DDT

DDT User Manual

DUM

Version:	4
Date:	2021-05-17
Reference:	CGI-MAN-00026
Total Pages:	99

CGI Deutschland B.V. & Co. KG, Borgmannstrasse 2, 44894 Bochum, Germany
Space Division, Tel.: +49 234 9258-0, Web: https://www.de.cgi.com/de/space

CGI

Document Signature Table

	Name	Function	Signature	Date
Author(s)	Carsten Mannel		Not signed	2021-05-17
Checked	Angelika Stalitza	DDT Product Assurance Manager	Original Signed	2021-05-17
Approval	Jean-Christophe Berthon	DDT Project Manager	Original Signed	2021-05-17

Table 1: Document Signature Table

Document Change Record

Issue	Date	Author	Reference	Changed Pages/Paragraphs
1	2020-04-17	C. Mannel		Initial version
2	2020-06-05	C. Mannel		New version for Alpha Release 2.0
		C. Mannel	RIX-PSI-04 RIX-PSI-07	Section 3.2 and 3.4: Updated new deployment using the private.lua file.
			RIX-PSI-05	Section 3.2: The phrase was removed, since it is irrelevant for the installation process.
				Section 3.3.1.x: Added three command line arguments to the Ddt Publisher simulator to allow configuring the ring buffer size and the directory from which images are loaded and the port over which notifications are sent.
		M. Grimm		Section 3.3.1.1.2: Added get_statistics() function to Table 9.
				Section 3.3.1.1.2: Updated section with a description of the DdtStatistics.
				Section 3.3.1.3: Updated the commandline parameters for the publisher simulator (publishing frequency) and notification port.
				Section 3.3.1.4: Updated the commandline parameters for the subscriber simulator (reading frequency).
				Added Section 3.3.1.1.3 (Configuration).
		C. Mannel		Section 3.5: Added new section on log configuration.
		M. Grimm		Section 3.3.1.1.2: Updated Table 9: Renamed the third parameter <subscription_uri> to <remote_broker_uri> and updated the description.
				Section 3.4: Updated Table 18: Renamed <publishing port for local broker> to <publishing URI> and updated the description.
				Section 3.3.1.1.1: Updated the table: Added the parameter notification port to the RegisterPublisher() function and updated the description.
				Section 3.3.1.1.1: Updated the table: Added arguments to the WriteData() function.
				Section 3.3.1.1.3: Updated the section. The environment variable is not eeltddtpath but DDT_TRANSFER_CONFIG.
3	2020-12-04	C. Mannel		New version for Image Handling and Widget and Dialogs Milestone
				Section 3.2: Updated the installation instructions

Issue	Date	Author	Reference	Changed Pages/Paragraphs
				Section 3.3.1: Updated API function descriptions for the Data Transfer and commandline arguments.
				Section 3.3.2.1: Update due to changes in deployment.
				Section 3.3.2.1: Updated since more widgets are part of the delivery now.
				Section 3.3.2.1.1: Updated description of Image Widget.
				Section 3.3.2.1.2: Updated description of the Data Stream Widget
				Section 3.3.2.1.3: Updated description of the Flip Rotate Widget
				Section 3.3.2.1.4: Updated description of the Scale Button Widget
				Section 3.3.2.1.5: Updated description of the Panning Widget
				Section 3.3.2.1.6: Replace section with new section on the Image Scale Widget
				Section 3.3.2.1.7 to .10: Added section on new widgets
				Section 3.3.2.2: Added content to section for Dialogs
				Section 3.4: Updated command line arguments
				Section 3.5: Updated logger configuration
		C. Bortlisz		Section 3.3.3: Updated Image Handling documentation
		M. Pfeil		Section 3.3.4 Added Python bindings documentation
4	2021-05-10	C.Mannel		New version for Provisional Acceptance Data Package
		M. Grimm		Section 3.3.1.2: Added that the path element of the URI Is optional
				Section 3.3.1.2: Added that CTRL-C stops the Broker
				Section 3.3.1.3: Replaced --frequency with –interval
				Section 3.3.1.3: --checksum parameter defaults to 1 (true)
				Section 3.3.1.3: Added that the path element of the URI Is optional
				Section 3.3.1.3: Changed the description of pressing CTRL-C
				Section 3.3.1.4: Replaced --frequency with –interval and updated the description
				Section 3.3.1.4: Added that the path element of the URI Is optional
				Section 3.3.1.4: Changed the description of pressing CTRL-C
				Section 3.3.1.4: Added that --broker and --datastream are required
				Section 3.4: Replaced --frequency with –interval and updated the description for both ddtPublisherSimulator and ddtSubscriberSimulator (Table 32: Commands List)
				Section 3.3.1.1.2: Updated the DdtStatistics struct and its description
		C. Mannel		Section 3.3.2.1: Figure updated and paragraph on base layout class added
				Section 3.3.2.1.1: Table 12 updated with new properties; table 12 updated with new signals and slots
				Section 3.3.2.1.3: Figure 4 updated
				Section 3.3.2.1.4: Figure 5 updated; description of zoom in / zoom out buttons updated
				Section 3.3.2.1.5: Figure 6 updated; paragraph on compass widget added; table 19 updated
				Section 3.3.2.1.6: Table 20 updated (auto scale function)
				Section 3.3.2.1.10: Updated description of functionality; Figure 11 updated

Issue	Date	Author	Reference	Changed Pages/Paragraphs
				Section 3.3.2.2: Updated configuration items of context menu; Table 25 contains new IDs
				Section 3.3.2.2.1: Figure 13 updated; updated section on magnification functionality
				Section 3.3.2.2.2: Figure 14 updated
				Section 3.3.2.2.3: Updated figure 15
				Section 3.3.2.2.4 to 3.3.2.2.15: New sections
				Section 3.3.2.4: Updated Figure 28; Update commandline interface of DDT Viewer; added remote control interface description
				Section 3.4: Table 46 updated with new arguments for DDT Viewer
				Section 3.3.2.3: Section updated (new content)
		J.-C. Berthon		Section 3.2: Updated requirements, added section 3.2.1 for MAL 1.2.0 installation steps, added section 3.2.2.1 for Shiboken special instructions as it is not fully supported by current ELT Dev Env used for our baseline. Finally, quickly restructured section 3.2 to accommodate the newly added sections.

Table 2: Document Change Record

Distribution List

Name	No. Copies
CGI Archive	1

Table 3: Internal Distribution List

Company/Organisation	Name	No. Copies
ESO	Mario Kiekebusch	Electronic

Table 4: External Distribution List

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

This document provides the instruction of a user manual for the Data Display Tool (DDT).

The Data Display Tool is a collection of libraries and applications that shall be used in the scope of the ESO ELT for the transfer, display and manipulation of data coming from the telescope.

The DDT is grouped into the components Data Transfer, Data Visualisation, Image Handling and Python Components.

In this manual these components will be described from the user's point of view. It will be described how these components can be built, deployed, configured and used. It contains an overview on the various command-line arguments and configuration items.

The manual will also contain a list of all library functions and interfaces (API). For the visualisation components it will be described how these elements are integrated in tools like the Qt Creator / Designer and how the user can use them to build customized applications.

The document is divided into sections for each of the components describing each component in detail. All sections should be subject to review.

2 Documents

2.1 Applicable Documents

Acronym	Reference	Title	Version
AD-1	ESO-324872	Statement of Work for Data Display Tool	1
AD-2	ESO-285808	ELT ICS Framework - Data Display Requirements	2
AD-3	ESO-288608	Control GUI Developer Guidelines	1
AD-4	ESO-254539	E-ELT Programming Language Coding Standards	2
AD-5	ESO-213265	Document Requirement Definition	2

Table 5: Applicable Documents

2.2 Reference Documents

Acronym	Reference	Title	Version
DD	SSYD-DER-00006	Design Description	4
ELT-INST	ESO-287339	E-ELT Linux Installation Guide	4.14
CENTOS_INST	https://docs.centos.org/en-US/centos/install-guide/	Installation Guide	1.1

Table 6: Reference Documents

2.3 Definition of Terms and Acronyms

Term	Description
Data Topic	The Data Topic defines the semantics of the data being exchanged. The data can be a scientific image, a multi-dimensional array or a user defined format.
Data Sample	The Data Sample defines a set of data sent through the data stream. The actual data set will be complemented with additional information (meta-data) to enable the subscriber to process the data.
Data Stream	The Data Stream defines a connection between publisher and subscriber. Upon establishing such a connection, the subscriber will receive information describing the type of data sent by the publisher.
Data Channel	Connection between two Data Brokers used to transport the data between brokers.
DDT	Data Display Tool
CII	Core Integration Infrastructure
MAL	Middleware Application Layer
ELT	Extremely Large Telescope
FITS	Flexible Image Transport System
SHM	Shared Memory – memory that is shared between the Data Broker and the local Publisher or Subscribers.
GUI	Graphical User Interface
API	Application Programming Interface
URI	Uniform Resource Identifier
ICD	Interface Control Document
OS	Operating System

Term	Description
CPL	Common Pipeline Library
WCS	World Coordinate System
RA / DEC	Right Ascension / Declination
DS	Data Sample
RMS	Root Mean Square
HDU	Header Data Unit
RPC	Remote Procedure Call

Table 7: Terms for the Data Display Tool Framework

3 Data Display Tool

3.1 Purpose of the software

The Data Display Tool (DDT) will be used in the scope of the Extremely Large Telescope (ELT) of ESO. The software can be used to transport the data (images or other types of data) from the sources of data, the so-called Publishers, to the consumers of data, the so-called Subscribers.

The DDT software is split into four major components. First is the Data Transfer. It can be used to transfer data through a network of computers. The Data Transfer Components contain software libraries that can be used by Publisher and Subscriber applications.

Data is relayed from Publishers to Subscribers using so-called Data Brokers. Data Brokers will be used to transfer data from Publishers to Subscribers or to other Data Brokers on other hosts.

The Data Transfer components also offer functions that allow the monitoring of the quality of the data transfer.

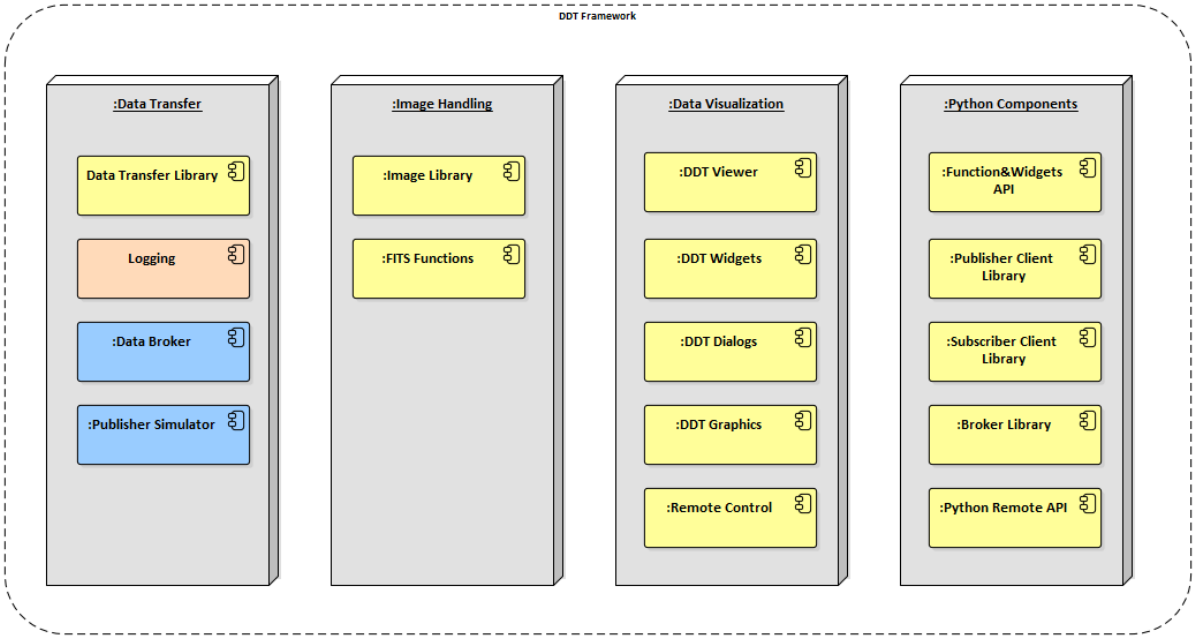


Figure 1: Data Display Tool Components

The other components of the DDT software are rather used for the display or manipulation of data that was transferred using the Data Transfer.

These components are the Data Visualisation and the Image Handling components. While the Data Visualisation is a library of GUI elements, so-called widgets, that can be used to build customer applications for accessing data, the Image Handling library offers a set of functions that can be used to manipulate (image) data.

Also included in the Data Visualisation component is an application, the DDT Standard Viewer, which is kind of a reference application that includes a set of DDT widgets.

The Python Components that are part of the DDT offer Python wrappers for the other DDT components that are all based on C++ code. The Python Components allow to integrate the libraries of the DDT software into Python applications.

3.2 Software Installation

The DDT software is built using the "`waf / wtools`" tool. CentOS 7.4 (1708) and the ESO dev environment version 2.2.5-3 is required, it is assumed the user have installed those prior to following this guide and as per instructions given by the respective projects (see [CENTOS-INST] and [ELT-INST]). On top the CII MAL needs to be upgraded separately to version 1.2.0. It is assumed that a version of MAL 1.2.0 is provided with the DDT installation.

3.2.1 Installing CII MAL 1.2.0 for ELT Dev Env 2.2.5-3

The steps are to remove the currently installed CII MAL version and to install the new one, then after a few files need to be adapted. It is assumed that the file `elt-mal-1.2.0.rpm` is available in the current working directory.

```
yum remove elt-mal
yum install ./elt-mal-1.2.0.rpm
```

Then the last step is to rename the LUA file in the directory `/eelt/System/modulefiles/mal/` from the old MAL version to the new MAL version:

```
mv /eelt/System/modulefiles/mal/1.1.0.lua /eelt/System/modulefiles/mal/1.2.0.lua
```

Finally, the `.version` file needs to be updated to reflect the newer MAL version. Use your favourite text editor to edit the file `/eelt/System/modulefiles/mal/.version` and change the line starting with "`set ModulesVersion`" to:

```
set ModulesVersion "1.2.0"
```

Then log out and log in again in your user account.

3.2.2 Building and Installing DDT

3.2.2.1 Installing Shiboken Support for ELT Dev Env

For the python bindings generated with shiboken two package-configuration files need to be added. The path to these files needs to be part of the `${PKG_CONFIG_PATH}`. So put them for example under `/eelt/System/pkgconfig/`. Paste the following content into the files:

`shiboken2.pc`

```
Name: shiboken2
Version: 5.14
Description: Shiboken2 library for python bindings.
prefix=/opt/anaconda3/lib/python3.7/site-packages
includedir=${prefix}/shiboken2_generator/include
libdir=${prefix}/shiboken2

Cflags: -I${includedir}
Libs: -L${libdir} -l:libshiboken2.abi3.so.5.14
```

`PySide2.pc`

```
Name: PySide2
Version: 5.14
Description: Qt for Python.
prefix=/opt/anaconda3/lib/python3.7/site-packages
```

```
includedir=${prefix}/PySide2/include
libdir=${prefix}/PySide2

Cflags: -I${includedir} -I${includedir}/QtCore -I${includedir}/QtGui -
I${includedir}/QtWidgets
Libs: -L${libdir} -l:libpyside2.abi3.so.5.14
```

3.2.2.2 Building and Installing DDT

To build the software the repository should be cloned into a subdirectory "`ddt`".

In the repository root a file "`private.lua`" will be included. This file needs to be copied to the directory `/home/<username>/modulefiles`. After copying the file, the user needs to logout and login again. The use of the lua-file will ensure that all environment variables are updated accordingly. In the `private.lua` file also the version number of the DDT software is set:

```
eeltddtpath = "/eelt/ddt/0.1"
```

Now in order to build the software from the console, in the repository root run the command

```
waf configure --mode=release build
```

It will build all libraries and applications into a "`build`"-subdirectory of the repository folder.

After this run as root-user the command:

```
waf install
```

It will deploy the software into the directory:

```
/eelt/ddt/0.1
```

Make sure the folder `/eelt/ddt/0.1/bin` has proper access rights for all users that will run the application after the installation.

The applications (like the Data Broker and the DDT Standard Viewer) are also part of the build-folder. They will be described in more detail in the following sections.

The configuration files required by the software will be stored in the location

```
/eelt/ddt/0.1/resource/config
```

This includes configuration for the Data Transfer and the DDT Standard Viewer as well as the logger configuration.

By default, log messages will be written to logfiles in the home folder of the current user. Default log level is `INFO`.

The logger configuration will be described in section 3.5 of this document. The rest of the configuration items will be described in the subsections of the corresponding software components.

3.3 Software components

3.3.1 Data Transfer

3.3.1.1 Publisher / Subscriber libraries

The Data Transfer component contains a dynamic library which is the DDT Data Transfer Library. It contains the classes required to create a Publisher or a Subscriber object that can be embedded in custom applications. They are also used in DDT simulator tools which can be used as Subscriber or Publisher.

The library is built as a single .so file: `libddt-transfer.so`

Due to the similarity of the features required by the Publisher and Subscriber library, these objects were placed in a single library.

3.3.1.1.1 Publisher library functions

When using the Data Transfer library to create a publisher, the following functions are available:

- **DdtDataTransferFactory**

Function	Arguments	Return type	Description
Create Publisher	DdtLogger* logger	static std::unique_ptr<DdtDataPublisher>	Creates an instance of the DdtDataPublisher. An instance of the DdtLogger object is given, if logging is required.

- **DdtDataPublisher**

Function	Arguments	Return type	Description
SetQoS	int latency (in ms) int deadline (in secs)	-	Sets the QoS parameters for the MAL connections.
SetBufferSize	int max_data_sample_size int number_of_samples	- .	Set the size hint for the data to be published. The size is made up of the maximum size in bytes required for one Data Sample and the number of samples that shall be stored in the ring buffer.
RegisterPublisher	string broker_uri, string data_stream_identifier, bool compute_checksum	int (-1 if registering fails)	Registers a Data Stream with the given Data Stream ID at the local broker with the given URI. A flag can be used to

Function	Arguments	Return type	Description
			switch the calculation of a checksum for the Data Samples that are transferred on or off. Returns an error code.
UnregisterPublisher	-	-	Unregisters the DDT Data Publisher from the local Data Broker.
WriteData	int32_t sample_id, ip::vector<uint8_t> datavec, ip::vector<uint8_t> metadata	-	Writes data to the shared memory using a Memory Accessor.
PublishData	-	-	Notifies the local Data Broker that it shall publish new data by sending the corresponding event.
set_topic_id	int topic_id	-	Sets the topic ID for the publisher.
get_connected_to_broker	-	Bool	Return true, if the publisher is connected to a Data Broker.

3.3.1.1.2 Subscriber library functions

When using the Data Transfer library to create a subscriber, the following functions are available:

- **DdtDataTransferFactory:**

Function	Arguments	Return type	Description
CreateSubscriber	DdtLogger* logger	static std::unique_ptr<DdtDataSubscriber>	Creates an instance of the DdtDataSubscriber. An instance of the DdtLogger object is given, if logging is required.

Table 8: Functions of DdtDataTransferFactory

- **DdtDataSubscriber:**

Function	Arguments	Return type	Description
SetQoS	int latency (in ms) int deadline (in secs)	-	Sets the QoS parameters for the MAL connections.
RegisterSubscriber	string broker_uri, string data_stream_identifier, string remote_broker_uri, int32_t reading_frequency (default is 10 ms)	int (-1 if registration fails)	Registers to a Data Stream with the given Data Stream ID at the local Data Broker with the given URI. For remote Subscribers the URI to the remote Broker needs to be specified. Returns an error code. Reading frequency (interval) can be used to simulate slow readers. The argument is optional. Default frequency will be 10 ms.

Function	Arguments	Return type	Description
UnregisterSubscriber	-	-	Unregisters the DDT Data Subscriber at the local Data Broker.
ReadData	-	DataSample* data_sample	Read data from the shared memory of the local Data Broker. Returns an error code.
StartNotificationSubscription	-	-	Starts MAL subscription for notifications in a separate thread.
StopNotificationSubscription	-	-	Stops the MAL subscription for notifications.
connect	const signal_t::slot_type &event_listener	boost::signals2 ::connection	Connects the event listener with the DataAvailableSignal from the memory accessor.
get_statistics	-	DdtStatistics	Returns the statistics computed by the Data Broker.

Table 9: Functions of DdtDataSubscriber

The function `get_statistics()` returns a struct of type `DdtStatistics` which has the following structure:

```
struct DdtStatistics {
  /**
   * Time of the last received data packet
   */
  std::chrono::system_clock::time_point last_received;
  /**
   * Time of the last received data packet as string
   */
  std::string last_received_str = "";
  /**
   * Number of connected subscribers
   */
  int32_t num_subscribers = 0;
  /**
   * Total number of received samples
   */
  uint64_t total_samples = 0;
  /**
   * Total number of received bytes
   */
  uint64_t total_bytes = 0;
  /**
   * Total latency of the data transfer [ms]
   */
  uint64_t total_latency = 0;
  /**
   * Queue capacity (number of elements in the ring buffer)
   */
  int32_t queue_capacity = 0;
  /**
   * Originating broker
   */
  std::string originating_broker = "";
};
```

This struct contains the raw values provided as simple counters. Counters are updated for each Data Sample received by a Data Broker over the network, and a timestamp is stored when the counters were updated. The data type of the timestamp is `std::chrono::system_clock::time_point`, but it is also provided as a `std::string` for convenience. With the help of the counters, clients can perform their own calculation of averages like the average sample frequency or the average latency. Furthermore, the queue capacity (which is the number of elements in the ring buffer) and the URI of the originating Data Broker are provided.

Data Brokers update the statistic counters whenever a new data notification is received from a Publisher. In the case a remote Subscriber is requesting the statistics, its Data Broker is updating the statistics whenever a Data Sample is received over the network from the remote Data Broker.

3.3.1.1.3 Configuration

It is possible to set some parameters of the DDT Data Transfer Library via a configuration file. This file is called `datatransfer.ini` and is copied to a config folder when the command `waf install` is executed (see section 3.2). The config folder is specified in the `DDT_TRANSFERCONFIG_PATH` environment variable which is set in the `private.lua` file. The file `datatransfer.ini` is structured as follows:

```
[datatransferlib]
max_age_data_sample=10000
reply_time=6
```

The first line specifies a name which simply groups the following lines into a section (`[datatransferlib]` in this case). This line should not be altered by the user.

The subsequent lines contain the parameters which can be configured. The parameters consist of a name and a value delimited by an equal sign. Currently the user can configure two parameters:

Parameter	Description
max_age_data_sample	Specifies the maximum age of a data sample in [ms]. This parameter is read from the DdtDataSubscriber and will drop a data sample if it is older than the age specified here. Data samples are dropped directly after they were read from the shared memory.
reply_time	Specifies a request timeout in [s]. This parameter is read from the DdtDataPublisher and sets the reply time of a MAL client instance. The reply time should only be increased if one expects a high response time from the Data Broker, which would be the case e.g. for transferring high resolution images.

Table 10: Datatransfer Library configuration file

3.3.1.2 Data Broker

The Data Broker is a command line application which is used to transfer the data from the Publisher to the Subscriber either on the same or on different hosts. When running the setup on different hosts, a single Data Broker needs to be started on each of the hosts.

The Data Broker can be started using the command line:

```
ddtBroker -U <Server URI of the broker> [--debug]
ddtBroker --uri <Server URI of the broker> [--debug]
```

An example for this would be:

```
ddtBroker --uri zpb.rr://*:5001/broker
```

Once the broker is running, local Publishers can connect to it. On the Subscriber side the Subscriber will connect to its local Data Broker and the Broker will then create the connection to the remote Broker running on the Publisher host.

The URI which needs to be specified is the URI on which local services (Publisher or Subscriber) can connect to the Data Broker. Note that the path element (`/broker` in the example above) is optional. If not specified by the user, this string is automatically appended.

