters** tab.

Declaration
```
int32_t CT400_ScanSavePowerSyncFile (uint64_t uiHandle,
                                     char *pcPath);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
pcPath	Path to the file to write (absolute or relative path). Possible extension: txt	char *

Return Value   • 0: the operation succeeded.
• −1: the operation failed.
Data type: int32

Example
```
CT400_ScanSavePowerSyncFile(uiHandle,"D:\\measurements\\powerout.txt");
```

2.4.10   CT400_ScanSavePowerResampledFile

Description   This function saves in a text file the output power values associated with the wavelength values measured by the CT400 after they have been re-sampled, so that the step between points corresponds to the resolution set in the CT400_SetSamplingResolution function (*p. 15*). It also saves the re-sampled transfer function on the enabled detectors.

In the CT400 GUI, this function is identical to the **Save** button in the **Transfer Functions** tab.

Declaration
```
int32_t CT400_ScanSavePowerResampledFile (uint64_t uiHandle,
                                          char *pcPath);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
pcPath	Path to the file to write (absolute or relative path). Possible extension: txt	char *

Return Value   • 0: the operation succeeded.
• −1: the operation failed.
Data type: int32

Example
```
CT400_ScanSavePowerResampledFile(uiHandle,"D:\\measurements\\powerout_resampled.txt");
```

2.4.11   CT400_ScanSaveDetectorFile

Description   This function saves in a text file the transfer function values measured by the CT400 on a selected detector associated with the recorded pulse number.

Declaration
```
int32_t CT400_ScanSaveDetectorFile(uint64_t uiHandle,
                                   rDetector eDetector,
                                   char *pcPath);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
eDetector	Detector number. Possible values: see section *CT400 Data Types, p. 11*.	rDetector
pcPath	Path to the file to write (absolute or relative path). Possible extension: txt	char *

Return Value
- 0: the operation succeeded.
- -1: the operation failed.

Data type: int32

Example
```
CT400_ScanSaveDetectorFile(uiHandle,DE_1,"D:\\measurements\\PoutD
et1.txt");
```

2.4.12   **CT400_ScanSaveDetectorResampledFile**

Description   This function saves in a text file the transfer function values on a selected detector associated with the wavelength values measured by the CT400 after they have been re-sampled, so that the step between points corresponds to the resolution set in the `CT400_SetSamplingResolution` function (*p. 15*).

Declaration   ```
int32_t CT400_ScanSaveDetectorResampledFile(uint64_t uiHandle,
                                    rDetector eDetector,
                                    char *pcPath);
```

**Parameters**

| Input Parameter | Description | Data Type |
|---|---|---|
| uiHandle | Handle returned from `CT400_Init` | unsigned int64 |
| eDetector | Detector number.<br>Possible values: see section *CT400 Data Types, p. 11*. | rDetector |
| pcPath | Path to the file to write (absolute or relative path).<br>Possible extension: txt | char * |

**Return Value**     • `0`: the operation succeeded.
                     • `-1`: the operation failed.

                     Data type: int32

**Example**          ```
CT400_ScanSaveDetectorResampledFile(uiHandle,DE_1,"D:\\measuremen
ts\\PoutDet1_resampled.txt");
```

2.4.13   **CT400_ReadPowerDetectors**

Description   This function reads the instantaneous optical power measured at the "Output" port and on the four detectors, and reads the voltage on the BNC port labeled "C". It provides the values implicitly.

In the CT400 GUI, this function is identical to the **Power Control** settings in the **Control/ Calibration** tab with power units in dBm.

Declaration
```
int32_t CT400_ReadPowerDetectors(uint64_t uiHandle,
                                 double *Pout,
                                 double *P1,
                                 double *P2,
                                 double *P3,
                                 double *P4,
                                 double *Vext);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from CT400_Init	unsigned int64
Output Parameter	**Description**	**Data Type**
Pout	Power on the Output port	double*
P1	Power on detector 1	double*
P2	Power on detector 2	double*
P3	Power on detector 3	double*
P4	Power on detector 4	double*
Vext	Voltage on the BNC port labeled "C", located on the rear panel.	double*

Return Value
- 0: the operation succeeded.
- –1: the operation failed.

