Matisse Programmer's Guide

Version 2.4

Contents

Chapter 1

Preface

Dear Programmer,

this manual will provide you with information on how to control *Matisse* laser devices programmatically. There are different approaches depending on which programming languages and environments you like to work with. In the following it is expected that you are familiar with the *Matisse Commander* control program and the concepts of controlling the various optical components of a *Matisse* laser system.

For communication between the *Matisse* laser device and a computer, the Universal Serial Bus (USB) interface is used. The *Matisse* implements a USB port according to the Universal Serial Bus Specifications Revision 2.0 (see www.usb.org for details). The *Matisse* conforms to the USB Test and Measurement class (USBTMC) according to the specification Revision 1.0. It can be used with National Instrument's NI-VISA USB INSTR class.

Based on National Instruments' VISA runtime a LabVIEW API library is provided, that implements all *Matisse* device commands. This library is presented in *Chapter 2*.

See *Chapter 3* for examples of other programming languages such as Python or C++.

Adapting and extending the functionality of the *Matisse Commander*, which is a LabVIEW application, will be explained in *Chapter 4*. The Wavemeter system and its applications are described in detail in *Chapter 5*.

Chapter 6 will provide a complete reference of the *Matisse* device commands, which are simple text messages. With these information it is possible, e.g., to use the C interface of National Instruments' VISA library to control the *Matisse* device instead of the LabVIEW environment.

Future editions of this manual can be certainly improved by feedback from its readers. So any suggestions or questions are welcome.

Chapter 2

LabVIEW Library

The *Matisse* API library (the combination of `LowLevelAPI.lvlibp` and `Matisse API.lvlibp`) introduces an interface to LabVIEW programmers to build applications controlling *Matisse* lasers. It provides VIs, that implement the *Matisse* device commands, and adds communication logic, e.g., a synchronized access to the laser device.

Requirements

Communication between a Computer and the *Matisse* device typically relies on the Virtual Instruments Software Architecture (VISA). A NI-VISA Runtime software is therefore required. During installation of the *Matisse Commander* Software a NI-VISA Runtime will be installed, if appropriate software is not already present.

Aside from VISA, you may also use a network connection to the *Matisse Commander* installed on the same or a different computer. All commands sent to the *Matisse Commander* over the network connection are relayed to the *Matisse* device.

The *Matisse* API library can be opened and used with LabVIEW 2020 and higher.

Introduction

You can find the Source code for the *Matisse Commander* and the following examples in the file `LabVIEW - Matisse Commander.zip` located in the Addons folder of your *Matisse Commander* installation or on your Sirah DVD. The following figure shows the block diagram of an example VI, that uses *Matisse* API VIs to set the laser to a specified wavelength with the help of the built-in calibration function:

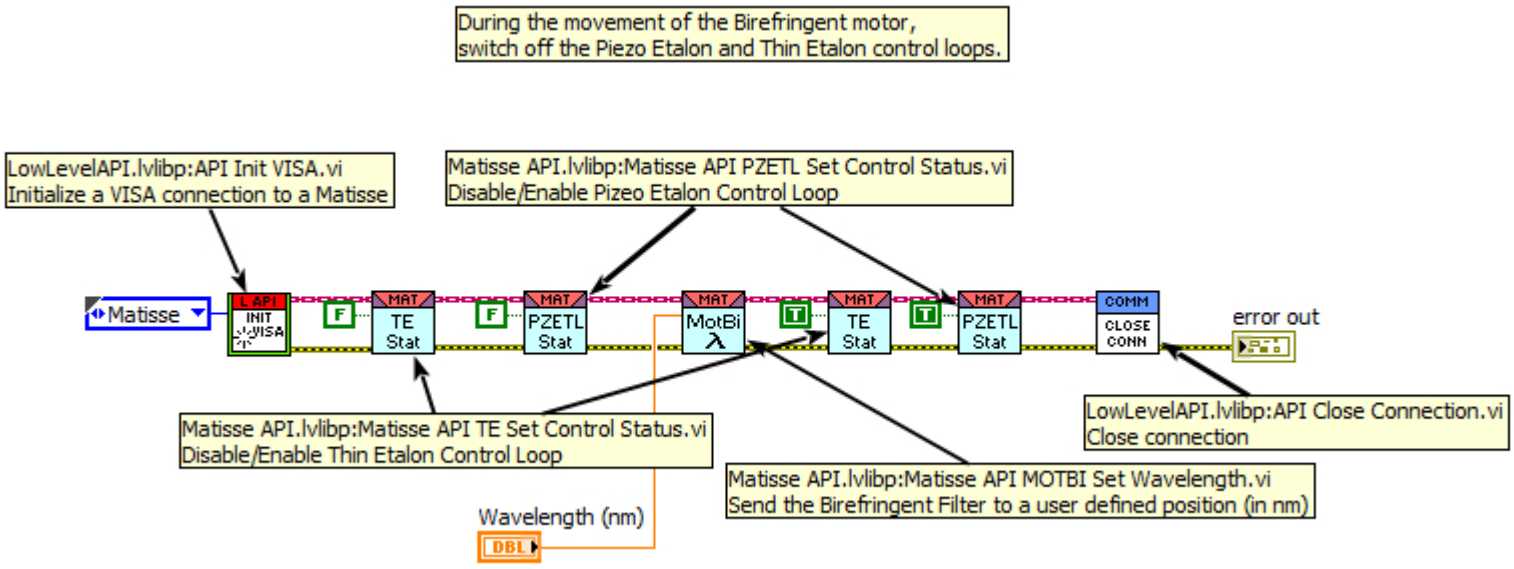


Figure 1: LabVIEW VISA Programming Example

If you always connect to the same *Matisse* device, you may use `LowLevelAPI.lvlibp:Add Visa Connection.vi` instead of the above introduced initializing VI. In this case, you need to connect an available

VISA resource so that LabVIEW does not need to search for VISA devices. This speeds up the runtime of the VISA initialization.

You may also use a network connection to a server program, that relays all *Matisse* commands. The main advantage of this network feature is the possibility to use two independent programs, such as the *Matisse Commander* and the LabVIEW Development Environment, which can be connected to the same device at the same time. The National Instruments implementation of VISA restricts one Windows process to connect to a VISA device. Please refer to the *Matisse Commander*'s User Guide describing the window **Matisse** > **Communication Options**.

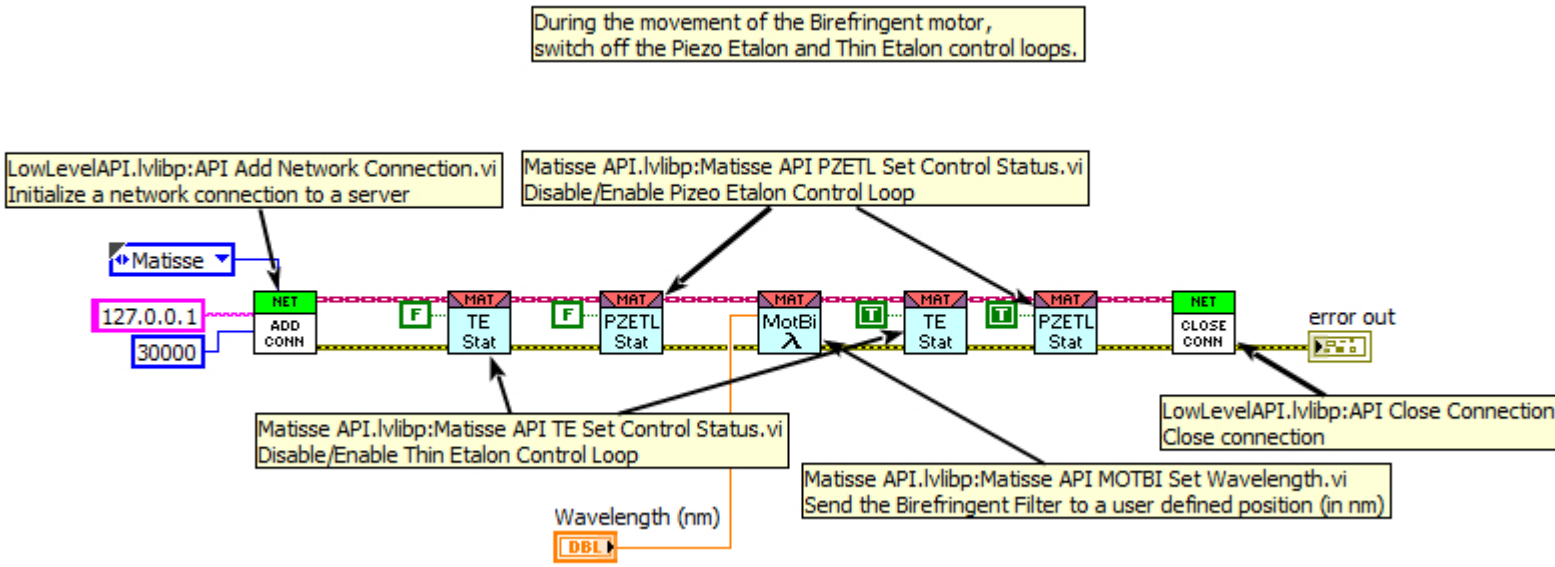


Figure 2: LabVIEW Network Programming Example

The example for a connection is very similar to the VISA connection. The main difference is the initialization of the connection. The VI closing the connection adapts to the type of connection automatically. Thus, if you replace `API Init VISA.vi` in the first example with `API Add Network Connection.vi` as shown in the second example, the icon of the closing connection VI changes.

Chapter 3

Access the laser via other programming languages

All communication with *Matisse* devices uses the VISA specification. Libraries for accessing VISA devices are available in many programming languages. With them it is possible to use all *Matisse* Device Commands (see *Chapter 6*) for direct control of the device. Examples for the languages *C#/C++* and *Python* can be found in the following section.

For more complex tasks involving additional instruments, you may want to leverage our code that is already written in LabVIEW. Fortunately, it is possible to compile LabVIEW into a *DLL-file* that can be run by virtually any programming language.

Please mind that there can only be active connection to a *Matisse* Device at any time, but it is possible to share a connection with the *Matisse Commander* Network Server.

Direct Access

VISA C(++) code

To utilize VISA in Visual C# or C++, please open a new project, create a windows console application and copy the C#/C++ example into a source file of your project.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include  "C:/Program Files/IVI Foundation/VISA/WinNT/include/visa.h"
#ifdef WIN64
 #pragma comment( lib, "C:/Program Files/IVI Foundation/VISA/WinNT/lib/
msc/visa64")
#else
 #pragma comment( lib, "C:/Program Files/IVI Foundation/VISA/WinNT/lib/
msc/visa32")
#endif

/* The VISA header must be included to define the VISA related commands.
 This header is usually in the
  "C:/Program Files/IVI Foundation/VISA/WinNT/include/"
   You also need to point to visa library, which is platform dependent.
   The .lib-file is usually in "C:/Program Files/IVI Foundation/VISA/
WinNT/lib/msc/".
*/

static char instrDescriptor[VI_FIND_BUFLEN];
static ViUInt32 numInstrs;
static ViFindList findList;
static ViSession defaultRM, instr;
static ViStatus status;
static ViUInt32 retCount;
static ViUInt32 writeCount;
static unsigned char buffer[100];
static char stringinput[512];
```

```
int main(void)
{
    /*
    First we will need to open the default resource manager. Check the
 status, if an error occurred. VISA status codes above 0 are warnings,
 below 0 errors and just 0 means everything is okay.
    */
    status = viOpenDefaultRM (&defaultRM);
    if (status < VI_SUCCESS)
    {
       printf("Could not open a session to the VISA Resource Manager!\n");
       exit (EXIT_FAILURE);
    }

    /* Now we search for Sirah laser devices. VISA Test and Measurement
 USB devices have resource format like
       USBn::0xvvvv::0xdddd::ssssss::INSTR
       where n is a rising integer, numbering each USB Test and
 Measurement USB device. vvvv stands for the vendor id in hexadecimal,
 which identifies the production company of this device. dddd is
 the hexadecimal code for the device id and ssssss is the device's
 serial number. A Sirah Matisse TR laser must have the format:
 USB[0-9]*::0x17E7::0x0101::??-??-??::INSTR with a unique serial number.
 Other Matisse types have different device ids:
       Matisse TR   0x0101
       Matisse TS   0x0102
       Matisse TX   0x0103
       Matisse DR   0x0104
       Matisse DS   0x0105
       Matisse DX   0x0106
       Matisse TX light  0x0107
    */
    status = viFindRsrc (defaultRM,
 "USB[0-9]*::0x17E7::0x0101::??-??-??::INSTR", &findList, &numInstrs,
 instrDescriptor);
    if (status ; VI_SUCCESS)
    {
       printf ("Error: %d: An error occurred while finding resources. A
 Sirah VISA device could not be found.\n",status);
       goto Close;
    }

    printf("Found device: %s \n",instrDescriptor);

    /* If you have more than Sirah device, you can check the number of
 found lasers:
          printf("%d Number Sirah devices found:\n\n",numInstrs);
       You may use
          status = viFindNext (findList, instrDescriptor);  // find next
 device
       to toggle findList to the VISA resource (or laser device) you want
 to use.
    */

    // Now we will open a session to the first instrument we just found.
    status = viOpen (defaultRM, instrDescriptor, VI_NULL, VI_NULL,
 &instr);
    if (status < VI_SUCCESS)
    {
       printf ("Error: %d: An error occurred opening a session to %s
\n",status,instrDescriptor);
    }
    else
    {
    //Set timeout value to 5000 milliseconds (5000 milliseconds).
    status = viSetAttribute (instr, VI_ATTR_TMO_VALUE, 5000);

    strcpy(stringinput,"IDN?");
```

```
   /*
    stringinput is the VISA command for the laser. You may also ask
 the laser for the current Thin Etalon motor position with "MOTTE:POS?"
 instead of "IDN?" or send it to a Thin Etalon motor position with
 "MOTTE:POS 15000".
    */
    status = viWrite (instr, (ViBuf)stringinput,
 (ViUInt32)strlen(stringinput), &writeCount);
    if (status < VI_SUCCESS)
    {
        printf("Error: %d: Error writing to the device\n",status);
        goto Close;
    }

    /*
       Now we will attempt to read back a response from the device to the
 identification query that was sent.  We will use the viRead function to
 acquire the data.  We will try to read back 100 bytes. After the data has
 been read the response is displayed.
    */
    status = viRead (instr, buffer, 100, &retCount);
    if (status < VI_SUCCESS)
    {
        printf("Error: %d: Error reading a response from the device
\n",status);
    }
    else
    {
       printf("Data read: %s\n",buffer);
    }
        viClose (instr);
    }
      viClose (instr);
    }

    Close:
    status = viClose(findList);
    status = viClose (defaultRM);
    printf ("\nHit enter to close.");
    fflush(stdin);
    getchar();

    return 0;
}
   Close:
   status = viClose(findList);
   status = viClose (defaultRM);
   printf ("\nHit enter to close.");
   fflush(stdin);
   getchar();