The "`--debug`" flag can be used to temporarily increase the log level of the Data Broker to "DEBUG".

The Data Broker also supports the command line argument:

```
ddtBroker --help
```

Pressing `CTRL+C` stops the Data Broker. In the case there are Publishers and Subscribers registered to the Data Broker, they get unregistered (and notified). If the Data Broker is restarted, Publishers and Subscribers get a notification and register again automatically.

The Data Broker uses a configuration file which can be used to configure the timeout behaviour and the network port range of the Data Broker.

The file will be deployed to:

```
/eelt/ddt/0.1/resource/config/databroker.ini
```

The file contains the following settings in the section [databroker]:

Parameter	Description
shm_timeout	The shm_timeout (in seconds) specifies the time after which the Data Broker deletes an allocated shared memory after a publisher was unregistered and when there are no more subscribers registered. The shared memory will otherwise be re-used, if the Publisher re-registers within this timeout.
waiting_time	Time in ms that the MAL publishers may use for establishing the communication. If the time is exceeded the connection attempt fails.
min_port	Minimum port number for the port range the Data Broker may use. Ports will be automatically assigned for communication between Publishers / Subscribers and the Data Broker for non-data exchange (notifications).
max_port	Maximum port number for the port range the Data Broker may use. Ports will be automatically assigned for communication between Publishers / Subscribers and the Data Broker for non-data exchange (notifications).
reply_time	Time in seconds used to establish a connection to a MAL server. If the time is exceeded the connection attempt fails.
heartbeat_interval	Interval in seconds for the heartbeat which is used to monitor the status of the connection between Data Broker and Subscriber/Publisher. By setting the heartbeat interval to 0 the heartbeat is deactivated for all Subscribers/Publishers that connect to the Data Broker.

Parameter	Description
heartbeat_timeout	Timeout in seconds for the heartbeat. If no heartbeat signal was received for the configured time the MAL client (Publisher or Subscriber) is unregistered.

Table 11: Data Broker configuration file

3.3.1.3 Publisher Simulator

The Publisher Simulator is an example for a publishing application that uses the DDT Publisher Library.

For now, the Publisher Simulator will send the content of FITS files or simulation data. In a later implementation it can either be used to publish data from files or data from a binary dump of a Data Stream.

The Publisher Simulator is a command line tool. It can be started using the following arguments:

```
ddtPublisherSimulator --broker <local broker URI> --datastream <data stream ID>
--interval <publishing interval> --mode <simulation-mode> --image_folder <folder
to FITS images> --buffer_size <ring buffer elements> --checksum <checksum flag>
[--debug]

ddtPublisherSimulator -b <local broker URI> -s <data stream ID> -f <publishing
interval> -m <simulation-mode> -i <folder to FITS images> -u <ring buffer
elements> -c <checksum flag> [-d]
```

An example of the command looks like this:

```
ddtPublisherSimulator --broker zpb.rr://127.0.0.1:5001/broker/Broker1 --
datastream ds1 --interval 1000 --mode 5 --image_folder /data/fitsimages/rotate -
-buffer_size 10 --checksum 1
```

The arguments are used to specify the following items:

`--broker` Allows to configure the URI to the local broker (required)

`--datastream` Defines the Data Stream ID which is used for publishing data (required)

`--interval` Publishing interval in ms for publishing of data samples

`--buffer_size` Size of the ring buffer, defaults to 4 elements, if not specified

`--image_folder` Folder containing FITS images (should be of same type/size)

`--mode` Optional, defaults to 1. The following modes are supported for testing:

Mode 1: Transfer of FITS images from the folder `--image_folder`

Mode 2: Oscilloscope use-case: Transfer of single dimensional array

Mode 3: Multidimensional Array use-case: Transfer of multi-dim. array

Mode 4: Configurable Map test scenario

Mode 5: Chunked image test scenario

`--checksum`	Flag, to switch on/off the checksum calculation (defaults to 1, true)
`--debug`	Temporarily increases the log level to `DEBUG`
`--help`	Gives an overview of the above listed options

Note that the path element of the `--broker` parameter (`/broker/Broker1` in the example above) is optional. If not specified by the user, this string is automatically appended.

The different modes can be used to test different use-cases. In mode 1 it is important to select a folder for image files (default is the current directory). The modes 2 and 3 can only be used in combination with a Subscriber Simulator. The modes 4 and 5 can only be used with a DDT Standard Viewer. Mode 1 can be used with both the Subscriber Simulator and the viewer.

When running the Publisher Simulator in mode 4 the connected DDT Standard Viewer will make use of a so called “Configuration Map”. These maps are stored in FITS format in the directory specified in `$DDT_CONFIGURATIONMAP_PATH`.

The definition for which configuration map is loaded depends on the meta data of the image.

When running the Publisher Simulator in mode 5 it will generate chunks of images with the proper meta-data also generated. Each chunk of the image will contain in its meta-data the proper x-y coordinates where this chunk is placed into the full image. Also, the last chunk has the “final” flag set in its meta-data. The meta-data also contains a “complete” flag. This is used to determine between chunked images and images that are not chunked.

Pressing `CTRL+C` unregisters the Publisher Simulator from its Data Broker and stops the Simulator. If the connection to the Data Broker was lost, Publishers get a notification and automatically resume publishing as soon as the connection was re-established.

3.3.1.4 Subscriber Simulator

The Subscriber Simulator is similar to the Publisher Simulator. It can be used to demonstrate the implementation of the DDT Subscriber Library.

The Subscriber Simulator is a command line tool that can be started like this:

```
ddtSubscriberSimulator [--remote <remote broker URI>] --broker <local broker
URI> --datastream <data stream ID> --frequency <reading period in ms> --mode
<simulator mode> [--statistics <0|1>] [--debug]

ddtSubscriberSimulator [-r <remote broker URI>] -b <local broker URI> -s <data
stream ID> -f <reading period in ms> -m <simulator mode> [-a <0|1>] [-d]
```

An example of the command looks like this:

```
ddtSubscriberSimulator --broker zpb.rr://127.0.0.1:5003/broker/Broker1
--datastream stream2 --remote zpb.rr://172.110.6.100:5001/broker/Broker1
```

The arguments are used to specify the following items:

`--broker`	URI of the local broker (required)
`--datastream`	Data Stream ID for which to subscribe (required)
`--interval`	Reading interval in ms (can be used to simulate a slow reader, default 10)
`--remote`	URI of a remote broker, only used for multi-host scenarios
`--mode`	Simulation mode. Optional, defaults to 1. The following modes exist:
	Mode 1: Receiving of images in FITS format
	Mode 2: Oscilloscope use-case: Receiving of single dimensional array
	Mode 3: Multidimensional Array use-case: Receiving of multi-dim. array
`--statistics`	Flag to switch off or on the calculation of transfer statistics (default off)
`--debug`	Temporarily increase the log level to `DEBUG`
`--help`	Show the available options

When the Subscriber is using data from the local host, the `--remote` argument is not required, it is optional in this case. When the Publisher is on a remote host, the `--remote` argument needs to be the URI of the remote Broker.

Note that the path element of the `--broker` parameter (`/broker/Broker1` in the example above) is optional. If not specified by the user, this string is automatically appended. This also applies for the `--remote` parameter.

Pressing `CTRL+C` unregisters the Subscriber Simulator from its local Data Broker and stops the Simulator. If the Data Broker or the Publisher was stopped, Subscribers get a notification and automatically subscribe again as soon as the connection is re-established.

Another example for a Subscriber application is the DDT Standard Viewer, which will be described in the following section.

3.3.2 Data Visualisation

3.3.2.1 DDT Widgets

The DDT Widgets library contains several widgets that are implemented as Qt Designer Plugins which can be used in the Qt Creator/Designer to build custom GUI applications. The widget library is made up of a shared library library (`libddt-widgets.so`) that contains the widget code and the plugins module (`libddt-plugins.so`) which is needed by the QT Creator/Designer.

All supported widgets are contained in a single shared library.

The central widget of the widget library is the Image Widget. The Image Widget can be used to display data either loaded from a FITS file or received via the Data Transfer.

Further auxiliary widgets are available in the library, but all of them are basically connected to one instance of the Image Widget.

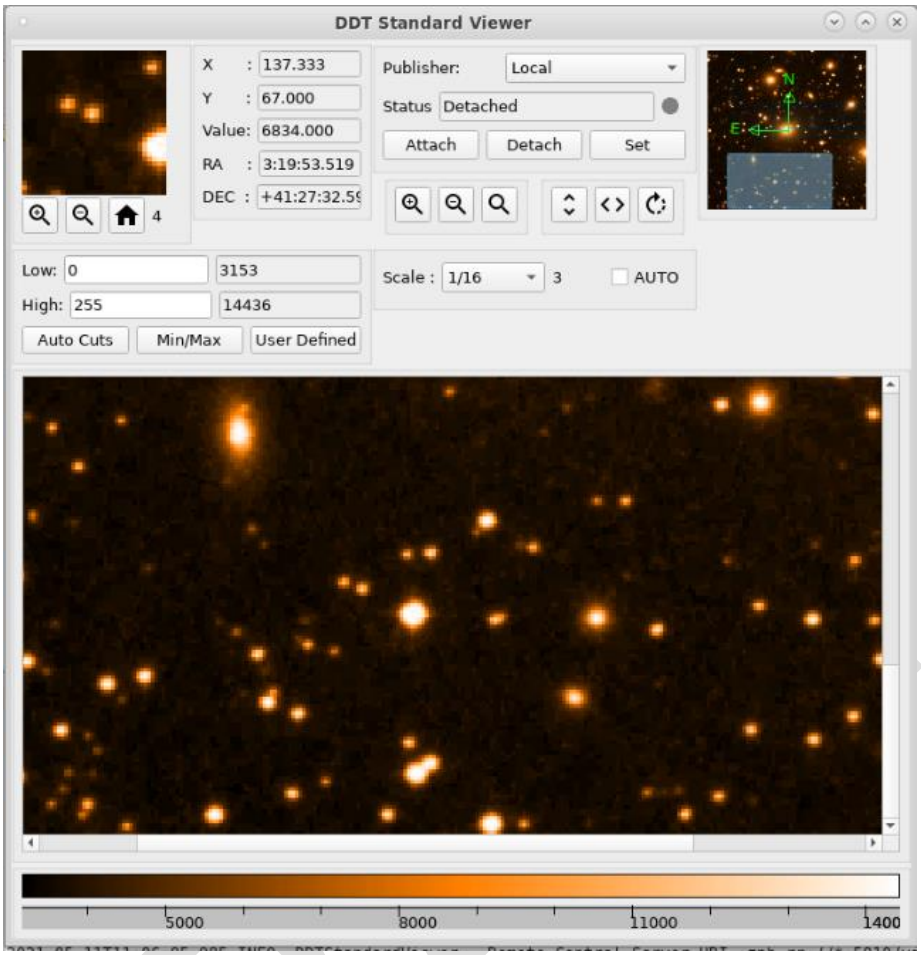


Figure 2: DDT Standard Viewer with DDT Widgets

The DDT Widgets all implement the interface of the `QDesignerCustomWidgetInterface`.

All widgets basically offer a "`CreateWidget`" method that will return a pointer to a `QWidget` object that can be used to access the object.

A code example for such a `CreateWidget` method looks like this:

```
void DdtPanningWidget::CreateWidget() {

  QHBoxLayout* layout = new QHBoxLayout;

  preview_image_label = new QLabel();
  preview_image_label->setAlignment(Qt::AlignCenter);
  layout->addWidget(preview_image_label);

  setLayout(DdtWidget::addParentLayout(layout));

  // Connect actions
  preview_image_label->setMouseTracking(true);
  preview_image_label->installEventFilter(this);
}
```

All widgets are using a base layout which is added by using the “addParentLayout(QLayout* child_layout)” method call. The base layout for example will create a box surrounding the widget component.

3.3.2.1.1 Image Widget

The Image Widget is used to display image data that either comes from a file or from a Data Publisher. In the current version the Image Widget can be used to display images in JPEG and FITS format.

Some properties of the Image Widget can be set as properties in the Qt Designer. These are:

Property	Default	Description
UseOpenGL	false	This flag allows enabling or disabling of OpenGL for the image display.
UseAntialiasing	false	This flag allows enabling or disabling of Antialiasing for the image display.
AutoScale	false	When set to true images will automatically be scaled to match the size of the Image Widget when loaded.
DefaultScale	1/2	This is the default scale which is selected when pressing the “Default scale” button in the related auxiliary widgets or that is used when loading a new image (and the auto scale flag is set to false).
ScaleFactorList	1/16,1/12,1/8,1/6,1/4, 1/2,1,2,4,6,8,12,16,20	The list of scale values that can be used by the various widgets that control the image scale. List of comma separated values.
DefaultColourmap	Real	Default colourmap that should be loaded (if not available the images will be using black/white colours).
ListContextMenu	-	A list of Dialog ID can be given (see Dialog section). The dialogs will be added to the context menu of the image widget. When the list is left empty, all supported dialogs will be offered in the context menu.
DefaultBiasImage	/data/fitsimages/default_bias_image.fits	Path to the default bias image, which will automatically be applied to images. If the image or the path do not exist, then no bias image is applied by default.
ShowScrollbars	true	Flag indicating, if scrollbars should be shown, when the image size exceeds the size of the Image Widget.

Table 12: Properties of the Image Widget

The Image Widget will be connected to the related auxiliary widgets using a list of signals and slots. For the Image Widget in the current version these are:

Widget Name	External Signal / Slot	Description (if required)
ddtImageWidget		
	Signals to the Colourmap Widget	
	Signal: SetCurrentColourmap(QVector<QRgb> colourtable)	Signal that sends the currently loaded colourtable to the colourbar display.
	Signal: UpdateColourbarAxis(double min, double max, int scaling_function)	Update the colourmap axis labels with the given min, max values and the selected scaling function.
	Signals to the Cursor Information Widget	
	Signal: CursorInfo(double x, double y, double pixelvalue, QString ra, QString decl)	Contains information on the coordinates of the current mouse pointer location.

Widget Name	External Signal / Slot	Description (if required)
	Slots for the Cursor Information Widget	
	Slot: CursorPosition(double x, double y, bool mouse_clicked)	Reacts to mouse movement or mouse clicks on the image.
	Signals to the Cut Values Widget	
	Signal: CurrentCutValues(double cut_min, double cut_max, ddt::ImageHandling::CutLevelType cut_type)	Send the current cut values for the current image.
	Slots for the Cut Value Widget	
	Slot: SetCutValue(double min, double max)	Sets the current cut values.
	Slot: SetAutoCuts()	Active auto cut levels.
	Slot: SetMinMaxCuts()	Active min-max-cut levels.
	Signals to the Data Stream Widget	
	Signal: CurrentConnectStatus(QString data_stream_id, ConnectionStatus status)	Signal that gives the current connection status for the Subscriber. It contains the Datastream ID and the current status.
	Signal: NewBoostDataEvent()	When receiving a boost signal from the Publisher Library this is forwared to the QT signal, so the widget can react on new data.
	Slots for the Data Stream Widget	
	Slot: AttachDataStream(QString data_stream_id)	Attach to a data stream.
	Slot: DetachDataStream(QString data_stream_id)	Detach from a data stream.
	Slot: HandleNewDataEvent(DataPacket data)	React on data received by the Subscriber Library.
	Slot: AttachDataFile(QString filename)	Attach data from a file to the image widget.
	Slot: AttachImageExtensionAsOne(QString filename)	Attach data from a FITS file with multiple extensions to load all data into a single image.
	Signals to the Flip Rotate Widget	
	Signal: UpdateFlipStatus(bool vertical, bool horizontal)	Send the current status to the widget (e.g. used when opening a dialog with flip / rotate functionality in parallel).
	Slots for the Flip Rotate Widget	
	Slot: FlipImage(bool vertical, bool horizontal)	Change the flip status of the image.
	Slot: RotateImage(int rotation_angle)	Rotate the image by the given angle.
	Signals to the Image Scale Widget	
	Signal: ScaleFactorListChanged(QList<QString> list)	The list of scale factors which can be configured as a property to the Image

Widget Name	External Signal / Slot	Description (if required)
		Widget will be send to the Image Scale Widget
	Signal: UpdateScaleFator(QString new_scale_factor)	The current scale factor was changed. Informs the widget of the update.
	Signal: UpdateAutoScale(bool new_state)	The auto scale flag was modified. Informs the widget of the current state.
	Slots for the Image Scale Widget	
	Slot: SelectNewScale(QString scale)	Sets the scale to the selected value.
	Slot: ToggleAutoScaleState()	Toggle the auto scale state for the image widget.
	Signals to the Magnification Widget	
	Signal: MagnifiedImage(QImage magnified_image)	Send the magnified part of the image at the current cursor position.
	Slots for the Magnification Widget	
	Slot: SetMagnificationFactor(QString magnification_factor)	Called when the selected magnification factor of the widget was changed.
	Signals to the Panning Widget	
	Signal: UpdatedImage(QImage*, QTransform&, bool show_axes, double rotation)	Informs the Panning Widget that the image in the Image Widget was updated. Also contains information needed to draw a compass in the widget.
	Signal: ImageWidgetViewChanged(QRect visible_image_rect, int current_image_width, int current_image_height)	The image was changed due to moving the scrollbars. Attached widgets may updated their view.
	Slots for the Pan Widget	
	Slot: UpdatePosition(double scroll_x, double scroll_y)	Update the position of the image once the position in the pan widget was changed.
	Slots for the Scale Buttons Widget	
	Slot: IncrementScale()	Increment the scale factor for the image.
	Slot: DecrementScale()	Decrement the scale factor for the image.
	Slot: SetToDefaultScale()	Sets the scale factor for the image to the default scale.
	Slot: SelectNewScale(QString next_scale)	Called when the user selects a new scale factor.
	Slot: ScaleFactorForNewImage()	Call when a new scale factor is set, especially when using the auto-scale
	Slot: QString FindAutoScale()	Called when loading a new image using the auto-scale function. Will return the best matching scale factor to match the image into the Image Widget.
	Signals to Dialogs	

Widget Name	External Signal / Slot	Description (if required)
	Signal: SetChangedDialogParameter(QString param_id, QVariant parameter)	A parameter in a connected dialog needs to be updated. The parameter is determined by the parameter ID and the value is stored in a QVariant.
	Slots for the dialogs	
	Slot: DialogParameterChanged(QString dialog_id, QString parameter_id, QVariant parameter)	Slot that reacts to changes in a dialog. The arguments are the ID of the dialog, the ID pf the parameter and the value of the parameter.
	Internal Signals / Slots	
	Signal: ContextMenuCommandSelected(QString menu_entry)	Signal when an entry in the context menu was selected.

Table 13: Signals and Slots of the Image Widget

3.3.2.1.2 Data Stream Widget

The Data Stream Widget can be used to connect to a data stream and monitor the status of the connection. The Data Stream can be selected by a reference name. The name can be selected from a combo box next to the label “Publisher:”.

The content of the combo box is read from a configuration file. The file needs to be placed in the directory `$DDT_TRANSFERCONFIG_PATH` using the file name: “publisher_uris.ini”.

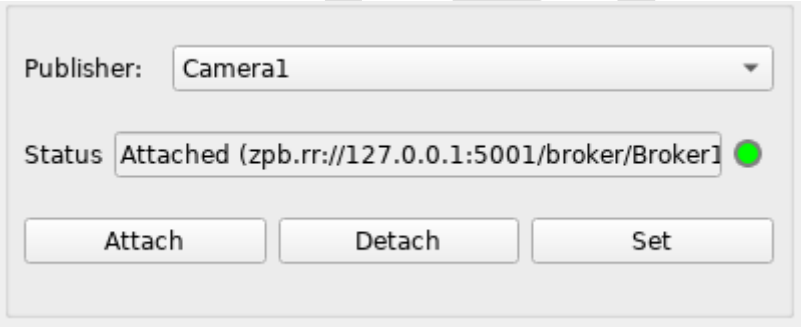


Figure 3: Data Stream Widget

The file should contain the list of known publishers in the format:

```
<Name>,<Local Broker URI>,<Datastream ID>[,<Remote Broker URI>]
<Name2>,<Local Broker URI>,<Datastream ID>[,<Remote Broker URI>]
…
```

- **Attach:**

When pressing the Attach button, the related Image Widget will be connected to the specified data stream. Alternatively, the command line option “`--datastream`” can be used to directly connect to a Data Stream. This entry will be added using the name “Commandline Publisher”.

- **Detach:**

When pressing the Detach button or closing the application the stream is disconnected again. If the data stream cannot be found or the connection fails, the DDT Viewer will report this in the log output.

The current status of the connection will be shown by a small LED icon. Green indicates an established connection, gray a disconnected link. When data is being received the light will flicker between green and gray.

- **Set:**

The Set button allows the user to change the name of the currently selected publisher (temporarily).

The Data Stream Widget can also be added to a viewer using the Qt Designer. Table 14: Properties of the Data Stream Widget

The Data Stream Widget can be connected to an Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtDataStreamWidget	Signal: AttachDataStream(string stream_id)	Stream was attached.
	Signal: DetachDataStream(string stream_id)	Stream was detached.
	Slot: AttachDataStream()	Called when stream was attached.
	Slot: DetachDataStream()	Called when a stream was detached.
	Slot: SetDataStream(QString stream)	Called at startup to set the startup data stream.
	Slot: CurrentStatus(QString stream_id, ConnectionStatus status)	Called when the connection status changes.
	Slot: FlickerStatus()	Used to show activity on the status when data is received.

Table 15: Signals and Slots of the Data Stream Widget

3.3.2.1.3 Flip / Rotate Widget

The Flip / Rotate Widget can be used to flip or rotate the image in the related Image Widget.

The widget offers three buttons – flip vertical, flip horizontal and rotate.

Figure 4: Flip / Rotate Widget

The Flip / Rotate Widget can be added using the Qt Designer. It has three properties for its configuration:

Property	Default	Description
FlipHorizontal	false	Image shall be flipped horizontally at startup (currently not supported).
FlipVertical	false	Image shall be flipped vertically at startup (currently not supported).
RotateClockwise	true	Flag to determine whether rotate will be clockwise or anti-clockwise.

Table 16: Properties of the Flip / Rotate Widget

The Flip Rotate Widget can be connected to a related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtFlipRotateWidget	Signal: FlipImage(bool vertical_axis, bool horizontal_axis)	Change flip state.
	Signal: RotateImage(int angle)	Rotate Image.
	Slot: FlipVertical()	Received to update the status of the flip buttons.
	Slot: FlipHorizontal()	Received to update the status of the flip buttons.
	Slot: Rotate()	Received when rotate button is pressed.

Table 17: Signals and slots of the Flip / Rotate Widget

3.3.2.1.4 Scale Buttons Widget

The Scale Buttons Widget can be used to zoom in and out of the image in the related image widget. In the current version the widget has no properties. The default scale and the possible scale factors are already defined in the Image Widget and will be handled there. The widget offers three buttons: Zoom in, Zoom out, Default scale.

Figure 5: Scale Buttons Widget

In the ‘Zoom in’ and ‘Zoom out’ button will change the current scale used for the image. Both buttons increase rsp. decrease the current zoom factor by 1 for values larger or equal to 1 and increase or decrease the denominator of the scale for values less then 1. So the scale values are changing like 1, 2, 3, 4, 5, … or 1, 1/2, 1/3, 1/4, … etc. Maximum scale factor will be 20, minimum scale factor 1/20.

The same functionality can be triggered by using the mouse scroll wheel while the mouse pointer is inside the Image Widget.

The ‘Default scale’ button will set the scale to the default scale configured in the related property of the Image Widget. Default here is “1/2”.

The current version of the Scale Buttons Widget can be connected to the related Image Widget using the following signals and slots:

Widget Name	External Signal / Slot	Description (if required)
ddtScaleButtonsWidget	Signal: IncrementScale()	Moves to the next scale value in the configured list.
	Signal: DecrementScale()	Moves to the previous scale value in the configured list.
	Signal: SetToDefaultScale()	Sets the scale value to the default scale which is configured in the properties of the Image Widget.