Data type: int32

Example
```
CT400_ReadPowerDetectors(uiHandle,&Pout,&P1,&P2,&P3,&P4,&Vext);
printf("P1:%f\n",P1);
printf("P2:%f\n",P2);
printf("P3:%f\n",P3);
printf("P4:%f\n",P4);
printf("Vext:%f\n",Vext);
```

2.4.14 CT400_GetNbInputs

Description   This function returns the number of available TLS inputs on the connected CT400 model.

Declaration   `int32_t CT400_GetNbInputs(uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64

Return Value
- ≥0: indicates the number of available inputs on the CT400.
- –1: the operation failed.

Example   `int32_t iNbofInputs =`
`CT400_GetNbInputs(uiHandle);`

2.4.15 CT400_GetNbDetectors

Description   This function returns the number of available optical power detectors on the connected CT400 model.

Declaration   `int32_t CT400_GetNbDetectors(uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64

Return Value
- ≥0: indicates the number of available detector ports on the CT400.
- –1: the operation failed.

Example   `int32_t iNbofDetectors =`
`CT400_GetNbDetectors(uiHandle);`

2.4.16   CT400_GetCT400Type

Description   This function returns the type CT400 model connected to your computer.

Declaration   `int32_t CT400_GetCT400Type(uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- `0`: the CT400 model type is SMF.
- `1`: the CT400 model type is PM13.
- `2`: the CT400 model type is PM15.
- `-1`: the operation failed.

For more details on the operating wavelength range of each CT400 model type, see the corresponding technical specifications in *CT400 User Manual*.

Example   `int32_t CT400Type =`
`CT400_GetCT400Type(uiHandle);`

2.4.17   CT400_GetNbDataPoints

Description   This function returns the number of available data points after a scan has been performed.

Declaration   `int32_t CT400_GetNbDataPoints (uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- `≥0`: indicates the number of available data points in the array.
- `-1`: the operation failed.

Example   `int32_t iPointsNumber =`
`CT400_GetNbDataPoints(uiHandle);`

2.4.18   CT400_GetNbDataPointsResampled

Description   This function returns the number of available re-sampled data points after a scan has been performed.

Declaration   `int32_t CT400_GetNbDataPointsResampled (uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- ≥0: indicates the number of available data points in the array.
- –1: the operation failed.

Example
```
int32_t iPointsNumber =
CT400_GetNbDataPointsResampled(uiHandle);
```

2.4.19   CT400_GetNbLinesDetected

Description   This function returns the number of spectral lines detected with heterodyne detection.

Declaration   `int32_t CT400_GetNbLinesDetected (uint64_t uiHandle);`

Parameters

Input Parameter	Description	Data Type
`uiHandle`	Handle returned from `CT400_Init`	unsigned int64

Return Value
- ≥0: indicates the number of spectral lines detected.
- –1: the operation failed.

Example
```
int32_t iLinesDetected =
CT400_GetNbLinesDetected(uiHandle);
```

2.4.20   **CT400_ScanGetLineDetectionArray**

Description   This function returns the values of spectral lines detected by heterodyne detection.
In the CT400 GUI, this function gets the results displayed in the **Line detection** tab.

Declaration
```
int32_t CT400_ScanGetLinesDetectionArray
      (uint64_t uiHandle,
      double *dArray,
      int32_t iArraySize);
```

Parameters

Input Parameter	Description	Data Type
uiHandle	Handle returned from `CT400_Init`	unsigned int64
dArray	Pointer over an initialized array.	double*
iArraySize	Size of the array. Recommended value is the result of the `CT400_GetNbLinesDetected` function (*p. 37*).	int32

Return Value
- ≥0: the array containing the values of spectral lines detected.
- -1: the operation failed.

