DDT

DDT User Manual

DUM

Version:	5
Date:	2022-05-24
Reference:	CGI-MAN-00026
Total Pages:	89

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	2022-05-24
Checked	Angelika Stalitza	DDT Product Assurance Manager		2022-05-24
Approval	Jean-Christophe Berthon	DDT Project Manager		2022-05-24

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	2021-11-25	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

Issue	Date	Author	Reference	Changed Pages/Paragraphs
				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
				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.
		C. Mannel		Section 3.3.2.2.9: Added section on Graphics Control Dialog
				Section 3.3.2.4: Updated the commandline options for the DDT Viewer and the screenshot of the GUI.
				Section 3.4: Update the commadline options for the DDT Viewer.
				Section 3.3.2.2: Completed list of dialog IDs
				Section 3.3.2.2.1: Updated screenshot of Pick Object Dialog and description of some features of the dialog. Also added description for new parameter IDs.
				Section 3.3.2.1: Updated screenshot of DDT Viewer.
				Section 3.3.2.2.8: Added parameters for rotation angle and line style to screenshot and description.
				Section 3.3.2.6: Section on rendering libraries added.
				Section 3.3.2.1.1: Added description of public methods.
				Section 3.3.2.1.10: Updated screenshot and description of properties
				Section 3.2.1.9: Formatting only
				Section 3.3.2.2.17: Section on Distance Dialog added
				Section 3.3.2.2.18: Section of Magnification Dialog added
				Section 3.3.2.1.6: Description of “AUTO” and “FIT” scales was updated.
		C. Bortlisz		Section 3.3.2.2.16 added (Save Image Dialog)

Issue	Date	Author	Reference	Changed Pages/Paragraphs
				Section 3.3.2.1.11 added (3D Cube Navigation Widget)
				Section 3.3.1.1.4 updated (added description of modes 6 and 7 of the Publisher Simulator)
				Section 3.3.2.3 updated (added graphical element overlay type DDT_OVERLAY_IMAGE)
				Section 3.3.1.1.5 updated (corrected and extended (data dump option) the command line arguments description for the Subscriber Simulator. Section 3.3.1.1.6 added (snapshot data publishing).
		M. Grimm		Section 3.3.2.5: Updated the complete section.
				Section 3.3.1.1.5: Renamed frequency to interval
				Section 3.3.1.1.1.3 Table 9: Functions of DdtDataSubscriber: Renamed frequency to interval and updated the description of the ReadData() function.
		M. Pfeil		Section 3.3.4: Updated introduction, added tables showing the content of the modules
				Section 3.3.4.2: Added this section for the broker module
				Section 3.3.4.3: Added this section for the data visualisation module
				Section 3.3.4.4: Added this section for the image handling module
				Section 3.3.4.5: Added this section for the remote API module
		C. Mannel		Section 3.2: Updated installation instructions.
				Section 3.3.1.2: Added section on meta-data encoding/decoding
5	2022-02-04	C.Mannel		New version for PAVV Closeout Datapack
		M. Grimm	MKI-10	Section 3.3.1.1.1: Added information about the latency definition.
			GZI-004	Section 3.3.1.1.1.2: Cells of RegisterPublisher() method merged.
			GZI-0027	Section 3.3.2.2.18: Fixed the reference.
			GZI-0021	Section 3.3.2.2.3: Updated figure 16 FITS Header Dialog.
			GZI-0028	Section 3.3.2.2.20 added (Mark / End mark position).
			PSI-2	Section 3.3.1.1.6: Added a note that the dump folder must exist.
			MKI-4	Section 3.3.1.2: Added the ‘labels’ field for multi-dimensional arrays.
			DDT-324	Section 3.3.1.2: Updated to current software implementation: Added ‘description’ field in MetaDataBase; Added ‘configuration_map_name’ field for multi-dimensional arrays. Removed ‘descrption’ field from MetaDataElementsImage3D.
			AHO-1	Section 3.3.1.1.2: New lines added in example code.
			AHO-2	Section 3.3.1.1.4: Corrected datastream and buffer_size options.
			AHO-5	Section 3.3.1.1.4: Updated publisher simulator example. Section 3.3.1.1.5: Updated subscriber simulator example
			AHO-28 GCH-11	Section 3.3.2.3: Updated Table 49: API function of the Overlay API

Issue	Date	Author	Reference	Changed Pages/Paragraphs
			GCH-02	Section 1.1: Removed the sentence.
			DDT-308	Section 3.3.1.1.1.2: Added overloaded WriteData() method.
			DDT-392	Section 3.3.1.1.3: Added a description for the "config_file" parameter and updated the command line example.
			DDT-301	Section 3.3.1.1.3: Removed the path element /broker from the example and specified of which elements the URI exists. Section 3.3.1.1.5: Updated the description of the URI.
			MKI-1 AHO-32	Sections 3.3.3.1 to 3.3.3.12 removed.
			MKI-3	Section 3.3.1.2: Moved the WCS information to the Image2D and Image3D structs.
			GCH-14	Section 3.3.3: Renamed imgHandling->ReprossImage() to imgHandling->ReprocessImage()
			AHO-29	Section 3.3.2.1: Changed “classed” to “classes”.
			AHO-8	Section 3.3.2.1: Added a sentence regarding the source and target of the signal / slot pairs.
			MCO-6	Section 3.3.2.3: Added a description with an example of how to configure the position of graphical elements.
			GZI-30	Section 3.3.4.1.2: Updated the section with new examples. Section 3.3.1.1.2: Added a more detailed description for the max_age_data_sample parameter.
			AHO-30	From section 3.3.4 to the end: Transferred all code listings to the correct format.
			AHO-27 GZI-31	Fixed all reference issues in the document.
		C. Bortlisz	DDT-310	Section 3.3.1.1.1.1 updated. Added additional functions to create a DdtDataPublisher / DdtDataSubscriber, taking a log4cplus logger as input.
			DDT-304	Section 3.3.2.2.6 updated. Renamed “FITS Table Dialog” to “Binary Table Dialog”. Corrected parameter list for the dialog.
			MCO-2	Section 3.3.5 added.
			GZI-8	Section 3.3.1.1.4: Added a sentence with a link to section 3.3.5.

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Table of Contents

1 Introduction 12
1.1 Purpose and Scope 12

2 Documents 13
2.1 Applicable Documents 13
2.2 Reference Documents 13
2.3 Definition of Terms and Acronyms 13

3 Data Display Tool 15
3.1 Purpose of the software 15
3.2 Software Installation 16
3.2.1 Building and Installing DDT 16
3.3 Software components 17
3.3.1 Data Transfer 17
3.3.2 Data Visualisation 30
3.3.3 Image Handling 74
3.3.4 Python Bindings 75
3.3.5 Configuration map handling 84
3.4 Commands and parameters 86
3.5 Log configuration 88

< Last Page of Document > 89

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

List of Tables

Table 1: Document Signature Table 2
Table 2: Document Change Record 6
Table 3: Internal Distribution List 7
Table 4: External Distribution List 7
Table 5: Applicable Documents 13
Table 6: Reference Documents 13
Table 7: Terms for the Data Display Tool Framework 14
Table 8: Environment variables defined by the DDT 17
Table 9: Functions of DdtDataTransferFactory 19
Table 10: Functions of DdtDataSubscriber 20
Table 11: Datatransfer Library configuration file 21
Table 12: Data Broker configuration file 23
Table 13: Properties of the Image Widget 32
Table 14: Public methods of the Image Widget 33
Table 15: Signals and Slots of the Image Widget 36
Table 16: Signals and Slots of the Data Stream Widget 37
Table 17: Properties of the Flip / Rotate Widget 37
Table 18: Signals and slots of the Flip / Rotate Widget 37
Table 19: Signals and slots of the Scale Buttons Widget 38
Table 20: Signals and slots of the Panning Widget 39
Table 21: Signals and Slots of the Image Scale Widget 40
Table 22: Signals and Slots of the Colourmap Widget 40
Table 23: Signals and Slots for the Cursor Information Widget 41
Table 24: Signals and Slots for the Cut Values Widget 42
Table 25: Signals and Slots of the Magnification Widget 43
Table 26: List of dialog IDs 44
Table 27: Signals and Slots of the dialog base class 45
Table 28: Pick modes in the Pick Object Dialog 46
Table 29: Buttons of the Pick Object Dialog 47
Table 30:Parameters of the Pick Object Dialog 47
Table 31: Parameters of the Colourmap Dialog 48
Table 32: Parameters of the FITS Header Dialog 49
Table 33: Parameters of the Data Stream Dialog 50
Table 34: Parameters of the HDU Dialog 51
Table 35: Parameters of the Binary Table Dialog 52
Table 36: Parameters of the Tabular Region Dialog 53
Table 37: Parameters of the Graphical Elements Dialog 54