Table 18: Signals and slots of the Scale Buttons Widget

3.3.2.1.5 Panning Widget

The Panning Widget allows to navigate the visible portion of the image in the related Image Widget. In the current version the widget has no properties.

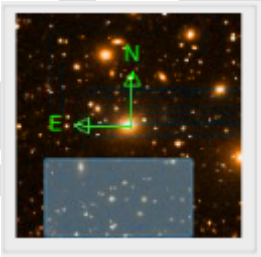


Figure 6: Panning Widget

The widget can simply be operated using the mouse. By dragging the shown rectangle in the preview image, the visible part of the image in the Image Widget is adjusted accordingly.

The widget uses a Qt Property “ShowAxes”. When the flag is set to true, a compass will be plotted into the pan preview window showing the north and east direction in the image. Currently the flag will be automatically set, when the image that was loaded contains the WCS coordinate information which are needed to determine the north direction for the image.

When the loaded image contains proper WCS coordinate information the pan widget will automatically draw a compass into the image. The rotation angle needs to be provided from the class sending the related signal. The compass can be switch on and off by using the right mouse button while clicking into the pan widget.

The Panning Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtPanWidget	Signal: UpdatePosition(int scroll_x, int scroll_y)	Pan movement in widget.
	Slot: SetImage(QImage* image, QTransform& transform, bool show, double rotation)	Image needs to be updated. Also contains formation, if the compass can be drawn and the required rotation angle.
	Slot: ImageWidgetViewChanged(QRect visible_image_rect, int current_width, int current_height)	View in image widget changed.

Table 19: Signals and slots of the Panning Widget

3.3.2.1.6 Image Scale Widget

The Image Scale Widget allows the user to directly select the current scale factor which shall be used for the Image Widget connected.

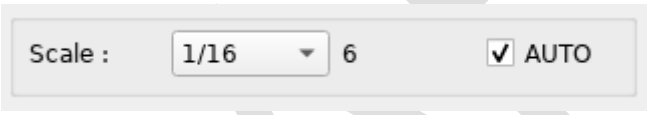


Figure 7: Image Scale Widget

In the widget a combo box offers a selection of scale factors the user can directly select so that they are applied to the image. The list of possible scale factors can be configured as a property in the Image Widget.

Next to the combo box the current scale factor used for the Image Widget is being displayed.

A checkbox offers the option to switch on / off the auto scale mode. When the auto scale mode is active, new images that are loaded from disk or received from a data stream are automatically re-scaled so the full image becomes visible (if possible) in the display. The auto scale function will use a value of the configured list of scale factors.

The Image Scale Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtImageScaleWidget	Signal: IncrementScale()	Signals the Image Widget to move one scale factor up in the list of scale factors.
	Signal: DecrementScale()	Signal the Image Widget to move one scale factor down in the list of scale factors.
	Signal: SetToDefaultScale()	Selects the default scale factor.
	Signal: SelectScale(QString scale)	Change the scale factor to the select value.

Widget Name	External Signal / Slot	Description (if required)
	Signal: SetAutoScale(bool new_auto_scale_state)	Toggles the auto scale selection.
	Slot: UpdateScaleLabel(QString new_scale_factor)	Updates the current scale factor display.
	Slot: NewScaleFactors(QList<QString> newScaleFactorList)	Populates the list of scale factors in the combo box.
	Slot: NewAutoScaleState(bool)	Slot which is called when the auto-scale state is modified.

Table 20: Signals and Slots of the Image Scale Widget

3.3.2.1.7 Colourmap Widget

The Colourmap Widget will give a graphical representation of the currently selected colourmap. The colourmap can be selected using the Colourmap Dialog.

Depending on the current cut values which are used for the display of the image, a scale will be automatically fitted to the colour bar. The axis depends on the current scaling function that was selected in the Colourmap Dialog (linear, logarithmic or square root).

A default colourmap can be configured in the properties of the Image Widget.

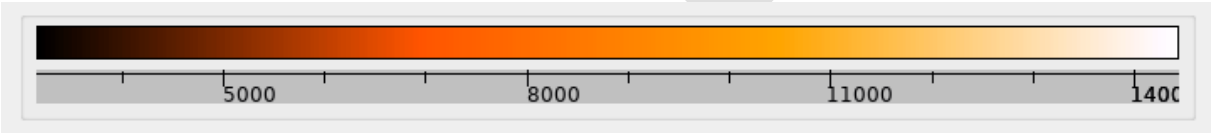


Figure 8: Colourmap Widget

The Colourmap Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtColourmapWidget	Slot: SetCurrentColourmap(QVector<QRgb> colourmap)	Receives the current colour map from the Image Widget.
	Slot: UpdateColourbarAxis(double min_value, double max_value, int scaling_function)	Receives minimum and maximum and the scaling function used to update the axis labels.

Table 21: Signals and Slots of the Colourmap Widget

3.3.2.1.8 Cursor Information Widget

The Cursor Information Widget displays information on the image point currently under the mouse pointer while the user moves the mouse pointer through the related Image Widget.

What information is displayed can be configured by the two properties “show_XY” and “show_RADEC” of the widget. By setting these flags it is possible to switch between the display using X, Y coordinates or WCS coordinates or both. A third property (“xyDigits”) can be used to configure the number of digits after the decimal point.

The cursor information contains the X and Y-coordinates of the original image (independent of rotation or flipping) and the original pixel value at that location.

Furthermore, when the image contains the related information to calculate the WCS coordinates the right ascension and declination for the image point is given.

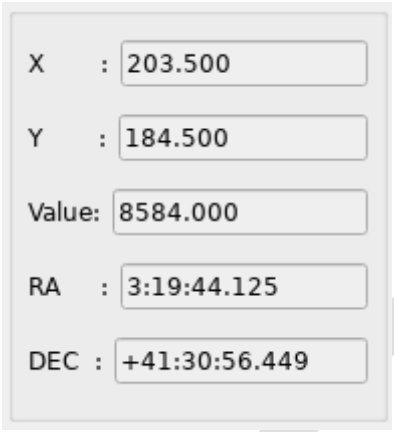


Figure 9: Cursor Information Widget

The Cursor Information Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtCursorInfoWidget	Slot: CursorInfo(double x, double y, double pixelvalue, QString ra, QString decl)	Slot that is called when updated information is available.

Table 22: Signals and Slots for the Cursor Information Widget

3.3.2.1.9 Cut Values Widget

The Cut Values Widget allows the user to specify the lower and upper limits used to display the image. The widget offers three options to specify the limit values:

- Auto Cuts: The lower and upper limit is calculated using a median filter on the image
- Min/Max: The lower and upper limit are the minimum and maximum pixel value
- User Defined: The user can specify the limits manually

For the user defined case two default values are shown (here 0 and 255). These default values can be set using two Qt Properties of the widget: “`default_low`” and “`default_high`”.

When the user enters new min/max values, it is possible to just press the `RETURN` key after inserting a new number or press the Min/Max button.

The values displayed in the right fields are the currently used values.

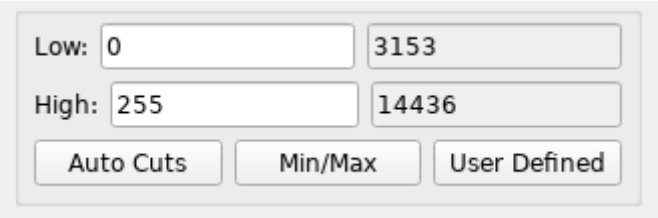


Figure 10: Cut Values Widget

The Cut Values Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtCutValuesWidgets	Signal: SetCutValues(double low, double high)	Signals the Image Widget that new cut values should be applied to the image.
	Signal: SetAutoCuts()	Signals that the Image Widget should calculate the auto cut values and apply them.
	Signal: SetMinMax()	Signals the Image Widget to calculate the minimum and maximum pixel value and apply them.
	Slot: CurrentCutValues(double min, double max, ddt::ImageHandling::CutLevelType)	Slot that can be used to update the current cut values and the method which is used for the calculation.

Table 23: Signals and Slots for the Cut Values Widget

3.3.2.1.10 Magnification Widget

The Magnification Widget shows an enlarged part of the image around the current mouse pointer position.

The widget can be configured using the following Qt Properties:

- `default_magnification_factors` Comma separated list of scale factors
- `region_size` Not yet implemented

The list of default magnification factors defaults to: 1, 2, 4, 6, 8, 10, 12, 16, 20

The widget allows the user to select the current magnification factor using the three buttons which will increase or decrease the magnification factor set the magnification factor to 1.

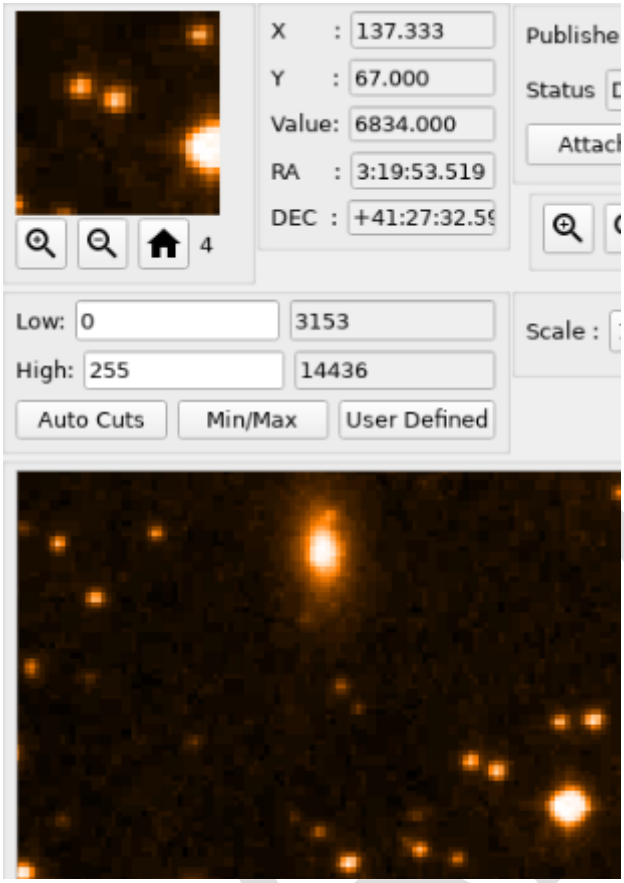


Figure 11: Magnification Widget

The Magnification Widget can be connected to the related Image Widget using the following signals and slots in the current version:

Widget Name	External Signal / Slot	Description (if required)
ddtMagnificationWidget	Signal: SetMagnificationFactor(QString factor)	Send when the magnification factor for the widget was changed.
	Slot: MagnifiedImage(QImage)	Receives the maginified image

Table 24: Signals and Slots of the Magnification Widget

3.3.2.2 DDT Dialogs

The DDT Dialogs library contains a number of dialogs which shall be accessible via a context menu of the Image Widget.

All dialogs can be created using a Dialog Factory class. Dialogs are created based on the Dialog ID. The same ID can be used to add dialogs to the context menu of the Image Widget.

The list of active dialogs is stored in the Qt Property "`ListContextMenu`" of the Image Widget. The list contains the comma-separated dialog IDs. All entries of the list will be selected for the context menu of the Image Widget. When the property is left empty by default all menu items will be displayed.

The dialogs will then be displayed in the context menu of the Image Widget:

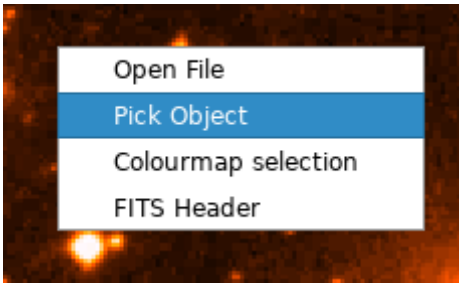


Figure 12: Context menu of the Image Widget

The list of available dialog ID is as follows:

Dialog ID	DDT Dialog
ddtColourmap	Selection of colourmap and scaling
ddtPickObject	Pick object dialog
ddtBias	Bias image dialog
ddtTabularRegion	Tabluar region dialog
ddtStatistics	Statistics dialog
ddtHDU	HDU display dialog
ddtFITSHeader	FITS header dialog
ddtFITSTable	FITS Table dialog
ddtReferenceLine	Reference line dialog
ddtPVCM	PVCM dialog
ddtGraphicalElements	Dialog for graphical elements
ddtOffset	Dialog for offset measurement
ddtSlit	Slit dialog
ddtCutValues	Cut value dialog
ddtDataStream	Data Stream dialog
ddtScaleRotateCut	Dialog for cut value, scale factor and flip / rotate
ddtFileOpen	This is basically not a DDT Dialog, but it allows access to a file open dialog
ddtFileOpenExt	This is also not a DDT Dialog, but it allows to select a file extension when loading FITS files with extensions
ddtFileSave	Can be used to save the current content as a file.
ddtMarkPositions	Also not a dialog, but allows the user to select the function to mark positions in the image using the mouse.
ddtEndMarkPosition	See above. This entry allows the user to end the selection of image positions.
ddtSeparator	Separator line for the context meu

Table 25: List of dialog IDs

All dialogs are created using a Factory class `DdtDialogFactory`. The main method of the factory class is the method to create a dialog:

```
static DdtDialog* createDialog(QString dialog_id)
```

All DDT Dialog are then subclassed from the common base class “DdtDialog”.

This class has the pure virtual method:

```
virtual void CreateDialog() = 0;
```

which is used to setup the dialog GUI elements.

Since all dialogs should be able to support the basic buttons “Confirm”, “Cancel” and “Quit” a method

```
virtual void AddDefaultButtonsToLayout(QBoxLayout* layout, bool
show_confirm_button, bool show_quit_button, bool show_cancel_button);
```

Here three flags can be used to either add or not add the required default buttons.

The buttons have the function:

Quit - Quits the dialog and returns an empty string

Cancel - Cancels the current operation

Confirm - The dialog is closed and the collected data of the dialog is returned

Furthermore, the dialogs have a virtual method:

```
virtual void SetInitialParameter(QString parameter_id, QVariant parameter) = 0;
```

This can be called to setup initial values in the dialog, when it is created. Each parameter will use a parameter ID to be identified. Initial values can be e.g. scale factors, images, image points etc.

The base dialog class also has a set of signals and slots:

Class Name	External Signal / Slot	Description (if required)
DdtDialog	Signal: ParameterChanged(QString dialog_id, QString parameter_id, QVariant parameter)	The signal is send, when a parameter of the dialog is changed that is required by the image handling backend. Parameters are again identified using a parameter ID.
	Slot: SetChangedParameter(QString param_id, QVariant parameter)	The slot is called, when the backend functions change some parameters used by the dialog. This could be e.g. the results of a calculation which was triggered by the dialog.

Table 26: Signals and Slots of the dialog base class

In the following sub sections the existing dialogs are described.

3.3.2.2.1 Pick Object Dialog

The Pick Object dialog can be used to calculate some statistical data for objects or points in the image. Therefore, the dialog offers two modes. One is the Object mode and one is the Cursor mode.

In the dialog these two modes can be selected using one of two radio buttons “Object” or “Cursor”.

Once a mode is selected the user needs to click on the “Pick” button. Then when clicking on any pixel in the image inside the Image Widget the following is shown.

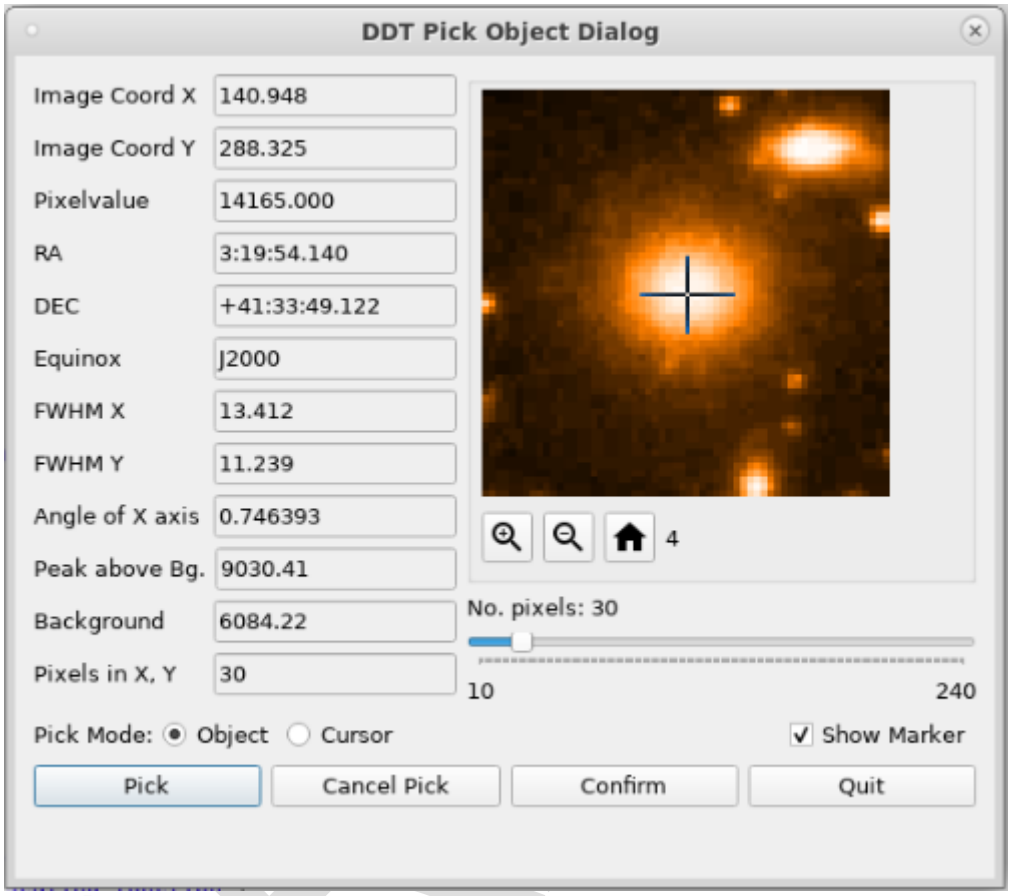


Figure 13: Pick Object Dialog

Mode	Clicked on	Result
Cursor Mode	Any pixel in the main image	The dialog will display the X and Y coordinate of the pixel (from the original image) plus the pixel value at that location. When the image in the display contains the necessary meta-data also the WCS coordinates (RA and DEC) will be displayed. All other values are set to 0.
Object Mode	Click on any pixel belonging to a star (or circular object)	The centre of the object is automatically located and the X, Y coordinates displayed in the dialog give the X, Y coordinate of the objects centre (not the coordinate the user clicked at). All other values like the Equinox, the FWHM in x and y direction, the angle of X axis, the peak above background and the background are calculated and displayed.
	Click outside a star	In this case again only the pixel coordinates and the pixel value are displayed.

Table 27: Pick modes in the Pick Object Dialog

The buttons that the dialog offers have the following functions:

Button	Function
Pick	Activates the Pick mode. The user can now click on pixels in the Image Widget. Depending on the selected pick mode the related information is being displayed whenever the user clicks in the image.
Cancel Pick	Only enabled, while the Pick mode is active. It cancels the pick mode. The last values are kept in the dialogs display. When the user now clicks in the Image Widget, the values are not modified. Also the Magnification Widget gets frozen. (The magnficication widget is not yet available in the current version.)
Confirm	Closes the dialog and returns the last results calculated as a string in the format: X Y PIXELVALUE RA DEC EQUINOX FWHMX FWHMY ANGLE_X PEAK_AB_BG BACKGR PIXELS_IN_X_Y
Quit	Closes the dialog and returns an empty string.

Table 28: Buttons of the Pick Object Dialog

The slider control in the Pick Object Dialog allows to select the size of the rectangle (or rather square) for which the calculation is done. The minimum value is fix at 10. The upper value will be dynamically set depending on the current magnification factor.

The “Show Marker” checkbox can be checked to display a cross on the selected object in the magnification view. The cross will be using the rotation angle and the FWHM values for x and y axis for its dimensions. It will mark the centre of the selected object.

The current version does not yet implement the entry of samples (currently only 1 sample is used independent of the selection made).

The Pick Object Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_PICK_MODE	Informs the Image Widget of the selected pick mode (Object or Cursor, active or inactive).
DDT_DIALOG_PARAM_PICK_RECTANGLESIZE	Signal a change of the number of pixels in x-y-direction used for the calculation (No. pixels)
DDT_DIALOG_PARAM_PICK_RETURN_VALUES	Used to return the calculated values as described in Table 28
DDT_DIALOG_PARAM_PICK_CANCELLED	Informs the Image Widget that the pick operation was cancelled.
DDT_DIALOG_PARAM_PICK_ACTIVATED	Informs the Image Widget that the pick operation was actived.
DDT_DIALOG_PARAM_PICK_MAGNIFIED_IMAGE	TBD
DDT_DIALOG_PARAM_PICK_MAGNIFY_FACTOR	TBD
DDT_DIALOG_PARAM_PICK_NUMBER_SAMPLES	TBD

Table 29: Parameters of the Pick Object Dialog

3.3.2.2.2 Colourmap Dialog

The Colourmap dialog can be used to select the false colouration map that should be used for the image colouring and the scaling function to use.

Colourmaps are read from the directory `$DDT_COLORMAP_PATH`. Colourmaps are files in ASCII format containing 256 lines of R-G-B values given in the range of 0.0 to 1.0.

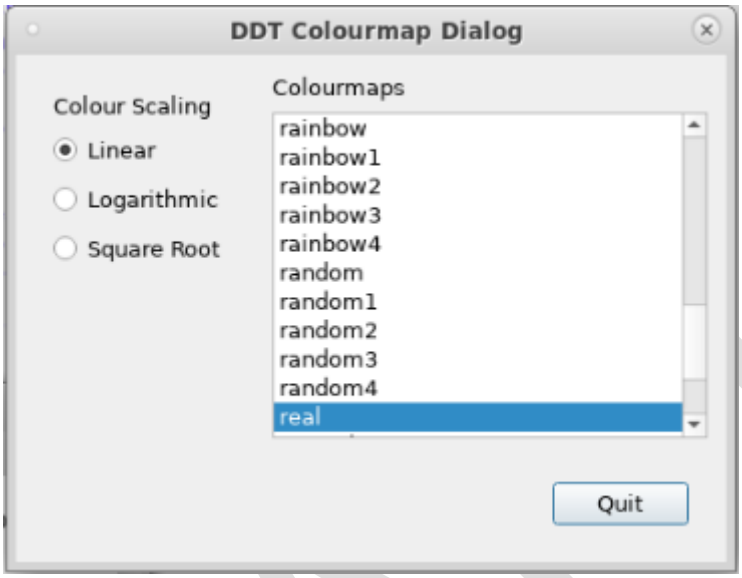


Figure 14: Colourmap Dialog

Colourmap files must use the extension ".lut" and the typical structure of the file is like this:

```
0.95686 0.58431 0.85490
0.95686 0.58824 0.85490
0.96078 0.59216 0.85490
0.96078 0.59608 0.85882
0.96078 0.60000 0.85882
….
(256 lines in total)
```

The colourmap will be applied to images in that way that the first R-G-B value is assigned to the minimum cut value and the last R-G-B value to the maximum cut value.

The 256 values are then being assigned according to the selected scaling function either as a linear scale, a logarithmic or a square root scale.

Colourmaps and scaling functions are automatically applied when the user selects them. A quit button can be used to close the dialog.

The Colourmap Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget.

The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_COLOURMAP_NAME	Informs the Image Widget of the selected colourmap name.
DDT_DIALOG_PARAM_SCALING_FUNCTION	Informs the Image Widget of the selected scaling function.
DDT_DIALOG_PARAM_COLOURMAP_LIST	Used to transfer the list of colourmaps to the dialog.

Table 30: Parameters of the Colourmap Dialog

3.3.2.2.3 FITS Header Dialog

The FITS Header Dialog can display the FITS Header of the main HDU when a FITS file is loaded in the Image Widget.

The dialog also reports the filename of the currently loaded file. A “Quit” button closes the dialog.