   return 0;
}
```

Python

The Python package PyVISA acts as wrapper for the same LabVIEW VISA library that is used by Matisse Commander (*https://pyvisa.readthedocs.io/en/latest/index.html*). Thus the necessary backend was already installed with *Matisse Commander* as long as you use Python with the same bitness (*32-bit*).

For an example please the file `MatissePyVisa_Ex.py` in the Examples folder of your Matisse Commander installation.

Native Python also provides a TCP/IP interface to the *Matisse Commander* Network Server. This allows you to connect to a running instance of *Matisse Commander* on your local network and send additional commands through the existing VISA connection. Thus giving you programmatic access and a user interface at the same time. A Python module implementing such a connection can be found in the following Git repo:

https://github.com/nelsond/sirah-matisse-commander

Access via a compiled DLL

Direct access is usually sufficient, but the more complex procedures such as GoTo Position or Extended Scan require a wavemeter and extensive logic. Matisse Commander implements these procedures in LabVIEW. To use them in other programming languages, you can compile them into a shared library (DLL). The following LabVIEW Knowledge article details the general approach:

https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YHxICAW

A sample DLL can be found in the folder `Examples\ExampleDLL` of the zip file `Addons\LabVIEW - Matisse Commander.zip` in your *Matisse Commander* installation. The LabVIEW example project exports 3 VIs into the DLL:

GoToPosition_Ex.vi	GoTo procedure that tunes the laser to a specific frequency using a Wavemeter.
GoToPosition_WithVisa.vi	Uses the given VISA identification string to select the laser to perform the GoTo procedure.
GoToPosition_Network.vi	Does the GoTo procedure on a Laser controlled by a Matisse Commander Network Server.

The DLL is pre compiled in the build folder. The files `DLLcall_ex.py` and `DLLcall_Network.py` serve as examples for using the DLL in Python.

Chapter 4

Matisse Commander Plug-ins

The functionality of the *Matisse Commander* software, that is developed with LabVIEW, can be enhanced by using LabVIEW application libraries ('Plug-ins'), that conform to a specific interface. Due to the constraints of the LabVIEW environment, these Plug-ins have to be compiled with the same LabVIEW version the *Matisse Commander* was produced with. At the time of writing this guide, this is LabVIEW 2020 *32-bit*.

Opening the Plug-in template

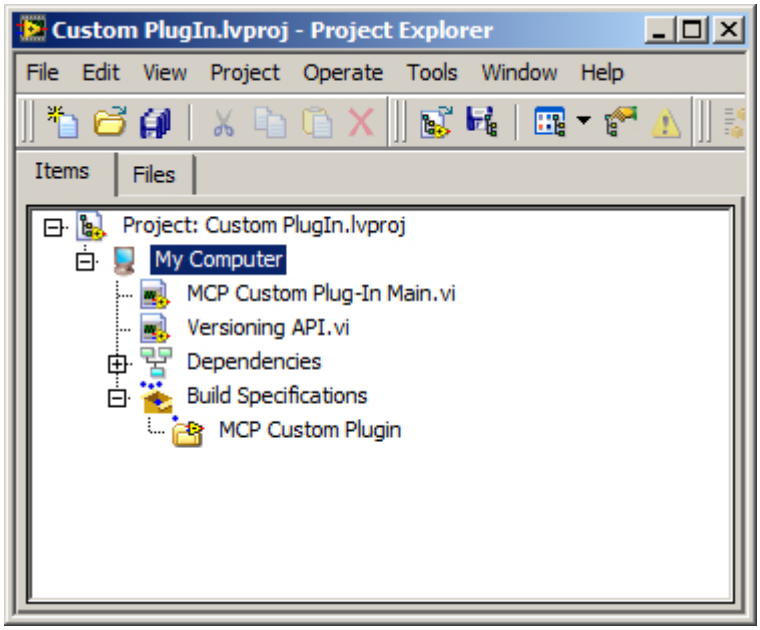


Figure 3: Custom Plug-in LabVIEW Project

To start a new *Matisse Commander* Plug-in, extract the file `Matisse API.zip` from the `Addons` subfolder of your *Matisse Commander* installation or Sirah DVD. In this zip file is a LabVIEW project (lvproj file), which is used to create a new Plug-in. Alternatively, you can create an empty project to start from scratch.

Upon *Matisse Commander* start, the program scans its home directory and all subdirectories for LabVIEW VI libraries (`*.llb` files). All libraries with filenames that conform to `MCP*.llb` are further investigated. Each of these libraries need to have a Main VI, which is identified by the naming convention `*Main.vi`. By using the Plug-in template `MCP Custom Plug-In Main.vi` this convention is met.

The main VI is the topmost VI in the Plug-in as it represents the communication interface with the *Matisse Commander*, that is called during the *Matisse Commander* start and for any menu selections (for details see *Plug-in Interface Specification*).

If you want to start from scratch, open an empty project, add an empty VI to the project and save it, e.g. using the filename `MCP Myownplugin Main.vi`. Proceed with this main VI as described in the following section.

Plug-in Interface Specification

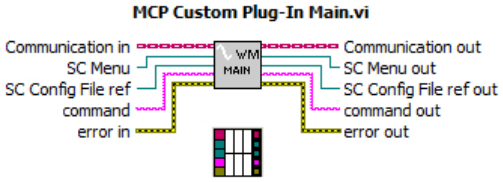


Figure 4: Plug-in Main VI Connector Pane

The Main VI has to use a specific connector pane as shown in *Figure 4*.

Only the Main VI is directly called from the *Matisse Commander*. There are different reasons for calling this main VI. The following list shows some predefined commands, that should be implemented in any *Matisse Commander* Plug-in.

`AppInit`	The main VI is called with this command upon the start of the *Matisse Commander*. Typically, the Plug-in will add some specific menu items to the menu bar of the *Matisse Commander* or read parameters stored into the *Matisse Commander* configuration file. For this reason, references to the menu bar and the configuration file are provided. Moreover you can initialize devices, that are used from your Plug-in.
`AppClose`	This command is passed to the Main VI when the *Matisse Commander* is about to terminate. Typically, this command will initiate saving of parameters to the configuration file and force the close down of dynamically called VIs.
`Version`	When opening the **About** window of the *Matisse Commander*, all installed Plug-ins are asked about their name and version. The string command out, that is answered on this command, will be shown in the **About** window.

Beside these three pre-defined commands, all item tags clicked on the menu bar are sent to the Plug-ins' main VI. In this way, the main VI can define Plug-in specific menu items and detect the selection of these items by the item tags send to the Main VI.

For your own commands, you have to create menu entries to make them accessible via the *Matisse Commander*. Please use sensible names like `MCP_*` for new menu item tags, where `*` denotes a meaningful specifier for your Plug-in.

Important: Menu item tag starting with `MC_*` are forbidden to use to avoid conflicts with *Matisse Commander* internal tags.

Catching Plug-in commands

To illustrate the usage of the Plug-in mechanism, let's take a look at the custom Plug-in, again. If you want to create a new main VI for your Plug-in, please ensure, you select the controls and indicators to the connector pane as indicated in the example below.

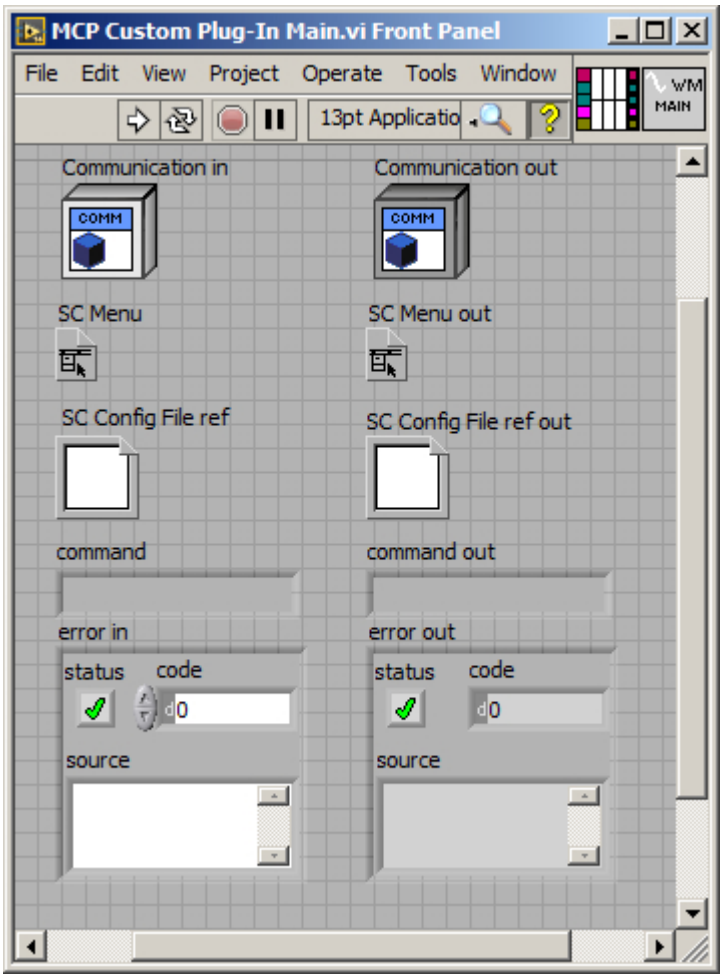


Figure 5: Plug-in Main VI Front Panel

On the block diagram, create a case structure that uses the command input for case selection. Define the Default case, where nothing happens.

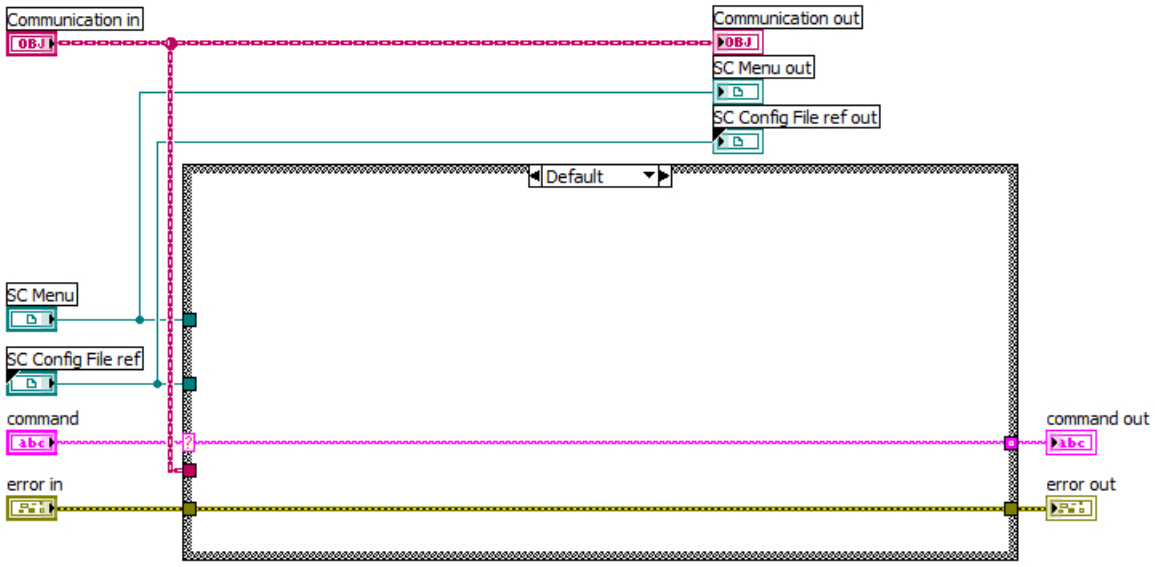


Figure 6: Plug-in Main VI block diagram Default case

Now, add a case for each command `AppInit`, `AppClose` and `Version`.

For the case `Version`, you may set command out to any string describing the version of your Plug-in. command out in the `Version` case is displayed in the *Matisse Commander*'s **About** window.

The `AppInit` case initializes the Plug-in during the *Matisse Commander* startup. Please put here all routines for initializing the Plug-in itself and all devices you might utilize in your Plug-in. Also place here the initialization of the menus of the Plug-in as shown in *Figure 7*.

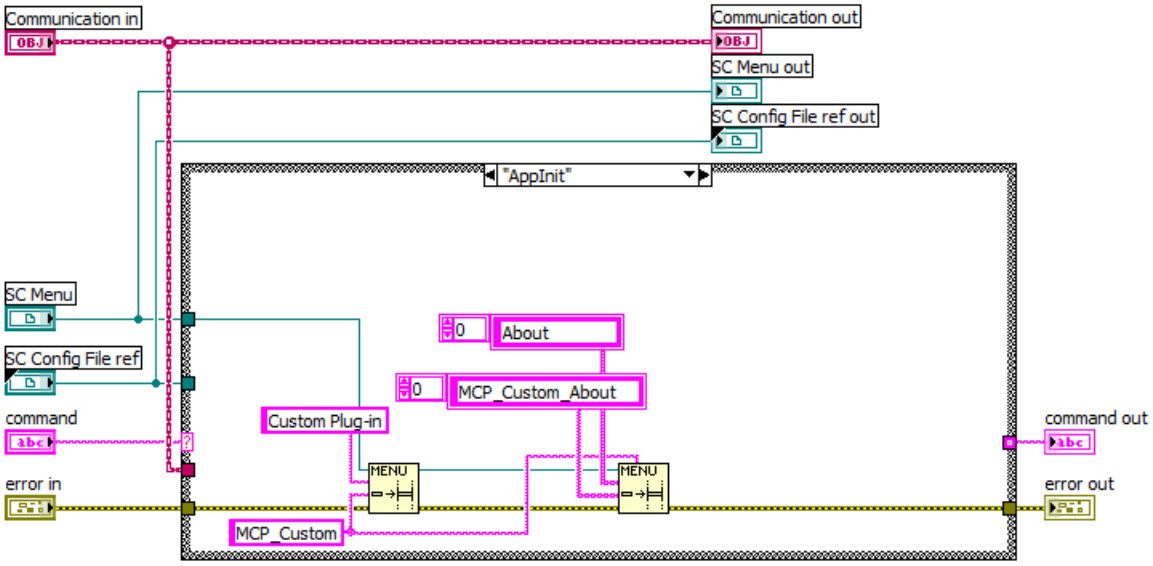


Figure 7: Plug-in Main VI block diagram `AppInit` case

The above example shows how to insert a new menu **Custom Plug-in** and filling it with the item **About**. You need to define for both, the menu and each menu item, an item name and an item tag. Please be aware, that the item tag should be unique for the entire *Matisse Commander* menu. You may add as many menu items as you want by adding new string values into the menu item name and tag arrays.

Important: Menu item tag starting with `MC_*` are forbidden to use to avoid conflicts with *Matisse Commander* internal tags.

Similar to the `AppInit` case, define a case for `AppClose`, e.g. for closing custom connections, that have been established beforehand by the Plug-in. This case is only executed, when the *Matisse Commander* is shutting down. You don't need to unload the menues loaded in the `AppInit` case.

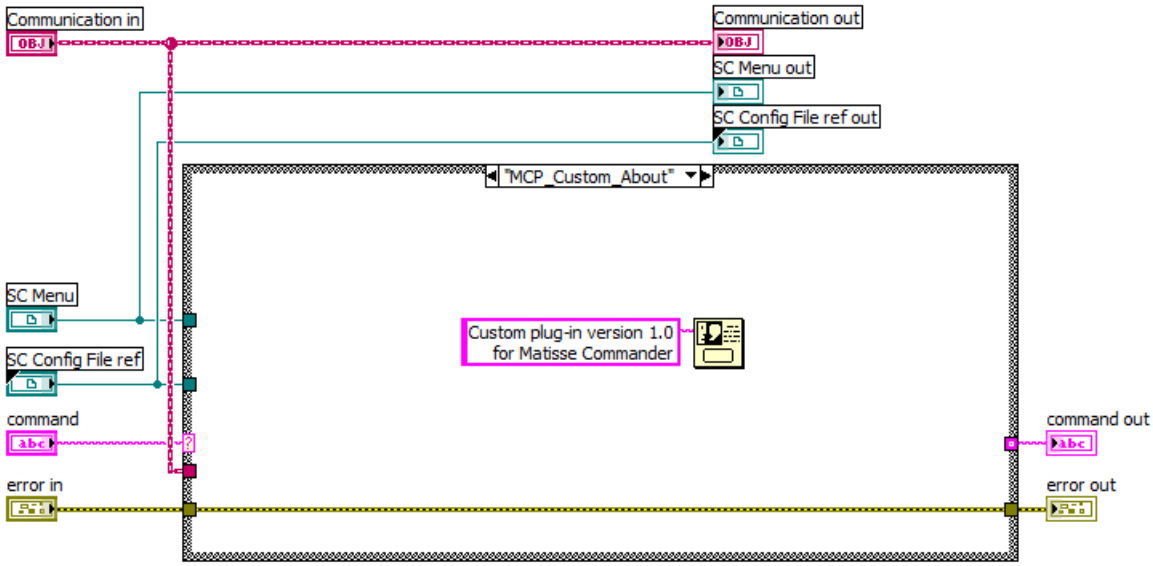


Figure 8: Plug-in Main VI block diagram `MCP_Custom_About` case

Now, we need to define what to do, when the user selects the newly created menu item **About**. So, we need to add an according case to the structure and define what to do. For selecting **Custom Plug-in** > **About** in the *Matisse Commander*, we just want to display a dialog box with a simple version information. When you select the **About** menu item, the *Matisse Commander* will feed the string `MCP_Custom_About` in the **command** input and call the main VI of all Plug-ins. All other Plug-ins except this one should default to the empty case, since

they don't know anything about the command `MCP_Custom_About`. But this Plug-in knows the command `MCP_Custom_About` and runs the code in the if-clause shown in *Figure 8*.

For each menu entry you declared in `AppInit`, add a case in the if-clause, that is called in correspondence to the menu item tag. In these if-cases, you may include VIs, that are visible during runtime. That enables you to show new windows in the *Matisse Commander*. Now, you can create the Plug-in library as described in *Application Library Creation*.

Application Library Creation

Note: The Plug-in libraries for the *Matisse Commander* must be created using a LabVIEW 2020 system.

To build an application library, you need to add your main VI and all relevant VIs to the project tree (as shown in *Figure 3*). You can only directly include VIs into the Plug-in, that are listed in this project tree. All dependencies of these VIs are included automatically. Thus usually, it's enough to add your main VI into this project tree. Save your project. To build your Plug-in, proceed through the following steps. If you use the Custom Plug-in template, start with step *6*.

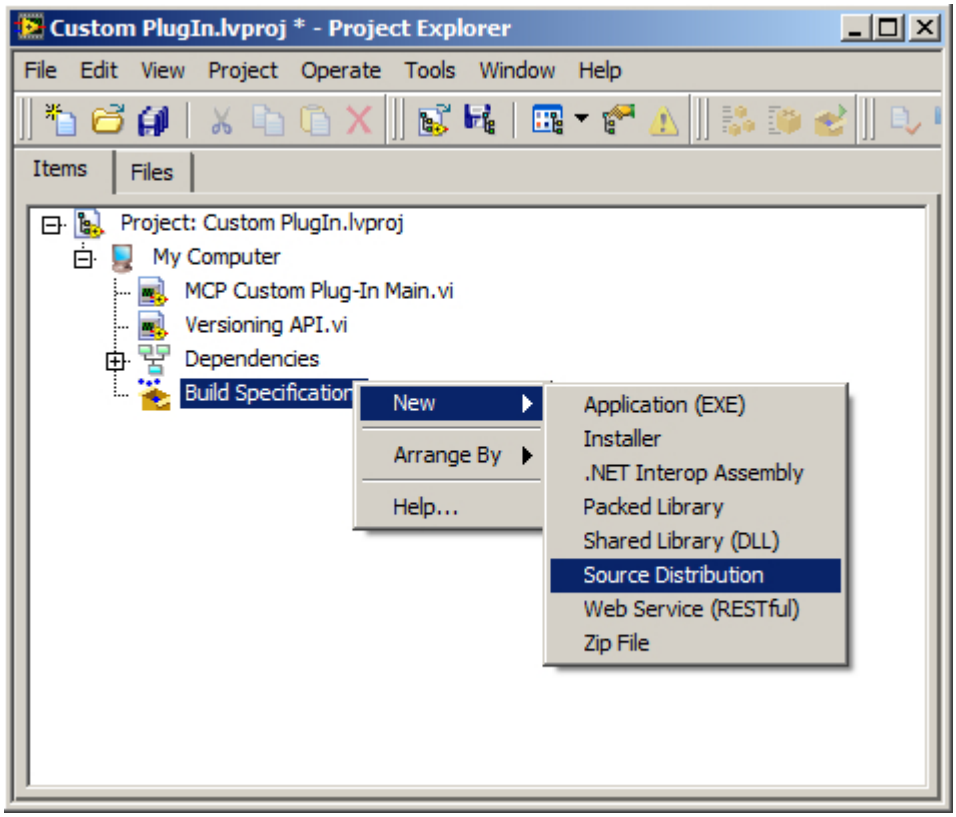
1. 

Figure 9: Create a new Build Specification by right clicking on Build Specification > New > Source Distribution

   Create a new **Build Specification** for your project: right-click on **Build Specification**. Select in the pop-up menu **New** > **Source Distribution**.
2. In the upcoming dialog, open the category **Information** and choose a new name for **Build specification name**, e.g. "MCP MyFunction".
3. In the **Source Files** category, select your main VI and add it to the **Always Included** list.
4. In the **Destinations** category, first select **LLB** as **Destination type** and set then your **Destination Path** as desired.
5. Close the dialog with **OK**.

6.

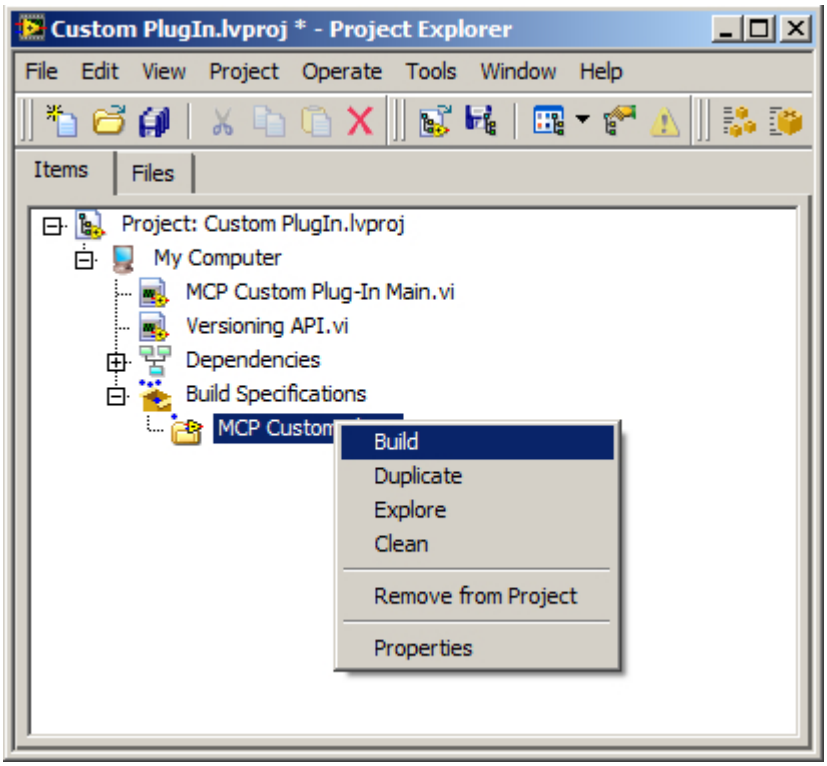


Figure 10: Building the LabVIEW library.

The Plug-in library is ready to build now. Right click on your newly created Build Specification and select **Build**. The llb file will be saved into the **Destination Path** previously set.

7. Copy the LabVIEW library into the *Matisse Commander* program directory and start the *Matisse Commander*. Your Plug-in should show up in the Plug-in list of **Matisse** > **Plug-ins**. Select your Plug-in and restart the *Matisse Commander*.

Tip: Also copy all needed external dependencies in the *Matisse Commander* directory, like dynamically linked libraries (DLLs), if they are not installed in directories, that are part of the systems search path, e.g., on Windows this is the PATH environment variable.

Chapter 5

Wavemeter Integration

Extending *Matisse Commander*'s functionality using devices capable of measuring the laser's current wavelength (further referred to as “Wavemeters”) are using a different Plug-in structure as explained above.

Use of a Wavemeter Plug-in

If you want to create your own *Matisse Commander* Plug-in, that should read the current wavelength from a Wavemeter, use the **WaveMeter.lvlibp:wm_singleton** class. It will be loaded and initialized at the start of *Matisse Commander* and manages all Wavemeter related operations. Add the `WaveMeter.lvlibp` and `wm_interface.lvlibp` files to your project. In the `WaveMeter.lvlibp` library locate the VIs: `Get Instance.vi` and `Get Read Wavelength extern Settings.vi`. To read the wavelength you have to first get an wm_singleton Instance and pass it to `Get Read Wavelength extern Settings.vi`. The code should look like this:

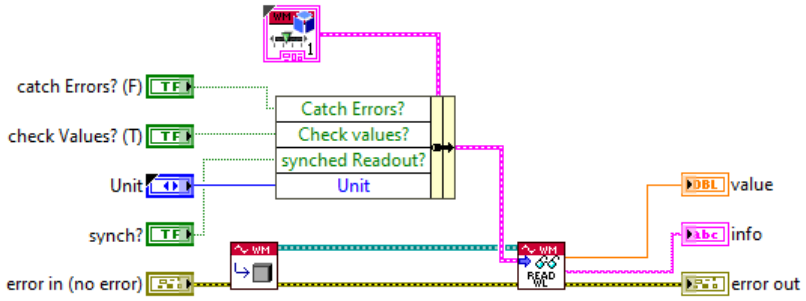


Figure 11: Readout code for Wavemeters (stored in `MCP WM std read wavelength.vi` of the `MCP Wavemeter.llb` Plug-in)

Unit	This control defines the unit of the measurement. Available units are wavelength (vacuum) in nm, wavelength (air) in nm, frequency in THz, wavenumber in cm^{-1} and Photon energy in eV
Catch WM errors?	This option allows to ignore all Wavemeter errors, if the boolean is set to TRUE. Be careful with disregarding errors.
Check WM Values?	Checks if the measurement result is within a reasonable range. If your Wavemeter sometimes tends to measure nonsense values, they should be filtered out with this option.