Data type: int32

Example
```
int32_t iLinesDetected;
double *dLinesValues = 0;
iLinesDetected = CT400_GetNbLinesDetected(uiHandle);
dLinesValues = (double *) calloc(iLinesDetected, sizeof(double));
CT400_ScanGetLinesDetectionArray(uiHandle, dLinesValues,
iLinesDetected);
for(i = 0; i < iLinesDetected; i++)
 {
  printf("Spectral line #%d : %f\n", i+1, dLinesValues[i]);
  }
free(dLinesValues);
```

3. Error and Warning Messages

This section lists all the possible error and warning messages, and how to handle them.

3.1   Warning Messages

Warning Code 100

Mode hops on the scan

- Cause:
  Mode hopping occurred during the scan on one TLS.

- Possible solution:
  Make sure the source performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.

Warning Code 101

Scan speed too low

- Possible causes:
  - The mean sweeping speed of the TLS is too low (< 8nm/s)
  - The calibration file is not located in the correct folder.

- Possible solution:
  - Raise the scan speed.
  - Make sure the calibration file is located in the proper **Calib** folder:
    - On Windows XP:
      C:\Documents and Settings\All Users\Documents\Yenista Optics\CT400\Calib
    - On Windows Vista and later versions:
      C:\Users\Public\Documents\Yenista Optics\CT400\Calib

Warning Code 102

Scan speed too high

- Cause:
  - The mean scanning speed of the TLS is too high (>120nm/s).
  - The calibration file is not located in the correct folder.

- Possible solution:
  - Decrease the scan speed.
  - Make sure the calibration file is located in the proper Calib folder:
    - On Windows XP: C:\Documents and Settings\All Users\Documents\Yenista Optics\CT400\Calib
    - On Windows Vista and later versions:
      C:\Users\Public\Documents\Yenista Optics\CT400\Calib

Warning Code 103

High dynamical changes at low level, reduce scan speed

- Cause:
  The scan speed and input power are not suited for the measurement.

- Possible solution:
  Decrease the scan speed.

Warning Code 104

Unexpected source behavior, check TLS performance

- Cause:
  The internal wavelength referencing has detected a troubling behavior.

- Possible solution:
  - Make sure the source performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.

  - Make sure that the input power meet the requirements detailed in *CT400 User Manual*, section *Technical Specifications, p. 6*.

Warning Code 106

Low power on one TLS input

- Cause:
  The power in the input port is below -16 dBm.

- Possible solution:
  - Make sure the jumper is clean and properly connected to the optical input of the CT400.
  - Make sure the TLS performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.

Warning Code 108

Numerous mode hops or multimoding behavior

- Cause:
  - The TLS is out of specification due to multimode behavior or numerous mode hops.
  - The calibration file is not located in the correct folder or is corrupted.

- Possible solution:
  - Make sure the source performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.
  - Make sure the calibration file is located in the proper Calib folder:
    - On Windows XP: C:\Documents and Settings\All Users\Documents\Yenista Optics\CT400\Calib
    - On Windows Vista and later versions: C:\Users\Public\Documents\Yenista Optics\CT400\Calib

  - Reset the calibration file as follows: in the **Calib** folder, delete the calibration file *filexxxxxxxxxxx.dat* file (where *xxxxxxxxxxx* is the serial number of the CT400) and restart the program. The original factory calibration file is automatically downloaded from the CT400.

Warning Code 109

High TLS input power variations: check TLS sources

- Cause:
  The power in the input port varies randomly above the specified value.

- Possible solution:
  Make sure the source performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.

Warning Code 110

Too high input power during the scan: check input power

- Cause:
  The power in the input port is above the specified value.