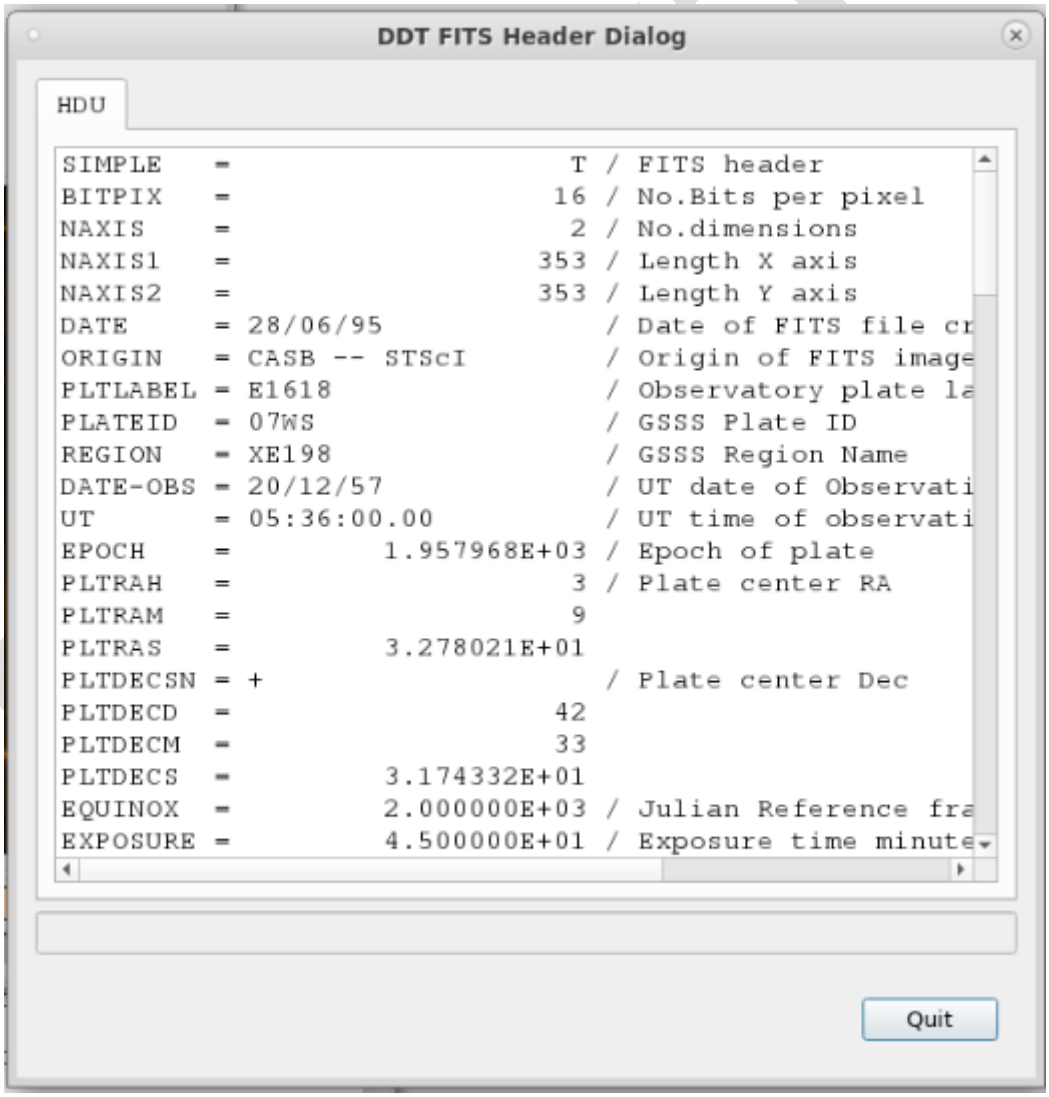


Figure 15: FITS Header Dialog

The FITS Header Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget.

The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_FITS_HEADER_DATA	Using this parameter the FITS Header data can be send to the dialog.
DDT_DIALOG_PARAM_FITS_HEADER_FILE	Using this parameter the name of the FITS file can be send to the dialog.

Table 31: Parameters of the FITS Header Dialog

3.3.2.2.4 Data Stream Dialog

The Data Stream Dialog contains the same functionality that is included in the Data Stream widget, see section 3.3.2.1.2. It is a dialog version of the widget that can be used in Viewer applications that do not contain the widget.

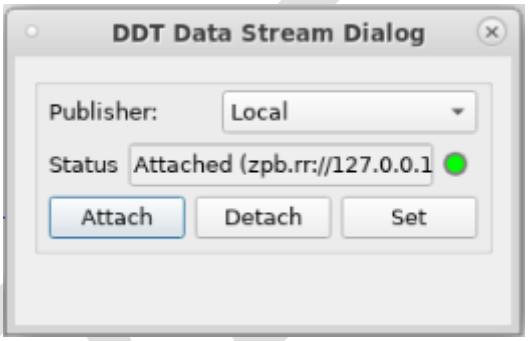


Figure 16: Data Stream Dialog

The Data Stream Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_ATTACH_DATA_STREAM	Informs the Image Widget that a data stream should be attached.
DDT_DIALOG_PARAM_DETACH_DATA_STREAM	Informs the Image Widget that a data stream should be detacted.
DDT_DIALOG_PARAM_STATUS_DATA_STREAM	Reports the status of the currently selected data stream.
DDT_DIALOG_PARAM_NAME_DATA_STREAM	Sets the name for the data stream.

Table 32: Parameters of the Data Stream Dialog

3.3.2.2.5 Image Header Data Units (HDU) Dialog

The Image HDU Dialog can be used to access additional HDUs of FITS files loaded in the Viewer. Once a FITS file containing several HDUs was opened the dialog offers a list of all HDUs showing the type, the name and information on the dimension of the HDU.

After selecting an entry in the list the “Open” button can be used to load the content into the viewer. HDUs of type image will be displayed in the Image Widget. Binary Tables will be displayed in a separate dialog (the FITS Table Dialog described in 3.3.2.2.6).

The additional button “Display as one image” allows the user to open all HDUs of type image into a single view in the Image Widget.

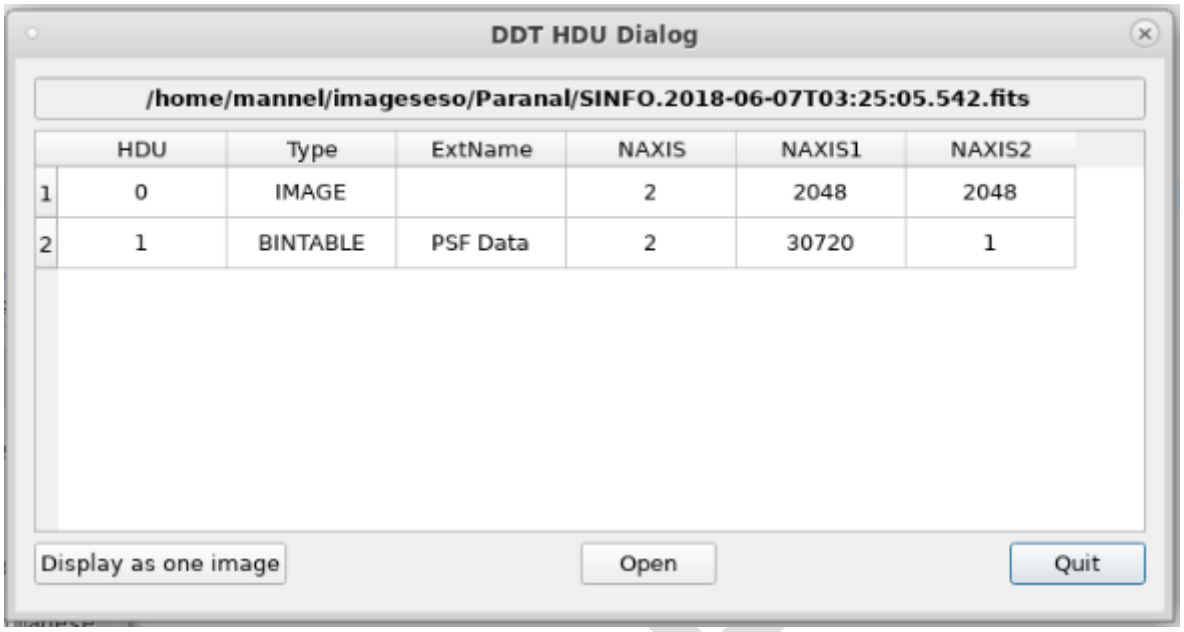


Figure 17: HDU Dialog

The HDU Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_HDU_FILE	Name of the FITS file containing several HDUs.
DDT_DIALOG_PARAM_HDU_TABLE_SIZE	Size of the selected table.
DDT_DIALOG_PARAM_HDU_TABLE_DATA	Data of the selected table.
DDT_DIALOG_PARAM_HDU_SINGLE_IMAGE	Send to open a single image selected from the table.
DDT_DIALOG_PARAM_HDU_ALL_AS_ONE	Send to open all images in a single view.

Table 33: Parameters of the HDU Dialog

3.3.2.2.6 FITS Table Dialog

The FITS Table Dialog is used to display the content of binary table extensions of FITS files.

Once a binary table was selected and opened in the HDU dialog (see 3.3.2.2.5) the content is displayed in this dialog. The format of the table depends on the content of the binary table.

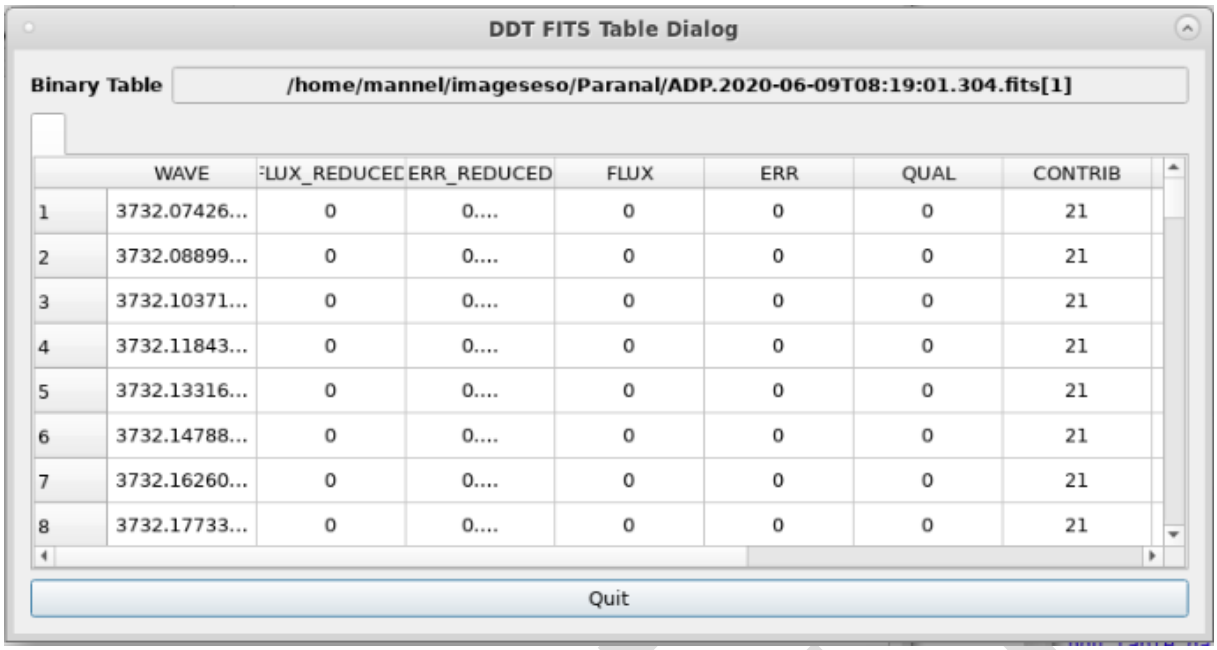


Figure 18: FITS Table Dialog

The FITS Header Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_FITS_HEADER_DATA	Data from the selected binary table.
DDT_DIALOG_PARAM_FITS_HEADER_FILE	Filename of the selected FITS file containing the binary table.

Table 34: Parameters of the FITS Table Dialog

3.3.2.2.7 Tabular Region Dialog

The Tabular Region Dialog can be used to display pixelvalues around the current mouse pointer position.

The dialog will show a table of nx x ny values around the mouse position. The size of the table can be configured by setting the values for nx and ny and pressing the “Resize Table” button.

In addition, the dialog will calculate some statistics on the selected data. These statistic values are:

- The minimum pixelvalue (min)
- The maximum pixelvalue (max)
- The average of the pixelvalues (ave)
- The root-mean-square value of the pixelvalues (rms)

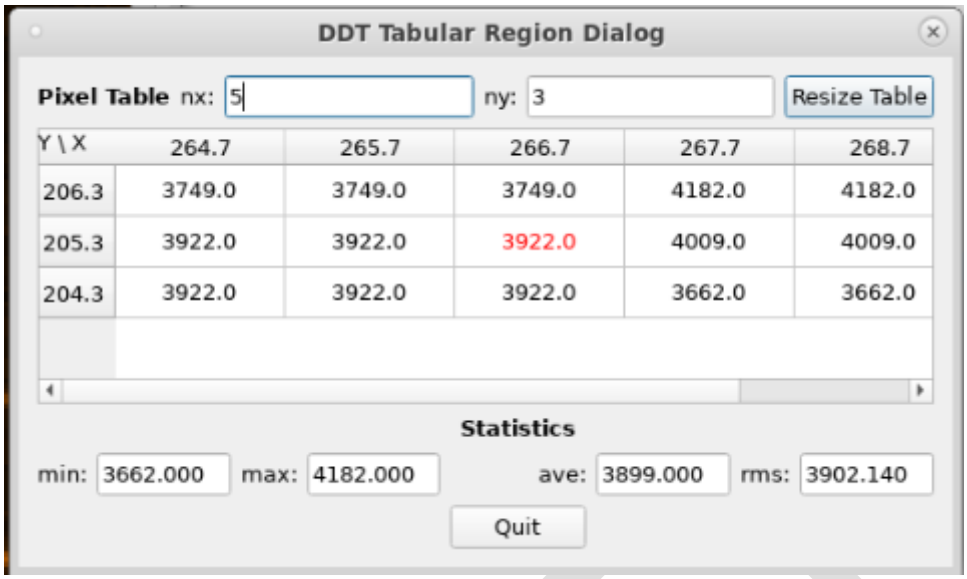


Figure 19: Tabular Region Dialog

The Tabular Region Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_TABULAR_REGION_DATA	This is used to the the pixelvalue data.
DDT_DIALOG_PARAM_TABULAR_REGION_ROWCOLDATA	This is used to resize the table to a new size.
DDT_DIALOG_PARAM_TABULAR_REGION_STATISTICDATA	Contains the statistics data.
DDT_DIALOG_PARAM_TABULAR_REGION_RESIZE	This signal is send, when the table size was changed.

Table 35: Parameters of the Tabular Region Dialog

3.3.2.2.8 Graphical Elements Dialog

The Graphical Elements Dialog can be used to draw overlay elements like ovals, crosses, rectangles, lines or text into the Image Widget.

The dialog offers buttons to select drawing any of those elements. While one of these buttons is selected, the user can draw this kind of overlay elements into the image.

The button showing an arrow can be used to select any of the previously drawn element, e.g. in order to delete this element.

The dialog also allows to define certain properties of the overlay elements. These are:

- Line thickness in pixel
- Font used for text
- Line colour
- Checkbox to select a fill colour plus the colour used for filling objects
- A tag (a string) that can be used to assign certain tags to overlay elements
- Threshold scale (a value which defines at which scale factor overlay elements will be hidden)

Figure 20: Graphical Elements Dialog

The tag can be used, e.g. by the Overlay API described in 3.3.2.3 in order to hide or show all elements using a given tag.

The dialog also offers a number of buttons. These can be used to delete a selected overlay element (“Delete”), to delete all elements of a specific type (“Delete all..”) or to close the dialog (“Quit”).

The Graphical Elements Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_LINE_THICKNESS	Sets the line thickness for overlay elements.
DDT_DIALOG_PARAM_FONT	Sets the font used for text overlays.
DDT_DIALOG_PARAM_FILL_COLOR	Sets the fill colour for overlay elements.
DDT_DIALOG_PARAM_FILL_ENABLED	Sets a flag, if the fill colour should be used.
DDT_DIALOG_PARAM_LINE_COLOR	Sets the line colour for the overlay elements.
DDT_DIALOG_PARAM_TAG	Sets the tag for overlay elements.
DDT_DIALOG_PARAM_DRAW_MODE	Sets the current draw mode (e.g. rectangle, ellipse, line etc.)
DDT_DIALOG_PARAM_DELETE_ELEMENT	Sends a signal that the selected element should be deleted.
DDT_DIALOG_PARAM_SCALE_THRESHOLD	Sets the threshold scale value. Overlay elements will be hidden below this threshold.
DDT_DIALOG_PARAM_SCALE_THRESHOLD_LIST	Gives the list of possible scale factors to the dialog.

Table 36: Parameters of the Graphical Elements Dialog

3.3.2.2.9 Cut Values Dialog

The Cut Values Dialog offers the same function as the Cut Values widget described in 3.3.2.1.9.

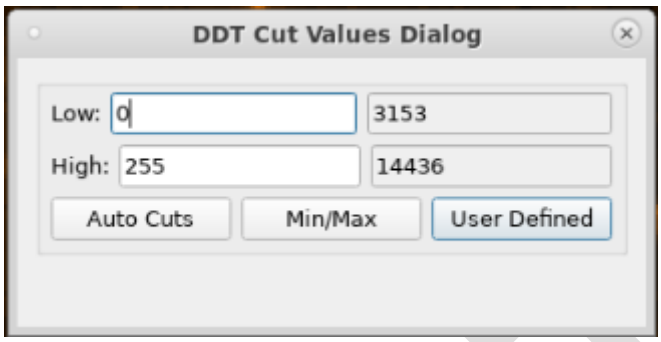


Figure 21: Cut Values Dialog

The Cut Values Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_CURRENT_CUT_VALUES	Sets the selected cut values.
DDT_DIALOG_PARAM_CURRENT_CUT_TYPE	Set the selected method for the cut values (auto, min/max, user defined)

Table 37: Parameters of the Cut Values Dialog

3.3.2.2.10 Bias Dialog

The Bias Dialog can be used to define a number of Bias images which can be applied to the image loaded in the Image Widget.

Bias images will be subtracted from the image loaded in the related Image Widget.

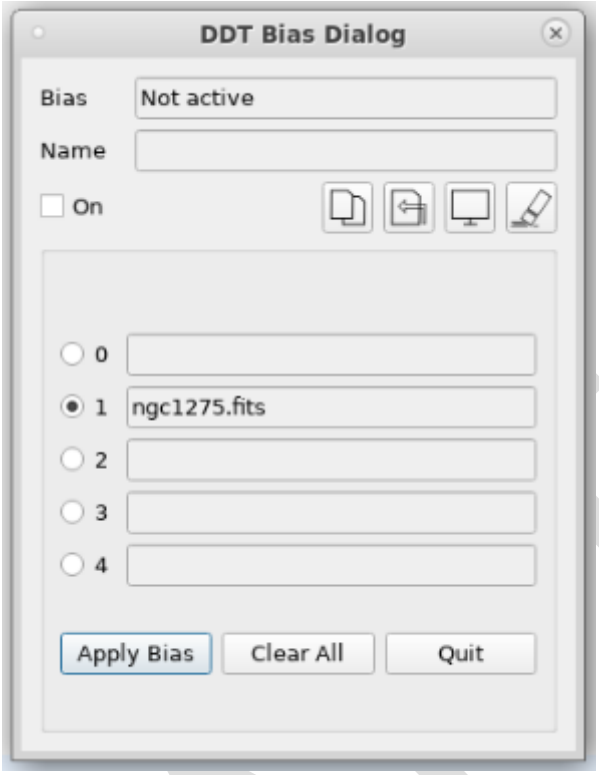


Figure 22: Bias Dialog

The dialog holds 5 slots for bias images. Bias images can either be captured from a attached data stream or can be loaded from the disk. The current bias image can be selected using the radio buttons on the table of bias images. By pressing the “Apply Bias” button the currently selected bias image is applied. The “Clear All” button allows to clear the list of bias images.

The buttons on top of the bias image list have the functions:

- Select currently loaded image as bias image (for the selected table entry)
- Load a FITS file as bias image to the selected slot
- Display the FITS image in the Image Widget
- Clear the selected table entry

By setting the “On” checkbox the bias function can be applied automatically to all new images loaded.

The Bias Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_BIAS_STORE_CURRENT_IMAGE	Send when the currently loaded image should be stored as bias image.
DDT_DIALOG_PARAM_BIAS_STORE_RETURN_VALUES	Update list of bias images.
DDT_DIALOG_PARAM_BIAS_APPLY_BIAS	Send when a bias image should be applied.
DDT_DIALOG_PARAM_BIAS_CLEAR_ALL	Send when all bias images should be cleared.

Parameter ID	Meaning
DDT_DIALOG_PARAM_BIAS_CLEAR_SELECTED	Send when only a selected entry shall be cleared.
DDT_DIALOG_PARAM_BIAS_LOAD_FROM_DISC	Send when a bias image shall be loaded from disk.
DDT_DIALOG_PARAM_BIAS_ENABLE_BIAS	Send when the bias function shall be enabled.
DDT_DIALOG_PARAM_BIAS_DISPLAY_SELECTED	Send when the selected bias image should be displayed.
DDT_DIALOG_PARAM_BIAS_CURRENT_SLOT_NAMES	Sends the current slot names.
DDT_DIALOG_PARAM_BIAS_CURRENT_SELECTED_SLOT	Sends the selected slot.

Table 38: Parameters of the Bias Dialog

3.3.2.2.11 Statistics Dialog

The Statistics Dialog allows the user to define a rectangular region in the image for which statistics should be calculated.

The dialog will display the following values after the user selected a rectangle:

- STARTX / STARTY / ENDX / ENDY: Corner coordinates of the rectangle
- MEAN: Mean value of the pixels in the rectangle
- RMS: Root-mean-square for the pixels
- MIN / MAX: Minimum and maximum pixel value
- PIXELS: Number of pixels in the rectangle

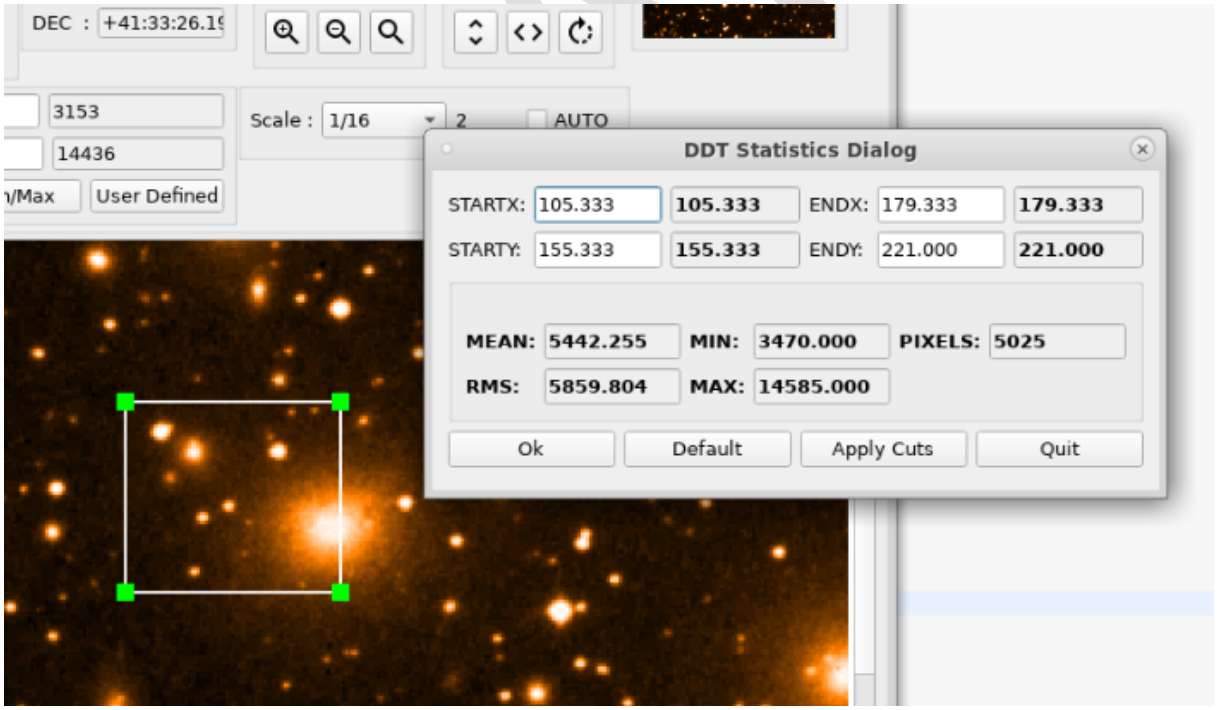


Figure 23: Statistics Dialog

The button “Default” can be used to set a default rectangle which is centred on the image.