Synchronous?	This boolean flag indicates, how the wavelength/frequency is read from the Wavemeter. In some use cases, it is necessary to measure an up to date value from the Wavemeter to avoid problems during time critical measurements. Under other circumstances, one wants a quick response and a few milliseconds old measurement value is sufficient, e.g. during the Birefringent Calibration. Hence, if this VI is called with **Synchronous?** = TRUE, the calling VI expects the next measured value. If **Synchronous?** = FALSE, the wavelength/frequency doesn't need to be that fresh

	(e.g. some ten milliseconds old) and you may implement a more time conserving way of reading the wavelength/frequency. If and how you implement this behaviour is left to you. The *Matisse Commander* Wavemeter Plug-in in general uses the synchronous option to read wavelengths, that offers the most recent values.
Timestamp	The time of readout. Useful for trouble-shooting timing issues.
Value	Read out of the Wavemeter in the unit you set in **Unit**.
Info	Is used for trouble-shooting and will contain additional information like an error status or warnings.

WM Plug-in Interface Specification

Class Diagram

To change or add code of the Wavemeter System of *Matisse Commander* please find the Source code in the file `LabVIEW - WM Selector Wavemeter PlugIns.zip`, that is located in the `Addons` folder of your *Matisse Commander* installation or on your Sirah DVD. The *wmp_custom class* is prepared to make the integration of your own Wavemeter as easy as possible. This class diagram can give you a general overview of the Wavemeter system.

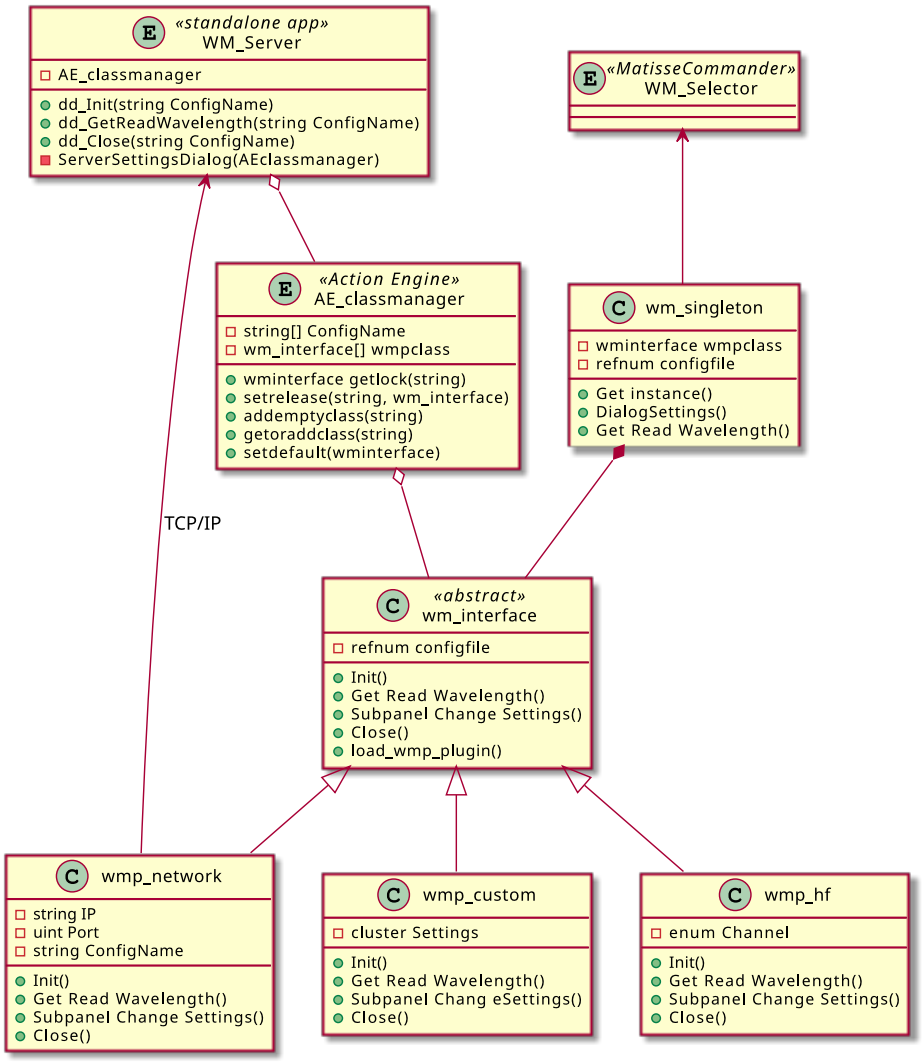


Figure 12: A simplified class diagram of the Wavemeter system of *Matisse Commander*

All Wavemeter Plug-ins must inherit from the wm_interface class, so they can be loaded dynamically by the wm_singleton class. This class is responsible for managing the Wavemeter support across the *Matisse Commander*.

Create your own Wavemeter Plug-in

If you use a Wavemeter, that is not supported by the *Matisse Commander*, you can build your own Wavemeter reading Plug-in. The template wmp_custom provides a good starting point for your code. Please use the LabVIEW 2020 Development Environment for compatibility to *Matisse Commander*. Access the template of the Custom Wavemeter Plug-in by extracting the file `LabVIEW - WM Selector Wavemeter Plug-ins.zip`, that is located in the `Addons` folder of your *Matisse Commander* installation or on your Sirah DVD. Open the LabVIEW project therein and locate the `Get Read Wavelength.vi` VI. The code of the VI is shown in *Figure 13*. You should replace the code in the red box and put the vacuum wavelength on the orange output wire. If the output unit does not match the requested unit it will be automatically converted. You can provide more units to skip this conversion step by adding the appropriate cases to the case structure.

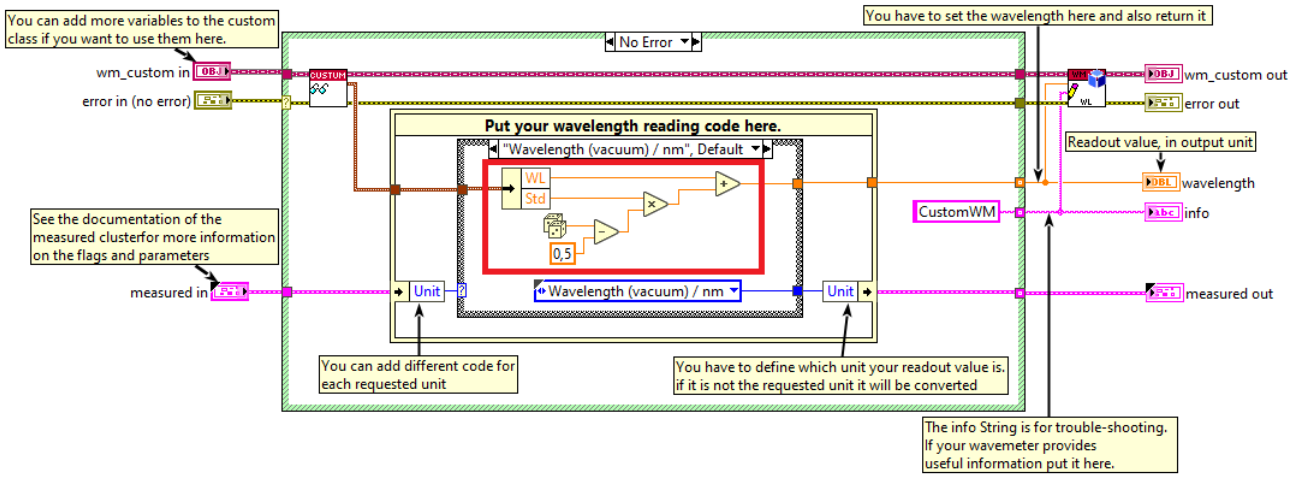


Figure 13: Sample Wavemeter reading LabVIEW code

The **wm_custom in** and **wm_custom out** objects are used by the class structure, so they have to be connected. You can use them to store additional variables you may need to operate the Wavemeter.

Error in and **Error out** are controls for the commonly used error cluster in LabVIEW. If an error occurs while reading the laser wavelength, you should output this error to **Error out**.

The **info** indicator is for trouble-shooting and should be filled with additional information like the error status or warnings.

The **measured in** cluster provides additional options for a better control of the Wavemeter measurements. These options are passed through to one of the LabVIEW VIs above mentioned, that should read the current laser wavelength/frequency. These parameters are the same as explained in *Use of a Wavemeter Plug-in*.

After implementing the changes, built your Plug-in as outlined in the section *Application Library Creation*. You can test the Plug-in by copying the newly created wmp_custom.lvlibp file to the `Matisse Commander/Plug-ins/Wavemeter` folder and activating it. If you have inserted the readout code successfully, your Wavemeter Plug-in will show up in the **WM Selector** > **Show Settings** window. You can open **WM Selector** > **Show Wavelength** to check the current readout.

Extended Scan and GoTo Procedure

The Extended Scan and GoTo procedure, that you can utilize in the *Matisse Commander* Plug-in Wavemeter, are also available for programming in your own *Matisse Commander* Plug-in or an independent LabVIEW project.

GoTo Position in a *Matisse Commander* Plug-in

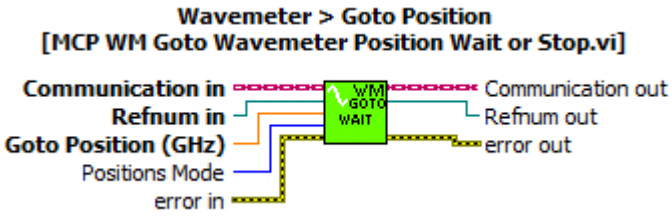


Figure 14: Wavemeter GoTo Position VI connector pane

To use **GoTo Position** procedure in your own *Matisse Commander* Plug-in, you need to copy the `MCP Wavemeter.llb` file from the *Matisse Commander*'s Plug-in folder. In this llb, there is a file called `MCP WM GoTo Wavemeter Position Wait or Stop.vi` providing the **GoTo Position**.

There are three controls, that must be connected to this VI. Within a *Matisse Commander* Plug-in the references for the **Communication in** resource and configuration file (**Refnum**) are provided by the *Matisse Commander*. Therefore you just need to set the destination frequency for the **GoTo Position** in GHz.

Optional controls are **Positions Mode** and **Create Report**. **Positions Mode** select, how the piezos of the *Matisse* are set after a GoTo. You can select upwards and downwards scan directions, so you are will be able to scan to higher or lower frequencies from the destination frequency, respectively. The mode **Center Position** sets the piezos in a way, you can scan upwards as good as downwards without reaching a mechanical limit of the piezos. **Current Position** uses the current piezo position for setting the new laser wavelength.

Create Report enables writing a report of the GoTo procedure to an HTML file. The path for the report file is set the *Matisse Commander* configuration file.

If you use this GoTo Position within a new *Matisse Commander* Plug-in, please make sure, that you have enabled a Wavemeter reading software such as the HighFinesse Plug-in.

Extended Scan

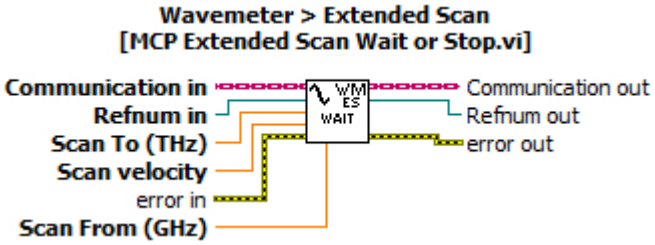


Figure 15: Extended Scan VI connector pane

Implementing the Extended Scan feature in your own *Matisse Commander* Plug-in works very similar to the GoTo procedure. To get access to the Extended Scan VI, you need to copy the `MCP Wavemeter.llb` file from the *Matisse Commander*'s Plug-in folder. In this llb, there is a file called `MCP Extended Scan Wait or Stop.vi`, which provides the Extended Scan.

There are five controls, that must be connected to this VI. Within a *Matisse Commander* Plug-in the references for the **Communication resource** and configuration file (**Refnum**) are provided by the *Matisse Commander*. Therefore you just need to set the destination frequency for the **GoTo Position** in GHz.

Use **Scan From (GHz)** and **Scan To (GHz)** to set the frequency limits in Gigahertz of the Extended Scan. Scan velocity selects the arbitrary unit for the scan speed, that is already used in the *Matisse Commander*.

An optional control is **Create Report**, that enables writing a report of the GoTo procedure to an HTML file. The path for the report file is set the *Matisse Commander* configuration file.

If you use the Extended Scan as a *Matisse Commander* Plug-in, please make sure, that you have enabled a Wavemeter reading software such as the HighFinesse Plug-in.

Extended Scan in a LabVIEW project

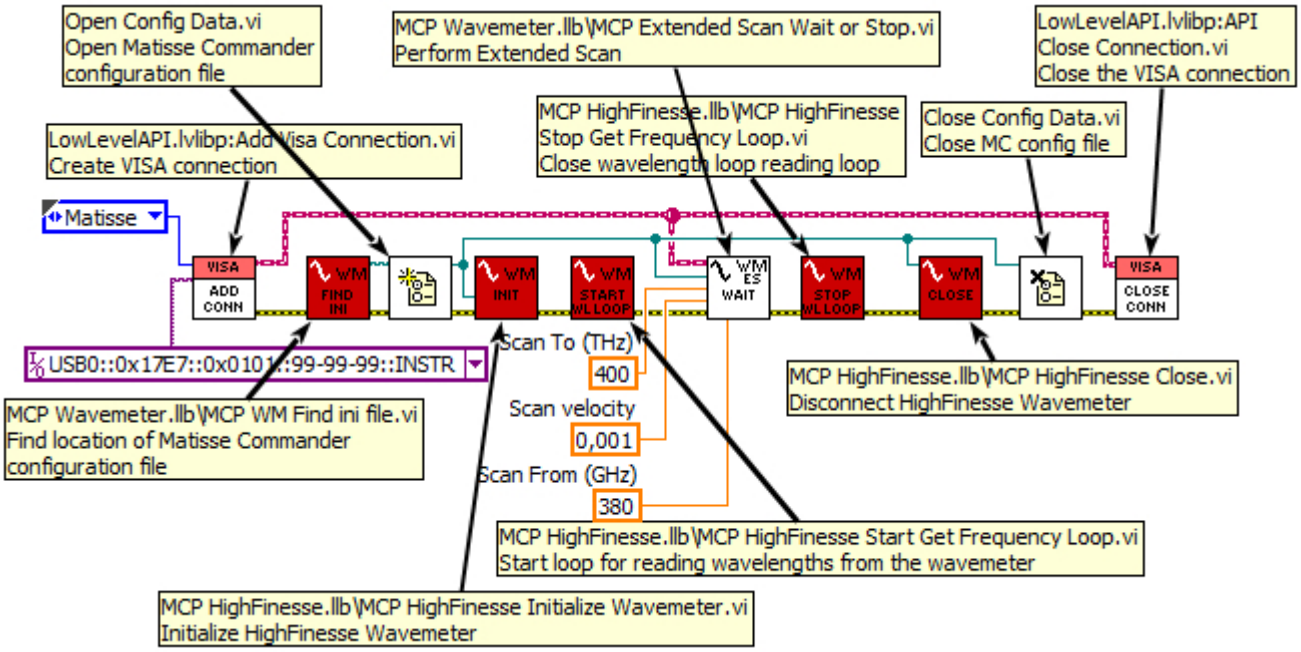


Figure 16: Extended Scan example

If you would like to use the Extended Scan in your own LabVIEW project, that is independent from the *Matisse Commander*, you have the advantage to utilize a different LabVIEW version than that, that was used to compile the *Matisse Commander*. You need to use at least LabVIEW 2020, so you can open the Extended Scan VI. You also need to make use of a Wavemeter reading Plug-in. In this example the HighFinesse Plug-in is used.

Therefore copy the files `MCP Wavemeter.llb` and `MCP HighFinesse.llb` from the *Matisse Commander*'s Plug-in directory and the *Matisse* API consisting of `LowLevelAPI.lvlibp` and `Matisse API.lvlibp` to a subdirectory of you project.

When starting your program, you need to initialize some references, that are required by the Extended Scan. In any case you must provide a communication resource, that is opened by `Add VISA Connection.vi`, which is part of LabVIEW. Moreover the GoTo Procedure needs to know the parameter set, that is stored in the *Matisse Commander* ini file. The `VI MCP WM Find ini file.vi` locates the ini file and passes its location to `Open Config Data.vi`, that is also part of LabVIEW. The VIs `MCP HighFinesse Initialize Wavemeter.vi` and `MCP HighFinesse Start Get Frequency Loop.vi` establishes a connection to the HighFinesse Wavemeter. From now on you can avail yourself of the Extended Scan, as long as you connect the references to the GoTo VI.

Before exiting the program, LabVIEW should close the references, it has opened before. Therefore please use `MCP HighFinesse Stop Get Frequency Loop.vi` and `MCP HighFinesse Close.vi`, that can be found in the `MCP HighFinesse.llb`. The `Close Config Data.vi` can be found in the LabVIEW function palettes while `API Close Connection.vi` is part of `LowLevelAPI.lvlibp`.

Chapter 6

General

This chapter explains some of the general concepts.

USB Interface

The *Matisse* uses a Universal Serial Bus (USB) interface for control purpose. The *Matisse* implements a USB port according to the *Universal Serial Bus Specifications Revision 2.0*.

The *Matisse* conforms to the USB Test and Measurement class (USBTMC) according to the specification Revision 1.0. It can be used with *National Instrument's* NI-VISA USB INSTR class. You can use the VISA Open, VISA Close, VISA Read, and VISA Write functions in the same way you would if you were communicating with a serial or IEEE 488 interfaced device.

Sirah's USB vendor ID is 17E7 hex (6119 dec).

Data Types

The following data types are used for the USB commands

<integer>

All integers are decimal numbers ranging from 0 .. 65535. Some commands operate on a limited subrange.

<float>

Float numbers need to be specified in decimal notation upon input (e.g. 0.001). All output float numbers are in exponential notation (0.1E-5). Please note that the DSP uses fixed-point representation for some values. The input value will be rounded to the nearest available fixed point number. Hence, values returned by the device may deviate from the value submitted.

<string>

All strings are enclosed in quotation marks (“”).

<parameter name>

This is a string with a maximum length of 22 characters. All strings are enclosed in quotation marks (“”).

<parameter value>

This is a string with a maximum length of 22 characters. All strings are enclosed in quotation marks (“”).

Matisse Device Commands

The following sections give a complete reference for all *Matisse* device commands.

PID Loops

A Proportional-Integral-Derivative controller or PID is a standard feedback loop component in control applications. It measures an *output* of a process and controls an *input*, with a goal of maintaining the output at a target value, which is called the *setpoint*. The basic idea is that the controller reads a sensor. Then it subtracts the measurement from a desired *setpoint* to determine an *error*.

The *error* is then treated in three different ways simultaneously:

Proportional

To handle the present, the *error* is multiplied by a proportional constant P, and sent to the output. P represents the band over which a controller's output is proportional to the error of the system.

Integral

To handle the past, the *error* is integrated (or averaged, or summed) over a period of time, and then multiplied by a constant I, and added to the proportional *output*. I represents the steady state error of the system. By using the Proportional term alone it is not possible to reach a steady *setpoint* temperature. Real world processes are not perfect and are subject to a number of environmental variables. As these variables are often constant they can be measured and compensated for. In practice, the Integral term of a controller only considers a relatively short history of the controller.

Derivative

To handle the future, the first derivative of the *error* (its rate of change) is calculated with respect to time, and multiplied by another constant D, and summed with the proportional and integral terms. The derivative term is used to govern a controller's response to a change in the system. The larger the derivative term the more rapidly the controller will respond to changes in the process value. This is a good thing when trying to dampen a controller's response to short term changes.

PID loops are susceptible to noise. Hence, the *input* is averaged to cancel out the influence of the noise. The Average controls the number of measurements averaged before a new PID iteration is calculated.

PID:PROTOCOL

Set the identifier number of the PID-loop to protocol.

PID ID	Usage
0	none
1	Thin Etalon
2	Thick Etalon
3	Slow Piezo
4	Fast Piezo

Command Syntax