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Table 38: Parameters of the Graphics Control Dialog 56
Table 39: Parameters of the Cut Values Dialog 56
Table 40: Parameters of the Bias Dialog 58
Table 41: Parameters of the Statistics Dialog 59
Table 42: Parameters of the Slit Dialog 60
Table 43: Parameters of the PVCM Dialog 61
Table 44: Parameters of the Reference Line Dialog 62
Table 45: Parameters of the Flip Rotate Scale Cut Values Dialog 63
Table 46: Parameters of the Distance Dialog 64
Table 47: Parameters of the Magnification Dialog 65
Table 48: Overlay element types 67
Table 49: API function of the Overlay API 68
Table 50: Functions for Remote Control 70
Table 51: Commands for Remote Control 71
Table 52: Commands for Remote Client 71
Table 53: Interfaces of the DdtRenderingPlugin 73
Table 54: Interfaces of the DdtImageGraphicsItem 74
Table 55: Data Transfer Components 75
Table 56: Data Visualisation Components 77
Table 57: Image Handling Components 77
Table 58: Utility Components 77
Table 59: Commands list 87

List of Figures

Figure 1: Data Display Tool Components 15
Figure 2: DDT Standard Viewer with DDT Widgets 31
Figure 3: Data Stream Widget 36
Figure 4: Flip / Rotate Widget 37
Figure 5: Scale Buttons Widget 38
Figure 6: Panning Widget 38
Figure 7: Image Scale Widget 39
Figure 8: Colourmap Widget 40
Figure 9: Cursor Information Widget 41
Figure 10: Cut Values Widget 41
Figure 11: Magnification Widget 42
Figure 12: 3D Cube Navigation Widget 43
Figure 13: Context menu of the Image Widget 44

Figure 14: Pick Object Dialog 46
Figure 15: Colourmap Dialog 48
Figure 16: FITS Header Dialog 49
Figure 17: Data Stream Dialog 50
Figure 18: HDU Dialog 51
Figure 19: Binary Table Dialog 52
Figure 20: Tabular Region Dialog 53
Figure 21: Graphical Elements Dialog 54
Figure 22: Graphics Control Dialog 55
Figure 23: Cut Values Dialog 56
Figure 24: Bias Dialog 57
Figure 25: Statistics Dialog 58
Figure 26: Slit Dialog 59
Figure 27: PVCM Dialog 61
Figure 28: Reference Line Dialog 62
Figure 29: Flip Rotate Scale Cut Values Dialog 63
Figure 30: Distance Dialog 64
Figure 31: DDT Magnification Dialog 65
Figure 32: Open File Context Menue Entry 65
Figure 33: Open File Dialog 66
Figure 34: ‘Mark position’ Context Menue Entry 67
Figure 35: Example for Python Broker 81
Figure 36: Example for Python remote client 84

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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	CGI-DER-00072	Design Description	1
ELT-INST	ESO-287339	E-ELT Linux Installation Guide	4.14
CENTOS_INST	https://docs.centos.org/enUS/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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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.

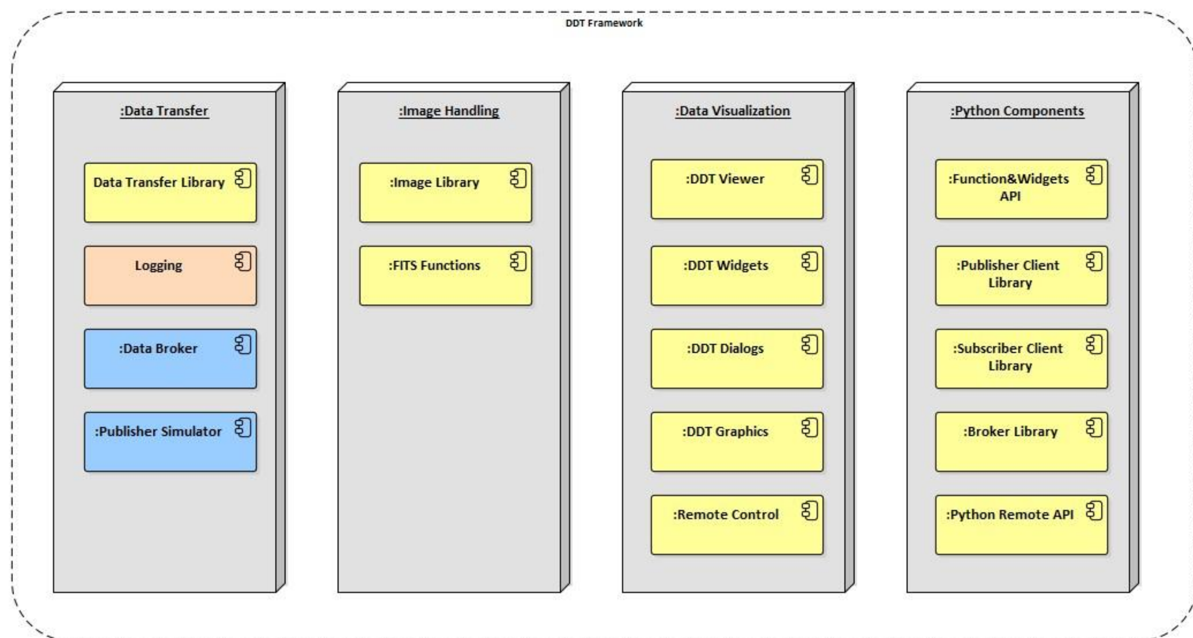


Figure 1: Data Display Tool Components

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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 2.2.6-5 is required, it is assumed the user has installed those prior to following this guide and as per instructions given by the respective projects (see [CENTOS-INST] and [ELT-INST]).

3.2.1 Building and Installing DDT

3.2.1.1 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
```

3.2.1.2 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.

In the private.lua file a number of environment variables will be set. These can be modified by the user:

Environment variable	Description
DDT_LOGCONFIG_PATH	Path to the logger configuration files.
DDT_TRANSFERCONFIG_PATH	Path to the configuration of the DDT Data Transfer.
DDT_COLORMAP_PATH	Path to the rgb colourmaps used by the DDT Datavisualisation.
DDT_CONFIGURATIONMAP_PATH	Path to configuration maps used by the DDT Datavisualisation.
DDT_TEST_DATA_DIR	Path to test data used by the testing.

Table 8: Environment variables defined by the DDT

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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:

3.3.1.1.1.1 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.
Create Publisher	log4cplus::Logger const& log4cplusLogger	static std::unique_ptr<DdtDataPublisher>	Creates an instance of the DdtDataPublisher. An instance of the log4cplus logger object is given. Note that in this case, the initialization and shutdown of the log4cplus must be done by the caller.

3.3.1.1.1.2 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 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.
WriteData	const int32_t sample_id, const uint8_t *const data, const int32_t	-	Writes the data packet into shared memory using a memory

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Function	Arguments	Return type	Description
	data_size, const uint8_t *const metadata, const int32_t metadata_size		accessor. The data is passed as a pointer.
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.

3.3.1.1.1.3 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.
CreateSubscriber	log4cplus::Logger const& log4cplusLogger	static std::unique_ptr< DdtDataSubscriber >	Creates an instance of the DdtDataSubscriber. An instance of the log4cplus logger object is given. Note that in this case, the initialization and shutdown of the log4cplus must be done by the caller.

Table 9: 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_interval (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 interval can be used to simulate slow readers. The argument

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Function	Arguments	Return type		Description
				is optional. Default interval will be 10 ms.
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.
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 10: 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;
```


EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
    /**
     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.