The “Apply Cuts” button will use the min/max values from the statistics as new cut values for the image display.

When pressing “Ok” the dialog will return the statistics values as a list:

<min> <max> <mean> <rms> <pixels> <startx> <starty> <endx> <endy>

The Statistics Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_STATISTIC_DRAW_MODE	Set the draw mode (for drawing the rectangle)
DDT_DIALOG_PARAM_STATISTIC_VALUES	Update of the statistics values.
DDT_DIALOG_PARAM_STATISTIC_COORDS	Update of the coordinate values.
DDT_DIALOG_PARAM_STATISTIC_CUT_VALUES	Cut value function selected.
DDT_DIALOG_PARAM_STATISTIC_RETURN_VALUES	Return the current statistics as a string.
DDT_DIALOG_PARAM_STATISTIC_DEFAULT_RECT	Set the selection rectangle to the image center.
DDT_DIALOG_PARAM_STATISTIC_INITIAL_VALUES	Sets the initial values.

Table 39: Parameters of the Statistics Dialog

3.3.2.2.12 Slit Dialog

The Slit Dialog can be used to calculate the offset of a position in the image to a defined slit object.

The user needs to select a point in the image and then the offset from this point to the slit object is calculated and displayed. The dialog will report the following values:

- Target X / Target Y: Position the user selected via mouse click
- Slit X / Slit Y: Centre position of the slit object
- X Offset / Y Offset: Offset from the target location to the slit location

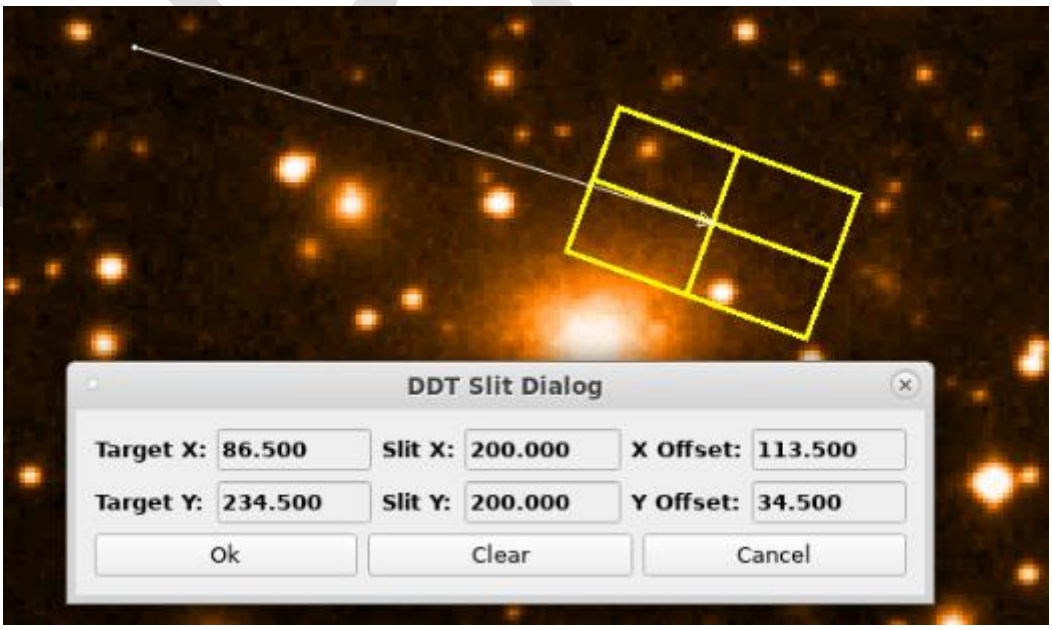


Figure 24: Slit Dialog

The slit object is defined by a configuration file. The file is called: slitparameter.ini

It is deployed to the resource/config-folder of the deployment directory.

The file has the following syntax:

[slitparameter]

slit_x=200.0

slit_y=200.0

slit_size_x=50.0

slit_size_y=30.0

slit_angle=20.0

slit_color=yellow

When pressing the “Ok” button the dialog will return the values for the offset calculation in the form:

Slit - <Offset X> <Offset Y> <Target X> <Target Y> <Slit X> <Slit Y>

The Slit Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_SLIT_DRAW_MODE	Sets the draw mode for drawing of the slit object
DDT_DIALOG_PARAM_SLIT_VALUES	Specifies the slit values.
DDT_DIALOG_PARAM_SLIT_RETURN_VALUES	Will contain the return values as a string.
DDT_DIALOG_PARAM_SLIT_INITIAL_VALUES	Sets the initial values.
DDT_DIALOG_PARAM_SLIT_DRAW_LINE	Send when the line was drawn.

Table 40: Parameters of the Slit Dialog

3.3.2.2.13 Pixel vs. Colourmap (PVCM) Dialog

The PVCM Dialog will show the distribution of pixelvalues versus the colourmap values.

The dialog will display the data as a diagram and allow the user some options to modify the range for the calculations.

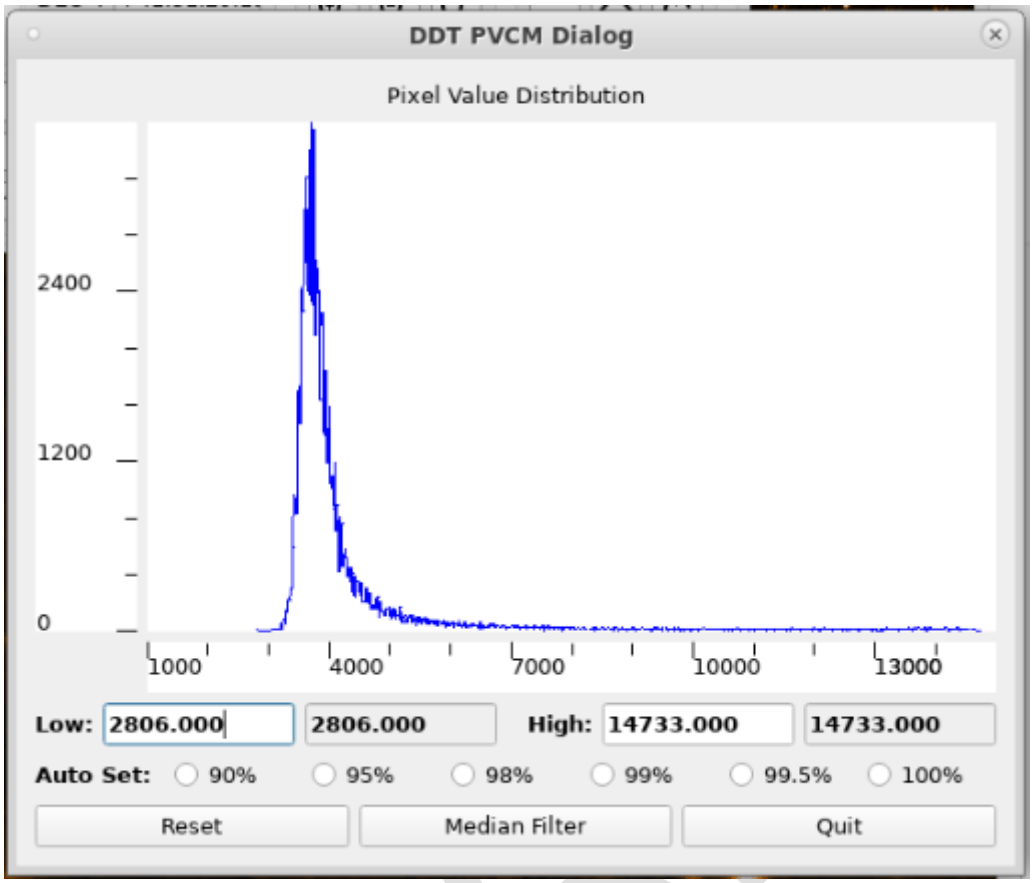


Figure 25: PVCM Dialog

The dialog will show the minimum and maximum values (Low / High) used for the diagram. The values are initially automatically calculated, but the user can manually specify new low and high values, selecting them via the RETURN key.

It is also possible to select the range for the histogram display using the radio buttons 90%, 95%, 98%, 99%, 99.5% and 100%. These specify the range that is taken into account for the display of the histogram.

In addition the button “Reset” will reset the display to its initial values and the button “Median Filter” to the pixel distribution.

The PVCM Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_PVCM_CUT_VALUES	Send when selecting the median filter option.
DDT_DIALOG_PARAM_PVCM_HISTOGRAM_VALUES	Contains the histogram values to be displayed.
DDT_DIALOG_PARAM_PVCM_RESET	Send when pressing the reset button.
DDT_DIALOG_PARAM_PVCM_MEDIAN	Send when pressing the median filter button.
DDT_DIALOG_PARAM_PVCM_AUTO_SET	Send when selecting one of the radio buttons.

Table 41: Parameters of the PVCM Dialog

3.3.2.2.14 Reference Line Dialog

The Reference Line Dialog can be used to display the pixel distribution along a line defined by the user.

When opening the dialog, the use can define a line in the Image Widget using the mouse. For the pixelvalues along the line then a diagram is shown. The diagram can be displayed using different smoothing algorithms: Step, Linear, Natural and Quadratic.

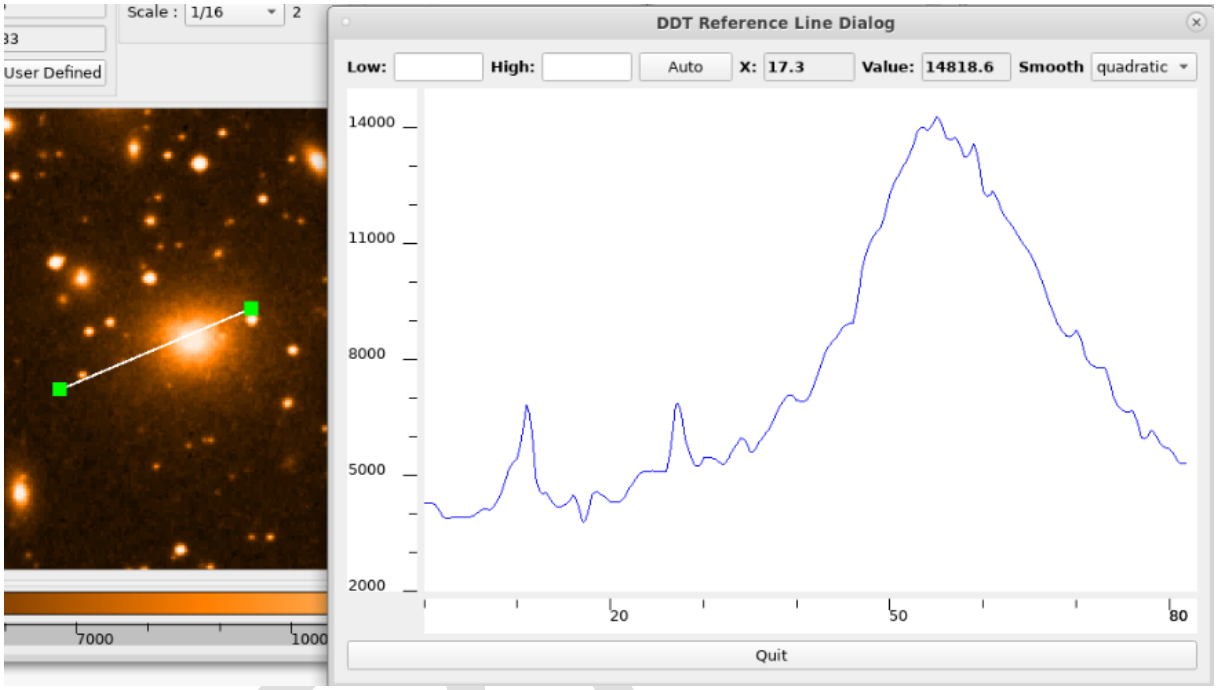


Figure 26: Reference Line Dialog

The user can also select the minimum and maximum of the values on the y-axis by manually entering values for the low/high values (and then pressing RETURN). By pressing the “Auto” button, the minimum and maximum are set automatically.

The user can also move the mouse pointer through the diagram to read the x and y-coordinates at a given point.

The Reference Line Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_REFLINE_RANGE_VALUES	Sets the min and max values.
DDT_DIALOG_PARAM_REFLINE_SPECTRUM_VALUES	Retrieves the pixel values for the diagram.
DDT_DIALOG_PARAM_REF_LINE_DRAW_MODE	Sets the draw mode in the Image Widget to Reference Line mode.

Table 42: Parameters of the Reference Line Dialog

3.3.2.2.15 Flip Rotate Scale Cut Values Dialog

This dialog is a container for the Flip / Rotate widget (see 3.3.2.1.3), the Cut Values widget (see 3.3.2.1.9), the Scale Buttons widget (see 3.3.2.1.4) and the Image Scale widget (see 3.3.2.1.6).

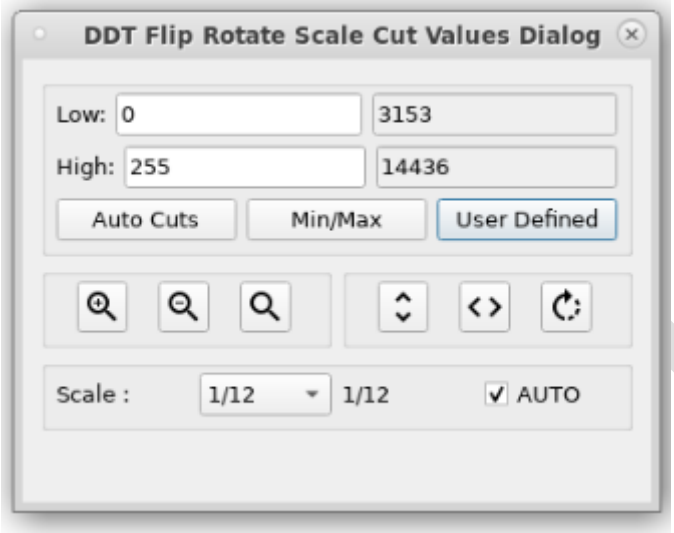


Figure 27: Flip Rotate Scale Cut Values Dialog

The Flip Rotate Scale Cut Values Dialog uses the signal:

```
ParameterChanged(QString dialog_id, QString param_id, QVariant parameter)
```

to communicate with the Image Widget. The following parameter IDs are supported:

Parameter ID	Meaning
DDT_DIALOG_PARAM_ROTATE_BY_ANGLE	Send when the image should be rotated.
DDT_DIALOG_PARAM_FLIP_VERTICAL	Send when the image should be flipped vertically.
DDT_DIALOG_PARAM_FLIP_HORIZONTAL	Send when the image should be flipped horizontally.
DDT_DIALOG_PARAM_INCREMENT_SCALE	Send when the image should be zoomed in.
DDT_DIALOG_PARAM_DECREMENT_SCALE	Send when the image should be zoomed out.
DDT_DIALOG_PARAM_DEFAULT_SCALE	Send when the image should be set to the default scale.
DDT_DIALOG_PARAM_NEW_SCALE	Send when the scale was changed.
DDT_DIALOG_PARAM_AUTO_SCALE	Send when auto-scale was selected.
DDT_DIALOG_PARAM_SCALE_LIST	Used to setup the list of possible scale factors.

Table 43: Parameters of the Flip Rotate Scale Cut Values Dialog

3.3.2.3 Graphical Elements Library

The Graphical Elements Library can be used to add overlay elements to the image displayed in the Image Widget.

Overlay elements are implemented as elements of the class “DdtGraphicalElement” which is derived from the class QGraphicsItem.

The Image Widget can then add those Graphics Items to the QGraphicsScene.

The Graphical Elements are defined by the DdtOverlayType:

Overlay Type ID	Meaning
DDT_OVERLAY_ELLIPSE	Ellipse objects
DDT_OVERLAY_RECTANGLE	Rectangle objects
DDT_OVERLAY_CROSS	Cross objects
DDT_OVERLAY_TEXT	Text overlays
DDT_OVERLAY_LINE	Line overlays
DDT_OVERLAY_SLIT	Slit element
DDT_OVERLAY_STAT_RECTANGLE	Resizable rectangle for image statistics
DDT_OVERLAY_COMPASS	Compass object

Table 44: Overlay element types

The overlay elements used are stored in an object of the type “DdtGraphicalOverlay”.

This class offers a number of public functions that can be used to handle overlay objects in the Image Widget (which holds an instance of this class):

API function	Description
`AddGraphicalElement(DdtGraphicalElement* element)`	Adds a new overlay object to the scene.
`RemoveGraphicalElement(DdtGraphicalElement* element)`	Removes an overlay object from the scene.
`QList<DdtGraphicalElement*>* GetListOfGraphicalElements()`	Returns a list of all overlay objects in the scene.
`QList<DdtGraphicalElement*> GetElementByTag(QString tag)`	Return a list of all overlay objects of a given tag.
`void ShowAllElements()`	Show all overlay elements.
`void HideAllElements()`	Hides all overlay elements.
`void ShowElementsOfType(DdtOverlayType type)`	Show all overlay elements of a given type.
`void HideElementsOfType(DdtOverlayType type)`	Hide all overlay elements of a given type.
`void ShowElementsOfTag(QString tag)`	Show all overlay elements of a given tag.
`void HideElementsOfTag(QString tag)`	Hide all overlay elements of a given tag.
`void RemoveElementsOfType(DdtOverlayType type)`	Remove all overlay elements of a given type.

Table 45: API function of the Overlay API

The overlay API can also be accessed graphically by using the DDT Graphical Elements dialog (see 3.3.2.2.8)

3.3.2.4 DDT Standard Viewer

The DDT Standard Viewer is a reference implementation of a DDT Subscriber GUI using all of the existing DDT Widgets.

In the current version the DDT Standard Viewer is mainly displaying one Image Widget plus some auxiliary widgets that are connected to the Image Widget.

Via the context menu of the Image Widget the user can access a number of DDT Dialogs plus a File Open dialog which allows loading FITS files from disk.

The functionality of the widgets and dialogs was described in detail in the sections above.

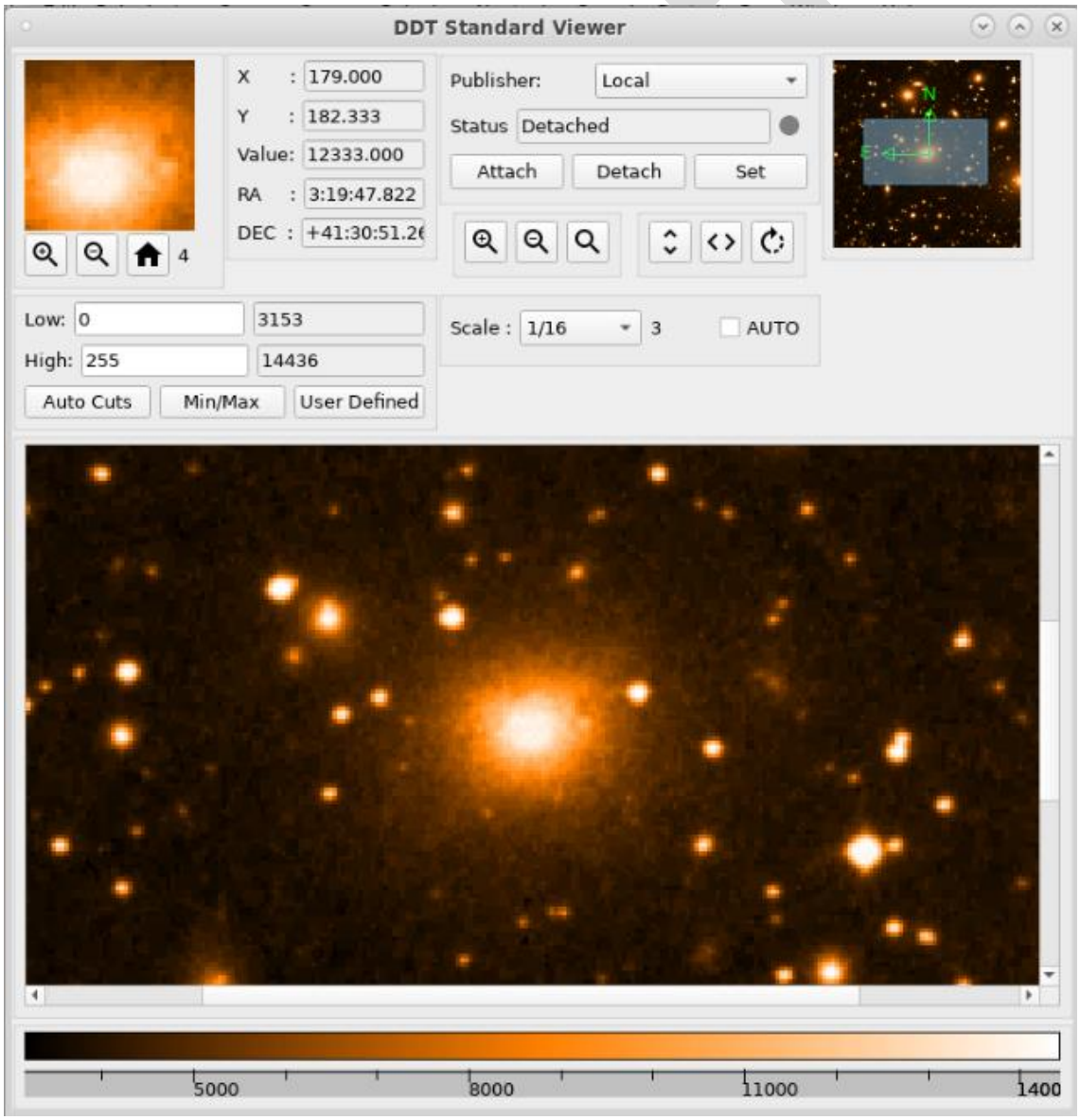


Figure 28: DDT Standard Viewer

The Standard Viewer can be started from the command line using the command:

```
ddtViewer [--filename=<path to image file> OR --datastream=<URI of data stream>]
[--debug] [--remotecontrol_uri <server URI for remote control>
```

The viewer can either be started giving the path to an image file (in JPEG or FITS format). This image will be loaded at start up.

The other option is to give the URI string to a Data Stream. This will start the viewer automatically attaching it to the given data stream. The two arguments are mutually exclusive.

The URI has the syntax:

```
<local broker URI> <datastream name> [<remote broker URI>]
```

An example for this would be:

```
--datastream “zpb.rr://127.0.0.1:5001 stream2 zpb.rr://192.10.10.200:5001”
```

The debug option can be set in order to set the log level temporarily to DEBUG.

The optional argument “remotecontrol_uri” will be described in the next section.

3.3.2.4.1 Remote Control Interface

The DDT Viewer supports an optional commandline argument for launching a remote control interface. By adding the argument “—remotecontrol_uri <URI>” the DDT Viewer will be running as a server. When running as a server the DDT Viewer opens a CII MAL server on the specified URI. It can receive commands from a CII MAL client.