```
PID:(PROT|PROTOCOL)?
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PID:PROT 1
Matisse> OK
```

PID:PROTOCOL?

Get the ID number of the PID loop which is currently written into the protocol.

PID ID	Usage
0	none
1	Thin Etalon
2	Thick Etalon
3	Slow Piezo
4	Fast Piezo

Command Syntax

```
PID:(PROT|PROTOCOL)?
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PID:PROT?
Matisse> :PID:PROT:0
```

PID:PROCESSSTATISTICS?

Evaluate some statistics for the process values stored in the PID protocol array. The values are calculated using the current contents of the 256 entry ring buffer. During the evaluation the recording of new PID protocol data is disabled.

Each request returns the following values: Minimal process value, maximal process value, average process value, root mean square deviation from the average value.

Command Syntax

```
PID:(PROCESSSTATISTICS|PSTAT)?
```

Reply Syntax

```
:PID:PSTAT: <float> <float> <float> <float>
```

Example

```
Example
```

Compatibility

```
Version 1.10
```

PID:ORDINAL?

Get the current value of the counter for the ordinal number of the protocol entries.

Command Syntax

```
PID:(ORDINAL|ORD)?
```

Reply Syntax

```
:PID:ORD: <integer>
```

Example

```
Example
```

Compatibility

```
Version 1.3
```

Error Handling

In case of an error the device will reply an error prompt. The error prompt consists of a text constant `!ERROR` an error code and a short description. The error code and the string are separated by a comma. The string is limited by quotation marks.

Please note the following. The error condition is cleared immediately after the error is prompted. Errors can occur independently from the issued commands, depending on the state of the laser, signals exceeding limits, and communication timeouts. Whenever an error condition occurs all errors that are raised later in time are ignored.

```
Cmd> x
Device> !ERROR 1,"general syntax error"
Cmd> *IDN?
Device> :IDN: "Matisse TR, S/N:15-12-99, DSP Rev.:00.00, Firmware:1.16c,
 Date:Dec 18 2015"
```

ERROR:CLEAR

Clears error conditions and information. This command does not affect error conditions from the motor controller.

Command Syntax

```
(ERROR|ERR):(CLEAR|CL)
```

Reply Syntax

```
OK
```

Example

```
Cmd> ERR:CL
Device> OK
```

Matisse Error Codes

Error Code	Description
1	general syntax error
4	factory command requires password
10	value out of limits
12	TWI bus busy
13	TWI bus transmit timeout
14	TWI bus no acknowledge received
15	TWI bus receive timeout
16	variable is read-only
17	variable is write-only
18	waveform is clipping
19	update sequence error
20	bad scan device id
21	can not write parameter set
22	can not convert wavelength / birefringent filter motor position
23	command input exceeds input buffer
24	piezo is locked by other command
30	parameter set name is already defined
31	parameter set name not found
34	no more free memory to execute command
35	parameter memory inconsistent
40	fast piezo not configured
50	communication error with Pound-Drever-Hall unit
51	Pound-Drever-Hall unit not configured
52	Pound-Drever-Hall amplifier not configured
53	Birefringent-Filter motor not configured
54	Thin etalon motor not configured
55	Slow piezo unit is not configured
56	Reference cell unit not configured
57	Thick etalon piezo unit is not configured
58	Device connected with variable not configured
59	Invalid scan limits (lower >= upper)
60	Variable gain reference cell not configured
61	Variable gain for reference needs to be non zero
203	internal error (read variable)
204	internal error (write variable)
205	internal error (write parameter)
206	internal error (read parameter)
207	device output buffer overflow; the USB message is too long for the device's buffer.
208	transfer buffer too small; the USB message will not fit into the host computer's transfer buffer.

Power Diode

These commands are used to request the status of the Integral Power Diode. This diode monitors a reflex that is proportional to the integral output power of the laser system.

DIODEPOWER:DCVALUE?

Get the DC-part of the integral laser output. The value is given in volts at the controller input. This is a read-only value.

Command Syntax

```
(DIODEPOWER|DPOW):(DCVALUE?|DC?)
```

Reply Syntax

```
:DPOW:DC: <float>
```

Example

```
Cmd> DPOW:DC?
Matisse> :DPOW:DC: 3.125000e-01
```

DIODEPOWER:WAVETABLE?

Get the current waveform of the AC-part of the integral laser output. The values are normalized to be in the range [-1,1]. The number of values is determined by the setting of `PIEZOETALON:OVERSAMPLING`.

Command Syntax

```
(DIODEPOWER|DPOW):(WAVETABLE?|WAVTAB?)
```

Reply Syntax

```
:DPOW:WAVTAB: {<float>}
```

Example

```
Cmd> DPOW:WAVTAB?
Matisse> :DPOW:WAVTAB: -4.000000e-01 4.000000e-02 2.000000e-01 -8.000001e-02
2.400000e-01 1.600000e-01 -1.000000e+00 -2.400000e-01 3.030303e-02
-3.030303e-01 3.030303e-02 4.848485e-01 1.212121e-01 -1.121212e+00
6.060606e-02 4.242424e-01
```

DIODEPOWER:LOW?

Get the current value of the low power level. When the signal at the integral power diode drops below this level all control loops are deactivated. Setting the level to 0 (zero) disables this function.

Command Syntax

```
(DPOW|DIODEPOWER):LOW?
```

Reply Syntax

```
:DPOW:LOW: <float>
```

Example

```
Cmd> DPOW:LOW?
Matisse> :DPOW:LOW: 3.400000e-01
```

DIODEPOWER:LOW

Set the low power level. When the signal at the Integral Power Diode drops below this level all control loops are deactivated. Setting the level to 0 (zero) disables this function.

Command Syntax

```
(DPOW|DIODEPOWER):LOW <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> DPOW:LOW 0.34
Matisse> OK
```

Birefringent Filter Motor

These commands are used to control the setting of the Birefringent Filter.

Motor commands need noticeable time to execute (10 ms .. 100 ms) due to communication overhead.

MOTORBIREFRINGENT:POSITION?

Get the current position of the Birefringent Filter's stepper motor position.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(POSITION|POS)?
```

Reply Syntax

```
:MOTBI:POS: <integer>
```

Example

```
Cmd> MOTBI:POS?
Matisse> :MOTBI:POS: 229960
```

MOTORBIREFRINGENT:POSITION

Move the stepper motor of the Birefringent Filter to an absolute position. The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(POSITION|POS)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:POS 120400
Matisse> OK
```

MOTORBIREFRINGENT:STATUS?

Retrieve the status and setting of the Birefringent Filter's motor controller. The status is binary coded into a single 16-bit integer value. The bits have the following meaning:

Bit	Usage
0 .. 6	current status of the controller
7	set in case of an error status of the controller
8	indicates that the motor is running

Bit	Usage
9	indicates the motor current is switched off
10	indicates an invalid motor position after hold was switched off
11	status of limit switch 1
12	status of limit switch 2
13	status of home switch
14	manual control enable / disable

The following status codes are defined:

Code	Usage
0x00	undefined status
0x01	system is initialized
0x02	system is waiting for command / button pressed
0x03	button is pressed
0x04	executing short move (manual control)
0x05	executing long move (manual control)
0x06	decelerating motor
0x07	finishing move
0x08	execute absolute move command
0x09	execute relative move command
0x0a .. 0x0f	executing origin search sequence
0x10	calculating acceleration table

Error codes are:

Code	Usage
0x00	no error
0x01	undefined command code
0x02	frequency out of range
0x03	ramp length invalid
0x04	limit switch triggered
0x05	command disabled when hold current is off
0x06	command disabled after hold current was off
0x07	position out of range
0x08	motor is running (command ignored)
0x20	internal error

Code	Usage
0x21	watchdog triggered
0x22	error writing to the on-chip flash memory
0x23	checksum error detected

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(STATUS|STA)?
```

Reply Syntax

```
:MOTBI:STA: <integer>
```

Example

```
Cmd> MOTBI:STA?
Matisse> :MOTBI:STA: 2
```

MOTORBIREFRINGENT:MAXIMUM?

Get the maximum position of the Birefringent Filter's stepper motor.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(MAXIMUM|MAX)?
```

Reply Syntax

```
:MOTBI:MAX: <integer>
```

Example

```
Cmd> MOTBI:MAX?
Matisse> :MOTBI:MAX: 464841
```

MOTORBIREFRINGENT:RELATIVE

Move the Birefringent Filter's stepper motor the given number of steps relative to its current position. The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(RELATIVE|REL) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:REL -150
Matisse> OK
```

MOTORBIREFRINGENT:HOME

Move the Birefringent Filter's stepper motor to its home position. The home position is defined by the home switch. The controller positions the stepper motor at the point where the home switch is actuated and resets the motor position to zero (0). The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):HOME
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:HOME
Matisse> OK
```

MOTORBIREFRINGENT:HALT

Stop a motion of the Birefringent Filter's stepper motor. The command will use a smooth deceleration to maintain accurate step tracking.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):HALT
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:HALT
Matisse> OK
```

Compatibility

Matisse, Version 1.10

MOTORBIREFRINGENT:CLEAR

Clear pending errors at the Birefringent Filter motor controller.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(CLEAR|CL)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:CLEAR
Matisse> OK
```

MOTORBIREFRINGENT:INCREMENT

Set the number of motor steps made by the Birefringent Filter when the manual control button is pressed for a short time.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(INCREMENT|INC) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:INC 25
Matisse> OK
```

MOTORBIREFRINGENT:INCREMENT?

Retrieve the number of steps the Birefringent Filter motor makes when the manual control button is pressed for a short time.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(INCREMENT|INC)?
```

Reply Syntax

```
:MOTBI:INC: <integer>
```

Example

```
Cmd> MOTBI:INC?
Matisse> :MOTBI:INC: 50
```

MOTORBIREFRINGENT:CONSTANTABSOLUTE

Move the stepper motor of the Birefringent Filter to an absolute position using the constant speed defined by *MOTBI:FREQ*. The command does not wait for completion of the motor movement. This command requires stepper motor driver firmware version R25, or higher

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(CONSTANTABSOLUTE|CABS)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:CABS 120400
Matisse> OK
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:CONSTANTRELATIVE

Move the stepper motor of the Birefringent Filter relative to its current position using the constant speed defined by *MOTBI:FREQ*. The command does not wait for completion of the motor movement. This commands requires stepper motor driver firmware version R25, or higher

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(CONSTANTRELATIVE|CREL)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:CREL -1000
Matisse> OK
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:FREQUENCY

Set the step frequency used for the Birefringent Filter motor when using constant speed scan commands (*MOTBI:CABS*, *MOTBI:CREL*). The lowest step frequency supported by the firmware of the stepper motor driver is about 60 Steps / sec. If a lower frequency is requested the motor will use the lowest possible frequency without extra warning or notice.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(FREQUENCY|FREQ) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTBI:FREQ 200
Matisse> OK
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:FREQUENCY?

Get the step frequency used for the Birefringent Filter motor when using constant speed scan commands (*MOTBI:CABS*, *MOTBI:CREL*). The lowest step frequency supported by the firmware of the stepper motor driver is about 60 Steps / sec. If a lower frequency is requested the motor will use the lowest possible frequency without extra warning or notice.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(FREQUENCY|FREQ)?
```

Reply Syntax

```
:MOTBI:FREQ: <integer>
```

Example

```
Cmd> MOTBI:FREQ?
Matisse> :MOTBI:FREQ: 150
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:WAVELENGTH

Move the Birefringent Filter to a wavelength position. The position is passed as nanometers. The resulting motor position needs to be in between 0 and the maximum motor position, as given by the `MOTORBIREFRINGENT:MAXIMUM` command.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(WAVELENGTH|WL) <float>
```

Reply Syntax

```
OK|!ERROR
```

Example

```
Cmd> MOTBI:POS 760.1
Matisse> OK
```

MOTORBIREFRINGENT:WAVELENGTH?

Get the current position of the Birefringent Filter in terms of a wavelength. The result is given in nanometers.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(WAVELENGTH|WL)?
```

Reply Syntax

```
:MOTBI:WL: <float>
```

Example

```
Cmd> MOTBI:WL?
Matisse> :MOTBI:WL: 7.60E2
```

MOTORBIREFRINGENT:CAVITYSCAN

Set the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Birefringent Filter motor.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(CAVITYSCAN|CAVSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

MOTORBIREFRINGENT:CAVITYSCAN?

Get the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Birefringent Filter motor.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(CAVITYSCAN|CAVSCN)?
```

Reply Syntax

```
:MOTBI:CAVSCN: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

MOTORBIREFRINGENT:REFERENCESCAN

Set the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Birefringent Filter motor. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(REFERENCESCAN|REFSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> MOTBI:REFSCN -56.288
Device> OK
SFC> MOTBI:REFSCN?
Device> :MOTBI:REFSCN: -5.628799e+01
SFC> REFCELL:GNC 2
Device> OK
SFC> MOTBI:REFSCN?
Device> :MOTBI:REFSCN: -1.407200e+02
```

Compatibility

```
Version 1.3
```

MOTORBIREFRINGENT:REFERENCESCAN?