- Possible solution:
  Decrease the input power to meet the specifications detailed in *CT400 User Manual*, section *Technical Specifications, p. 6*.

**Warning Codes
111 to 114**

Power on DET1/DET2/DET3/DET4 too high: check set up

- Cause:
  The power on the detector is above specified value.

- Possible solution:
  Decrease the input power to meet the specifications detailed in *CT400 User Manual*, section *Technical Specifications, p. 6*.
  Do not use and optical amplifier before the detector.

**Warning Code
115**

Spurious input power: check set-up

- Cause:
  Optical power has been detected where it is not supposed to happen.

- Possible solution:
  Verify that the lasers are connected to the correct input ports of the CT400.

**Warning Codes
117 to 120**

Power too low on IN port 1/port 2/port 3/port 4

- Cause:
  The power in the TLS input port is below the specified value.

- Possible solution:
  - Make sure the jumper is clean and properly connected to the corresponding optical inputs
  - Make sure the TLS performances meet the requirements detailed in *CT400 User Manual*, section *Product Features, p. 5*.

**Warning Codes
121 to 124**

Power too high on IN port 1/port 2/port 3/port 4

- Cause:
  The power in the TLS input port is above the specified value.

- Possible solution:
  Decrease the input power to meet the specifications detailed in *CT400 User Manual*, section *Technical Specifications, p. 6*.

**Warning Code
999**

The current sampling resolution does not allow to memorize all the points. The sampling resolution has been set to x pm.

- Cause:
  The resolution is too high for the running conditions. Scan is performed.

- Possible solution:
  Modify the parameters (wavelength range, number of detectors).

3.2   Error Messages

Error Code 1   **The measurement has been canceled by the user**

This error only occurs if the `CT400_ScanStop` function has been called by the user.

Error Code 2   **Failure in data exchange with the DSP**

- Cause:
  A failure occurred during the communication between the computer and the internal DSP.
  For example, the number of points expected by the computer does not correspond to the number of points sent by the DSP.

- Possible solution:
  - Check the USB connection.
  - Contact the **Yenista Optics** customer support service (see section *Contact Information, p. 4*).

Error Code 3   **Error in the wavelength referencing**

- Possible cause:
  - The calibration file is not saved in the correct folder.
  - If more than one laser is used, at least one input port is inversed.

- Possible solution:
  - Check the input power of the laser
  - Make sure the calibration file is located in the proper Calib folder:
    - On Windows XP: C:\Documents and Settings\All Users\Documents\Yenista Optics\CT400\Calib
    - On Windows Vista and later versions: C:\Users\Public\Documents\Yenista Optics\CT400\Calib

- Verify that the lasers are connected to the correct input port of the CT400, so that the wavelength ranges follow the port order.

Error Code 4   **Switch failure**

- Cause:
  Hardware failure on the optical switch.

- Possible solution:
  Contact the **Yenista Optics** customer support service (see section *Contact Information, p. 4*).

Error Code 5   **Failure in the communication with the DSP**

- Cause:
  No communication at all is established between the computer and the internal DSP.

- Possible solution:
  - Check the USB connection.
  - Contact the **Yenista Optics** customer support service (see section *Contact Information, p. 4*).

4. Importing the CT400_lib.dll Library in LabVIEW

Subject   This section explains how to create LabVIEW wrappers for DLL functions using the built-in import shared library wizard in LabVIEW.

The CT400_lib.dll contains multiple functions with different types of parameters and return types, ranging from integers, arrays and strings.

The following procedure explains how to create LabVIEW wrappers for DLL functions using the built-in import shared library wizard in LabVIEW. It creates (or updates) a VI for each function of the selected DLL.