The latency is defined as follows: In the local case it is the time difference of the data sample written into shared memory and the time the data broker was notified from the publisher about new data. In the remote case it is the time difference of the data sample written into shared memory and the time the remote broker received the data sample via network.

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.

3.3.1.1.2 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. Note: The sample is not removed from the shared memory. The data is kept until it is overwritten by the publisher.
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 11: Datatransfer Library configuration file

3.3.1.1.3 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> -o <configuration file> [--debug]
ddtBroker --uri <Server URI of the broker> --config_file <configuration file> [--
debug]
```

An example for this would be:

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

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. The URI consists of the protocol (e.g. zpb.rr), the host (e.g. *) and the port (e.g. 5001).

The "`--config_file`" parameter is optional. If this parameter is not specified by the user, a default configuration file is used (see below). This parameter can be used to specify a different file which allows to run multiple Data Brokers on the same host with different configurations. If a specified file is invalid or does not exist, the Data Broker reports a warning and uses the default configuration instead.

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]:

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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.
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 12: Data Broker configuration file

3.3.1.1.4 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 --datastream ds1 --interval 1000 --mode 1 --image_folder /data/fitsimages/rotate --buffer_size 10 --debug
```

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

	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
	Mode 6: Transfer of 16 bit unsinged data
	Mode 7: Transfer of 32 bit unsigned data
`--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, 5, 6 and 7 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. For detailed information on the configuration map handling, please refer to section 3.3.5.

When running the Publisher Simulator in mode 5 it will generate chunks of images with the proper metadata 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 metadata. The meta-data also contains a “complete” flag. This is used to determine between chunked images and images that are not chunked.

The modes 6 and 7 will generate a data stream consisting of data that cover the range for 16 rsp. 32 bit unsigned data, with the first value (image coordinate 1 / 1) set to 0 and the last value (image coordinate width / height) set to a value near the maximum possible value for the data type in question. The data type is specified in the meta data.

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.

3.3.1.1.5 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> --interval <reading period in ms> --mode
<simulator mode> [--statistics <0|1>] [--dump_data <0|1>] [--dump_folder <folder
specification>] [--debug]
```

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

An example of the command looks like this:

```
ddtSubscriberSimulator --broker zpb.rr://127.0.0.1:5001 --datastream ds1 --mode 1 -
-debug
```

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)
--dump_data	Flag to switch off or on the dumping of the incoming data stream into FITS files (default off). This flag is only considered in simulation mode 1.
--dump_folder	Specification of folder to dump the FITS files to when the dump_data flag is set (default: /tmp)
--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 URI of the `--broker` and `--remote` parameters must not include any path elements. The URI only consists of the protocol (e.g. zpb.rr), the host (e.g. 127.0.0.1) and the port (e.g. 5001).

When started in mode 1, it is possible to store the received data in FITS files. To do this, the option –dump_data has to be provided with a value of 1. In addition, it is then possible to specifiy the folder where the FITS files are stored using the –dump_folder option. When not set, the folder defaults to "/tmp". The created FITS files are named as follows: <timestamp>_subscriber_dump.fits, where <timestamp> is the UTC timestamp taken from the meta data of the data package.

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.

3.3.1.1.6 Publish Data from a Snapshot

In order to publish data from a previously recorded snapshot, the following procedure can be used:

- Use the Subscriber Simulator (see 3.3.1.1.5) with the dump_data and dump_folder options to store the received data stream into FITS files. Use a newly created or an empty dump_folder for this purpose. Note that the specified dump folder must exist.
- After stopping the Subscriber Simulator, make a compressed archive from the dump folder by executing:

```
tar -czf <snapshot_file.tgz> <dump_folder>
```

- In order to publish the stored snaphot, unpack the tgz snapshot file, by executing:

```
tar -xzf <snapshot_file.tgz>
```

- Start a Publisher Simulator (see 3.3.1.1.4) to start a data transfer in mode 1, providing the previously unpacked data folder as image_folder argument.

3.3.1.2 Meta-Data definition

For the encoding/decoding of meta-data encoder/decoder classes exist. These make use of a meta-data definition for different data types: image data with 2 dimensions, image data with 3 dimensions and multi-array data with x dimension.

The related classes are derived from a single base class. The base class offer the following methods:

```
class DdtEncDec {
 public:
  /**
   * Constructor
   */
  explicit DdtEncDec();
  /**
   * Destructor
   */
  virtual ~DdtEncDec() = 0;

  /**
   * Sets the meta data length
   */
  virtual void setMetaDataLength(const int mdl);

  /**
   * Sets the topic id
   */
  void setTopicId(const int ti);

  /**
   * Return the meta data length
   */
  virtual int getMetaDataLength();

  /**
   * Return the topic id
   */
  virtual int getTopicId();

  /**
   * Return the bytes_per_pixel member.
   */
  virtual uint32_t getBytes_per_pixel() const;

  /**
   * Return the number_dimensions member.
   */
```

```
virtual uint32_t getNumber_dimensions() const;

/**
 * Return the utc_timestamp member.
 */
virtual std::string getUtc_timestamp() const;

/**
 * Return the complete_flag member.
 */
virtual bool getComplete_flag() const;

/**
 * Return the last_segment member.
 */
virtual bool getLast_segment() const;

/**
 * Return the byte_order_little_endian member.
 */
virtual bool getByte_order_little_endian() const;

/**
 * Return the data_type member.
 */
virtual uint32_t getData_type() const;

/**
* Return the description member.
*/
Virtual std::string getDescription() const;

/**
 * Return the reference_point_x member.
 */
virtual float getReference_point_x() const;

/**
 * Return the reference_point_y member.
 */
virtual float getReference_point_y() const;

/**
 * Return the ra_reference_point member.
 */
virtual float getRaReference_point() const;

/**
 * Return the dec_reference_point member.
 */
virtual float getDecReference_point() const;

/**
 * Return the arcsec_pixel_x member.
 */
virtual float getArcSec_pixel_x() const;

/**
 * Return the arcsec_pixel_y member.
 */
virtual float getArcSec_pixel_y() const;

/**
 * Return the rotation_x_axis member.
```

```
   */
  virtual float getRotation_x() const;

  /**
   * Return the cd1_1 member.
   */
  virtual float getCd1_1() const;

  /**
   * Return the cd1_2 member.
   */
  virtual float getCd1_2() const;

  /**
   * Return the cd2_1 member.
   */
  virtual float getCd2_1() const;

  /**
   * Return the cd2_2 member.
   */
  virtual float getCd2_2() const;

  /**
   * Return the epoch_equinox member.
   */
  virtual float getEpochEquinox() const;

  /**
   * Return the type 1 projection member.
   */
  virtual std::string getType_1() const;

  /**
   * Return the type 2 projection member.
   */
  virtual std::string getType_2() const;

 protected:
  /**
   * Return the current time including milliseconds
   */
  std::string getCurrentTime() const;

  /**
   * The topic ID is used to distinguish meta data shapes from each other
   */
  int topicId = 0;

  /**
   * The length of the meta data block
   */
  int metaDataLength = 0;
};
```

Additional derived classes exist:

- DdtEncDecImage2D
- DdtEncDecImage3D
- DdtEncDecBinaryxD

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

These derived classes offer getters for the related meta-data field defined for those data types.

The meta-data is grouped into structs. The base meta-data includes the following elements:

```
struct MetaDataBase {
  uint32_t bytes_per_pixel;
  uint32_t number_dimensions;
  std::string utc_timestamp;
  bool complete_flag;
  bool last_segment;
  bool byte_order_little_endian;
  uint32_t data_type;
  std::string description;
};


struct WcsInformation {
  float reference_point_x;
  float reference_point_y;
  float ra_reference_point;
  float dec_reference_point;
  float arcsec_pixel_x;
  float arcsec_pixel_y;
  float rotation_x_axis;
  float cd1_1;
  float cd1_2;
  float cd2_1;
  float cd2_2;
  float epoch_equinox;
  std::string type_1;
  std::string type_2;
};
```

The struct WcsInformation is required by 2D and 3D images to calculate WCS coordinates.

The derived data types contain some additional information:

Meta-Data for 2D image data:

```
struct MetaDataElementsImage2D {
  MetaDataBase meta_data_base;
  uint32_t number_pixels_x;
  uint32_t number_pixels_y;
  int32_t binning_factor_x;
  int32_t binning_factor_y;
  uint32_t first_pixel_x;
  uint32_t first_pixel_y;
  uint32_t number_chunks_x;
  uint32_t number_chunks_y;
  uint32_t image_id;
  WcsInformation wcs_info;
};
```

Meta-Data for 3D image data:

```
struct MetaDataElementsImage3D {
  MetaDataBase meta_data_base;
  uint32_t number_pixels_x;
  uint32_t number_pixels_y;
  int32_t binning_factor_x;
  int32_t binning_factor_y;
  uint32_t number_layers;
  uint32_t item_size;
```


EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
  WcsInformation wcs_info;
};
```

Meta-Data for multi dimensional arrays:

```
struct MetaDataElementsBinaryxD {
  MetaDataBase meta_data_base;
  std::string array_dimensions;
  std::string configuration_map_name;
  std::string labels;
};
```

More information on the meta-data definition can be found in the Design Description [DD] in section 3.1.6.

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. Note: All signals of the auxiliary widgets have the Image Widget as target while all slots have the Image Widget as source.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

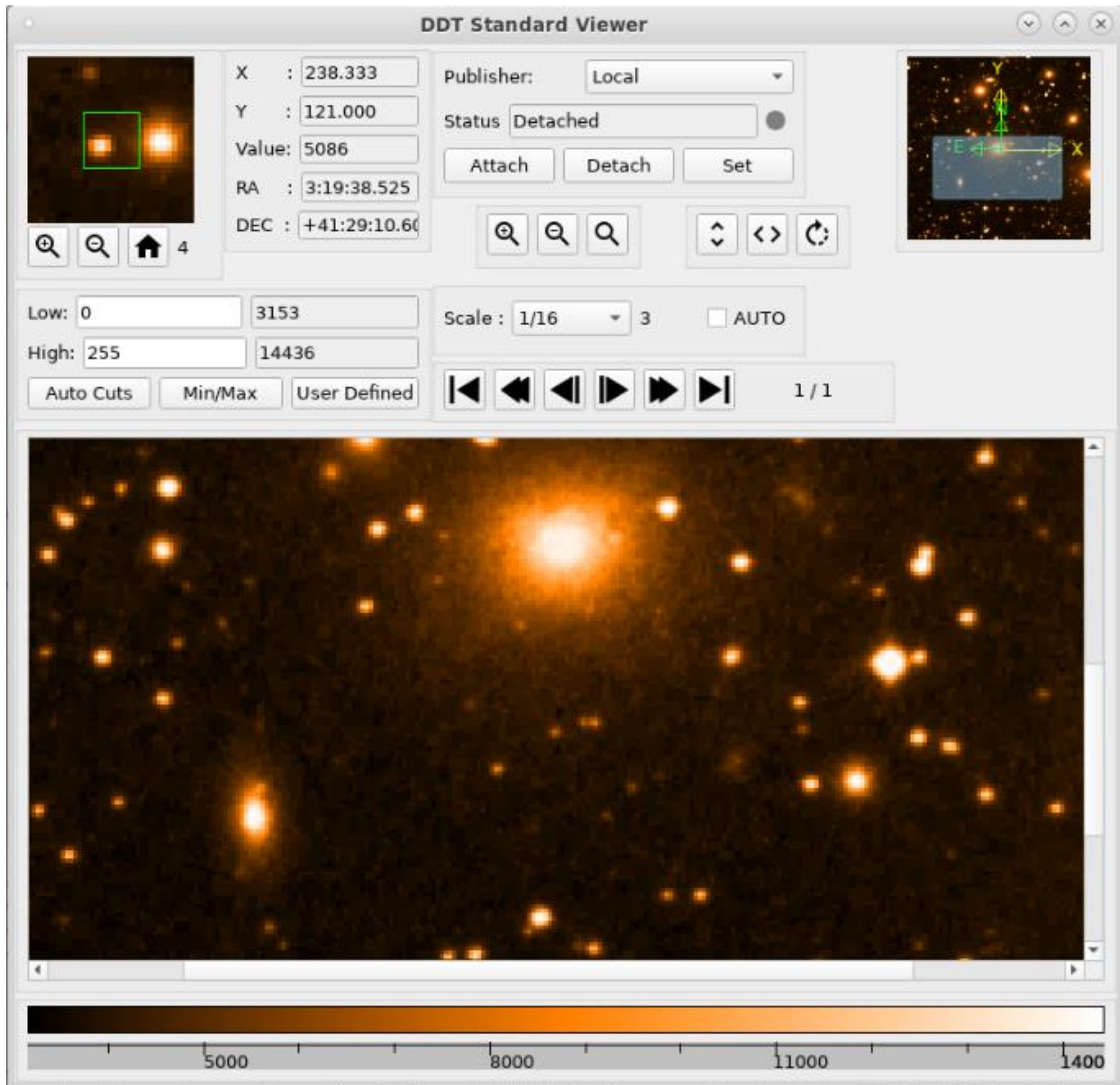


Figure 2: DDT Standard Viewer with DDT Widgets

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 13: Properties of the Image Widget

The Image Widget offers a number of public methods:

Function Name	Return Type	Arguments	Description
ActivateTimestampDisplay	void	-	Activates debug output of the data sample timestamp in the image.
AddRenderingPlugins	void	DdtRenderingPlugin* const new_plugin	This function can be called to add a new rendering function into the Image Widget. Additional rendering functions can be used to render image data in different ways.
CloseAllDialogs	**void**	**-**	Closes all open dialogs.
ConvertCanvasToImage	**void**	**const double x_canvas** **const double y_canvas** **double* x_image** **double* y_image**	Converts the x,y canvas coordinates to image coordinates (returned by reference).

Function Name	Return Type	Arguments	Description
ConvertImageToCanvas	**void**	**const double x_image** **const double y_image** **double* x_canvas** **double* y_canvas**	Converts the x,y image coordinates to canvas coordinates (returned by reference).
CutLevelChanged	**void**	**-**	Handling the change of cut levels.
EndMarkPosition	**QString**	**-**	Ends the marking of position in the image and returns the list of selected points as string.
FetchDialogName	**QString**	**Const QString dialog_id**	Fechtes the name of a dialog with a given dialog ID. The name can be used e.g. in the context menu.
HandleNewBoostDataEvent	**void**	**-**	Handles event coming from the data transfer library informing about the availability of new data.
InitializeDialogMap	**void**	**const QMap<QString, DdtDialog*> insert_map**	Initialize the map of dialogs which can be called from the Image Widget. A map of dialogs is given as argument. The map contains the dialog objects indexed by the dialog IDs.
MarkPositions	**void**	**-**	Start marking positions in the image.
ParseFractionString	**float**	**QString fraction_str**	Parse the string representation of a fraction to a floating point value. E.g. 1/4 is converted to 0.25.
ProcessRemoteCommand	**void**	**const std::string& command_name** **const std::vector<std::string>& command_arguments**	Called when processing a remote command. Arguments contain the command name and a vector of strings holding the command arguments.
RegisterLastSegmentCallback	**void**	**std::function<void()> const& lastSegmentFunction**	Can be used to register a function which will then be called when a last segment of a segmented image arrives.
ReloadGraphicsItem	**void**	**-**	Reloads the graphics items into the scene.
SetActiveRenderingPlugin	**void**	**const int rendering_plugin_id**	Select the active rendering plugin id. The activated rendering plugin will be used for rendering new data.
SetOverlayImageFile	**void**	**QString overlay_file**	Can be used to add a (transparent) png file as overlay.
SortedScaleFactors	**QList<QString>**	**QMap<QString, float> scale_map**	Returns a sorted list of scale factors. The input is a maps of scale factors in floating point format mapped the the string representation of these scale factors.
UpdataAllStatistics	**void**	**-**	Updates all statistics like cursor information, object information etc.
UpdateThumbnailImages	**void**	**-**	Updates all attached thumbnail images, e.g. pan widget, magnification widget etc.