The remote interface then allows to send a command “HandleRemoteCommand" with the arguments:

- image_widget_name (string): Name of the Image Widget for which a command is send
- command_name (string): Name of the remote command
- command_arguments (string array): Array of strings containing the arguments for the command

The command will return a response as a string.

In the current version the handling of the command is not yet implemented. The command interface is prepared, but the commands are not handled by the DDT Viewer yet.

3.3.3 Image Handling

The Image Handling provides image processing capabilities that can be accessed by a simple API. It makes use of the ESO Common Pipeline Library (CPL) for image processing. The functions provided are roughly divided into the following domains:

- I/O: handle access to the filesystem, i.e. reading and writing images in the FITS file format
- Image Processing: provide operations like flip, rotate, cut level application, configurable map support etc.
- Arithmetical Computations: provide basic arithmetical operations like addition, subtraction, multiplication, division.
- Analysis & Statistics: provide statistical information like min / max, mean, RMS, sigma, FWHM, circular object detection etc.
- Coordinate Conversion: convert between coordinate systems and between image and canvas coordinates

The starting point to use the Image Handling library would be the instantiation of an object of the class `ddt::ImageHandling()` (see sample code below). This object then offers functions to attach to a data stream, to open a data file, to access the image data and perform various operations on these data:

```
ddt::ImageHandling* imgHandling = new ddt::ImageHandling();

imgHandling->LoadColorMaps(colourmap_directory, default_colourmap);

imgHandling->LoadFile(filename);

imgHandling->ReprossImage();

cpl_image* image = imgHandling->get_Image();
```

Note: the base for the following sub-sections was auto generated by the doxygen tool from the software source files.

3.3.3.1 ddt::ConfigurationMaps Class Reference

```
#include <configurationMaps.hpp>
```

3.3.3.1.1 Public Member Functions

- **ConfigurationMaps** ()
- virtual **~ConfigurationMaps** ()
- void **set_logger** (ddt::DdtLogger ***logger**)
- std::list< std::string > **get_ConfigurationMaps** ()
- cpl_image * **get_ConfigurationMap** (std::string identification)
- void **LoadConfigurationMaps** (std::string config_map_source)

3.3.3.1.2 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.1.3 Detailed Description

Class to wrap the usage of configuration map access classes.

3.3.3.1.4 Constructor & Destructor Documentation

3.3.3.1.4.1 ConfigurationMaps::ConfigurationMaps ()

Constructor

3.3.3.1.4.2 ConfigurationMaps::~ConfigurationMaps ()[virtual]

Destructor

3.3.3.1.5 Member Function Documentation

3.3.3.1.5.1 cpl_image * ConfigurationMaps::get_ConfigurationMap (std::string *identification*)

Get a configuration map in form of a CPL image.

3.3.3.1.5.1.1 Parameters

identification	string to identify the configuration map that is to be loaded

3.3.3.1.5.1.2 Returns

a cpl_image object containing the configuration map

3.3.3.1.5.2 std::list< std::string > ConfigurationMaps::get_ConfigurationMaps ()

Get a list of configuration maps. The maps are identified using a string.

3.3.3.1.5.2.1 Returns

a list of configuration map identification strings

3.3.3.1.5.3 void ConfigurationMaps::LoadConfigurationMaps (std::string *config_map_source*)

Load the configuration maps from the specified configuration map source.

3.3.3.1.5.3.1 Parameters

config_map_source	the source of the configuration maps

3.3.3.1.5.4 void ConfigurationMaps::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.1.5.4.1 Parameters

logger	The logger object

3.3.3.1.6 Member Data Documentation

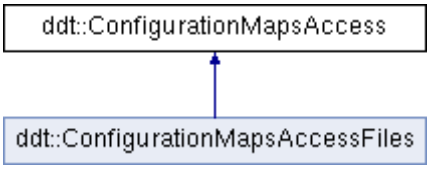
3.3.3.1.6.1 ddt::DdtLogger* ddt::ConfigurationMaps::logger[protected]

The logger object

3.3.3.2 ddt::ConfigurationMapsAccess Class Reference

```
#include <configurationMapsAccess.hpp>
```

Inheritance diagram for ddt::ConfigurationMapsAccess:



3.3.3.2.1 Public Member Functions

- **ConfigurationMapsAccess** ()
- virtual **~ConfigurationMapsAccess** ()
- virtual void **set_logger** (ddt::DdtLogger ***logger**)
- virtual std::list< std::string > **get_ConfigurationMaps** ()
- virtual cpl_image * **get_ConfigurationMap** (std::string identification)
- virtual void **LoadConfigurationMaps** (std::string config_map_source)=0

3.3.3.2.2 Protected Attributes

- ddt::DdtLogger * **logger**
- std::unordered_map< std::string, cpl_image * > **configurationMaps**

3.3.3.2.3 Detailed Description

Class to wrap the usage of configuration map access functions.

3.3.3.2.4 Constructor & Destructor Documentation

3.3.3.2.4.1 ConfigurationMapsAccess::ConfigurationMapsAccess ()[explicit]

Constructor

3.3.3.2.4.2 ConfigurationMapsAccess::~ConfigurationMapsAccess ()[virtual]

Destructor

3.3.3.2.5 Member Function Documentation

3.3.3.2.5.1 cpl_image * ConfigurationMapsAccess::get_ConfigurationMap (std::string *identification*)[virtual]

Get a configuration map in form of a CPL image.

3.3.3.2.5.1.1 Parameters

identification	string to identify the configuration map that is to be loaded

3.3.3.2.5.1.2 Returns

a cpl_image object containing the configuration map

3.3.3.2.5.2 std::list< std::string > ConfigurationMapsAccess::get_ConfigurationMaps ()[virtual]

Get a list of configuration maps. The maps are identified using a string.

3.3.3.2.5.2.1 Returns

a list of configuration map identification strings

3.3.3.2.5.3 virtual void ddt::ConfigurationMapsAccess::LoadConfigurationMaps (std::string *config_map_source*)[pure virtual]

Load the configuration maps from the specified configuration map source.

3.3.3.2.5.3.1 Parameters

config_map_source	the source of the configuration maps

Implemented in **ddt::ConfigurationMapsAccessFiles**.

3.3.3.2.5.4 void ConfigurationMapsAccess::set_logger (ddt::DdtLogger * *logger*)[virtual]

Set logger

3.3.3.2.5.4.1 Parameters

logger	The logger object

3.3.3.2.6 Member Data Documentation

3.3.3.2.6.1 std::unordered_map<std::string, cpl_image*> ddt::ConfigurationMapsAccess::configurationMaps[protected]

The collection of configuration maps

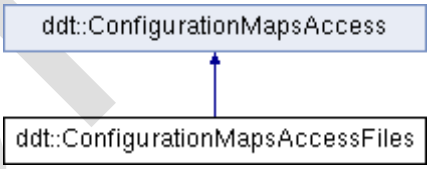
3.3.3.2.6.2 ddt::DdtLogger* ddt::ConfigurationMapsAccess::logger[protected]

The logger object

3.3.3.3 ddt:ConfigurationMapsAccessFiles Class Reference

```
#include <configurationMapsAccessFiles.hpp>
```

Inheritance diagram for ddt::ConfigurationMapsAccessFiles:



3.3.3.3.1 Public Member Functions

- **ConfigurationMapsAccessFiles** ()
- **~ConfigurationMapsAccessFiles** ()
- void **LoadConfigurationMaps** (std::string config_map_source)

3.3.3.3.2 Additional Inherited Members

3.3.3.3.3 Detailed Description

Class to wrap the usage of configuration maps that are stored in files.

3.3.3.3.4 Constructor & Destructor Documentation

3.3.3.3.4.1 ConfigurationMapsAccessFiles::ConfigurationMapsAccessFiles ()

Constructor

3.3.3.3.4.2 ConfigurationMapsAccessFiles::~ConfigurationMapsAccessFiles ()

Destructor

3.3.3.3.5 Member Function Documentation

3.3.3.3.5.1 void ConfigurationMapsAccessFiles::LoadConfigurationMaps (std::string *config_map_source*)[virtual]

Load the configuration maps from the specified configuration map source.

3.3.3.3.5.1.1 Parameters

config_map_source	the source of the configuration maps

Implements **ddt::ConfigurationMapsAccess**.

3.3.3.4 CplStatistics Struct Reference

```
#include <imageStats.hpp>
```

3.3.3.4.1 Public Attributes

- double **min_val**
- double **max_val**
- double **mean_val**
- double **rms_val**
- int **pix_num**

3.3.3.4.2 Detailed Description

This file is part of the DDT Image Handling Library and provides functions to analyse and calculate statistic values for CPL images.

The usage is as follows:

`#include "ddt/imageStats.hpp" ... using namespace ddt;` Structure to bundle the main statistic values

3.3.3.4.3 Member Data Documentation

3.3.3.4.3.1 double CplStatistics::max_val

maximum pixel value

3.3.3.4.3.2 double CplStatistics::mean_val

mean pixel value

3.3.3.4.3.3 double CplStatistics::min_val

minimum pixel value

3.3.3.4.3.4 int CplStatistics::pix_num

number of pixel used for calculation

3.3.3.4.3.5 double CplStatistics::rms_val

RMS (root mean square) value

3.3.3.5 ddt::DataAcquisition Class Reference

```
#include <dataAcquisition.hpp>
```

3.3.3.5.1 Public Member Functions

- **DataAcquisition** ()
- virtual **~DataAcquisition** ()
- void **set_logger** (ddt::DdtLogger ***logger**)
- void **AttachDataStream** (std::string data_stream_id)
- void **DetachDataStream** (std::string data_stream_id)
- void **ProcessNewData** ()
- signal_t * **DataAvailableSignal** ()
- ddt::DataSample * **get_DataSample** ()
- bool **get_AttachedToStream** ()

3.3.3.5.2 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.5.3 Detailed Description

Class to wrap the usage of data stream acquisition functions.

This class offers functions to attach to and detach from a data stream. Attaching to the data stream is realized by a Data Subscriber that connects to a Data Broker. In case of new data available from the subscriber, a corresponding signal is sent out.

3.3.3.5.4 Constructor & Destructor Documentation

3.3.3.5.4.1 DataAcquisition::DataAcquisition ()

Constructor

3.3.3.5.4.2 DataAcquisition::~DataAcquisition ()[virtual]

Destructor

3.3.3.5.5 Member Function Documentation

3.3.3.5.5.1 void DataAcquisition::AttachDataStream (std::string *data_stream_id*)

Attach to a data stream

3.3.3.5.5.1.1 Parameters

data_stream_id	**The ID of the data stream (could be the URI)**

3.3.3.5.5.2 signal_t * DataAcquisition::DataAvailableSignal ()

Access to the data available signal

3.3.3.5.5.3 void DataAcquisition::DetachDataStream (std::string *data_stream_id*)

Detach from a data stream

3.3.3.5.5.3.1 Parameters

data_stream_id	**The ID of the stream to be disconnected**

3.3.3.5.5.4 bool DataAcquisition::get_AttachedToStream ()

Get the attached to stream flag

3.3.3.5.5.4.1 Returns

the flag indicating if the data acquisition is attached to a data stream

3.3.3.5.5.5 ddt::DataSample * DataAcquisition::get_DataSample ()

Get the currently available data sample

3.3.3.5.5.5.1 Returns

the available DataSample

3.3.3.5.5.6 void DataAcquisition::ProcessNewData ()

Event handler for new data on data stream

3.3.3.5.5.7 void DataAcquisition::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.5.5.7.1 Parameters

logger	**The logger object**

3.3.3.5.6 Member Data Documentation

3.3.3.5.6.1 ddt::DdtLogger* ddt::DataAcquisition::logger[protected]

The logger object

3.3.3.6 ddt::DataFile Class Reference

```
#include <dataFile.hpp>
```

3.3.3.6.1 Public Member Functions

- **DataFile** ()
- virtual **~DataFile** ()
- void **set_logger** (ddt::DdtLogger *__logger__)
- bool **OpenFile** (const std::string &file_name, int position=0)
- cpl_vector * **OpenAndLoadVector** (const std::string &file_name, int position=0)
- cpl_image * **OpenAndLoadImage** (const std::string &file_name, int position=0)
- cpl_image * **OpenAndLoadCube** (const std::string &file_name, int plane_number=0, int position=0)
- cpl_vector * **LoadVector** ()
- cpl_image * **LoadImage** (int plane_number=0)
- cpl_propertylist * **LoadPropertyList** ()
- cpl_wcs * **LoadWCS** ()
- int **get_FilePosition** ()
- int **get_NumberAxis** ()
- int **get_NumberPlanes** ()
- int **get_ImageWidth** ()
- int **get_ImageHeight** ()
- bool **get_IsVector** ()
- bool **get_IsImage** ()
- bool **get_IsCube** ()

3.3.3.6.2 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.6.3 Detailed Description

Class to wrap the usage of FITS file functions.

This class offers access to FITS files, using functions from the CPL library. The files are opened and the properties are read. According to the contained data (number of axis), various functions are available to load the 1D, 2D or 3D data.

3.3.3.6.4 Constructor & Destructor Documentation

3.3.3.6.4.1 DataFile::DataFile ()

Constructor

3.3.3.6.4.2 DataFile::~DataFile ()[virtual]

Destructor

3.3.3.6.5 Member Function Documentation

3.3.3.6.5.1 int DataFile::get_FilePosition ()

Get the current file position (extension; number of HDU opened (0: first))

3.3.3.6.5.1.1 Returns

the current file position

3.3.3.6.5.2 int DataFile::get_ImageHeight ()

Get the image height (number of y pixels)

3.3.3.6.5.2.1 Returns

the image height

3.3.3.6.5.3 int DataFile::get_ImageWidth ()

Get the image width (number of x pixels)

3.3.3.6.5.3.1 Returns

the image width

3.3.3.6.5.4 bool DataFile::get_IsCube ()

Get the flag indicating if the image contains cube data

3.3.3.6.5.4.1 Returns

the cube flag

3.3.3.6.5.5 bool DataFile::get_IsImage ()

Get the flag indicating if the image contains image data

3.3.3.6.5.5.1 Returns

the image flag

3.3.3.6.5.6 bool DataFile::get_IsVector ()

Get the flag indicating if the image contains vector data

3.3.3.6.5.6.1 Returns

the vector flag

3.3.3.6.5.7 int DataFile::get_NumberAxis ()

Get the number of axis (vector data: 1, image data: 2, cube data: 3)

3.3.3.6.5.7.1 Returns

the number of axis

3.3.3.6.5.8 int DataFile::get_NumberPlanes ()

Get the number of planes (relevant only for a 3D image)

3.3.3.6.5.8.1 Returns

the number of planes

3.3.3.6.5.9 cpl_image * DataFile::LoadImage (int *plane_number* = 0)

Load image data. Note that the file has to be opened before using **OpenFile()** function.

3.3.3.6.5.9.1 Parameters

plane_number	number of plane to load. This defaults to 0 (the only option for 2D images), which means the first plane.

3.3.3.6.5.9.2 Returns

a cpl_image object, or NULL on error

3.3.3.6.5.10 cpl_propertylist * DataFile::LoadPropertyList ()

Load the CPL property list from the file. The returned object must be destroyed using cpl_propertylist_delete().

3.3.3.6.5.10.1 Returns

the cpl_propertylist object, or NULL on error

3.3.3.6.5.11 cpl_vector * DataFile::LoadVector ()

Load vector data. Note that the file has to be opened before using **OpenFile()** function.

3.3.3.6.5.11.1 Returns

a cpl_vector object, or NULL on error

3.3.3.6.5.12 cpl_wcs * DataFile::LoadWCS ()

Load the CPL WCS object from the file's property list. The returned object must be destroyed using cpl_wcs_delete().

3.3.3.6.5.12.1 Returns

the cpl_wcs object, or NULL on error

3.3.3.6.5.13 cpl_image * DataFile::OpenAndLoadCube (const std::string & *file_name*, int *plane_number* = 0, int *position* = 0)

Open file and load image data. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0). The function first opens the file and checks the image properties. If the file contains cube data (number of axis = 3), the image data of the specified plane is loaded.

3.3.3.6.5.13.1 Parameters

file_name	
plane_number	number of plane to load. This defaults to 0, meaning the first plane.
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.6.5.13.2 Returns

a cpl image object, or NULL on error (file not existing, no image data)

3.3.3.6.5.14 cpl_image * DataFile::OpenAndLoadImage (const std::string & *file_name*, int *position* = 0)

Open file and load image data. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0). The function first opens the file and

checks the image properties. If the file contains image data (number of axis = 2), the image data is loaded.

3.3.3.6.5.14.1 Parameters

file_name	
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.6.5.14.2 Returns

a cpl image object, or NULL on error (file not existing, no image data)

3.3.3.6.5.15 cpl_vector * DataFile::OpenAndLoadVector (const std::string & *file_name*, int *position* = 0)

Open file and load vector data. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0). The function first opens the file and checks the image properties. If the file contains vector data (number of axis = 1), the vector data is loaded.

3.3.3.6.5.15.1 Parameters

file_name	
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.6.5.15.2 Returns

a cpl vector object, or NULL on error (file not existing, no vector data)

3.3.3.6.5.16 bool DataFile::OpenFile (const std::string & *file_name*, int *position* = 0)

Open file. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0). The function reads the property list of the file and checks the number of axis in the file and the image dimensions. This information can be retrieved by the corresponding functions in order to load the image rsp. vector data.

3.3.3.6.5.16.1 Parameters

file_name	the FITS file name to load
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.6.5.16.2 Returns

true if opening was successful, false otherwise

3.3.3.6.5.17 void DataFile::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.6.5.17.1 Parameters

logger	The logger object

3.3.3.6.6 Member Data Documentation

3.3.3.6.6.1 ddt::DdtLogger* ddt::DataFile::logger[protected]

The logger object

3.3.3.7 ddt::ImageBuffer Class Reference

```
#include <imageBuffer.hpp>
```

3.3.3.7.1 Public Member Functions

- **ImageBuffer** ()
- virtual **~ImageBuffer** ()
- void **set_logger** (ddt::DdtLogger *__logger__)
- bool **LoadFile** (const std::string &file_name, int position=0)
- bool **LoadActualConfigurationMap** (std::string configuration_map_name)
- bool **LoadConfigurationMaps** (std::string configuration_map_source)
- void * **get_ImageData** ()
- cpl_image * **get_Image** ()
- int **get_ImageWidth** ()
- int **get_ImageHeight** ()
- int **get_NumberPlanes** ()
- int **get_NumberAxis** ()
- cpl_wcs * **get_Wcs** ()
- cpl_propertylist * **get_PropertyList** ()
- double **get_PixelValue** (int x_image, int y_image, int *is_rejected)
- bool **get_AttachedToStream** ()
- bool **get_LastSegment** ()
- bool **get_FirstSegmentReceived** ()
- void **AttachDataStream** (std::string data_stream_id)
- void **DetachDataStream** (std::string data_stream_id)
- signal_t * **ImageDataAvailableSignal** ()

3.3.3.7.2 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.7.3 Detailed Description

Class to wrap the internal CPL image. This is created from the incoming data stream rsp. from a loaded file.

It offers access to image properties.

3.3.3.7.4 Constructor & Destructor Documentation

3.3.3.7.4.1 ImageBuffer::ImageBuffer ()

Constructor

3.3.3.7.4.2 ImageBuffer::~ImageBuffer ()[virtual]

Destructor

3.3.3.7.5 Member Function Documentation

3.3.3.7.5.1 void ImageBuffer::AttachDataStream (std::string *data_stream_id*)

Attach to a data stream

3.3.3.7.5.1.1 Parameters

data_stream_id	**The ID of the data stream (could be the URI)**

3.3.3.7.5.2 void ImageBuffer::DetachDataStream (std::string *data_stream_id*)

Detach from a data stream

3.3.3.7.5.2.1 Parameters

data_stream_id	**The ID of the stream to be disconnected**

3.3.3.7.5.3 bool ImageBuffer::get_AttachedToStream ()

Get the attached to stream flag from the data acquisition

3.3.3.7.5.3.1 Returns

the flag indicating if the data acquisition is attached to a data stream

3.3.3.7.5.4 bool ImageBuffer::get_FirstSegmentReceived ()

Get the first segment received flag, indicating if the current image is the first segment of a segmented image

3.3.3.7.5.4.1 Returns

the flag indicating the first segment received

3.3.3.7.5.5 cpl_image * ImageBuffer::get_Image ()

Get the image data

3.3.3.7.5.6 void* ddt::ImageBuffer::get_ImageData ()

Get the image data

3.3.3.7.5.7 int ImageBuffer::get_ImageHeight ()

Get the image height

3.3.3.7.5.8 int ImageBuffer::get_ImageWidth ()

Get the image width

3.3.3.7.5.9 bool ImageBuffer::get_LastSegment ()

Get the last segment flag, indicating if the current image is complete or the last segment of a segmented image

3.3.3.7.5.9.1 Returns

the flag indicating the last segment

3.3.3.7.5.10 int ImageBuffer::get_NumberAxis ()

Get the number of axis in the image

3.3.3.7.5.11 int ImageBuffer::get_NumberPlanes ()

Get the number of planes in the image

3.3.3.7.5.12 double ImageBuffer::get_PixelValue (int *x_image*, int *y_image*, int * *is_rejected*)

Get the pixel value from the internal image at a certain position

3.3.3.7.5.12.1 Parameters

x_image	the pixel x position
y_image	the pixel y position
is_rejected	set to 1 if the pixel is bad, 0 if good, or -1 on error

3.3.3.7.5.12.2 Returns

double pixel value (cast to double) or undefined

3.3.3.7.5.13 cpl_propertylist * ImageBuffer::get_PropertyList ()

Get the property list

3.3.3.7.5.13.1 Returns

the property list

3.3.3.7.5.14 cpl_wcs * ImageBuffer::get_Wcs ()

Get the WCS object

3.3.3.7.5.14.1 Returns

the WCS object

3.3.3.7.5.15 signal_t * ImageBuffer::ImageDataAvailableSignal ()

Provide the image data available signal

3.3.3.7.5.16 bool ImageBuffer::LoadActualConfigurationMap (std::string *configuration_map_name*)

Load the supplied configuration map as the actual configuration map.

3.3.3.7.5.16.1 Parameters

configuration_map_name	the name of the configuration map to be loaded.

3.3.3.7.5.16.2 Returns

true if loading was successful, false otherwise

3.3.3.7.5.17 bool ImageBuffer::LoadConfigurationMaps (std::string *configuration_map_source*)

Load the configuration maps from the specified configuration map source.

3.3.3.7.5.17.1 Parameters

config_map_source	the source of the configuration maps

3.3.3.7.5.18 bool ImageBuffer::LoadFile (const std::string & *file_name*, int *position* = 0)

Load file. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0).

3.3.3.7.5.18.1 Parameters

file_name	the FITS file name to load
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.7.5.18.2 Returns

true if loading was successful, false otherwise

3.3.3.7.5.19 void ImageBuffer::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.7.5.19.1 Parameters

logger	The logger object

3.3.3.7.6 Member Data Documentation

3.3.3.7.6.1 ddt::DdtLogger* ddt::ImageBuffer::logger[protected]

The logger object

3.3.3.8 ddt::ImageColor Class Reference

```
#include <imageColor.hpp>
```

3.3.3.8.1 Public Member Functions

- **ImageColor** ()
- virtual **~ImageColor** ()
- void **set_logger** (ddt::DdtLogger ***logger**)
- void **LoadColorMaps** (std::string color_map_folder)
- std::list< std::string > **get_ColorMapsList** ()
- **ddt::colorMap_t** * **get_ColorMap** (std::string color_map_name)
- **ddt::colorMapARGB_t** * **get_ColorMapARGB** (std::string color_map_name)
- **ddt::colorMap_t** * **get_GrayScaleColorMap** ()
- **ddt::colorMapARGB_t** * **get_GrayScaleColorMapARGB** ()
- **ddt::scalingLut_t** * **get_LinearScalingLut** (const double cut_level_min, const double cut_level_max)
- **ddt::scalingLut_t** * **get_LogarithmicScalingLut** (const double cut_level_min, const double cut_level_max)
- **ddt::scalingLut_t** * **get_SqrtScalingLut** (const double cut_level_min, const double cut_level_max)

3.3.3.8.2 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.8.3 Detailed Description

This class is part of the DDT Image Handling Library and provides functions for color map purposes.