You can use any LabVIEW version that includes this wizard. In the following procedure, LabVIEW version 13.0 is used. You can find a similar procedure on creating wrappers on the National Instrument website.
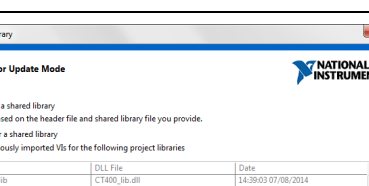
Procedure   1. Open LabVIEW and start the **Import Shared Library Wizard** as follows:

In the **Tools** menu, select **Import** -> **Shared Library (.dll)**

The **Import Shared Library Wizard** appears.
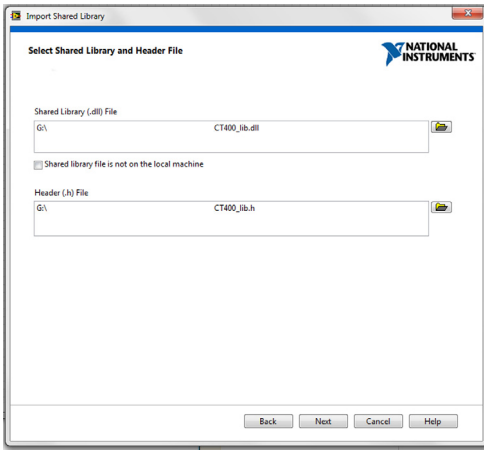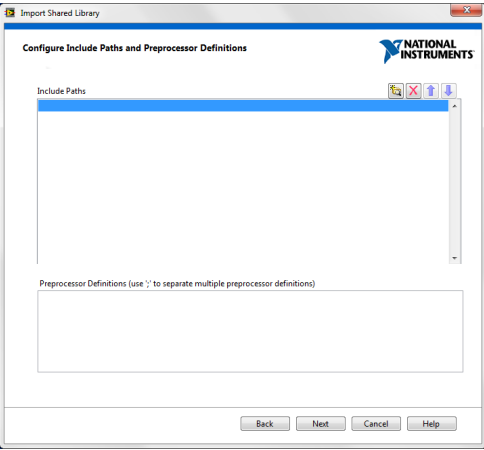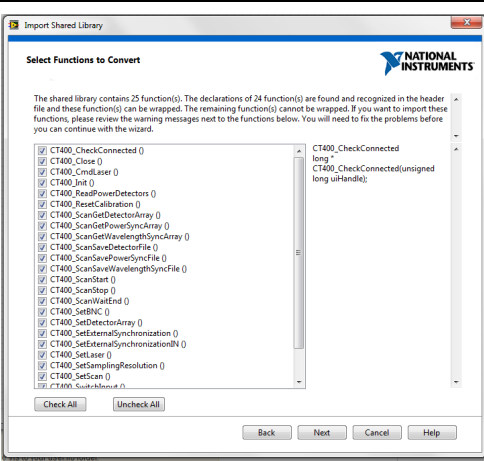
2. Follow the instructions given in the following table:

When this window appears...	Do the following...
	1. Do one of the following: • If you are importing the CT400 DLL for the first time, select **Create VIs for a shared library**. • If you are updating the CT400 DLL, select **Update VIs for a shared library**. 2. Click **Next**.