Table 14: Public methods 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.
	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 forwarded 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	
	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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Widget Name	External Signal / Slot	Description (if required)
	Internal Signals / Slots	
	Signal: ContextMenuCommandSelected(QString menu_entry)	Signal when an entry in the context menu was selected.

Table 15: 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”.

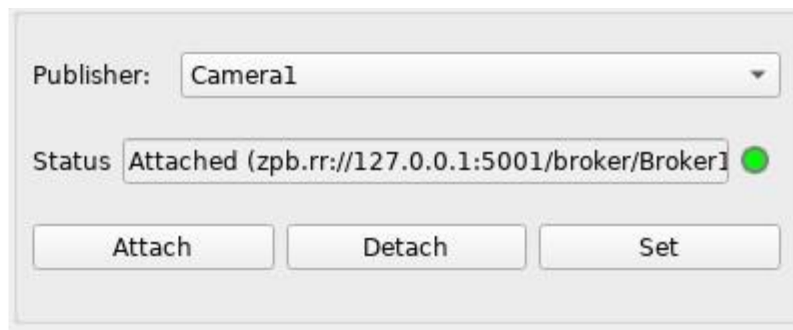


Figure 3: Data Stream Widget

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 16: 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 17: 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 18: Signals and slots of the Flip / Rotate Widget

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 19: 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.

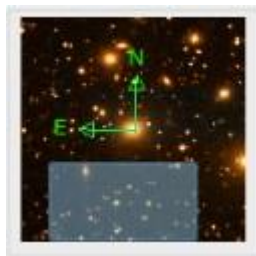


Figure 6: Panning Widget

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 20: 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. One entry in the combo box is the entry “FIT”. When selecting the “FIT” entry the current image will be automatically re-scaled so that the full image becomes visible 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. As long as the “AUTO” checkbox is enabled, it will not be possible to change the scale factor. In order to change the scale factor again, the “AUTO” checkbox needs to be disabled.

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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Widget Name	External Signal / Slot	Description (if required)
	Signal: SelectScale(QString scale)	Change the scale factor to the select value.
	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 autoscale state is modified.

Table 21: 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.

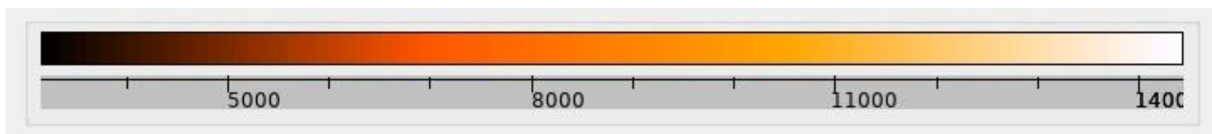


Figure 8: Colourmap Widget

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 22: 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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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

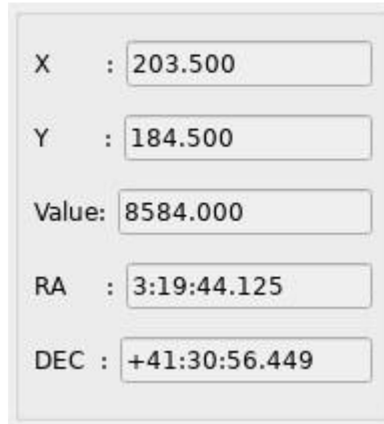


Figure 9: Cursor Information Widget

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 23: 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.

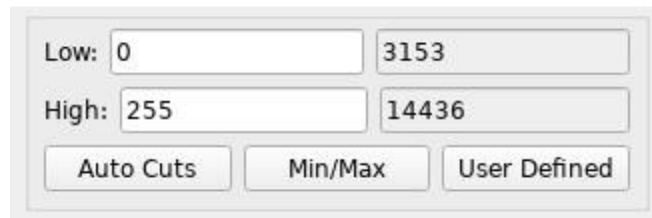


Figure 10: Cut Values Widget

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 24: 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` Size of a rectangle displayed in the center of the magnified image (default size 10 pixels)

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.

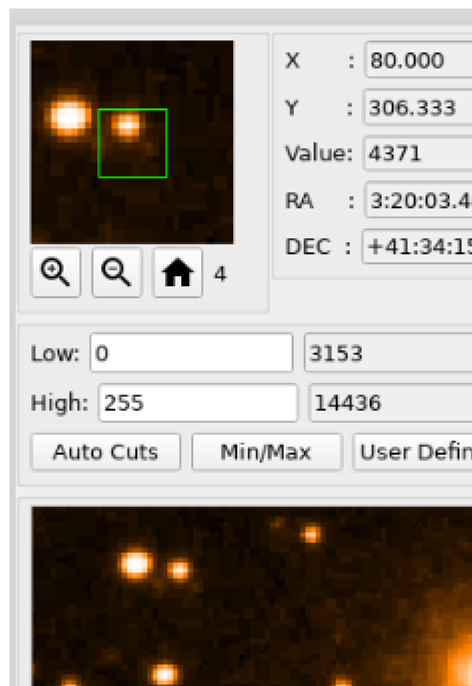


Figure 11: Magnification Widget

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 25: Signals and Slots of the Magnification Widget

3.3.2.1.11 3D Cube Navigation Widget

The 3D Cube Navigation Widget can be used to navigate through the planes of a 3D image or image extension. After opening such an image or image extension (either by using the extension notation “[<extension number>]” when specifying the filename in the “Open File” dialog (e.g. SCI-GUM43_COMBINED_CUBE_010.fits[2]) or by selecting the extension in the DDT HDU Dialog), the 3D Cube Navigation Widget looks similar to this:

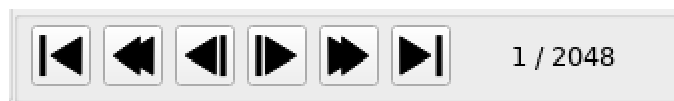
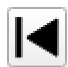


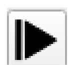
Figure 12: 3D Cube Navigation Widget

In this example, the numbers “1 / 2048” mean that plane number 1 from a total of 2048 planes is currently displayed. With help of the arrow buttons it is then possible to navigate through the planes. Use the buttons in the following way:

 Skip to the first plane.

Fast-rewind by 10 planes.

Rewind by 1 plane.

 Forward by 1 plane.

Fast-forward by 10 planes.

Skip to the last plane.

Clicking the rewind and forward buttons will result in one skip operation. It is also possible to keep the mouse button pressed on these buttons, which will result in a continuous skipping of planes.

Note that the fast-skipping buttons will not skip if the resulting plane number would be out of the range.