It offers functions to create colormaps from colormap files.

3.3.3.8.4 Constructor & Destructor Documentation

3.3.3.8.4.1 ImageColor::ImageColor ()

Constructor

This class is part of the DDT Image Handling Library and provides functions for coordinate conversion purposes.

It offers functions to to create colormaps from colormap files.

3.3.3.8.4.2 ImageColor::~ImageColor ()[virtual]

Destructor

3.3.3.8.5 Member Function Documentation

3.3.3.8.5.1 ddt::colorMap_t * ImageColor::get_ColorMap (std::string *color_map_name*)

Get a color map array, identified by the color map name

3.3.3.8.5.1.1 Parameters

color_map_name	the name of the color map (taken from the color map file name)

3.3.3.8.5.1.2 Returns

a colorMap object (an array of 256 RGB triples (array of 3 float values, representing the red, green and blue part, all expressed as a value between 0.0 and 1.0)

3.3.3.8.5.2 ddt::colorMapARGB_t * ImageColor::get_ColorMapARGB (std::string *color_map_name*)

Get a color map array, identified by the color map name

3.3.3.8.5.2.1 Parameters

color_map_name	the name of the color map (taken from the color map file name)

3.3.3.8.5.2.2 Returns

a colorMapARGB object (an array of 256 unsigned int values, each value representing an alpha channel and the red, green and blue parts, all expressed as a values between 0 and 255)

3.3.3.8.5.3 std::list< std::string > ImageColor::get_ColorMapsList ()

Returns a list of all loaded colormaps

3.3.3.8.5.3.1 Returns

list containing the names of all loaded colormaps

3.3.3.8.5.4 ddt::colorMap_t * ImageColor::get_GrayScaleColorMap ()

Get a default gray scale color map array

3.3.3.8.5.4.1 Returns

a colorMap object (an array of 256 RGB triples (array of 3 float values, representing the red, green and blue part, all expressed as a value between 0.0 and 1.0)

3.3.3.8.5.5 ddt::colorMapARGB_t * ImageColor::get_GrayScaleColorMapARGB ()

Get a default gray scale color map array

3.3.3.8.5.5.1 Returns

a colorMapARGB object (an array of 256 unsigned int values, each value representing an alpha channel and the red, green and blue parts, all expressed as a values between 0 and 255)

3.3.3.8.5.6 ddt::scalingLut_t * ImageColor::get_LinearScalingLut (const double *cut_level_min*, const double *cut_level_max*)

Get the LUT for linear color scaling

3.3.3.8.5.6.1 Parameters

cut_level_min	the minimum cut level to be applied for the LUT creation
cut_level_max	the minimum cut level to be applied for the LUT creation

3.3.3.8.5.6.2 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.8.5.7 ddt::scalingLut_t * ImageColor::get_LogarithmicScalingLut (const double *cut_level_min*, const double *cut_level_max*)

Get the LUT for logarithmic color scaling

3.3.3.8.5.7.1 Parameters

cut_level_min	the minimum cut level to be applied for the LUT creation
cut_level_max	the minimum cut level to be applied for the LUT creation

3.3.3.8.5.7.2 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.8.5.8 ddt::scalingLut_t * ImageColor::get_SqrtScalingLut (const double *cut_level_min*, const double *cut_level_max*)

Get the LUT for square root color scaling

3.3.3.8.5.8.1 Parameters

cut_level_min	the minimum cut level to be applied for the LUT creation
cut_level_max	the minimum cut level to be applied for the LUT creation

3.3.3.8.5.8.2 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.8.5.9 void ImageColor::LoadColorMaps (std::string *color_map_folder*)

Load the colormaps.

3.3.3.8.5.9.1 Parameters

color_map_folder	folder containing color map files

3.3.3.8.5.10 void ImageColor::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.8.5.10.1 Parameters

logger	The logger object

3.3.3.8.6 Member Data Documentation

3.3.3.8.6.1 ddt::DdtLogger* ddt::ImageColor::logger[protected]

The logger object

3.3.3.9 ddt::ImageCoords Class Reference

```
#include <imageCoords.hpp>
```

3.3.3.9.1 Public Member Functions

- **ImageCoords** (int image_width, int image_height, bool rotate_flag, bool flip_x_flag, bool flip_y_flag)
- virtual **~ImageCoords** ()=default
- void **set_logger** (ddt::DdtLogger *__logger__)
- bool **WorldCoordinatesDegreesFromCanvas** (cpl_wcs *wcs, const double x_canvas, const double y_canvas, double *alpha, double *delta)
- bool **WorldCoordinatesDegreesFromImage** (const cpl_wcs *wcs, const double x_image, const double y_image, double *alpha, double *delta)
- const cpl_matrix * **WorldCoordinatesCdMatrix** (const cpl_wcs *wcs)
- void **ConvertCanvasToImage** (const double x_canvas, const double y_canvas, double *x_image, double *y_image)
- void **ConvertImageToCanvas** (const double x_image, const double y_image, double *x_canvas, double *y_canvas)
- void **set_RotateFlag** (const bool rot_flag)
- bool **get_RotateFlag** ()
- void **set_FlipXFlag** (const bool flip_flag)
- bool **get_FlipXFlag** ()
- void **set_FlipYFlag** (const bool flip_flag)
- bool **get_FlipYFlag** ()
- int **get_ImageWidth** ()
- void **set_ImageWidth** (const int image_width)
- int **get_ImageHeight** ()
- void **set_ImageHeight** (const int image_height)

3.3.3.9.2 Static Public Member Functions

- static std::string **RaDegToHMS** (double deg)
- static std::string **DecDegToDMS** (double deg)

3.3.3.9.3 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.9.4 Detailed Description

This file is part of the DDT Image Handling Library and provides functions for coordinate conversion purposes.

It offers functions to convert canvas coordinates to image coordinates and to retrieve world coordinates from canvas coordinates.

3.3.3.9.5 Constructor & Destructor Documentation

3.3.3.9.5.1 ImageCoords::ImageCoords (int *image_width*, int *image_height*, bool *rotate_flag*, bool *flip_x_flag*, bool *flip_y_flag*)

Constructor

3.3.3.9.5.2 virtual ddt::ImageCoords::~ImageCoords ()[virtual], [default]

Destructor

3.3.3.9.6 Member Function Documentation

3.3.3.9.6.1 void ImageCoords::ConvertCanvasToImage (const double *x_canvas*, const double *y_canvas*, double * *x_image*, double * *y_image*)

Converts canvas coordinates to image coordinates

3.3.3.9.6.1.1 Parameters

x_canvas	the canvas x position
y_canvas	the canvas y position
x_image	the resulting image x position
y_image	the resulting image y position

3.3.3.9.6.2 void ImageCoords::ConvertImageToCanvas (const double *x_image*, const double *y_image*, double * *x_canvas*, double * *y_canvas*)

Converts image coordinates to canvas coordinates

3.3.3.9.6.2.1 Parameters

x_image	the image x position
y_image	the image y position
x_canvas	the resulting canvas x position
y_canvas	the resulting canvas y position

3.3.3.9.6.3 std::string ImageCoords::DecDegToDMS (double *deg*)[static]

Get declination from degrees as a +D:M:S string representation

3.3.3.9.6.3.1 Parameters

deg	the declination in degrees

3.3.3.9.6.3.2 Returns

string containing the declination converted into +D:M:S representation

3.3.3.9.6.4 bool ImageCoords::get_FlipXFlag ()

Get the flip X flag.

3.3.3.9.6.4.1 Returns

the flip X flag (indicating if the image had been flipped horizontally)

3.3.3.9.6.5 bool ImageCoords::get_FlipYFlag ()

Get the flip Y flag.

3.3.3.9.6.5.1 Returns

the flip Y flag (indicating if the image had been flipped vertically)

3.3.3.9.6.6 int ImageCoords::get_ImageHeight ()

Get the image height

3.3.3.9.6.7 int ImageCoords::get_ImageWidth ()

Get the image width

3.3.3.9.6.8 bool ImageCoords::get_RotateFlag ()

Get the rotate flag.

3.3.3.9.6.8.1 Returns

the rotate flag (indicating if the image had been rotated)

3.3.3.9.6.9 std::string ImageCoords::RaDegToHMS (double *deg*)[static]

Get right ascension from degrees as a H:M:S string representation

3.3.3.9.6.9.1 Parameters

deg	the right ascension in degrees

3.3.3.9.6.9.2 Returns

string containing the right ascension converted into H:M:S representation

3.3.3.9.6.10 void ImageCoords::set_FlipXFlag (const bool *flip_flag*)

Set the flip X flag.

3.3.3.9.6.10.1 Parameters

flip_flag	the flip x flag to set

3.3.3.9.6.11 void ImageCoords::set_FlipYFlag (const bool *flip_flag*)

Set the flip Y flag.

3.3.3.9.6.11.1 Parameters

flip_flag	the flip y flag to set

3.3.3.9.6.12 void ImageCoords::set_ImageHeight (const int *image_height*)

Set the image height

3.3.3.9.6.12.1 Parameters

image_height	the image height to set

3.3.3.9.6.13 void ImageCoords::set_ImageWidth (const int *image_width*)

Set the image width

3.3.3.9.6.13.1 Parameters

image_width	the image width to set

3.3.3.9.6.14 void ImageCoords::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.9.6.14.1 Parameters

logger	The logger object

3.3.3.9.6.15 void ImageCoords::set_RotateFlag (const bool *rot_flag*)

Set the rotate flag.

3.3.3.9.6.15.1 Parameters

rot_flag	the rotate flag to set

3.3.3.9.6.16 const cpl_matrix * ImageCoords::WorldCoordinatesCdMatrix (const cpl_wcs * *wcs*)

Get the CD matrix from the WCS data.

3.3.3.9.6.16.1 Returns

a cpl_matrix with the CD information from the WCS data, or NULL on error

3.3.3.9.6.17 bool ImageCoords::WorldCoordinatesDegreesFromCanvas (cpl_wcs * *wcs*, const double *x_canvas*, const double *y_canvas*, double * *alpha*, double * *delta*)

Get the world coordinates in degrees from canvas coordinates

3.3.3.9.6.17.1 Parameters

wcs	the WCS object related to the image
x_canvas	the canvas x position
y_canvas	the canvas y position
alpha	the resulting world coordinates alpha value
delta	the resulting world coordinates delta value

3.3.3.9.6.17.2 Returns

bool indicating success or failure of coordinate fetching

3.3.3.9.6.18 bool ImageCoords::WorldCoordinatesDegreesFromImage (const cpl_wcs * *wcs*, const double *x_image*, const double *y_image*, double * *alpha*, double * *delta*)

Get the world coordinates in degrees from image coordinates

3.3.3.9.6.18.1 Parameters

wcs	the cpl_wcs object, containing the WCS information related to a CPL image
x_image	the image x position

wcs	the cpl_wcs object, containing the WCS information related to a CPL image
y_image	the image y position
alpha	the resulting world coordinates alpha value, or 0.0 on error
delta	the resulting world coordinates delta value, or 0.0 on error

3.3.3.9.6.18.2 Returns

bool indicating success or failure of coordinate fetching

3.3.3.9.7 Member Data Documentation

3.3.3.9.7.1 ddt::DdtLogger* ddt::ImageCoords::logger[protected]

The logger object

3.3.3.10 ddt::ImageHandling Class Reference

```
#include <imageHandling.hpp>
```

3.3.3.10.1 Public Types

- enum **CutLevelType** { **AUTO**, **MINMAX**, **USERDEFINED** }
- enum **ColorScalingType** { **LINEAR_SCALE**, **LOG_SCALE**, **SQRT_SCALE** }

3.3.3.10.2 Public Member Functions

- **ImageHandling** ()
- virtual **~ImageHandling** ()
- void **set_logger** (ddt::DdtLogger ***logger**)
- bool **LoadFile** (const std::string &file_name, int position=0)
- void **set_CutLevelType** (const **CutLevelType** cut_level_type)
- **CutLevelType get_CutLevelType** ()
- std::string **get_CutLevelTypeStr** ()
- bool **ApplyCutLevelsMinMax** ()
- bool **ApplyCutLevelsAuto** ()
- bool **ApplyCutLevelsManual** (const double min_value, const double max_value)
- bool **ResetCutLevels** ()
- bool **LoadColorMaps** (std::string color_map_folder, std::string color_map_name="standard")
- bool **LoadActualColorMap** (std::string color_map_name)
- bool **LoadActualConfigurationMap** (std::string configuration_map_name)
- bool **LoadConfigurationMaps** (std::string configuration_map_source)
- void **set_RotateFlag** (const bool rot_flag)
- bool **get_RotateFlag** ()
- void **set_FlipXFlag** (const bool flip_flag)
- bool **get_FlipXFlag** ()
- void **set_FlipYFlag** (const bool flip_flag)
- bool **get_FlipYFlag** ()
- void * **get_ImageData** ()
- cpl_image * **get_Image** ()
- int **get_ImageWidth** ()
- int **get_ImageHeight** ()
- void **set_CutLevelMin** (const double min_value)

- void **set_CutLevelMax** (const double max_value)
- double **get_CutLevelMin** ()
- double **get_CutLevelMax** ()
- double **get_ImagePixelMin** ()
- double **get_ImagePixelMax** ()
- std::list< std::string > **get_ColorMaps** ()
- **ddt::colorMap_t** * **get_ActualColorMap** ()
- **ddt::colorMapARGB_t** * **get_ActualColorMapARGB** ()
- std::string **get_ActualColorMapName** ()
- **ddt::scalingLut_t** * **get_ScalingLut** ()
- **ddt::scalingLut_t** * **get_LinearScalingLut** ()
- **ddt::scalingLut_t** * **get_LogarithmicScalingLut** ()
- **ddt::scalingLut_t** * **get_SqrtScalingLut** ()
- **ColorScalingType get_ColorScalingType** ()
- void **set_ColorScalingTypeLinear** ()
- void **set_ColorScalingTypeLogarithmic** ()
- void **set_ColorScalingTypeSquareRoot** ()
- void **set_ColorScalingType** (**ColorScalingType** scaling_type)
- std::string **get_ColorScalingTypeStr** ()
- double **ImagePixelFromImage** (const double x_image, double y_image)
- double **ImagePixelFromCanvas** (const double x_canvas, const double y_canvas, double *x_image, double *y_image)
- void **ConvertCanvasToImage** (const double x_canvas, const double y_canvas, double *x_image, double *y_image)
- void **ConvertImageToCanvas** (const double x_image, const double y_image, double *x_canvas, double *y_canvas)
- bool **WorldCoordinatesDegreesFromImage** (const double x_image, const double y_image, double *alpha, double *delta)
- bool **WorldCoordinatesDegreesFromCanvas** (const double x_canvas, const double y_canvas, double *alpha, double *delta)
- bool **get_AttachedToStream** ()
- bool **ReprossImage** ()
- void **AttachDataStream** (std::string data_stream_id)
- void **DetachDataStream** (std::string data_stream_id)
- cpl_apertures * **GetCircularObjects** (double sigma)
- bool **GetObjectCentroidPos** (cpl_apertures *apertures, double x_image, double y_image, double *x_centroid, double *y_centroid)
- bool **GetObjectInformation** (double x_image, double y_image, int size, double *x_axis_angle, double *background, double *peak_above_background, double *fwhm_x, double *fwhm_y)
- bool **GetGaussianParameters** (double x_image, double y_image, int size, double *background, double *gaussian_volume, double *correlation, double *gaussian_coord_x, double *gaussian_coord_y, double *sigma_x, double *sigma_y)
- std::string **GetEquinox** ()
- signal_t * **ImageAvailableInBufferSignal** ()
- signal_t * **CutLevelChangedSignal** ()
- std::vector< std::string > **GetFITSHeader** ()

3.3.3.10.3 Protected Attributes

- ddt::DdtLogger * **logger**

3.3.3.10.4 Detailed Description

Class to wrap the usage of image handling functions.

This class allows to load a data file or attach to a data stream. It offers access to image properties and the possibility to process the image data (e.g. flip / rotate etc.).

3.3.3.10.5 Member Enumeration Documentation

3.3.3.10.5.1 enum ddt::ImageHandling::ColorScalingType

To distinguish between color scaling types.

3.3.3.10.5.1.1 Enumerator:

LINEAR_SCALE	**Linear scaling. The LUT for pixel values is setup using a linear scaling. The LUT will scale the pixels inside the cut level range to the range 0-255.**
LOG_SCALE	Logarithmic scaling. The LUT for pixel values is setup using a logarithmic scaling. The LUT will scale the pixels inside the cut level range to the range 0-255.
SQRT_SCALE	Square root scaling. The LUT for pixel values is setup using a square root scaling. The LUT will scale the pixels inside the cut level range to the range 0-255.

3.3.3.10.5.2 enum ddt::ImageHandling::CutLevelType

To distinguish between cut level types.

3.3.3.10.5.2.1 Enumerator:

AUTO	**Auto cut level. The cut levels are determined by taking the min/max values of the median filtered image.**
MINMAX	Min/Max cut level. The cut levels are determined by taking the min/max values of the image.
USERDEFINED	Used defined cut levels.

3.3.3.10.6 Constructor & Destructor Documentation

3.3.3.10.6.1 ImageHandling::ImageHandling ()

Constructor

3.3.3.10.6.2 ImageHandling::~ImageHandling ()[virtual]

Destructor

3.3.3.10.7 Member Function Documentation

3.3.3.10.7.1 bool ImageHandling::ApplyCutLevelsAuto ()

Apply cut levels based on the median filtered image min/max values

3.3.3.10.7.1.1 Returns

true if applying the cut levels was successful, false otherwise

3.3.3.10.7.2 bool ImageHandling::ApplyCutLevelsManual (const double *min_value*, const double *max_value*)

Apply cut levels based manually

3.3.3.10.7.2.1 Returns

true if applying the cut levels was successful, false otherwise

3.3.3.10.7.3 bool ImageHandling::ApplyCutLevelsMinMax ()

Apply cut levels based on the image min/max values

3.3.3.10.7.3.1 Returns

true if applying the cut levels was successful, false otherwise

3.3.3.10.7.4 void ImageHandling::AttachDataStream (std::string *data_stream_id*)

Attach to a data stream

3.3.3.10.7.4.1 Parameters

data_stream_id	The ID of the data stream (could be the URI)

3.3.3.10.7.5 void ImageHandling::ConvertCanvasToImage (const double *x_canvas*, const double *y_canvas*, double * *x_image*, double * *y_image*)

Converts canvas coordinates to image coordinates

3.3.3.10.7.5.1 Parameters

x_canvas	the canvas x position
y_canvas	the canvas y position
x_image	the resulting image x position
y_image	the resulting image y position

3.3.3.10.7.6 void ImageHandling::ConvertImageToCanvas (const double *x_image*, const double *y_image*, double * *x_canvas*, double * *y_canvas*)

Converts image coordinates to canvas coordinates

3.3.3.10.7.6.1 Parameters

x_image	the image x position
y_image	the image y position
x_canvas	the resulting canvas x position
y_canvas	the resulting canvas y position

3.3.3.10.7.7 signal_t * ImageHandling::CutLevelChangedSignal ()

Provide the cut level changed signal

3.3.3.10.7.8 void ImageHandling::DetachDataStream (std::string *data_stream_id*)

Detach from a data stream

3.3.3.10.7.8.1 Parameters

data_stream_id	The ID of the stream to be disconnected

3.3.3.10.7.9 ddt::colorMap_t * ImageHandling::get_ActualColorMap ()

Get the actual color map

3.3.3.10.7.9.1 Returns

the actual color map

3.3.3.10.7.10 ddt::colorMapARGB_t * ImageHandling::get_ActualColorMapARGB ()

Get the actual color map as ARGB value array

3.3.3.10.7.10.1 Returns

the actual color map

3.3.3.10.7.11 std::string ImageHandling::get_ActualColorMapName ()

Get the name of the current colourmap

3.3.3.10.7.11.1 Returns

name of the current colourmap

3.3.3.10.7.12 bool ImageHandling::get_AttachedToStream ()

Get the attached to stream flag from the data acquisition

3.3.3.10.7.12.1 Returns

the flag indicating if the data acquisition is attached to a data stream

3.3.3.10.7.13 std::list< std::string > ImageHandling::get_ColorMaps ()

Get the list of loaded color maps;

3.3.3.10.7.13.1 Returns

the list of currently loaded color maps

3.3.3.10.7.14 ImageHandling::ColorScalingType ImageHandling::get_ColorScalingType ()

Get the color scalingtype.

3.3.3.10.7.14.1 Returns

the currently selected color scaling type

3.3.3.10.7.15 std::string ImageHandling::get_ColorScalingTypeStr ()

Get a string representaiton of the color scaling type.

3.3.3.10.7.15.1 Returns

the currently selected color scaling type as string

3.3.3.10.7.16 double ImageHandling::get_CutLevelMax ()

Get the cut level maximum

3.3.3.10.7.17 double ImageHandling::get_CutLevelMin ()

Get the cut level minimum

3.3.3.10.7.18 ImageHandling::CutLevelType ImageHandling::get_CutLevelType ()

Get the cut level type.

3.3.3.10.7.18.1 Returns

the currently selected cut level type

3.3.3.10.7.19 std::string ImageHandling::get_CutLevelTypeStr ()

Get a string representation of the cut level type.

3.3.3.10.7.19.1 Returns

the currently selected cut level type as string

3.3.3.10.7.20 bool ImageHandling::get_FlipXFlag ()

Get the flip X flag.

3.3.3.10.7.20.1 Returns

the flip X flag (indicating if the image had been flipped horizontally)

3.3.3.10.7.21 bool ImageHandling::get_FlipYFlag ()

Get the flip Y flag.

3.3.3.10.7.21.1 Returns

the flip Y flag (indicating if the image had been flipped vertically)

3.3.3.10.7.22 cpl_image * ImageHandling::get_Image ()

Get the image data

3.3.3.10.7.23 void * ImageHandling::get_ImageData ()

Get the image data

3.3.3.10.7.24 int ImageHandling::get_ImageHeight ()

Get the image height

3.3.3.10.7.25 double ImageHandling::get_ImagePixelMax ()

Get the image pixel maximum

3.3.3.10.7.26 double ImageHandling::get_ImagePixelMin ()

Get the image pixel minimum

3.3.3.10.7.27 int ImageHandling::get_ImageWidth ()

Get the image width

3.3.3.10.7.28 ddt::scalingLut_t * ImageHandling::get_LinearScalingLut ()

Get the LUT for linear color scaling

3.3.3.10.7.28.1 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.10.7.29 ddt::scalingLut_t * ImageHandling::get_LogarithmicScalingLut ()

Get the LUT for logarithmic color scaling

3.3.3.10.7.29.1 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.10.7.30 bool ImageHandling::get_RotateFlag ()

Get the rotate flag.