When this window appears...	Do the following...
	1. Select the location of the **CT400_lib.dll** and **CT400_lib.h** files on your computer according to the platform used (Win 32 or Win 64). 2. Make sure the **CT400_Types.h** is located in the same directory as the **CT400_lib.h** file. 3. Click **Next**.
	1. Leave the default parameters. 2. Click **Next**. The wizard parses your header file and lists all the functions available in the shared library file.
	1. Select the functions you want to import. If the wizard can not import a function, the function appears with a glyph beside the function name 2. Click **Next**.
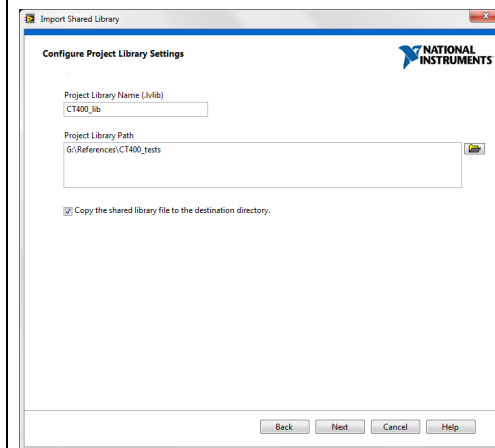
When this window appears...	Do the following...
	1. Type a name for the library of wrapper VIs and select its location. **2.** Click **Next**.
	1. In the **Error Handling Mode** list, select **Simple Error Handling**. **2.** Click **Next**.
	This window allows you to set up the Call Library Function Node settings for each individual function. **1.** Leave the default parameters. You will be able to go into the individual VIs and modify the **Call Library Function Node** at a later time. **2.** Click **Next**.
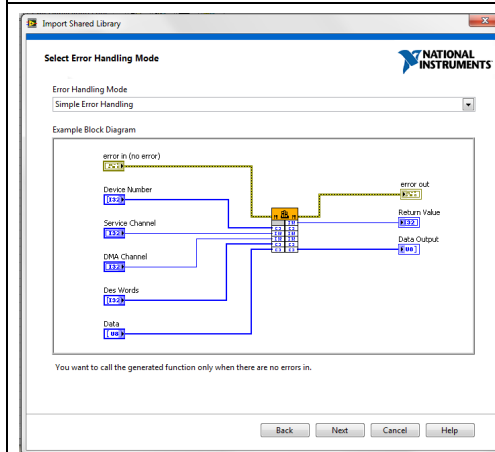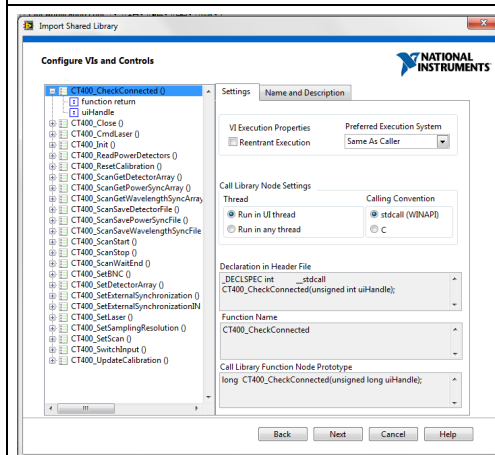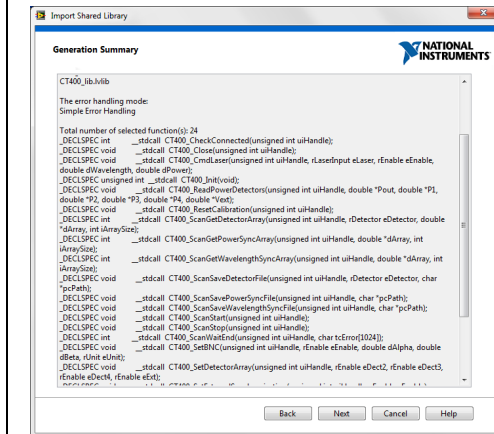
When this window appears...	Do the following...
	• Click **Next**.

3. Click **Finish** to view the generated library containing the wrappers VIs.
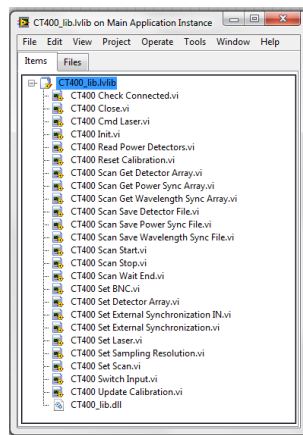
4. To run any of the created VIs, copy the **SiUSBxp.dll** file corresponding to the platform used (Win32 or Win64) in the folder created in the **Configure Project Library Settings** window. This file enables the communication with the CT400.