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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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:

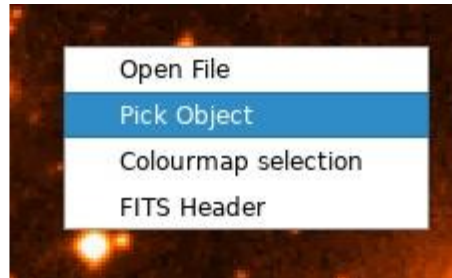


Figure 13: Context menu of the Image Widget

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	Binary Table dialog
ddtReferenceLine	Reference line dialog
ddtDistance	Dialog for distance measurement
ddtPVCM	PVCM dialog
ddtGraphicalElements	Dialog for graphical elements
ddtGraphicsControl	Dialog for control of graphical elements
ddtSlit	Slit dialog
ddtCutValues	Cut value dialog
ddtDataStream	Data Stream dialog
ddtScaleRotateCut	Dialog for cut value, scale factor and flip / rotate
ddtMagnification	Magnification dialog
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 26: List of dialog IDs

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 27: 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.

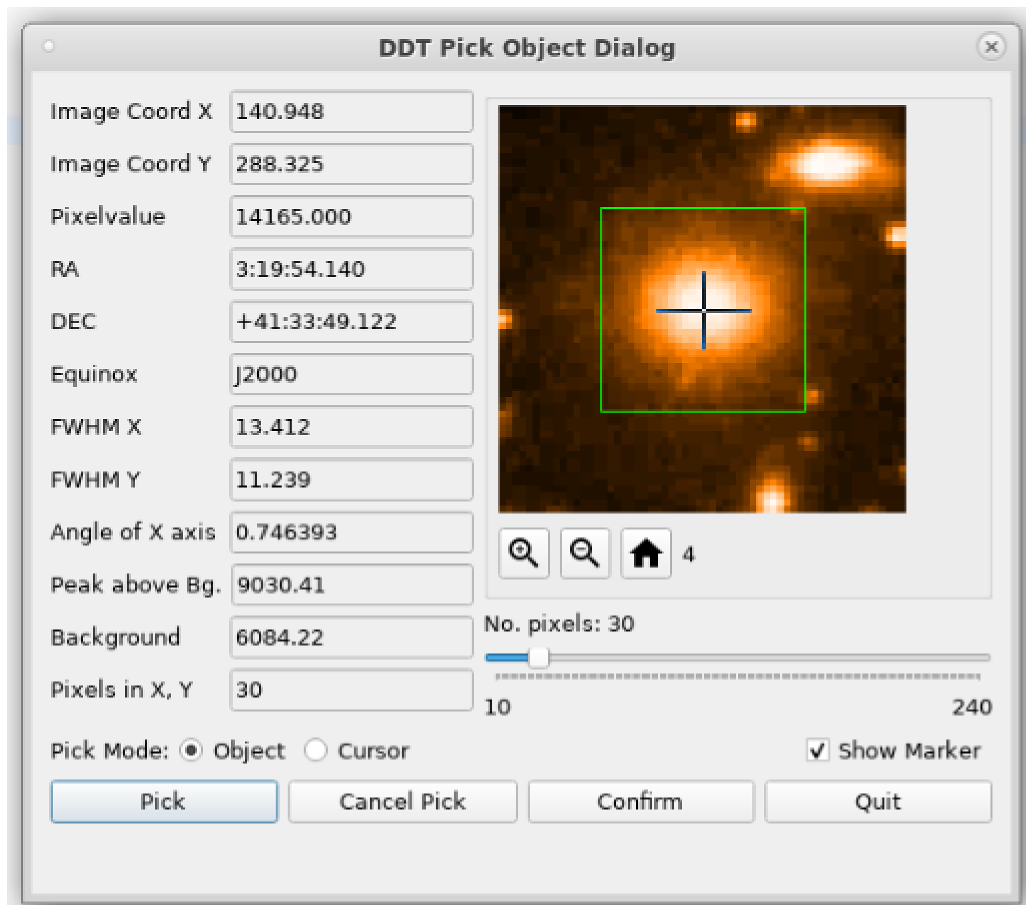


Figure 14: Pick Object Dialog

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 28: 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 a pixel in the Image Widget. Depending on the selected pick mode the related information is being displayed whenever the user clicks in the image. The Pick mode is then deactivated again.
Cancel Pick	Only enabled, while the Pick mode is active. It cancels the pick mode. The displayed values are cleared.
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 29: 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. In the magnification widget a rectangle will be drawn around the center with the dimensions according to the selected number of pixels.

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	Used to receive the magnified image for the thumbnail view of the magnification widget.
DDT_DIALOG_PARAM_PICK_MAGNIFY_FACTOR	Used to send the current magnification factor of the magnification widget.
DDT_DIALOG_PARAM_PICK_NUMBER_SAMPLES	Used to send the number of subsequent samples that shall be used for the calculation of the object statistics.

Table 30: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.

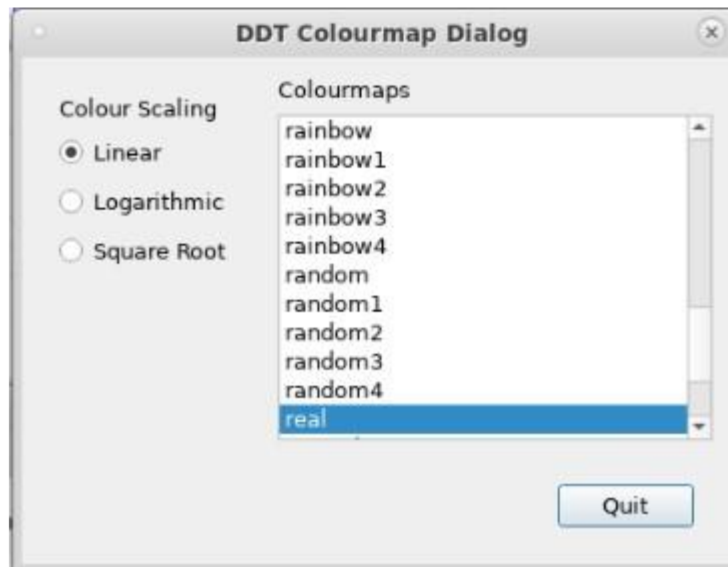


Figure 15: Colourmap Dialog

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 31: Parameters of the Colourmap Dialog

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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.