Get the proportional factor that controls how a scan of the Reference Cell influences the position of the Birefringent Filter motor. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI):(REFERENCESCAN|REFSCN)?
```

Reply Syntax

```
:MOTBI:REFSCN: <float>
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> MOTBI:REFSCN -56.288
Device> OK
SFC> MOTBI:REFSCN?
Device> :MOTBI:REFSCN: -5.628799e+01
SFC> REFCELL:GNC 2
Device> OK
SFC> MOTBI:REFSCN?
Device> :MOTBI:REFSCN: -1.407200e+02
```

Compatibility

```
Version 1.3
```

Wavelength Birefringent Filter

For the following sections it is important to understand how the BiFi wavelength is calculated from the BiFi Motor position. This is described by the following formula:

$$\lambda_{BiFi} = \frac{(8.5 + 1.5 \cdot 0.65) \cdot 10^{-3} \cdot f}{f \cdot 0.65 \cdot 10^{-6} + 1}$$

Depending on this, the factor f is required, which is determined by BiFi Thickness, Bifi Order, Bifi Position, Bifi Motorfactor and Bifi Motoroffset.

$$f = \frac{2 \cdot n_0}{\pi} \cdot BiFi_{Thickness} \cdot \sin^2(\gamma) \cdot \left(\frac{1}{\cos(\beta)} \cdot \frac{1}{BiFi_{Order}} \right) \cdot 10^6$$

$$\gamma = \arccos(\cos(\rho) \cdot \sin(\beta))$$

$$\rho = BiFi_{position} \cdot BiFi_{Factor} + BiFi_{Offset}$$

$$\beta = \arcsin\left(\frac{\sin(\arctan(n_0))}{n_0} \right)$$

$$n_0 = 1.544 \ (refractive\ index)$$

So, to understand how to set the different variables, you should refer to these formulas.

In Versions older than 1.13, the BiFi motor moved a lever. To support those old BiFi constructions set the $BiFi_{Thickness}$ to zero. The functional relation between Wavelength and Motor Position was determined by the following formula:

$$\lambda = \lambda_{Off} + \rho \sin^2[\arctan(L_{Lever}(POS + L_{Off}))]$$

where:

Variable	Command	Description
λ	Wavelength	emission wavelength of laser
POS	Position	step motor position
λ_{Off}	WavelengthOffset	fit parameter
p	WavelengthFactor	fit parameter
L_{Lever}	LeverLength	fit parameter
L_{Off}	LinearOffset	fit parameter

MOTORBIREFRINGENT:MOTOROFFSET

Set the calibration parameter $BiFi_{Offset}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(MOTOROFFSET:MOTOFF) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:MOTOROFFSET?

Get the calibration parameter $BiFi_{Offset}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(MOTOROFFSET|MOTOFF)?
```

Reply Syntax

```
:MOTBI:MOTOFF: <float>
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:MOTORFACTOR

Set the calibration parameter $BiFi_{Factor}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(MOTORFACTOR|MOTFAC) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:MOTORFACTOR?

Get the calibration parameter $BiFi_{Factor}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(MOTORFACTOR|MOTFAC)?
```

Reply Syntax

```
:MOTBI:FACTOR: <float>
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:THICKNESS

Set the calibration parameter $BiFi_{Thickness}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(THICKNESS|THNS) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:THICKNESS?

Get the calibration parameter $BiFi_{Thickness}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(THICKNESS|THNS)?
```

Reply Syntax

```
:MOTBI:THNS: <float>
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:ORDER

Set the calibration parameter $BiFi_{Order}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(ORDER) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

Version 1.13

MOTORBIREFRINGENT:ORDER?

Get the calibration parameter $BiFi_{Order}$ for the step motor position to wavelength conversion.

Command Syntax

```
(MOTORBIREFRINGENT|MOTBI)|(ORDER)?
```

Reply Syntax

```
:MOTBI:ORDER: <float>
```

Example

```
N/A
```

Compatibility

Version 1.13

Thin Etalon Motor

These commands are used to control the setting of the Thin Etalon.

Motor commands need noticeable time to execute (10 ms .. 100 ms) due to communication overhead.

MOTORTHINETALON:POSITION

Move the stepper motor of the Thin Etalon to an absolute position. The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(POSITION|POS)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:POS 12000
Matisse> OK
```

MOTORTHINETALON:POSITION?

Get the current absolute position of the stepper motor.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(POSITION|POS)?
```

Reply Syntax

```
:MOTTE:POS: <integer>
```

Example

```
Cmd> MOTTE:POS?
Matisse> :MOTTE:POS: 12000
```

MOTORTHINETALON:STATUS?

Get the status of the Thin Etalon's stepper motor controller. The status is binary coded into a single 16-bit value. The bits have the following meaning:

Bit	Usage
0 .. 6	current status of the controller

Bit	Usage
7	set in case of an error status of the controller
8	indicates that the motor is running
9	indicates the motor current is switched off
10	indicates an invalid motor position after hold was switched off
11	status of limit switch 1
12	status of limit switch 2
13	status of home switch
14	manual control enable / disable

The following status codes are defined:

Code	Usage
0x00	undefined status
0x01	system is initialized
0x02	system is waiting for command / button pressed
0x03	button is pressed
0x04	executing short move (manual control)
0x05	executing long move (manual control)
0x06	decelerating motor
0x07	finishing move
0x08	execute absolute move command
0x09	execute relative move command
0x0a .. 0x0f	executing origin search sequence
0x10	calculating acceleration table

Error codes are:

Code	Usage
0x00	no error
0x01	undefined command code
0x02	frequency out of range
0x03	ramp length invalid
0x04	limit switch triggered
0x05	command disabled when hold current is off
0x06	command disabled after hold current was off
0x07	position out of range

Code	Usage
0x08	motor is running (command ignored)
0x20	internal error
0x21	watchdog triggered
0x22	error writing to the on-chip flash memory
0x23	checksum error detected

Command Syntax

```
(MOTORTHINEATLON|MOTTE):(STATUS|STA)?
```

Reply Syntax

```
:MOTTE:STA: <integer>
```

Example

```
Cmd> MOTTE:STA?
Matisse> :MOTTE:STA: 2
```

MOTORTHINETALON:MAXIMUM?

Get the maximum position of the Thin Etalon's stepper motor.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(MAXIMUM|MAX)?
```

Reply Syntax

```
:MOTTE:MAX: <integer>
```

Example

```
Cmd> MOTTE:MAX?
Matisse> :MOTTE:MAX: 40000
```

MOTORTHINETALON:RELATIVE

Move the Thin Etalon's stepper motor the given number of steps relative to its current position. The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(RELATIVE:REL) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

MOTORTHINETALON:FREQUENCY

Set the step frequency used for the Thin Etalon motor when using constant speed scan commands (*MOTTE:CABS*, *MOTTE:CREL*). The lowest step frequency supported by the firmware of the stepper motor

driver is about 60 Steps / sec. If a lower frequency is requested the motor will use the lowest possible frequency without extra warning or notice.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(FREQUENCY|FREQ) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:FREQ 200
Matisse> OK
```

Compatibility

Version 1.18

MOTORTHINETALON:FREQUENCY?

Get the step frequency used for the Thin Etalon motor when using constant speed scan commands (*MOTTE:CABS*, *MOTTE:CREL*). The lowest step frequency supported by the firmware of the stepper motor driver is about 60 Steps / sec. If a lower frequency is requested the motor will use the lowest possible frequency without extra warning or notice.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(FREQUENCY|FREQ)?
```

Reply Syntax

```
:MOTTE:FREQ: <integer>
```

Example

```
Cmd> MOTTE:FREQ?
Matisse> :MOTTE:FREQ: 150
```

Compatibility

Version 1.18

MOTORTHINETALON:CONSTANTABSOLUTE

Move the stepper motor of the Thin Etalon to an absolute position using the constant speed defined by *MOTTE:FREQ*. The command does not wait for completion of the motor movement. This commands requires stepper motor driver firmware version R25, or higher

Command Syntax

```
(MOTORTHINETALON|MOTTE):(CONSTANTABSOLUTE|CABS)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:CABS 120400
Matisse> OK
```

Compatibility

Version 1.18

MOTORTHINETALON:CONSTANTRELATIVE

Move the stepper motor of the Thin Etalon relative to its current position using the constant speed defined by *MOTTE:FREQ*. The command does not wait for completion of the motor movement. This command requires stepper motor driver firmware version R25, or higher

Command Syntax

```
(MOTORTHINETALON|MOTTE):(CONSTANTRELATIVE|CREL)<integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:CREL -1000
Matisse> OK
```

Compatibility

Version 1.18

MOTORTHINETALON:HOME

Move the Thin Etalon's stepper motor to its home position. The home position is defined by the home switch. The controller positions the stepper motor at the point where the home switch is actuated and resets the motor position to zero (0). The command does not wait for completion of the motor movement.

Command Syntax

```
(MOTORTHINETALON|MOTTE):HOME
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:HOME
Matisse> OK
```

MOTORTHINETALON:HALT

Stop a motion of the Thin Etalon's stepper motor. The command will use a smooth deceleration to maintain accurate step tracking.

Command Syntax

```
(MOTORTHINETALON|MOTTE):HALT
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:HALT
Matisse> OK
```

Compatibility
Matisse, Version 1.10

MOTORTHINETALON:CLEAR
Clear pending errors at the motor controller.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(CLEAR|CL)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> MOTTE:CL
Matisse> OK
```

MOTORTHINETALON:INCREMENT
Set the number of steps the Thin Etalon's stepper motor makes when the manual control button is pressed for a short period of time.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(INCREMENT:INC) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

MOTORTHINETALON:INCREMENT?
Get the number of steps the Thin Etalon's stepper motor makes when the manual control button is pressed for a short period of time.

Command Syntax

```
(MOTORTHINETALON|MOTTE):(INCREMENT|INC)?
```

Reply Syntax

```
:MOTTE:INC: <integer>
```

Example

```
N/A
```

TWI Bus

The DSP unit communicates with a number of sub-units, such as the stepper motor controller, via a two-wire-interface (TWI). The TWI is also known as I^2C bus.

TWOWIREINTERFACE:SEND
Send a command to a device attached to the DSP's TWI bus. Only bytewide communication is supported. `<adr>` is the device's TWI address (7bit address mode required), `<length>` the number of bytes to be sent (in the range 1..16), `<data>` the respective data bytes.

Command Syntax

```
(TWI|TWOWIREINTERFACE):SEND <adr> <length> {<data>}
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> TWI:SEND 196 1 1
Matisse> OK
```

TWOWIREINTERFACE:READ

Read information from a device attached to the DSP's TWI bus. <adr> is the device's address (7bit mode), <length> the number of bytes to read (in the range 1..16).

Command Syntax

```
(TWOWIREINTERFACE|TWI):READ <adr> <length>
```

Reply Syntax

```
:TWI:READ: {<integer>}
```

Example

```
Cmd> TWI:READ 196 2
Matisse> :TWI:READ: 1 2
```

Piezo Etalon Control

The Piezo Etalon ensures that all except one longitudinal mode have losses that prevent laser activity from these modes. Therefore, the spacing of the etalon must be matched to an exact multiple of the favoured longitudinal mode's wavelength. A stack of piezoelectric ceramics is used to control the etalon spacing. Because of the high precision required and in order to be able to perform a scan, the spacing is actively controlled. The control scheme used to achieve this is a phase lock loop (PLL).

The PLL evaluates the response of the laser to an intentionally introduced perturbation. The perturbation is a slight variation of the etalon spacing. The variation follows the amplitude of a sine wave, further on called "Dither Frequency" f_D. The response of the laser is the variation in the total laser power, measured at the power diode. f_D is synthesized in a digital-to-analog converter operating with an adjustable sample output rate ranging from 8 kHz to 96 kHz. The number of samples used to synthesize the sine wave is controlled by the product of the sample output rate and the parameter OVERSAMPLING.

f_D = *sample rate / #samples*

Choose f_D in a way that you have at least 8 samples per period, more samples improve the result of the calculations, more than 64 samples make no difference in the accuracy of the process. Avoid f_D values that coincide with the line frequency (50/60 Hz) if possible. The PLL loop measures the phase-relation between the reference signal and the laser's intensity. The phase between the two is used as an error signal for a PID loop that controls the voltage of the piezo.

PIEZOETALON:OVERSAMPLING

Set the number of sample points for sine interpolation. The minimum value is 8, the maximum value is 64 samples per period.

Command Syntax

```
(PIEZOETALON|PZETL):(OVERSAMPLING|OVER) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

PIEZOETALON:OVERSAMPLING?

Get the number of sample points used for sine interpolation.

Command Syntax

```
(PIEZOETALON|PZETL):(OVERSAMPLING|OVER)?
```

Reply Syntax

```
:PZETL:OVER: <integer>
```

Example

```
N/A
```

PIEZOETALON:BASELINE

Set the baseline of the modulation waveform to a new value. The value needs to be within the interval [-0.95,0.95].

Command Syntax

```
(PIEZOETALON|PZETL):(BASELINE|:BASE) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PZETL:BASE 0.3
Matisse> OK
```

PIEZOETALON:BASELINE?

Get the baseline of the modulation waveform.

Command Syntax

```
(PIEZOETALON|PZETL)|(BASELINE|BASE)?
```

Reply Syntax

```
:PZETL:BASE: <float>
```

Example

```
N/A
```

PIEZOETALON:AMPLITUDE

Set the amplitude of the modulation of the Piezo Etalon.

Command Syntax

```
(PIEZOETALON|PZETL):(AMPLITUDE|AMP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

PIEZOETALON:AMPLITUDE?

Get the amplitude of the modulation of the Piezo Etalon.

Command Syntax

```
(PIEZOETALON|PZETL):(AMPLITUDE|AMP)?
```

Reply Syntax

```
:PZETL:AMP: <float>
```

Example

```
N/A
```

PIEZOETALON:CONTROLSTATUS

Start or stop the **P** (see *PID Loops*) control loop that controls the baseline value of the modulation. If the control is stopped the amplitude of the modulation is set to zero.

Command Syntax

```
(PIEZOETLAON|PZETL):(CONTROLSTATUS|CNTRSTA)  (RUN|RU|STOP|ST)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

PIEZOETALON:CONTROLSTATUS?

Get the status of the **P** (see *PID Loops*) control loop that controls the baseline value of the modulation.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLSTATUS|CNTRSTA)?
```

Reply Syntax

```
:PZETL:CNTRSTA:  (RUN|STOP)
```

Example

```
N/A
```

PIEZOETALON:CONTROLPROPORTIONAL

Set the proportional gain of the P (see PID Loops) control loop that controls the baseline value of the modulation.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLPROPORTIONAL|CNTRPROP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

PIEZOETALON:CONTROLPROPORTIONAL?

Get the proportional gain of the **P** (see *PID Loops*) control loop that controls the baseline value of the modulation.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLPROPORTIONAL|CNTRPROP)?
```

Reply Syntax

```
:PZETL:CNTRPROP: <float>
```

Example

```
N/A
```

PIEZOETALON:CONTROLAVERAGE

Set the number of waveforms averaged before a P (see *PID Loops*) control loop iteration is performed.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLAVERAGE|CNTRAVG) <integer>
```

Reply Syntax

```
:PZETL:CNTRAVG: <integer>
```

PIEZOETALON:CONTROLAVERAGE?

Get the number of waveforms averaged before a P (see *PID Loops*) control loop iteration is performed.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLAVERAGE|CNTRAVG)?
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

PIEZOETALON:CONTROLPHASESHIFT

Set the phaseshift that is used for the PLL calculation.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLPHASESHIFT|CNTRPHSF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

PIEZOETALON:CONTROLPHASESHIFT?

Get the phaseshift value used for the PLL calculation.

Command Syntax

```
(PIEZOETALON|PZETL):(CONTROLPHASESHIFT|CNTRPHSF)?
```

Reply Syntax

```
:PZTEL:CNTRPHSF: <integer>
```

Example

```
Cmd> PZETL:CNTRPHSF?
Matisse> :PZETL:CNTRPHSF: 0
```

PIEZOETALON:CAVITYSCAN

Set the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Piezo Etalon. The factor results in an immediate piezo movement, even without the **P** (see *PID Loops*) control loop enabled.

Command Syntax

```
(PIEZOETALON|PZTEL):(CAVITYSCAN|CAVSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

PIEZOETALON:CAVITYSCAN?

Get the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Piezo Etalon. The factor results in an immediate stepper motor movement, even without the **P** (see *PID Loops*) control loop enabled.

Command Syntax

```
(PIEZOETALON|PZETL):(CAVITYSCAN|CAVSCN)?
```

Reply Syntax

```
:PZETL:REFSCN: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

PIEZOETALON:REFERENCESCAN

Set the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Piezo Etalon. The factor results in an immediate piezo movement, even without the *P control loop* enabled. On the other hand, a value of 0 for this parameter corresponds to a control loop only operation of the Piezo Etalon. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(PIEZOETALON|PZETL):(REFERENCESCAN|REFSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> PZETL:REFSCN -1.700378
Device> OK
SFC> PZETL:REFSCN?
Device> :PZETL:REFSCN: -1.700363e+00
SFC> REFCELL:GNC 2
Device> OK
SFC> PZETL:REFSCN?
Device> :PZETL:REFSCN: -4.250931e+00
```

Compatibility

```
Version 1.3
```

PIEZOETALON:REFERENCESCAN?

Get the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Thin Etalon. The factor results in an immediate stepper motor movement, even without the **P** (see *PID Loops*) control loop enabled.