    IMPORTANT   You must locate your user programs in the same folder.

    The created wrappers VIs are available through the **Functions** tool by right-clicking in the block diagram as shown in the following figure:
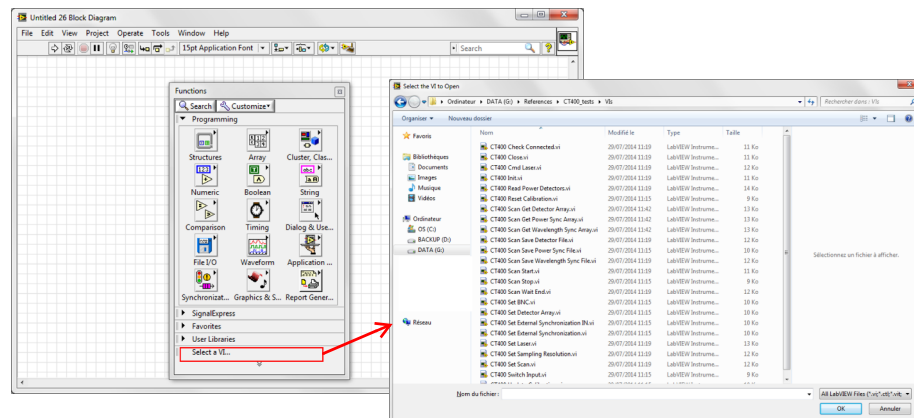
5. Program Examples

Subject

Yenista Optics provides an example on how to use the CT400_lib.dll in C, Python and LabVIEW programming languages. An example is available for each platform (Win32 and Win64).

- CT400_testwrap.c is an example of the DLL use in C language.
  It has been edited, compiled and tested using the IDE Netbeans v. 8.0 along with the C compiler MinGW4.7 for Win32, and MSVC++ 2010 Express for Win64.

- CT400_testwrap.py is an example of CT400 control file in Python language.
  It has been edited, compiled and tested using the IDLE and Python v. 2.7.7 for Win32 and v. 3.4.2 for Win64.
  CT400_testwrap.vi is an example of the DLL use in LabVIEW version 9.

  It has been edited and tested using the LabVIEW v. 13.0. The example is provided in LabVIEW version 9 for both Win32 and Win64 platforms. Before opening this example, you must have created wrappers as explained in section *Importing the CT400_lib.dll Library in LabVIEW, p. 43*.

Example Description

The provided examples perform the following operations:

1. It initializes the DLL library and verifies if the communication between the computer and the CT400 has been established.

   - If the communication is established, the configuration of the CT400 is carried out.
   - If the communication is not established, the program ends by closing the communication with the DLL.

2. The configuration detects the number of available inputs and detectors of the CT400, the type of CT400 and sets the following:

   - the sampling resolution of the measurement to be made,
   - the laser type and laser parameters (e.g. lower and upper wavelengths, speed of scan, GPIB address etc.),
   - the wavelength scan range
   - detector of the CT400 in active state

3. A scan is performed over the wavelength range, previously set in the configuration section of the program, and awaits for the scan to finish.

   If an error occurred during this process, the program displays the error or warning message generated by the CT400.

4. Once the scan is finished:
   - The measured wavelength, corresponding optical power output and IL data (both in standard and re-sampled versions) are saved in the in the files named "Lambda_Sync.txt", "Output_Sync.txt" and "Output_Detector1_Sync.txt", respectively.
   - The values of the detected spectral lines are displayed.
   - The measured wavelength, optical power and corresponding IL data are assigned to the corresponding output arrays and displayed.

5. The programs end by reading and displaying the static power on the detectors named "Pout", "P1," "P2", "P3", "P4" and the voltage on BNC port labeled "C" named "Vext".
   These operations may differ from example to example.

6. The communication between the CT400 and the computer is closed and the memory allocated by the dll is released.

List of Functions