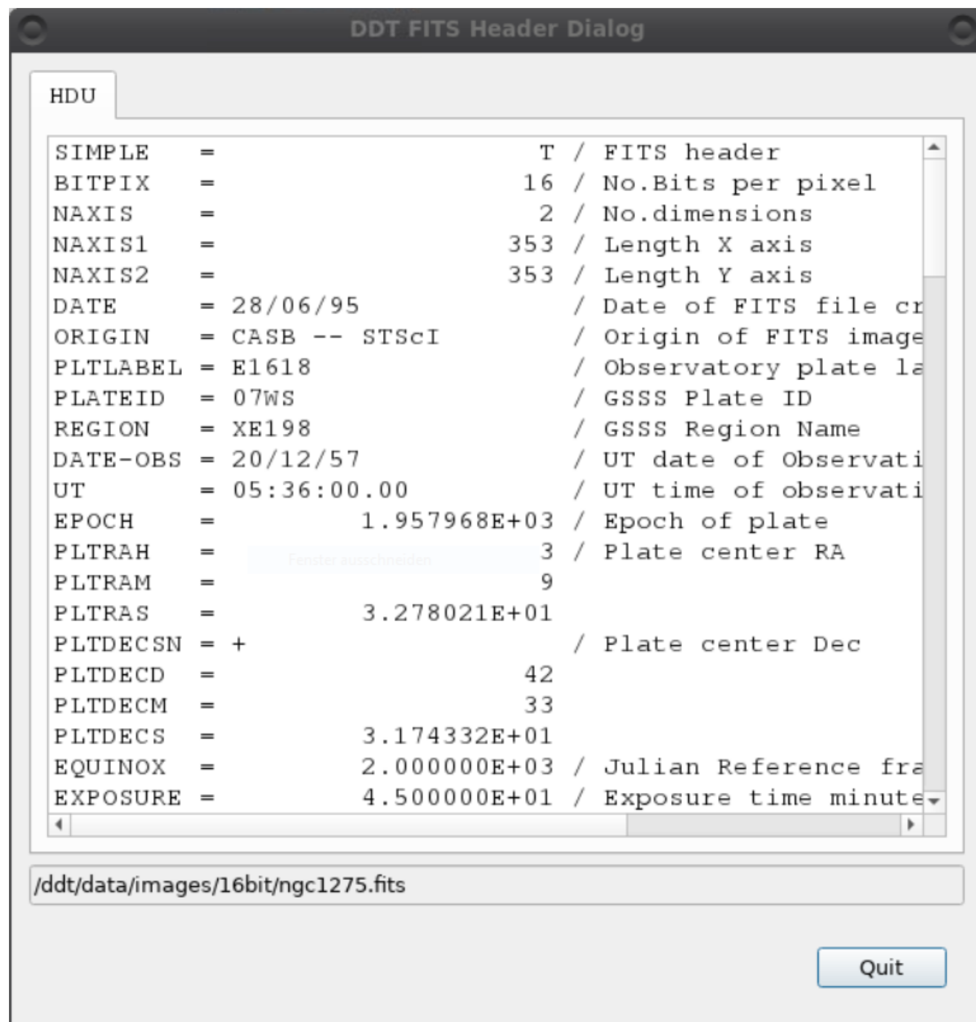


Figure 16: FITS Header Dialog

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 32: 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.

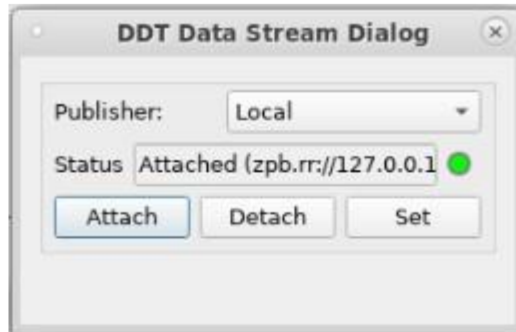


Figure 17: Data Stream Dialog

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 33: 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 Binary 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.

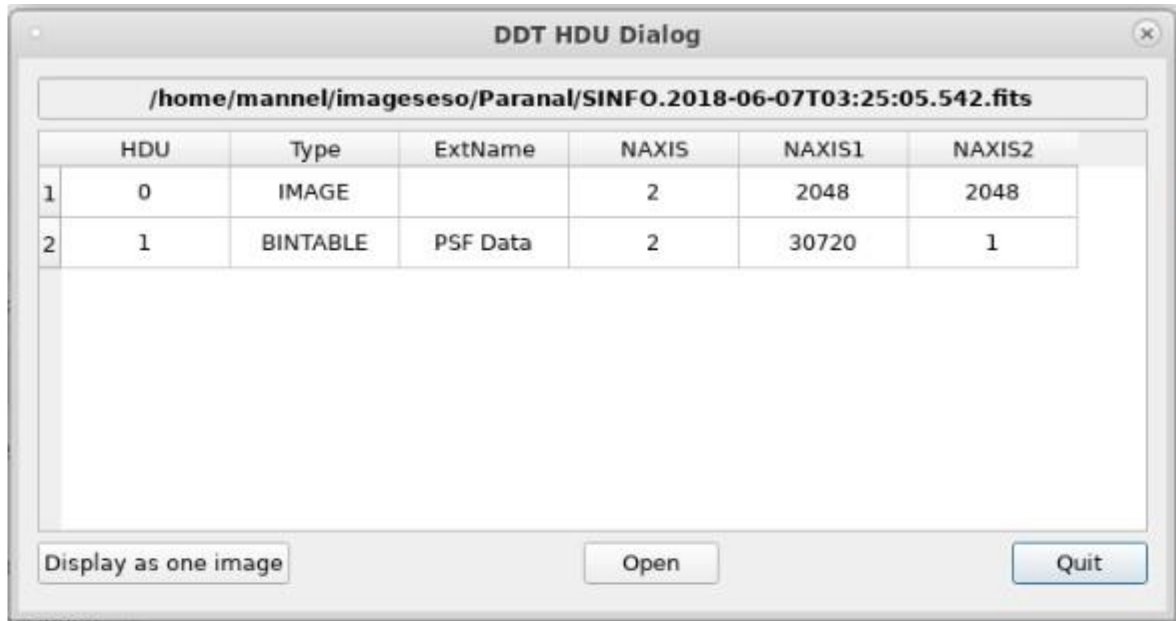


Figure 18: HDU Dialog

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 34: Parameters of the HDU Dialog

3.3.2.2.6 Binary Table Dialog

The Binary 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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

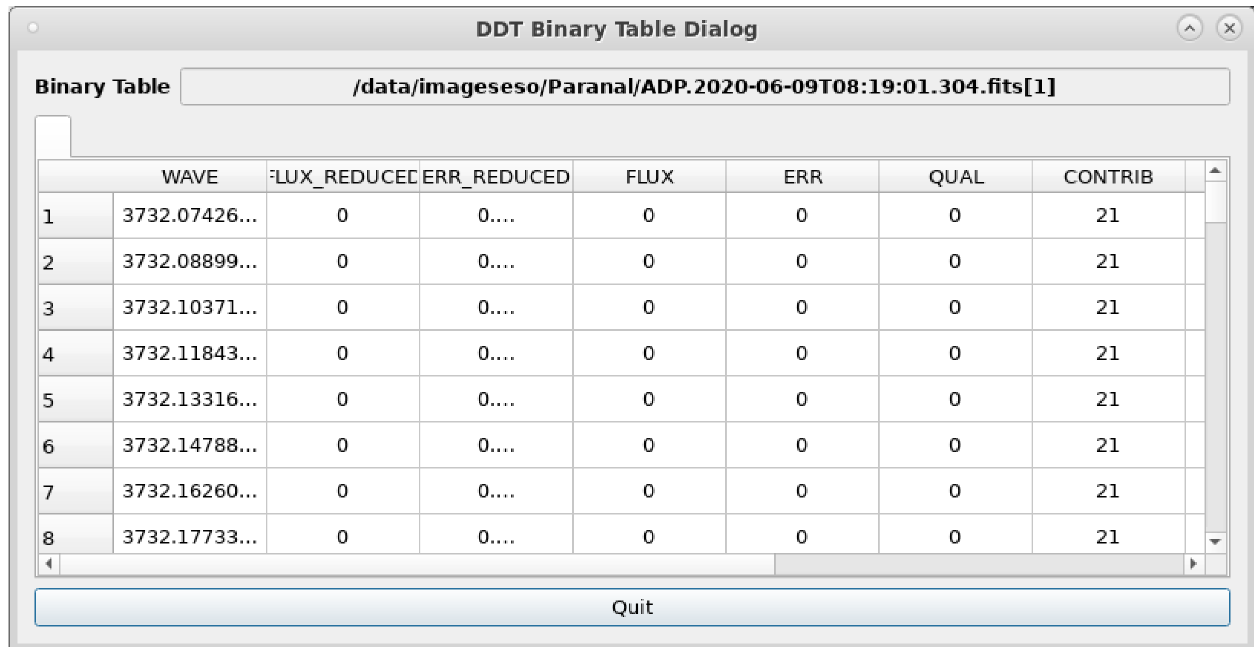


Figure 19: Binary Table Dialog

The Binary Table 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_TABLE_DATA	Data from the selected binary table.
DDT_DIALOG_PARAM_FITS_TABLE_SIZE	Size (columns, rows) of the binary table.
DDT_DIALOG_PARAM_FITS_TABLE_COL_NAMES	Names of the table columns.
DDT_DIALOG_PARAM_FITS_TABLE_CLEAR	Clear the table dialog
DDT_DIALOG_PARAM_FITS_TABLE_CLEAR_AND_PREPARE	Clear the table dialog and prepare display of next table (fill table name field and prepare necessary tabs).