3.3.3.10.7.30.1 Returns

the rotate flag (indicating if the image had been rotated)

3.3.3.10.7.31 ddt::scalingLut_t * ImageHandling::get_ScalingLut ()

Get the scaling LUT, depending on the current scaling type

3.3.3.10.7.31.1 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.10.7.32 ddt::scalingLut_t * ImageHandling::get_SqrtScalingLut ()

Get the LUT for square root color scaling

3.3.3.10.7.32.1 Returns

a scalingLut object (an array of 64K uint values)

3.3.3.10.7.33 cpl_apertures * ImageHandling::GetCircularObjects (double *sigma*)

Get a list of objects from a CPL image

3.3.3.10.7.33.1 Parameters

sigma	the detection level

3.3.3.10.7.33.2 Returns

list of cpl_apertures objects

3.3.3.10.7.34 std::string ImageHandling::GetEquinox ()

Return the current equinox information related to the image, if available

3.3.3.10.7.34.1 Returns

the equinox information

3.3.3.10.7.35 std::vector< std::string > ImageHandling::GetFITSHeader ()

Get the FITS header items as a vector of strings

3.3.3.10.7.35.1 Returns

a vector of strings containing the FITS header items

3.3.3.10.7.36 bool ImageHandling::GetGaussianParameters (double *x_image*, double *y_image*, int *size*, double * *background*, double * *gaussian_volume*, double * *correlation*, double * *gaussian_coord_x*, double * *gaussian_coord_y*, double * *sigma_x*, double * *sigma_y*)

Get parameters from 2D gaussian fit

3.3.3.10.7.36.1 Parameters

x_image	the image x position
y_image	the image y position
size	the size of the rectangle around the image position to consider
background	the background level, or 0.0 on error
gaussian_volume	the gaussian volumne, or 0.0 on error
correlation	the correlation, or 0.0 on error
gaussian_coord_x	the x coordinate of the mid of the gaussian fit
gaussian_coord_y	the y coordinate of the mid of the gaussian fit
sigma_x	the variance of gaussian fit in x direction
sigma_y	the variance of gaussian fit in y direction

3.3.3.10.7.36.2 Returns

true on success, otherwise false

3.3.3.10.7.37 bool ImageHandling::GetObjectCentroidPos (cpl_apertures * *apertures*, double *x_image*, double *y_image*, double * *x_centroid*, double * *y_centroid*)

Find an aperture object that matches certain image coordinates and return the centroid position

3.3.3.10.7.37.1 Parameters

apertures	list of cpl_apertures objects
x_image	the image x position
y_image	the image y position
x_centroid	the centroid x coordinate, or -1.0 on error
y_centroid	the centroid y coordinate, or -1.0 on error

3.3.3.10.7.37.2 Returns

true on success, otherwise false

3.3.3.10.7.38 bool ImageHandling::GetObjectInformation (double *x_image*, double *y_image*, int *size*, double * *x_axis_angle*, double * *background*, double * *peak_above_background*, double * *fwhm_x*, double * *fwhm_y*)

Get information about an aperture object

3.3.3.10.7.38.1 Parameters

x_image	the image x position
y_image	the image y position
size	the size of the rectangle around the image position to consider

x_image	the image x position
x_axis_angle	the angle of the x_axis (degrees), or 0.0 on error
background	the background level, or 0.0 on error
peak_above_background	the peak above background value, or 0.0 on error
fwhm_x	the FWHM along the x axis, or -1.0 on error
fwhm_y	the FWHM along the y axis, or -1.0 on error

3.3.3.10.7.38.2 Returns

true on success, otherwise false

3.3.3.10.7.39 signal_t * ImageHandling::ImageAvailableInBufferSignal ()

Provide the image available in buffer signal

3.3.3.10.7.40 double ImageHandling::ImagePixelFromCanvas (const double *x_canvas*, const double *y_canvas*, double * *x_image*, double * *y_image*)

Get the image pixel value from the position expressed as canvas coordinates

3.3.3.10.7.40.1 Parameters

x	the canvas x position
y	the canvas y position
x_image	the resulting image x position
y_image	the resulting image y position

3.3.3.10.7.40.2 Returns

the pixel value (converted to double) from the canvas coordinates

3.3.3.10.7.41 double ImageHandling::ImagePixelFromImage (const double *x_image*, double *y_image*)

Get the image pixel value from the position expressed as image coordinates

3.3.3.10.7.41.1 Parameters

x_image	the image x position
y_image	the image y position

3.3.3.10.7.41.2 Returns

the pixel value (converted to double) from the image coordinates

3.3.3.10.7.42 bool ImageHandling::LoadActualColorMap (std::string *color_map_name*)

Load the supplied color map as actual color map.

3.3.3.10.7.42.1 Parameters

color_map_name	the name of the color map to be loaded. The color map names are derived from the color map file names.

3.3.3.10.7.42.2 Returns

true if loading was successful, false otherwise

3.3.3.10.7.43 bool ImageHandling::LoadActualConfigurationMap (std::string *configuration_map_name*)

Load the supplied configuration map as the actual configuration map.

3.3.3.10.7.43.1 Parameters

configuration_map_name	the name of the configuration map to be loaded.

3.3.3.10.7.43.2 Returns

true if loading was successful, false otherwise

3.3.3.10.7.44 bool ImageHandling::LoadColorMaps (std::string *color_map_folder*, std::string *color_map_name* = "standard")

Load the color maps from the specified folder into the color_maps member. If possible, the supplied color map is then loaded as the actual color map.

3.3.3.10.7.44.1 Parameters

color_map_folder	the folder containing color map files. These files (with extension .lut or .lasc) should contain 256 RGB triples (in the form: r.rrrr g.gggg b.bbbb, where each value is in the range 0.0 and 1.0).
color_map_name	the name of the color map to be loaded. The color map names are derived from the color map file names.

3.3.3.10.7.44.2 Returns

true if loading was successful, false otherwise

3.3.3.10.7.45 bool ImageHandling::LoadConfigurationMaps (std::string *configuration_map_source*)

Load the configuration maps from the specified configuration map source.

3.3.3.10.7.45.1 Parameters

config_map_source	the source of the configuration maps

3.3.3.10.7.46 bool ImageHandling::LoadFile (const std::string & *file_name*, int *position* = 0)

Load file. This function takes a CPL file (FITS) name and a position. The position indicates the HDU to use. The default HDU is the first (position 0).

3.3.3.10.7.46.1 Parameters

file_name	the FITS file name to load
position	the position indicating the HDU to use. This defaults to 0 (the first HDU).

3.3.3.10.7.46.2 Returns

true if loading was successful, false otherwise

3.3.3.10.7.47 bool ImageHandling::ReprossImage ()

Apply all processing flags and cut levels on the image

3.3.3.10.7.48 bool ImageHandling::ResetCutLevels ()

Reset cut levels

3.3.3.10.7.48.1 Returns

true if resetting the cut levels was successful, false otherwise

3.3.3.10.7.49 void ImageHandling::set_ColorScalingType (ColorScalingType *scaling_type*)

Set the actual color scaling type

3.3.3.10.7.49.1 Parameters

scaling_type	the scaling type to be set

3.3.3.10.7.50 void ImageHandling::set_ColorScalingTypeLinear ()

Set the scaling type to LINEAR_SCALE.

3.3.3.10.7.51 void ImageHandling::set_ColorScalingTypeLogarithmic ()

Set the scaling type to LOG_SCALE.

3.3.3.10.7.52 void ImageHandling::set_ColorScalingTypeSquareRoot ()

Set the scaling type to SQRT_SCALE.

3.3.3.10.7.53 void ImageHandling::set_CutLevelMax (const double *max_value*)

Set the cut level maximum

3.3.3.10.7.54 void ImageHandling::set_CutLevelMin (const double *min_value*)

Set the cut level minimum

3.3.3.10.7.55 void ImageHandling::set_CutLevelType (const CutLevelType *cut_level_type*)

Set the cut level type.

3.3.3.10.7.56 void ImageHandling::set_FlipXFlag (const bool *flip_flag*)

Set the flip X flag.

3.3.3.10.7.57 void ImageHandling::set_FlipYFlag (const bool *flip_flag*)

Set the flip Y flag.

3.3.3.10.7.58 void ImageHandling::set_logger (ddt::DdtLogger * *logger*)

Set logger

3.3.3.10.7.58.1 Parameters

logger	The logger object

3.3.3.10.7.59 void ImageHandling::set_RotateFlag (const bool *rot_flag*)

Set the rotate flag.

3.3.3.10.7.60 bool ImageHandling::WorldCoordinatesDegreesFromCanvas (const double *x_canvas*, const double *y_canvas*, double * *alpha*, double * *delta*)

Get the world coordinates in degrees from canvas coordinates

3.3.3.10.7.60.1 Parameters

x_canvas	the canvas x position
y_canvas	the canvas y position
alpha	the resulting world coordinates alpha value
delta	the resulting world coordinates delta value

3.3.3.10.7.60.2 Returns

bool indicating success or failure of coordinate fetching

3.3.3.10.7.61 bool ImageHandling::WorldCoordinatesDegreesFromImage (const double *x_image*, const double *y_image*, double * *alpha*, double * *delta*)

Get the world coordinates in degrees from image coordinates

3.3.3.10.7.61.1 Parameters

x_image	the image x position
y_image	the image y position
alpha	the resulting world coordinates alpha value
delta	the resulting world coordinates delta value

3.3.3.10.7.61.2 Returns

bool indicating success or failure of coordinate fetching

3.3.3.10.8 Member Data Documentation

3.3.3.10.8.1 ddt::DdtLogger* ddt::ImageHandling::logger[protected]

The logger object

3.3.3.11 ddt Namespace Reference

3.3.3.11.1 Classes

- class **ConfigurationMaps**
- class **ConfigurationMapsAccess**
- class **ConfigurationMapsAccessFiles**
- class **DataAcquisition**
- class **DataFile**
- class **ImageBuffer**
- class **ImageColor**
- class **ImageCoords**
- class **ImageHandling**
- struct **scalingLut_t**

3.3.3.11.2 Typedefs

- typedef std::array< std::array< float, 3 >, 256 > **colorMap_t**
- typedef std::array< unsigned int, 256 > **colorMapARGB_t**
- typedef struct **ddt::scalingLut_t scalingLut_t**

3.3.3.11.3 Typedef Documentation

3.3.3.11.3.1 typedef std::array<std::array<float, 3>, 256> ddt::colorMap_t

Color map type definition. The color map type consists of a 256 element array. Each array element itself consists of an array of 3 float values, where these float values represent the red, green and blue part, all expressed as a value between 0.0 and 1.0)

3.3.3.11.3.2 typedef std::array<unsigned int, 256> ddt::colorMapARGB_t

Definition of a color map type containing the RGB values as unsigned int values. This color map type consists of a 256 element array. Each array element itself consists of an unsigned int value, where each values represents an ARGB quadruplet in the form #AARRGBB with AA being the alpha channel that defaults to FF (opaque). The red, green and blue parts are between 0 and 255)

3.3.3.11.3.3 typedef struct ddt::scalingLut_t ddt::scalingLut_t

The scaling Lookup-Table type definition. The scaling LUT consists of a 65536 element array of uint values. The LUT is used to assign each pixel value another pixel values that e.g. was computed using a logarithmic function. In addition, an offset and a factor are contained. These are used to adjust value ranges that do not fit to the given LUT type and dimension.

3.3.3.12 ddt::scalingLut_t Struct Reference

```
#include <imageColor.hpp>
```

3.3.3.12.1 Public Attributes

- std::array< uint, 65536 > **lut**
- double **offset**
- double **factor**

3.3.3.12.2 Detailed Description

The scaling Lookup-Table type definition. The scaling LUT consists of a 65536 element array of uint values. The LUT is used to assign each pixel value another pixel value that e.g. was computed using a logarithmic function. In addition, an offset and a factor are contained. These are used to adjust value ranges that do not fit to the given LUT type and dimension.

3.3.3.12.3 Member Data Documentation

3.3.3.12.3.1 double ddt::scalingLut_t::factor

Factor to multiply the pixel values with in order to adjust the image pixel value range to the ushort value range.

3.3.3.12.3.2 std::array<uint, 65536> ddt::scalingLut_t::lut

The lookup table. It consists of 65536 uint values.

3.3.3.12.3.3 double ddt::scalingLut_t::offset

Offset to be added to the pixel values in order to shift the image pixel value range it into the positive.

3.3.4 Python Bindings

The python bindings are in an early stage of development and we currently cannot extract a proper documentation for them. We will provide the full documentation in an upcoming release. For the moment, we give a very short introduction to them and a small example usage to get started and invite the developers to check the code for the available APIs.

The Python bindings can be used to access the public API of the DDT software. Currently there are bindings for the classes `DdtDataPublisher`, `DdtDataSubscriber`, `DdtLogger` and `DdtStatistics`. Other bindings will be provided with future releases.

The bindings for the `DdtDataPublisher` and the `DdtDataSubscriber` can be found in the Python module `DdtDataTransfer`. This module also contains some utility bindings for non-standard types that are used by the `DdtDataPublisher` or the `DdtDataSubscriber`. We present after a very short introduction to this module.

Bindings for the `DdtLogger` and the `DdtStatistics` can be found in the Python module `DdtUtils`. This module is not yet described in this documentation but will be done in an upcoming release.

3.3.4.1 DdtDataTransfer module

The core of this module are the bindings for the `DdtDataPublisher` and the `DdtDataSubscriber` which provide the possibility to create Python publisher and subscriber. A list of available methods can be found in 3.3.1.1. The only difference is the construction of the `DdtDataPublisher` / `DdtDataSubscriber`, since no factory is used on the Python side (see section Example Usage below).

The module also contains bindings for the `DataSample` which is returned by the `ReadData()` method of the subscriber. Its constructor takes three input values: The ID of the sample, the length of the meta data vector and the length of the data vector. The fields, which can be seen in the code snippet below, are all accessible from Python side. Instead of `std::vector` a python list is used in Python code, which gets automatically translated.

```
/**
* Topic ID to identify the kind of meta data
*/
int32_t topic_id;
/**
* The length of the meta data blob
*/
int32_t meta_data_length;
/**
* The meta data as vector of bytes
*/
std::vector<uint8_t> meta_data;
/**
* Sample ID to uniquely identify the data sample
*/
int32_t sample_id;
/**
* The image data as vector of bytes
*/
std::vector<uint8_t> data;
```

Attributes of the DataSample class

The meta data used by the `DataSample` can be created by using the bindings of one of the classes `DdtEncDecBiDim`, `DdtEncDecMultiDim` or `DdtEncDecMultiLayer`. Alternatively, the class `DdtEncDec` can be extended. For encoding the meta data the classes provide the `encode(…)` method which also adds a UTC timestamp to the meta data vector. These vectors can be decoded with the `decode(length, metaData)` method, which takes the length of the meta data vector and the vector itself as input. After encoding or decoding the values can be accessed via the corresponding getter-methods of the classes.

For the connection management the module also contains bindings for `boost::signals2::connection`. The connection is returned by the method `connect()` of the `DdtDataSubscriber`. The only bound method however is `disconnect()` which closes the connection and removes the subscriber from the list of listeners for available data.

3.3.4.2 Example Usage

For using the bindings, both the `DdtUtils` and the `DdtDataTransfer` modules must be imported. When creating a subscriber (`DdtDataSubscriber`) or a publisher (`DdtDataPublisher`), a logger (`DdtLogger`) needs to be instantiated before.

```
>>> logger = DdtUtils.DdtLogger("testLogger")
>>> publisher = DdtDataTransfer.DdtDataPublisher(logger)
```

Before registering the publisher to the broker, the size of the buffer needs to be set. Otherwise the registration will fail.

The subscriber needs to receive the `DataAvailable` signal. To bind a method to this signal, which is called each time new data arrives, the `connect` method of the subscriber needs to be called with the call-back method as argument.

```
>>> def call_this():
...     print("Hi there!")
...
>>> connection = subscriber.connect(call_this)
```

For sending data from the publisher to the subscriber the publisher needs to write data and explicitly publish it. After calling the `publishData` method the call-back method of the subscriber will be called.

A full example can be seen in the following images:

```
>>> import DdtUtils
>>> import DdtDataTransfer
>>> logger = DdtUtils.DdtLogger("testLogger")
>>> publisher = DdtDataTransfer.DdtDataPublisher(logger)
DEBUG - DDT_LOGGER - DdtDataPublisher: config file: /eelt/ddt/0.1/resource/config/datatransferlib.ini
DEBUG - DDT_LOGGER - reply time for MAL clients [s]: 6
>>> publisher.SetBufferSize(1000, 10)
>>> publisher.RegisterPublisher("zpb.rr://127.0.0.1:5001/broker/Broker1", "stream1")
DEBUG - DDT_LOGGER - DdtDataPublisher parameters:
DEBUG - DDT_LOGGER - Data_stream_identifier: stream1
DEBUG - DDT_LOGGER - Latency: 10000 [ms]
DEBUG - DDT_LOGGER - Deadline: 10 [s]
DEBUG - DDT_LOGGER - Max_data_sample_size: 1000
DEBUG - DDT_LOGGER - Number_of_samples: 10
DEBUG - DDT_LOGGER - DdtDataPublisher: Broker connection established
DEBUG - DDT_LOGGER - DdtDataTransferLib: HeartbeatThread started
DEBUG - DDT_LOGGER - DdtMemoryAccessor: Opened shared memory with id stream1
DEBUG - DDT_LOGGER - DdtDataPublisher: DdtMemoryAccessor created
1
>>> publisher.WriteData(1, [42], [1])
DEBUG - DDT_LOGGER - DdtMemoryWriter:
DEBUG - DDT_LOGGER - circ_buffer->at(0).data_stream_identifier: stream1
DEBUG - DDT_LOGGER - circ_buffer->at(0).checksum: 0
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample_length: 1
DEBUG - DDT_LOGGER - circ_buffer->at(0).writer_index: 0
DEBUG - DDT_LOGGER - circ_buffer->at(0).timestamp: 1606931577398
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample.sample_id: 1
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample.data: 42
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample.topic_id: 0
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample.meta_data_length: 1
DEBUG - DDT_LOGGER - circ_buffer->at(0).sample.metadata: 1
>>> publisher.PublishData()
```

Figure 29: Example for Python publisher

```
>>> import DdtUtils
>>> import DdtDataTransfer
>>> logger = DdtUtils.DdtLogger("testLogger")
>>> subscriber = DdtDataTransfer.DdtDataSubscriber(logger)
DEBUG - DDT_LOGGER - DdtDataSubscriber: config file: /eelt/ddt/0.1/resource/config/datatransferlib.ini
DEBUG - DDT_LOGGER - max age data sample [ms]: 2000
DEBUG - DDT_LOGGER - reply time for MAL clients [s]: 6
>>> subscriber.RegisterSubscriber("zpb.rr://127.0.0.1:5001/broker/Broker1", "stream1")
DEBUG - DDT_LOGGER - DdtDataSubscriber: Broker connection established
DEBUG - DDT_LOGGER - DdtDataSubscriber parameters:
DEBUG - DDT_LOGGER - subscriber_uuid: 56693e13-c1ab-4721-a69a-b3ca93e676d3
DEBUG - DDT_LOGGER - data_stream_identifier: stream1
DEBUG - DDT_LOGGER - remote_broker_uri:
DEBUG - DDT_LOGGER - latency [ms]: 10000
DEBUG - DDT_LOGGER - deadline [s]: 10
DEBUG - DDT_LOGGER - DdtMemoryAccessor: Opened shared memory with id stream1
DEBUG - DDT_LOGGER - DdtDataSubscriber: DdtMemoryAccessor created
INFO - DDT_LOGGER - DdtDataSubscriber: Start reading...
DEBUG - DDT_LOGGER - DdtDataTransferLib: HeartbeatThread started
DEBUG - DDT_LOGGER - DdtDataSubscriber: notification_uri: zpb.ps://127.0.0.1:5101/stream1
DEBUG - DDT_LOGGER - DdtDataSubscriber: NotificationSubscriber initialized
INFO - DDT_LOGGER - DdtDataSubscriber: Subscriber registered
1
DEBUG - DDT_LOGGER - DdtDataSubscriber: Notification subscription started...
>>> def call_this():
...     print("Hi there!")
...
>>> connection = subscriber.connect(call_this)
>>> Hi there!

>>> sample = subscriber.ReadData()
WARN - DDT_LOGGER - Data Sample is dropped because the specified time window is exceeded.
WARN - DDT_LOGGER - writer index: 0
>>> connection.disconnect()
```

Figure 30: Example for Python subscriber

3.4 Commands and parameters

In this section a summary of the available command line tools and their arguments is given.

Command	Description
ddtBroker --uri <Server URI of the broker> [--debug]	Starts a Data Broker on the host. The argument specifies the server URI of the broker that can be used by connecting Publisher / Subscribers.
ddtPublisherSimulator --broker <local broker URI> --datastream <data stream ID> --interval <publishing interval> --buffer_size <Ring buffer elements> --mode <simulator mode> --image_folder <path to sample FITS files for the transfer> --checksum <0/1>	Starts a Publisher Simulator which will connect to the local broker with the specified local broker URI. Data for the specified data stream ID will be published. The connected broker shall publish the data via the network. Internally a notification port is used to notify local subscribers about new data. This port is taken from a configurable port range (see above). The publishing interval is set in milliseconds. The number of elements of the ring buffer for the shared memory can be set. It defaults to 4. The mode argument can be used to switch between different simulation modes. Mode 1 requires the specification of an image folder containing FITS images. For more details check the section on the Data Transfer

Command	Description
	above. The checksum argument can be used to enable/disable the calculation of checksums for Data Samples.
ddtSubscriberSimulator --broker <Local Broker URI> --datastream <Data Stream ID> --interval <reading interval> [--remote <URI Remote Broker>] –mode <mode> [--statistics <0/1>]	Starts a Subscriber Simulator. It will connect to the local broker on the local broker URI. It will subscribe for data of the given stream identifier. When it should connect to a remote Publisher the URI to the broker of this publisher can be specified optionally. The reading interval in milliseconds is optional and should only be used to simulate slow readers. The mode argument can select different simulation modes. The subscriber simulator should always use the same mode as its publisher simulator. The statistics flag can be used to enable / disable the calculation of transfer statistics (latency etc.).
ddtViewer [--filename <path to image file> OR –datastream “<connection details of data stream>”] [--debug] [--remotecontrol_uri <uri>]	Start the DDT Standard Viewer. An optional filename to an image file can be given to load the file at start-up of the viewer. Instead of this it is also possible to add the URI path to a Data Stream to automatically attach to that stream at start-up. The connection string is made up of the local broker URI, the datastream ID and the optional remote broker URI. The optional DEBUG flag can be set to increase the log level. The optional flag for the remote control uri can be used to activate a CII MAL server on the given URI over which remote commands can be send to the viewer.

Table 46: Commands list

All applications also support the option “`--help`" to give a list of possible command line options. The help text also contains examples for the various arguments.

3.5 Log configuration

The log configuration is done in a configuration file for Log4CPlus.

The configuration file can be edited in a text editor.

The default configuration contains 2 loggers, one console logger and one file logger.

The log configuration files are stored in the directory `$DDT_LOGCONFIG_PATH`. Each application uses its own configuration file. The filename of the configuration file is made up of the prefix "`log4cplus_`" followed by the application name in lower case letters and the extension "`.properties`" (e.g. `log4cplus_ddtsubscribersimulator.properties`)

An example of the log configuration looks like this:

```
# Set options for appender named "ROLLING"
# ROLLING should be a RollingFileAppender, with maximum file size of 10 MB
# using at most one backup file. ROLLING's layout is TTCC, using the
# ISO8061 date format with context printing enabled.
log4cplus.rootLogger=INFO, ROLLING, STDOUT

log4cplus.appender.ROLLING=log4cplus::RollingFileAppender
log4cplus.appender.ROLLING.MaxFileSize=10MB
log4cplus.appender.ROLLING.MaxBackupIndex=1
log4cplus.appender.ROLLING.layout=log4cplus::PatternLayout
log4cplus.appender.ROLLING.layout.ConversionPattern=%d{%FT%T.%q} %-5p %m%n
log4cplus.appender.ROLLING.File=${HOME}/ddtDataBroker.log

log4cplus.appender.STDOUT=log4cplus::ConsoleAppender
log4cplus.appender.STDOUT.layout=log4cplus::PatternLayout
log4cplus.appender.STDOUT.layout.ConversionPattern=%d{%FT%T.%q} %-5p %m%n
```

The log level for all log messages can be defined to use the level `DEBUG`, `INFO`, `WARNING`, `ERROR` or `FATAL`. In the example there is only one log level set. It could also be set differently for different loggers.

The console logger in the example uses the name "`STDOUT`", while the file logger uses a rolling file and is called "`ROLLING`".

The filename of the rolling file is configured in the item "`log4cplus.appender.ROLLING.File`". In the example it is set to a file in the user's home directory. The maximum size of the logfile is set to 10 MB and the number of backup files is set to 1.

Various other logger settings can be configured according on the Log4CPlus documentation on https://github.com/log4cplus/log4cplus.

< Last Page of Document >