Command Syntax

```
(PIEZOETALON|PZETL):(REFERENCESCAN|REFSCN)?
```

Reply Syntax

```
:TE:REFSCN: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

PIEZOETALON:SAMPLERATE

Set the sample rate for the Piezo Etalon control loop. The product of samplerate and oversampling determines the modulation frequency of the etalon. The samplerate parameter uses the following codes:

Code	Sample Rate
0	8 kHz
1	32 kHz
2	48 kHz
3	96 kHz

Command Syntax

```
(PIEZOETALON|PZETL):(SAMPLRATE|SRATE) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.7
```

PIEZOETALON:SAMPLERATE

Get the sample rate for the Piezo Etalon control loop. The product of samplerate and oversampling determines the modulation frequency of the etalon. The samplerate parameter uses the following codes:

Code	Sample Rate
0	8 kHz
1	32 kHz
2	48 kHz
3	96 kHz

Command Syntax

```
(PIEZOETALON|PZETL):(SAMPLRATE|SRATE)?
```

Reply Syntax

```
:PZETL:SRATE: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.7
```

FEEDFORWARD:AMPLITUDE

Set the amplitude for the feed forward of the Piezo Etalon's modulation to the fast stabilization piezo. Negative values for the amplitude will result in a feed forward signal with twice the frequency of the piezo modulation. This behaviour is useful for setting the feedforward parameters. The parameter is only available when the Piezo Etalon is installed and configured.

Command Syntax

```
(FEEDFORWARD|FEF):(AMPLITUDE|AMP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> FEF:AMP 1.5
Matisse> OK
```

Compatibility

```
Version 1.8
```

FEEDFORWARD:AMPLITUDE?

Get the amplitude for the feed forward of the Piezo Etalon's modulation to the fast stabilization piezo. Negative values for the amplitude result in a feed forward signal with twice the frequency of the piezo modulation. This behaviour is useful for setting the feedforward parameters. The parameter is only available when the Piezo Etalon is installed and configured.

Command Syntax

```
(FEEDFORWARD|FEF):(AMPLITUDE|AMP)?
```

Reply Syntax

```
:FEF:AMP: <float>
```

Example

```
Cmd> FEF:AMP?
Matisse> :FEF:AMP: 1.500000e+00
```

Compatibility

```
Version 1.8
```

FEEDFORWARD:PHASESHIFT

Set the phase shift for the feed forward of the Piezo Etalon's modulation to the fast stabilization piezo. Useful values for this parameter range from 0 to the `OVERSAMPLING` of the Piezo Etalon's modulation signal. The parameter is only available when the Piezo Etalon is installed and configured.

Command Syntax

```
(FEEDFORWARD|FEF):(PHASESHIFT|PHSF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> FEF:PHSF 14
Matisse> OK
```

Compatibility

```
Version 1.8
```

FEEDFORWARD:PHASESHIFT?

Get the phase shift for the feed forward of the Piezo Etalon's modulation to the fast stabilization piezo. Useful values for this parameter range from 0 to the `OVERSAMPLING` of the Piezo Etalon's modulation signal. The parameter is only available when the Piezo Etalon is installed and configured.

Command Syntax

```
(FEEDFORWARD|FEF):(PHASESHIFT|PHSF)?
```

Reply Syntax

```
:FEF:PHSF: <integer>
```

Example

```
Cmd> FEF:PHSF?
```

```
Matisse> :FEF:PHSF: 14
```

Compatibility

```
Version 1.8
```

Thin Etalon Control

The position of the Thin Etalon needs to be controlled for two reasons:

- Avoid mode hopping when the laser is scanned.
- Maintain a high output energy over a long period of time.

To ensure a proper setting of the Thin Etalon a *PID Loop* is used. The input for the loop is the intensity of the reflex of the Thin Etalon. A high intensity of the reflex corresponds to high losses from the Thin Etalon. So, a low intensity corresponding to few losses for the Thin Etalon is desirable. The input for the *PID Loop* is the intensity of the Etalon reflex (THINETALON:DCVALUE) divided by the integral output (DIODEPOWER:DCVALUE) of the laser.

THINETALON:DCVALUE?

Get the DC-part of the reflex of the Thin Etalon. The value is given in volts at the controller input.

Command Syntax

```
(THINETALON|TE):(DCVALUE|DC)?
```

Reply Syntax

```
:TE:DC: <float>
```

Example

```
Cmd> TE:DC?
Matisse> :TE:DC: 1.34E-1
```

THINETALON:CONTROLSTATUS

Set the status of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLSTATUS|CNTRSTA) (RUN|RU|STOP|ST)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> TE:CNTRSTA RUN
Matisse> OK
```

THINETALON:CONTROLSTATUS

Get the status of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLSTATUS|CNTRSTA)?
```

Reply Syntax

```
:TE:CNTRSTA: (RUN|STOP)
```

Example

```
Cmd> TE:CNTRSTA?
Matisse> :TE:CNTRSTA: RUN
```

THINETALON:CONTROLPROPORTIONAL

Set the proportional gain of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLPROPORTIONAL|CNTRPROP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

THINETALON:CONTROLPROPORTIONAL?

Get the proportional gain of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLPROPORTIONAL|CNTRPROP)?
```

Reply Syntax

```
:TE:CNTRPROP: <float>
```

Example

```
N/A
```

THINETALON:CONTROLERROR?

Get the current error value of the Thin Etalon control loop.

Command Syntax

```
(THINETALON|TE):(CONTROLERROR|CNTRERR)?
```

Reply Syntax

```
:TE:CNTRERR: <float>
```

Example

```
N/A
```

THINETALON:CONTROLAVERAGE

Set the number of measurements averaged of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLAVERAGE|CNTRAVG) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

THINETALON:CONTROLAVERAGE?

Get the number of measurements averaged of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLAVERAGE|CNTRAVG)?
```

Reply Syntax

```
:TE:CNTRAVG: <integer>
```

Example

```
N/A
```

THINETALON:CONTROLSETPOINT

Set the control goal of the Thin Etalon control loop. The error signal of the *PI control loop* is calculated according to:

```
Error = TE:CNTRSP - (TE:DCVALUE)/(DPOW:DCVALUE)
```

Command Syntax

```
(THINETALON|TE):(CONTROLSETPOINT|CNTRSP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

THINETALON:CONTROLSETPOINT?

Get the control goal of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLSETPOINT|CNTRSP)?
```

Reply Syntax

```
:TE:CNTRSP: <float>
```

Example

```
Cmd> TE:CNTRSP?
Matisse> :TE:CNTRSP: 3.457942e-01
```

THINETALON:CONTROLINTEGRAL

Set the integral gain of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLINTEGRAL|CNTRINT) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> TE:CNTRINT 4.7
Matisse> OK
```

THINETALON:CONTROLINTEGRAL?

Get the integral gain of the Thin Etalon *PI control loop*.

Command Syntax

```
(THINETALON|TE):(CONTROLINTEGRAL|CNTRINT)?
```

Reply Syntax

```
:TE:CNTRINT: <float>
```

Example

```
Cmd> TE:CNTRINT?
Matisse> :TE:CNTRINT: 4.700000e+00
```

THINETALON:CAVITYSCAN

Set the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Thin Etalon. The factor results in an immediate stepper motor movement, even without the *PI control loop* enabled.

Command Syntax

```
(THINETALON|TE):(CAVITYSCAN|CAVSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> TE:CNTRINT 4.7
Matisse> OK
```

Compatibility

```
Version 1.3
```

THINETALON:CAVITYSCAN?

Get the proportional factor that controls how a scan of the slow cavity piezo influences the position of the Thin Etalon. The factor results in an immediate stepper motor movement, even without the *PI control loop* enabled.

Command Syntax

```
(THINETALON|TE):(CAVITYSCAN|CAVSCN)?
```

Reply Syntax

```
:TE:CAVSCN: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

THINETALON:REFERENCESCAN

Set the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Thin Etalon. The factor results in an immediate stepper motor movement, even without the *PI control loop* enabled. On the other hand, a value of 0 for this parameter corresponds to a control loop only operation of the Piezo Etalon. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(THINETALON|TE):(REFERENCESCAN|REFSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> TE:REFSCN 2670
Device> OK
SFC> TE:REFSCN?
Device> :TE:REFSCN: 2.670000e+03
SFC> REFCELL:GNC 2
Device> OK
SFC> TE:REFSCN?
Device> :TE:REFSCN: 6.675000e+03
```

Compatibility

```
Version 1.3
```

THINETALON:REFERENCESCAN?

Get the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Thin Etalon. The factor results in an immediate stepper motor movement, even without the *PI control loop* enabled. On the other hand, a value of 0 for this parameter corresponds to a control loop only operation of the Piezo Etalon. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(THINETALON|TE):(REFERENCESCAN|REFSCN)?
```

Reply Syntax

```
:TE:REFSCN: <float>
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> TE:REFSCN 2670
Device> OK
SFC> TE:REFSCN?
Device> :TE:REFSCN: 2.670000e+03
SFC> REFCELL:GNC 2
Device> OK
```

```
SFC> TE:REFSCN?
Device> :TE:REFSCN: 6.675000e+03
```

Compatibility

```
Version 1.3
```

Scan Control

These commands control the scanning mirror, or if used, the Reference Cell.

SCAN:STATUS

Start or stop a scan.

Command Syntax

```
SCAN:(STATUS|STA) (RUN|RU|STOP|ST)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:STATUS?

Get current status of the scan

Command Syntax

```
SCAN:(STATUS|STA)?
```

Reply Syntax

```
:SCAN:STA: (RUN|STOP)
```

Example

```
Cmd> SCAN:STA?
Matisse> :SCAN:STA: RUN
```

SCAN:LOWERLIMIT

Set the lower limit of the scan pattern. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:(LOWERLIMIT|LLM) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:LOWERLIMIT?

Get the lower limit of the scan pattern. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:(LOWERLIMIT|LLM)?
```

Reply Syntax

```
:SCAN:LLM: <float>
```

Example

```
N/A
```

SCAN:UPPERLIMIT

Set the upper limit of the scan pattern. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:(UPPERLIMIT|ULM) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:UPPERLIMIT?

Get the upper limit of the scan pattern. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:(UPPERLIMIT|ULM)?
```

Reply Syntax

```
:SCAN:ULM: <float>
```

Example

```
N/A
```

SCAN:NOW

Set the current scan position. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:NOW <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:NOW?

Get the current scan position. Scan positions are within the interval [0,0.7].

Command Syntax

```
SCAN:NOW?
```

Reply Syntax

```
:SCAN:NOW: <float>
```

Example

```
N/A
```

SCAN:MODE

Set the current scan mode. The scan mode determines the direction of the scan and whether it stops at one of the limits. The behaviour is coded into the bits of this variable. When the scan device reaches one of the limit values, the direction is inverted. As a next step the scan is stopped at the limit, if the appropriate control bit is set.

Bit	Action if bit is set
0	increase voltage, stop at neither limit
1	decrease voltage, stop at neither limit
2	increase voltage, stop at lower limit
3	decrease voltage, stop at lower limit
4	increase voltage, stop at upper limit
5	decrease voltage, stop at upper limit
6	increase voltage, stop at either limit
7	decrease voltage, stop at either limit

Command Syntax

```
SCAN:MODE <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:MODE?

Get the current scan mode. The scan mode determines the direction of the scan and whether it stops at one of the limits. The behaviour is coded into the bits of this variable. When the scan device reaches one of the limit values, the direction is inverted. As a next step the scan is stopped at the limit, if the appropriate control bit is set.

Bit	Action if bit is set
0	increase voltage, stop at neither limit
1	decrease voltage, stop at neither limit
2	increase voltage, stop at lower limit
3	decrease voltage, stop at lower limit
4	increase voltage, stop at upper limit

Bit	Action if bit is set
5	decrease voltage, stop at upper limit
6	increase voltage, stop at either limit
7	decrease voltage, stop at either limit

Command Syntax

```
SCAN:MODE?
```

Reply Syntax

```
:SCAN:MODE: <integer>
```

Example

```
N/A
```

SCAN:DEVICE

Set the device that controls the scan of the *Matisse* laser. This device is the master that controls the tuning of the system, all other devices follow the master device either by open-loop (e. g. Birefringent Filter) or closed- loop control (e. g. Piezo Etalon). If the specified device is already used by another command e.g. the `SLOWPIEZO` control loop, an error message will be returned.

Code	Device
0	no device
1	slow cavity piezo
2	reference cell piezo

Command Syntax

```
SCAN:(DEVICE|DEV) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SCAN:DEVICE?

Get the device that controls the scan of the *Matisse Commander* laser. This device is the master that controls the tuning of the system, all other devices follow the master device either by open-loop (e. g. Birefringent Filter) or closed- loop control (e. g. Piezo Etalon).

Code	Device
0	no device
1	slow cavity piezo
2	reference cell piezo

Command Syntax

```
SCAN:(DEVICE|DEV)?
```

Reply Syntax

```
:SCAN:DEV: <integer>
```

Example

```
N/A
```

SCAN:RISINGSPEED

Set the speed of the voltage ramp-up of the scan mirror.

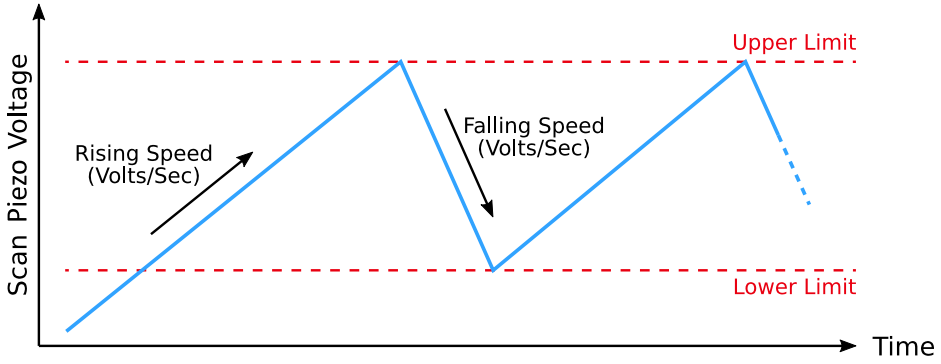


Figure 17: Scan Timing

Command Syntax

```
SCAN:(RISINGSPEED|RSPD) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> SCAN:RSPD 0.01
Matisse> OK
```

SCAN:RISINGSPEED?

Get the speed of the voltage ramp-up of the scan mirror.

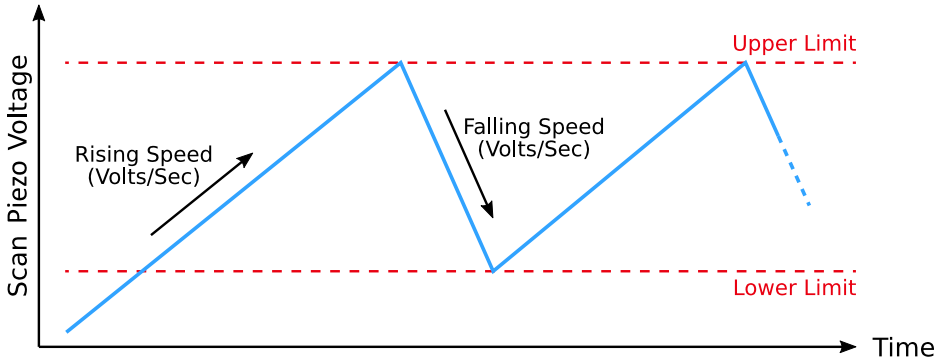


Figure 18: Scan Timing

Command Syntax

```
SCAN:(RISINGSPEED|RSPD)?
```

Reply Syntax

```
:SCAN:RSPD: <float>
```

Example

```
Cmd> SCAN:RSPD 0.01
Matisse> OK
```

SCAN:FALLINGSPEED

Set the speed of the voltage ramp-down of the scan mirror.

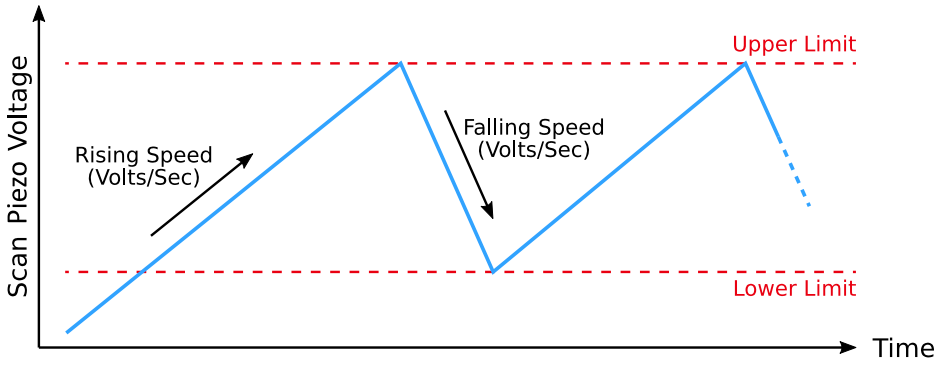


Figure 19: Scan Timing

Command Syntax

```
SCAN:(FALLINGSPEED|FSPD) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> SCAN:FSPD 0.01
Matisse> OK
```

SCAN:FALLINGSPEED?

Get the speed of the voltage ramp-down of the scan mirror.

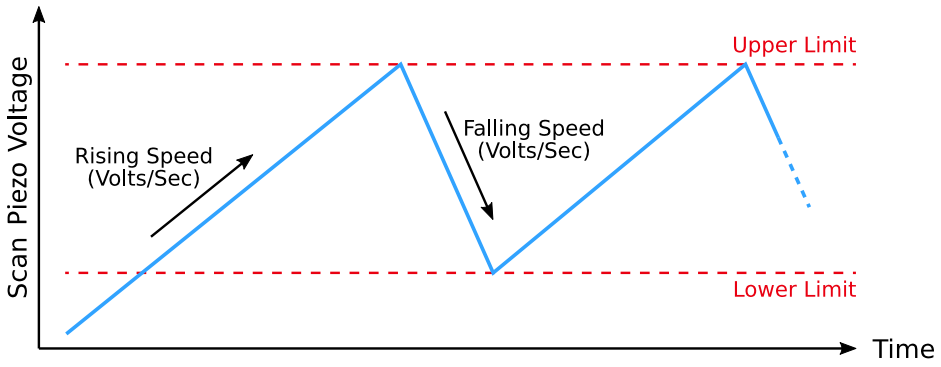


Figure 20: Scan Timing

Command Syntax

```
SCAN:(FALLINGSPEED|FSPD)?
```

Reply Syntax

```
:SCAN:FSPD: <float>)
```

Example

```
Cmd> SCAN:FSPD?
Matisse> :SCAN:FSPD: 2.0E-3
```

SCAN:REFERENCECALIBRATION

Set the scan device calibration factor for Reference Cell controlled scans. The value is stored into the laser's flash memory but has no further influence on the operation.

Command Syntax

```
SCAN:(REFERENCECALIBRATION|REFCAL) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> SCAN:REFCAL 1525.3
Matisse> OK
```

Compatibility

Version 1.13

SCAN:REFERENCECALIBRATION?

Get the scan device calibration factor for Reference Cell controlled scans. The value is stored into the laser's flash memory but has no further influence on the operation.

```
SCAN:(REFERENCECALIBRATION|REFCAL)?
```

Reply Syntax

```
:SCAN:REFCAL: <float>)
```

Example

```
Cmd> SCAN:REFCAL?
Matisse> :SCAN:REFCAL: 1.5260E3
```

Compatibility

Version 1.13

SCAN:CAVITYCALIBRATION

Set the scan device calibration factor for cavity scans. The value is stored into the laser's flash memory but has no further influence on the operation.

Command Syntax

```
SCAN:(CAVITYCALIBRATION|CAVCAL) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> SCAN:CAVCAL 1546.5
Matisse> OK
```

Compatibility
Version 1.13

SCAN:CAVITYCALIBRATION?

Get the scan device calibration factor for cavity scans. The value is stored into the laser's flash memory but has no further influence on the operation.

Command Syntax

```
SCAN:(CAVITYCALIBRATION|CAVCAL)?
```

Reply Syntax

```
:SCAN:CAVCAL: <float>)
```

Example

```
Cmd> SCAN:CAVCAL?
Matisse> :SCAN:CAVCAL: 1.2560E3
```

Compatibility
Version 1.13

Fast Piezo Control

One of the laser's cavity mirrors is mounted on a Fast Piezo stack. This piezo is used to compensate fluctuations in the output wavelength (cavity length). The inertia of the masses involved (and the speed of sound) limits the bandwidth of this element. This element is also known as the "tweeter" opposed to the Slow Piezo or "woofer".

FASTPIEZO:LOCK?

This variable tracks the locking status of the laser. The laser is considered to be locked whenever the tweeter is within 5%...95% of its tuning range.

The criterion is based on the following concept: if the laser is not locked, it will not react to the tweeter. So any error will be integrated until the tweeter is on the lower or upper end of its tuning range. Retrieving this value automatically resets the variable value to TRUE. Whenever the tweeter is within the last 5% of its tuning range the variable is set `FALSE`.

Command Syntax

```
(FASTPIEZO|FPZT):LOCK?
```

Reply Syntax

```
:FPZT:LOCK: (TRUE|FALSE)
```

Example

```
N/A
```

FASTPIEZO:CONTROLSETPOINT
Set the control goal of the Fast Piezo control loop.

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLSETPOINT|CNTRSP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> FPZT:CNTRSP 0.235
Matisse> OK
```

FASTPIEZO:CONTROLSETPOINT?
Get the control goal of the Fast Piezo control loop.

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLSETPOINT|CNTRSP)?
```

Reply Syntax

```
:FPZT:CNTRSP: <float>
```

Example

```
Cmd> FPZT:CNTRSP?
Device> :FPZT:CNTRSP: 2.349854e-01
```

FASTPIEZO:CONTROLSTATUS
Start or stop the PID loop that controls the Fast Piezo (tweeter).

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLSTATUS|CNTRSTA) (RUN|RU|STOP|ST)
```

Reply Syntax

```
OK|!ERROR
```

Example

```
N/A
```

FASTPIEZO:CONTROLSTATUS?
Get the status of the PID loop that controls the Fast Piezo (tweeter).

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLSTATUS|CNTRSTA)?
```

Reply Syntax

```
:FPZT:CNTRSTA: (RUN|STOP)
```

Example

```
N/A
```

FASTPIEZO:INPUT?
Get the current value of the diode at the Reference Cell (or the current voltage at the external input of the DSP board). The value is normalized to be in the range -1..1. This is a read only value.

Command Syntax

```
(FPZT|FASTPIEZO):(INPUT|INP)?
```

Reply Syntax

```
:FPZT:INP: <float>
```

Example

```
Cmd> FPZT:INPUT?
Matisse> :FPZT:INP: 0.000000e+00
```

Compatibility

```
Version 1.3
```

FASTPIEZO:CONTROLINTEGRAL

Set the integral gain of the Fast Piezo control loop.

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLINTEGRAL|CNTRINT) <float>
```

Reply Syntax

```
OK|!ERROR
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

FASTPIEZO:CONTROLINTEGRAL?

Get the integral gain of the Fast Piezo control loop.

Command Syntax

```
(FASTPIEZO|FPZT):(CONTROLINTEGRAL|CNTRINT)?
```

Reply Syntax

```
:FPZT:CNTRINT: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

FASTPIEZO:LOCKPOINT

Set the value for the initial control goal value. When the laser needs to perform an initial lock or a re-locking, the control loop will lock the laser to the value given by `FASTPIEZO:LOCKPOINT`. After the laser is stabilized to the `FASTPIEZO:LOCKPOINT` value, the control loop will change its control goal to `FASTPIEZO:CONTROLSETPOINT` after a while. The change will be smooth.

Command Syntax

```
(FPZT|FASTPIEZO):(LOCKPOINT|LKP) <float>
```

Reply Syntax

```
OK|!ERROR
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

FASTPIEZO:LOCKPOINT

Get the value to which the Fast Piezo control locks the laser before it smoothly moves the Fast Piezo control from the lockpoint to the setpoint.

Command Syntax

```
(FASTPIEZO|FPZT):(LOCKPOINT|LKP)?
```

Reply Syntax

```
:FPZT:LKP: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

FASTPIEZO:NOW

Set the current position of the Fast Piezo. The value should be in the range 0..1. An active control loop (`FASTPIEZO:CONTROLSTATUS = RUN`) will overwrite the value after a short time.

Command Syntax

```
(FPZT|FASTPIEZO):NOW <integer>
```

Reply Syntax

```
OK|!ERROR
```

Example

```
N/A
```

FASTPIEZO:NOW?

Get the current position of the Fast Piezo. The value is in the range 0..1.

Command Syntax

```
(FPZT|FASTPIEZO):NOW?
```

Reply Syntax

```
:FPZT:NOW: <float>
```

Example

```
N/A
```

Reference Cell Control

This command group is used for controlling the Reference Cell of the *Matisse* laser.

REFERENCECELL:TABLE?

This command starts the process to perform a Reference Cell scan and measure the intensity on the reference diode at the same time. The scan will start at the position defined by `REFERENCECELL:LOWERLIMIT` and end at the position defined by `REFERENCECELL:UPPERLIMIT`. The intensity will be measured at a number of intermediate positions, the number being defined by `REFERENCECELL:OVERSAMPLING`.

At the end of the measurement the scan piezo will be reset to the lower limit position. If the Reference Cell piezo is already used by another command e.g. the `SCAN` command, an error message is returned.

Command Syntax

```
(REFERENCECELL|REFCELL):TABLE?
```

Reply Syntax

```
:REFCELL:TABLE: {<float>}
```

Example

```
N/A
```

REFERENCECELL:OVERSAMPLING

Set the number of points sampled during the `REFERENCECELL:TABLE?` command. The minimum value is 4 the maximum value is 512 samples.

Command Syntax

```
(REFERENCECELL|REFCELL):(OVERSAMPLING|OVER) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

REFERENCECELL:OVERSAMPLING?

Get the number of points sampled during the `REFERENCECELL:TABLE?` command.

Command Syntax

```
(REFERENCECELL|REFCELL):(OVERSAMPLING|OVER)?
```

Reply Syntax

```
:REFCELL:OVER: <integer>
```

Example

```
N/A
```

REFERENCECELL:LOWERLIMIT

Set the lower limit for the scan that is performed during the `REFERENCECELL:TABLE?` command.

Command Syntax

```
(REFERENCECELL|REFCELL):(LOWERLIMIT|LLM) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

REFERENCECELL:LOWERLIMIT?

Get the lower limit for the scan that is performed during the `REFERENCECELL:TABLE?` command.

Command Syntax

```
(REFERENCECELL|REFCELL):(LOWERLIMIT|LLM)?
```

Reply Syntax

```
:REFCELL:LLM: <float>
```

Example

```
N/A
```

REFERENCECELL:UPPERLIMIT

Set the upper limit for the scan that is performed during the `REFERENCECELL:TABLE?` command.

Command Syntax

```
(REFERENCECELL|REFCELL):(UPPERLIMIT|ULM) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

REFERENCECELL:UPPERLIMIT?

Get the upper limit for the scan that is performed during the `REFERENCECELL:TABLE?` command.

Command Syntax

```
(REFERENCECELL|REFCELL):(UPPERLIMIT|ULM) <float>
```

Reply Syntax

```
:REFCELL:ULM: <float>
```

Example

```
N/A
```

REFERENCECELL:MODE

Set the measurement mode for the data acquisition during the scan that is performed during the `REFERENCECELL:TABLE?` command.

Code	Mode
0	None
1	Average
2	Minimum
3	Maximum

Command Syntax

```
(REFERENCECELL|REFCELL):MODE <int>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

REFERENCECELL:MODE?

Get the measurement mode for the data acquisition during the scan that is performed during the `REFERENCECELL:TABLE?` command.

Code	Mode
0	None
1	Average
2	Minimum
3	Maximum

Command Syntax

```
(REFERENCECELL|REFCELL):MODE?
```

Reply Syntax

```
:REFCELL:MODE: <int>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

REFERENCECELL:NOW

Set the position of the Reference Cell piezo.

Command Syntax

```
(REFERENCECELL|REFCELL):NOW <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.11
```

REFERENCECELL:NOW?

Get the position of the Reference Cell piezo.

Command Syntax

```
(REFERENCECELL|REFCELL):NOW?
```

Reply Syntax

```
:REFCELL:NOW: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.11
```

REFERENCECELL:GAINCODE

Set the gaincode of a variable gain Reference Cell piezo amplifier. The Reference Cell piezo amplifier is optionally equipped with a gain select. Different codes select different gains for this amplifier. Lower gains result in higher resolution of the Reference Cell. Higher gains result in an increased scan range.

Code	Gain
-1	invalid / not defined
0	0 / grounded
1	2
2	5
3	10
4	32

Command Syntax

```
(REFERENCECELL|REFCELL):(GAINCODE|GNC) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.14
```

REFERENCECELL:GAINCODE?

Get the gaincode of a variable gain Reference Cell piezo amplifier. The Reference Cell piezo amplifier is optionally equipped with a gain select. Different codes select different gains for this amplifier. Lower gains result in higher resolution of the Reference Cell. Higher gains result in an increased scan range.

Code	Gain
-1	invalid / not defined
0	0 / grounded

Code	Gain
1	2
2	5
3	10
4	32

Command Syntax

```
(REFERENCECELL|REFCELL):(GAINCODE|GNC) "?"
```

Reply Syntax

```
:REFCEL:GNC: |(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.14
```

REFERENCECELL:GAIN?

Get the current gain of the Reference Cell amplifier.

Command Syntax

```
(REFERENCECELL|REFCELL):GAIN "?"
```

Reply Syntax

```
:REFCELL:GAIN: <float> |(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.14
```

Slow Piezo Control

SLOWPIEZO:LOCKPROPORTIONAL

Set the proportional gain of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects that the laser is locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(LOCKPROPORTIONAL|LPROP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

SLOWPIEZO:LOCKPROPORTIONAL?

Get the proportional gain of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects that the laser is locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(LOCKPROPORTIONAL|LPROP)?
```

Reply Syntax

```
:SPZT:LPROP: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

SLOWPIEZO:LOCKLINTEGRAL

Set the integral gain of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects that the laser is locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(LOCKINTEGRAL|LINT) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

SLOWPIEZO:LOCKLINTEGRAL?

Get the integral gain of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects that the laser is locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(LOCKINTEGRAL|LINT)?
```

Reply Syntax

```
:SPZT:CNTRINT: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.3
```

SLOWPIEZO:FREESPEED
Set the speed of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects that the laser is not locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(FREESPEED|FRSP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.13
```

SLOWPIEZO:FREESPEED?
Get the speed of the Slow Piezo (cavity scan piezo) control loop. This value is used when the control loop detects the laser is not locked to the Reference Cell.

Command Syntax

```
(SLOWPIEZO|SPZT):(FREESPEED|FRSP)?
```

Reply Syntax

```
:SPZT:FRSP: <float>
```

Example

```
N/A
```

Compatibility

```
Version 1.13
```

SLOWPIEZO:CONTROLSETPOINT
Set the control goal of the Slow Piezo (cavity scan piezo) control loop.

Command Syntax

```
(SLOWPIEZO|SPZT):(CONTROLSETPOINT|CNTRSP) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SLOWPIEZO:CONTROLSETPOINT?
Get the control goal of the Slow Piezo (cavity scan piezo) control loop.

Command Syntax

```
(SLOWPIEZO|SPZT):(CONTROLSETPOINT|CNTRSP)?
```

Reply Syntax

```
:SPZT:CNTRSP: <float>
```

Example

```
N/A
```

SLOWPIEZO:CONTROLSTATUS

Start or stop the PID loop that controls the Slow Piezo (cavity scan piezo). The Slow Piezo control will only be active when the Fast Piezo control is running at the same time. If the slow cavity piezo is already used by another command e.g. the `SCAN` command, an error message is returned.

Command Syntax

```
(SLOWPIEZO|SPZT):(CONTROLSTATUS|CNTRSTA) (RUN|RU|STOP|ST)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SLOWPIEZO:CONTROLSTATUS?

Get the status of the PID loop that controls the Slow Piezo (cavity scan piezo).

Command Syntax

```
(SLOWPIEZO|SPZT):(CONTROLSTATUS|CNTRSTA)?
```

Reply Syntax

```
:SPZT:CNTRSTA: (RUN|STOP)
```

Example

```
N/A
```

SLOWPIEZO:NOW

Set the position of the Slow Piezo (cavity scan piezo).

Command Syntax

```
(SLOWPIEZO|SPZT):NOW <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

SLOWPIEZO:NOW?

Get the current position of the Slow Piezo (cavity scan piezo).

Command Syntax

```
(SLOWPIEZO|SPZT):NOW?
```

Reply Syntax

```
:SPZT:NOW: <float>
```

Example

```
N/A
```

SLOWPIEZO:REFERENCESCAN

Set the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Slow Piezo. The factor results in an immediate piezo movement, even without the PID loop enabled. On the other hand, a value of 0 for this parameter corresponds to a control loop only operation of the Piezo Etalon. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(SLOWPIEZO|SPZT):(REFERENCESCAN|REFSCN) <float>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
SFC> REFCELL:GNC 1
Device> OK
SFC> SPZT:REFSCN -0.9121
Device> OK
SFC> SPZT:REFSCN?
Device> :SPZT:REFSCN: -9.121000e-01
SFC> REFCELL:GNC 2
Device> OK
SFC> SPZT:REFSCN?
Device> :SPZT:REFSCN: -2.280250e+00
```

Compatibility

```
Version 1.3
```

SLOWPIEZO:REFERENCESCAN?

Get the proportional factor that controls how a scan of the Reference Cell piezo influences the position of the Slow Piezo. The factor results in an immediate piezo movement, even without the PID loop enabled. On the other hand, a value of 0 for this parameter corresponds to a control loop only operation of the Piezo Etalon. If a Reference Cell piezo amplifier with variable gain is installed, this value will be changed according to the selected gain.

Command Syntax

```
(SLOWPIEZO|SPZT):(REFERENCESCAN|REFSCN)?
```

Reply Syntax

```
:SPZT:REFSCN: <float>
```

Example

```
SFC> REFCELL:GNC 1
```

```
Device> OK
SFC> SPZT:REFSCN -0.9121
Device> OK
SFC> SPZT:REFSCN?
Device> :SPZT:REFSCN: -9.121000e-01
SFC> REFCELL:GNC 2
Device> OK
SFC> SPZT:REFSCN?
Device> :SPZT:REFSCN: -2.280250e+00
```

Compatibility

```
Version 1.3
```

Pound-Drever-Hall Unit

The Pound-Drever-Hall unit is used for high performance stabilization of the laser system. To generate the Pound-Drever-Hall error signal a fraction of the laser beam is modulated with the external Electro-Optical Modulator (EOM) using a constant frequency of 20 MHz. This mixture of beams is directed to a HighFinesse Etalon. The beams have different frequencies. Hence, they have a different degree of resonance with the Etalon. This introduces different phase-shifts to the reflected beams which generate a beat-note signal on a fast photo diode.

The fast diode's signal is analysed by mixing it with the driving frequency of the external EOM (this is variably phase shifted by a factor determined by the parameter `POUNDDREVERHALL:PHASESHIFT`). The resulting waveform is called the Pound-Drever-Hall error signal. An offset voltage is added (`POUNDDREVERHALL:DSPOFFSET`) and the error signal is transmitted to the input of the digital signal processor.

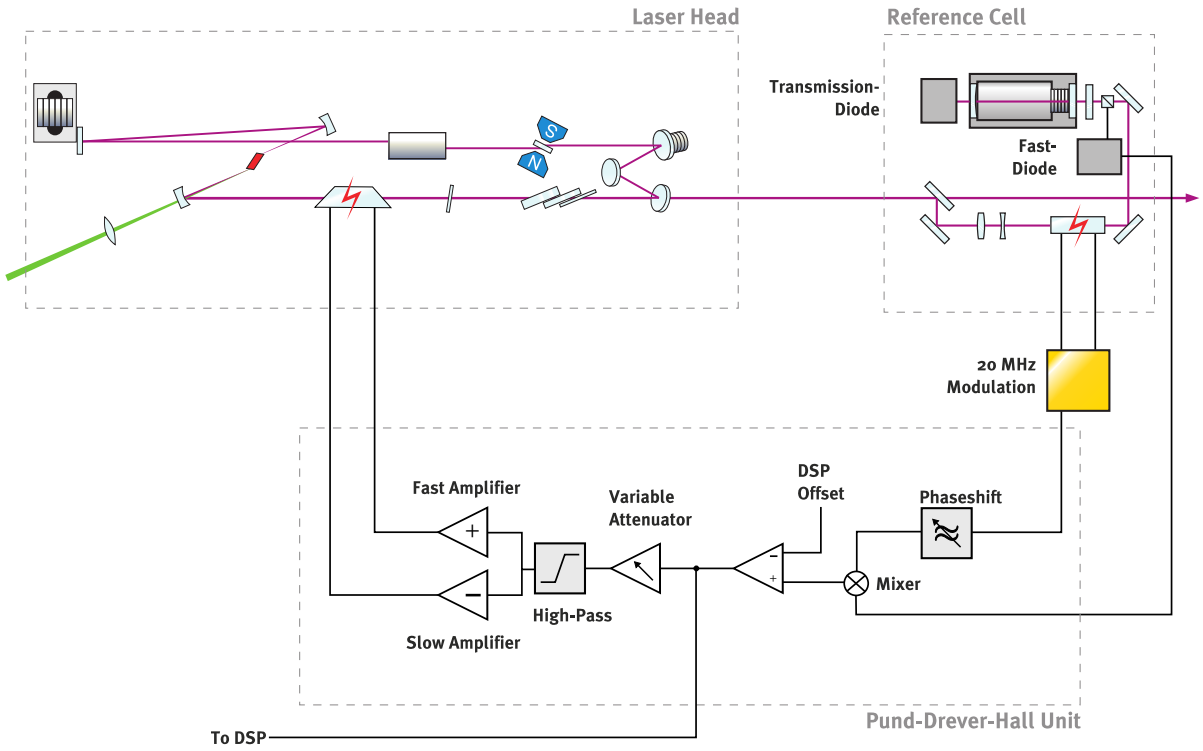


Figure 21: Matisse TX Setup

POUNDDREVERHALL:ATTENUATOR

Set the attenuation factor for the error signal before it is amplified and feed to the intra-cavity Electro-Optical Modulator. The value may be set to a value from 0 to 256. Values outside of the range are coerced to be within limits. A value of zero corresponds to no attenuation. Each increment attenuates the input by 0.5 dB.

This commands needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier.

Command Syntax

```
(POUNDDREVERHALL|PDH):(ATTENUATOR|ATT) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:ATTENUATOR?

Get the attenuation factor for the error signal before it is amplified and feed to the intra-cavity Electro-Optical Modulator. The value ranges from 0 to 256.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier.

Command Syntax

```
(POUNDDREVERHALL|PDH):(ATTENUATOR|ATT)?
```

Reply Syntax

```
:PDH:ATT: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:CONTROL

Get the signal source for the multiplexer and Electro-Optical Modulator enable status.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board.

Bit 4 switches the input for the error signal from the internal Pound-Drever-Hall detection to the SMA connector.

The board needs to be fitted with such a connector for this bit to have an effect.

Bit	Purpose
0	Multiplexer channel selection
1	Multiplexer channel selection
2	Intra cavity EOM on/off
3	20 MHz modulation on/off
4	Use internal/external error signal input

The first two bits encode multiplexer input:

Value	Input
0	Slow Side Signal for intra-cavity EOM
1	Diode Signal
2	Mixer Output
3	Transmission Diode Signal

Command Syntax

```
(POUNDDREVERHALL|PDH):(CONTROL|CTRL) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:CONTROL?

Control signal source for the multiplexer and Electro-Optical Modulators enable status.

This command needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board.

Bit 4 switches the input for the error signal from the internal Pound-Drever-Hall detection to the SMA connector. The board needs to be fitted with such a connector for this bit to have an effect.

Bit	Purpose
0	Multiplexer channel selection
1	Multiplexer channel selection
2	Intra cavity EOM on/off
3	20 MHz modulation on/off
4	Use internal/external error signal input

The first two bits encode multiplexer input:

Value	Input
0	Slow Side Signal for intra-cavity EOM
1	Diode Signal
2	Mixer Output
3	Transmission Diode Signal

Command Syntax

```
(POUNDDREVERHALL|PDH):(CONTROL|CTRL)?
```

Reply Syntax

```
:PDH:CTRL: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:DSPOFFSET

Set the value for the offset compensation of the error signal that is feed into the DSP input. The value may be set to a value from 0 to 255. Values outside of the range are coerced to be within limits.

This commands needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board.

Command Syntax

```
(POUNDDREVERHALL|PDH):(DSPOFFSET|DSPOFF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:DSPOFFSET?

Get the value for the offset compensation of the error signal that is feed into the DSP input. The value ranges from 0 to 255.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board

Command Syntax

```
(POUNDDREVERHALL|PDH):(DSPOFFSET|DSPOFF)? <integer>
```

Reply Syntax

```
:PDH:DSPOFF: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:FASTOFFSET

Set the value for the offset compensation of the fast side driver of the intra-cavity Electro-Optical Modulator. The value may be set to a value from 0 to 255. Values outside of the range are coerced to be within limits.

This command needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier

Command Syntax

```
(POUNDDREVERHALL|PDH):(FASTOFFSET|FOFF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:FASTOFFSET?

Get the value for the offset compensation of the fast side driver of the intra-cavity Electro-Optical Modulator. The value ranges from 0 to 255.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier.

Command Syntax

```
(POUNDDREVERHALL|PDH):(FASTOFFSET|FOFF)?
```

Reply Syntax

```
:PDH:FOFF: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:PHASESHIFT

Set the phaseshift in between 20 MHz generator signal and detected back reflection signal from the Reference Cell. The value may be set to a value from 0 to 255. Values outside of the range are coerced to be within limits.

This command needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board.

Command Syntax

```
(POUNDDREVERHALL|PDH):(PHASESHIFT|PHSF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:PHASESHIFT?

Get the phaseshift in between 20 MHz generator signal and detected back reflection signal from the Reference Cell. The value ranges from 0 to 255.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board.

Command Syntax

```
(POUNDDREVERHALL|PDH):(PHASESHIFT|PHSF)?
```

Reply Syntax

```
:PDH:PHSF: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:RESET

Reset all parameters of the Pound-Drever-Hall device to their current values. Due to the independent power supply of the Pound-Drever-Hall box it is possible to have deviating values. Use this command to synchronize the box and the laser control.

This commands needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

Command Syntax

```
(POUNDDREVERHALL|PDH):(RESET|RST)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:SLOWOFFSET

Set the value for the offset compensation of the slow side driver of the intra-cavity Electro-Optical Modulator. The value may be set to a value from 0 to 255. Values outside of the range are coerced to be within limits.

This command needs noticeable time to execute (10 ms .. 100 ms) due to the slow communication with the Pound-Drever-Hall controller board.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier.

Command Syntax

```
(POUNDDREVERHALL|PDH):(SLOWOFFSET|SOFF) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

POUNDDREVERHALL:SLOWOFFSET?

Get the value for the offset compensation of the slow side driver of the intra-cavity Electro-Optical Modulator. The value ranges from 0 to 255.

This variable is enabled when the laser is configured to have a Pound-Drever-Hall board with amplifier.

Command Syntax

```
(POUNDDREVERHALL|PDH):(SLOWOFFSET|SOFF)?
```

Reply Syntax

```
:PDH:SOFF: <integer>
```

Example

```
N/A
```

Compatibility

```
Version 1.4
```

Parameter

The device's behaviour is controlled by a number of parameters. All values of these parameters are called a set. The control software has a number of commands that allow to store, manipulate and retrieve these parameter sets.

The DSP card is able to store about 20 different parameter sets.

PARAMETER:NOW

Set all parameters to the values of the parameter set stored with the given name.

Command Syntax

```
(PARAMETER|PAR):NOW <parameter name>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PAR:NOW?
Device> :PAR:NOW: "FACTORY_ONE"
Cmd> PAR:NOW "USER_ONE"
Device> OK
```

```
Cmd> PAR:NOW?
Device> :PAR:NOW: "USER_ONE"
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:NOW?

Get the name of the currently active parameter set.

Command Syntax

```
(PARAMETER|PAR):NOW?
```

Reply Syntax

```
:PAR:NOW: <parameter name>
```

Example

```
Cmd> PAR:NOW?
Device> :PAR:NOW: "FACTORY_ONE"
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:DEFAULT

Determine the parameter set that is loaded during cold start of the device. To check the currently active default parameter set you have to use the PAR:LIST command and scan for the parameter name with angle brackets.

Note: During the execution of this command, control of the Piezo Etalon is disabled. Command execution can take 5 .. 10 sec.

Command Syntax

```
(PARAMETER|PAR):(DEFAULT|DEF) <parameter name>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PAR:LIST?
Device> :PAR:LIST: "[*FACTORY_ONE][*FACTORY_TWO]
[*FACTORY_THREE]<USER_ONE>[USER_TWO][USER_THREE]"
Cmd> PAR:DEF "FACTORY_ONE"
Device> OK
Cmd> PAR:LIST?
Matisse> :PAR:LIST:"<*FACTORY_ONE>[*FACTORY_TWO]
[*FACTORY_THREE][USER_ONE][USER_TWO][USER_THREE]"
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:SET

Store the current parameter setting using the given name.

Note: During the execution of this command, control of the Piezo Etalon is disabled. Command execution can take 5 .. 10 sec. Parameter names are limited to 24 characters, letters are converted to uppercase.

Command Syntax

```
(PARAMETER|PAR):SET <parameter name>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PAR:LIST?
Device> :PAR:LIST: "[*FACTORY_ONE][*FACTORY_TWO]
[*FACTORY_THREE]<USER_ONE>[USER_TWO]"
Cmd> PAR:DEF "USER_THREE"
Device> OK
Cmd> PAR:LIST?
Device> :PAR:LIST: "[*FACTORY_ONE][*FACTORY_TWO]
[*FACTORY_THREE]<USER_ONE>[USER_TWO][USER_THREE]"
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:LIST?

Retrieve a list of all currently defined parameter sets. The result is a string enclosed in quotation marks ("). Each parameter name is surrounded by square brackets ([..]), except for the current default set which will be enclosed in sharp brackets (<..>). Factory sets will have a star (*) in front of their names.

Command Syntax

```
(PARAMETER|PAR):LIST?
```

Reply Syntax

```
:PAR:LIST: <string>
```

Example

```
Cmd> PAR:LIST?
Device> :PAR:LIST: "[*FACTORY_ONE][*FACTORY_TWO]
[*FACTORY_THREE]<USER_ONE>[USER_TWO][USER_THREE]"
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:MEMORY?

Retrieves the contents of the flash memory starting at the location given as argument. The command returns the next 16 bytes in hexadecimal notation.

Syntax

```
(PARAMETER|PAR):LIST?
```

Example

```
Example
```

PARAMETER:DEVICE

Set which devices are configured during the start up and operation of the *Matisse*. The devices are coded into the respective bits of the argument. Only devices that were selected during the power on cycle of the *Matisse* are initialized properly. Hence, to activate a new PARAMETER:DEVICE value you need to create a *parameter set* and make it the new *default parameter set*. Otherwise you might experience strange behaviour of the *Matisse* controller, or error messages.

Bit Number

	Device
0	DAC for slow piezo
1	DAC for reference cell
2	Motor controller for birefringent filter
3	Motor controller for thin etalon
4	Fast Piezo
5	Piezo Etalon
6	Pound-Drever-Hall error signal generation
7	Pound-Drever-Hall EOM amplifier unit
8	Reference cell variable amplifier

Command Syntax

```
(PAR|PARAMETER):(DEV|DEVICE) <integer>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PARAMETER:DEVICE 67
Device> OK
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:DEVICE?

Get the current setting of the device configuration.

Command Syntax

```
(PARAMETER|PAR):(DEVICE|DEV)?
```

Reply Syntax

```
:PAR:DEV: <integer>
```

Example

```
Example
```

Compatibility

```
Matisse, Version 1.3
```

PARAMETER:DELETE

Delete a parameter set. When the name *ALL is used, all user parameters sets are deleted.

Note: During the execution of this command, control of the Piezo Etalon is disabled. Command execution can take 5 .. 10 sec.

Command Syntax

```
(PARAMETER|PAR):(DELETE|DEL) <parameter name>
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> PAR:LIST?
Device> :PAR:LIST:
"<*FACTORY_ONE>[*FACTORY_TWO]
[*FACTORY_THREE][USER_ONE][USER_TWO][USER_THREE]"
Cmd> PAR:DEL "USER_TWO"
Device> OK
Cmd> PAR:LIST?
Device> :PAR:LIST: "<*FACTORY_ONE>[*FACTORY_TWO]
[*FACTORY_THREE][USER_ONE][USER_THREE]"
Cmd>PAR:DEL "*ALL"
Device> OK
Cmd>PAR:LIST?
Device> :PAR:LIST: ""
```

Compatibility

```
Matisse, Version 1.3
EagleEye, Version 1.0
```

Miscellaneous

These commands are used for error processing and housekeeping.

IDENTIFICATION?

Return the identification string of the device. The identification string consists of the following components: model name, serial number, board version, firmware version, and version date.

Command Syntax

```
(IDENTIFICATION|IDN)?
```

Reply Syntax

```
:IDN: <string>
```

Example

```
Cmd> IDN?
Matisse> :IDN: "Matisse TS, S/N:05-25-20, DSP Rev.
01.00, Firmware: 1.6, Date: Apr 23 2007"
```

Compatibility

```
Matisse, Version 1.3
```

UPDATE:RESET

Reset the update procedure. Clear the internal software image.

Command Syntax

```
(UPDATE|UPD):RESET
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Example
```

UPDATE:DATA

Read a data line for the firmware update. The data consists of a stream of 16-bit words in ASCII-Hex format. Each of the byte is stored at successive address locations in a buffer memory. The data is not written to the actual program memory until the command UPDATE:EXECUTE is performed.

Command Syntax

```
(UPDATE|UPD):DATA "<string>"
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Cmd> UPDATE:DATA "0004 0000 4c5f 5f000034"
Device> OK
```

UPDATE:EXECUTE

Load the firmware update into the flash memory. A cold start of the system is required to ensure proper operation of the device.

Command Syntax

```
(UPDATE|UPD):(EXECUTE|EXE)
```

Reply Syntax

```
OK|(!ERROR <integer>)
```

Example

```
Example
```

UPDATE:CHECKSUM?

Read the checksum of the firmware update loaded into the memory. The checksum is calculated using a 32-bit unsigned integer accumulator. Please note that the checksum is returned in decimal notation.

Command Syntax

```
(UPDATE|UPD):(CHECKSUM?|CHS?)
```

Reply Syntax

```
:UPD:CHS: <integer>
```

Example

```
Cmd> UPDATE:RESET
Device> OKCmd>
UPDATE:DATA "0004 0000 4c5f 5f00 0034"
Device> OK
Cmd> UPDATE:DATA "6c00 0008 fd11 c001 0012"
Device> OK
Cmd> UPDATE:CHECKSUM?Device> :UPD:CHS: 185539
```

Compatibility

```
Matisse, Version 1.3
```

RESET

Initiate a restart of the entire controller unit. Note, that the USB connection will be lost during that process.

Syntax

```
RESET
```

Example

```
N/A
```

Compatibility

```
Matisse, Version 1.3
```

Version History

Version 1.3

The firmware version 1.3 has many structural changes compared to its predecessor version 1.2. The changes are:

1. Completely reworked functionality of the parameter set functions. The *Matisse* supports now multiple parameter sets. The individual sets are identified by names. To update from version 1.2 the entire parameter (and device identification) flash memory needs to be re-initialized. All information about the user- and factory parameter sets will be lost.
2. The control loop for the Thin Etalon.
3. The control loop for the Slow Piezo.
4. The control loop for the Fast Piezo.
5. Better performance of the view Reference Cell waveform function.
6. Better performance with fast scans.
7. Reset capability of the controller. (New command: RESET)
8. New command: PID:ORDINAL?
9. The UPTIME? command is removed.
10. Removed bug from scan direction control.
11. CONTROLSCAN parameter is split into two variables CAVITYSCAN and REFERENCESCAN
12. FPZT:INPUT? introduced

Version 1.4

Version 1.4 introduces commands necessary to control the Pound-Drever-Hall stabilization and lock electronics, improves the performance of the Fast Piezo control loop, and fixes some minor bugs of the previous version. The changes are:

1. New commands for controlling the Pound-Drever-Hall unit:

   POUNDDREVERHALL:ATTENUATOR,

   POUNDDREVERHALL:ATTENUATOR?

   POUNDDREVERHALL:CONTROL

   POUNDDREVERHALL:CONTROL?

   POUNDDREVERHALL:DSPOFFSET

   POUNDDREVERHALL:DSPOFFSET?

   POUNDDREVERHALL:FASTOFFSET

   POUNDDREVERHALL:FASTOFFSET?

   POUNDDREVERHALL:PHASESHIFT,

   POUNDDREVERHALL:PHASESHIFT?

   POUNDDREVERHALL:SLOWOFFSET

   POUNDDREVERHALL:SLOWOFFSET?

   POUNDDREVERHALL:RESET
2. The control loop for the Fast Piezo is re-programmed. It features now two cascaded lag-lead compensators. On the other hand the derivative part of the PID control is now removed. The measurement rate of the Fast Piezo loop is set to 40 kHz. The control implements now a lockpoint and a setpoint.
3. The Slow Piezo control idles when the Fast Piezo is in proximity to its setpoint.
4. Removed minor bugs.

Version 1.5

Version 1.5 introduced a subtle change to the Fast Piezo control loop. When the loop is disabled the Fast Piezo is moved to the position defined as setpoint of the Slow Piezo loop.

Version 1.6

Changes in firmware version 1.6 are:

1. Wavetables can have as much as 512 entries.
2. Read access to parameters does not influence the Piezo Etalon control. To enable this feature the data structure of the parameter entries is changed and parameter sets have to be deleted and re-written when updating from prior versions. Write access is now more time consuming.
3. Access of disabled hardware components raises an error.
4. Numerous parameters perform a range check when set by a command.
5. The Pound-Drever-Hall unit is divided into the error signal detection and the amplifier section.

This version is based on version 5.31 of TI's DSP/BIOS and Code Composer Studio 3.3. The entire USB communication was reprogrammed.

Version 1.7

Changes in firmware version 1.7 are:

1. Reprogrammed control loop for the Piezo Etalon. The SAMPLERATE parameter is introduced. The feed-forward input from the scan control is now applied at every iteration of the control cycle.
2. Bugs were fixed for the IDN and FPZT:NOW command.

This version is based on version 5.32 of TI's DSP/BIOS and Code Composer Studio 3.3.

Version 1.8

Changes in firmware 1.8 are:

1. Changed behaviour of the Fast Piezo control loop. The lag-lead control and the associated variables are disabled. The control does not support averaging any more.

2. The feed forward logic is introduced which feeds the Piezo Etalon modulation directly to the Fast Piezo. The amplitude and phase of the feed are controlled by the parameters FEEDFORWARD:AMPLITUDE and FEEDFORWARD:PHASESHIFT.

3. The USB activity LED is switched on while the firmware update is executing.

This firmware is based on version 5.32 of TI's DSP/BIOS and Code Composer Studio 3.3.

Version 1.9

Changes in firmware 1.9 are:

1. The pulse duration for the wake-up signal for the Pound-Drever-Hall unit was increased. This avoids error conditions for the first command that is send to a PDH unit in power down mode.

The firmware is based on DSP BIOS version 5.33.01, compiled with TI's toolchain version 4.3.0, using Code Composer Studio 3.3.

Version 1.10

Changes in firmware 1.10 are:

1. The HALT commands for the ***Birefringent Filter*** motor and ***Thin Etalon*** motor are added.
2. The ***PID:PROCESSSTATISTIC*** command is added.
3. The sample rate for the ***Fast Piezo control loop*** is increased to 125 kHz.
4. The PARAMETER:ADDITIONAL commands are obsoleted. The command PARAMETER:DELETE "*ALL" is introduced. A critical bug in the parameter garbage collection is fixed.

The firmware is based on DSP BIOS version 5.41.04, compiled with TI's toolchain version 4.3.6, using Code Composer Studio 3.3.

Version 1.11

Changes in firmware 1.11 are:

1. Lowered sample rate for the ***Fast Piezo control loop*** the increased rate from the prior version tended to stall the firmware.
2. Introduced the commands REFERENCECELL:NOW and REFERENCECELL:NOW? for direct control of the Reference Cell's piezo.
3. Increased overall performance and stability.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.12

Changes in firmware 1.12 are:

1. Changed the USB descriptor to manufacturer "Sirah Lasertechnik"
2. Worked on the controlscan-functionality for Birefringent Filter, Thin Etalon, and slow scan piezo.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.13

Changes in firmware 1.13 are:

1. New Stepper-Motor firmware (V25) introduced with constant speed capabilities. Introduced MOTBI:CABS, MOTBI:CREL, and MOTBI:FREQ commands.
2. Store scandevice calibration factors in *Matisse* flash memory
3. New Formula for Birefringent Filter tuning.
4. Scan piezo wraps round when searching for lock. Use a constant search speed.
5. Test upper and lower limits on scan commands.
6. General bug fixes.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.14

Changes in firmware 1.14 are:

1. New device: variable gain Reference Cell amplifier for PARAMETER:DEVICE
2. New commands: REFERENCECELL:GAIN?, REFERENCECELL:GAINCODE?, REFERENCECELL:GAINCODE
3. General bug fixes.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.15

Changes in firmware 1.15 are:

1. General bug fixes.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.16

Changes in firmware 1.16 are:

1. Fix bug for Reference Cell control scan parameters during start-up.
2. Now PZETL:CNTRSTA STOP automatically sets the modulation amplitude to zero. Restarting it sets the amplitude to the PZETL:AMPL value.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.17

Changes in firmware 1.17 are:

1. Revised error prompts. Error codes are returned immediately. Errors cleared after being prompted
2. General bug fixes.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.18

Changes in firmware 1.18 are:

1. Added constant move commands for the Thin Etalon.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.

Version 1.19

Changes in firmware 1.19 are:

1. Fixed Piezo Etalon output Signal was sometimes unreasonably high.

The firmware is based on DSP BIOS version 5.41.13, compiled with TI's toolchain version 4.3.9, using Code Composer Studio 3.3.