Table 35: Parameters of the Binary 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)

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

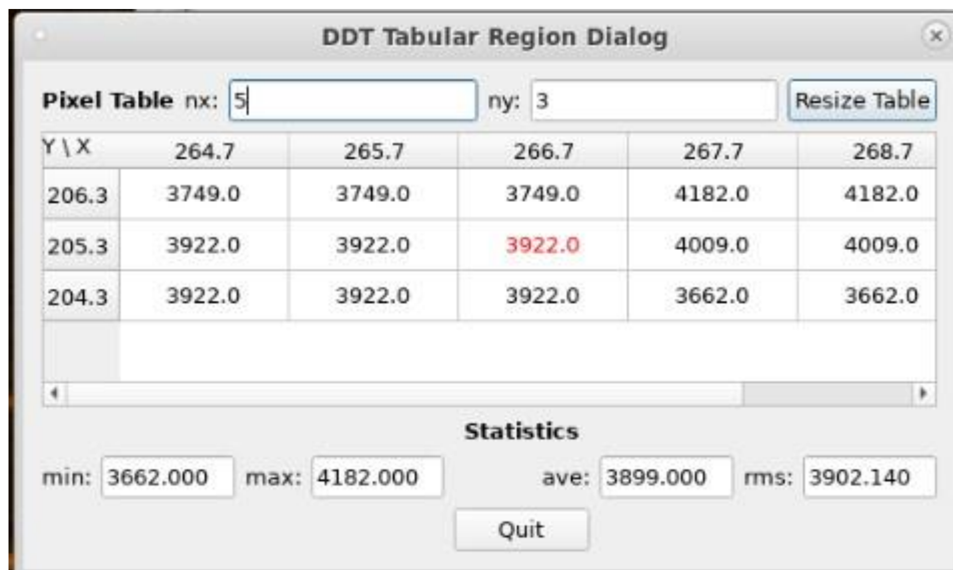


Figure 20: Tabular Region Dialog

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 36: 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
- Line Style (Solid, Dashed, Dotted, Dash-dotted)
- 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)
- Rotation angle

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

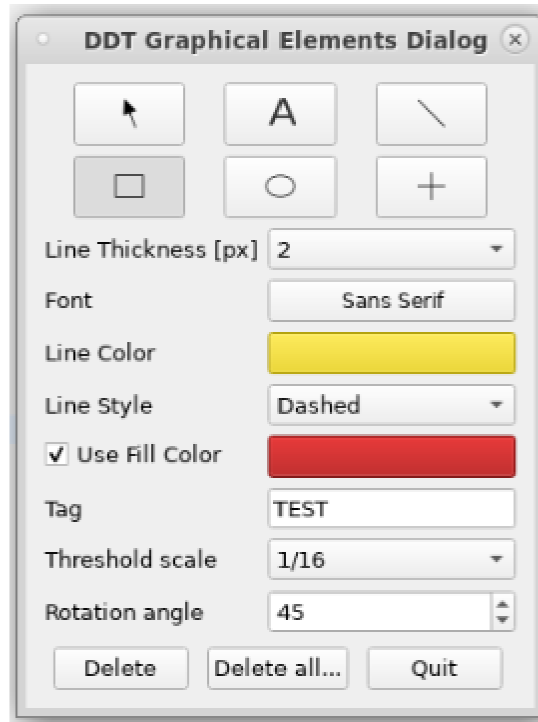


Figure 21: Graphical Elements Dialog

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.
DDT_DIALOG_PARAM_LINE_STYLE	Sets the line style for the overlay elements.
DDT_DIALOG_PARAM_ROTATION_ANGLE	Sets the rotation angle for the overlay elements.

Table 37: Parameters of the Graphical Elements Dialog

3.3.2.2.9 Graphics Control Dialog

The Graphics Control Dialog can be used to select which Graphical Elements (see previous chapter) should currently be visible in the display.

The dialog offers various options to control the display of Graphical Elements.

First of all the checkbox “Enable graphical elements” can be used to enable or disable the display of all Graphical Elements. When this checkbox is unchecked, all elements will be hidden in the display. When it is active the elements will be displayed based on the further selections done in this dialog.

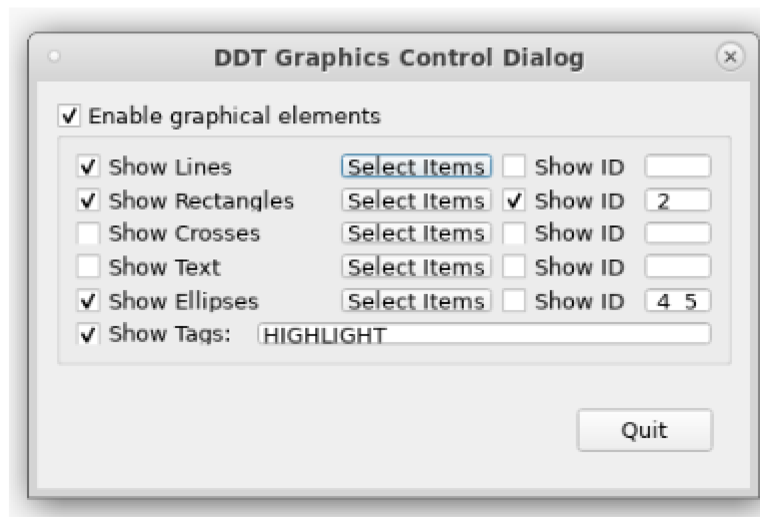


Figure 22: Graphics Control Dialog

The remaining elements will either be grouped by the type of Graphical Elements (like Lines, Rectangles, etc.) or related to the tags used for the Graphical Elements.

The first category consists of the following elements – shown for the example of Rectangles:

For each type of Graphical Element these options can be selected:

- Show <Item Type>: Show or hide all elements of a given type
- Select Items: A button which shows a list of all element IDs of this type
- Show ID: allows to show / hide only the elements with the selected ID

When pressing the “Select Items” button a list of all currently displayed elements will be shown. The user can select any number of IDs from that list to place them into the edit box on the right. When selecting the “Show ID” checkbox either only the elements from the list of IDs are shown or only those of the ID are hidden.

Finally the option “Show Tags” allows to show or hide elements with a given tag. Tags can be specified as property of the Graphical Elements when drawing them. The edit box next to the checkbox allows the entry of tags (separated by white space).

The Graphics Control 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:

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Parameter ID	Meaning
DDT_DIALOG_PARAM_ENABLE_ALL_ELEMENTS	Select if all elements should be hidden or shown (when matching the other selections).
DDT_DIALOG_PARAM_SHOW_LINES	Show or hide line elements.
DDT_DIALOG_PARAM_SHOW_RECTANGLE	Show or hide rectangle elements.
DDT_DIALOG_PARAM_SHOW_CROSSES	Show or hide crosses.
DDT_DIALOG_PARAM_SHOW_TEXT	Show or hide text.
DDT_DIALOG_PARAM_SHOW_ELLIPSES	Show or hide ellipses.
DDT_DIALOG_PARAM_SHOW_TAG	Show or hide elements by tag.

Table 38: Parameters of the Graphics Control Dialog

3.3.2.2.10 Cut Values Dialog

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

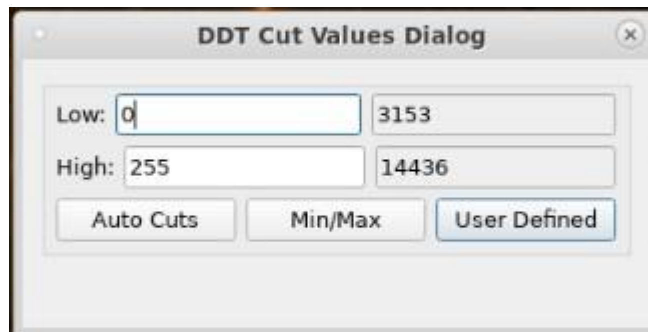


Figure 23: Cut Values Dialog

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 39: Parameters of the Cut Values Dialog

3.3.2.2.11 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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

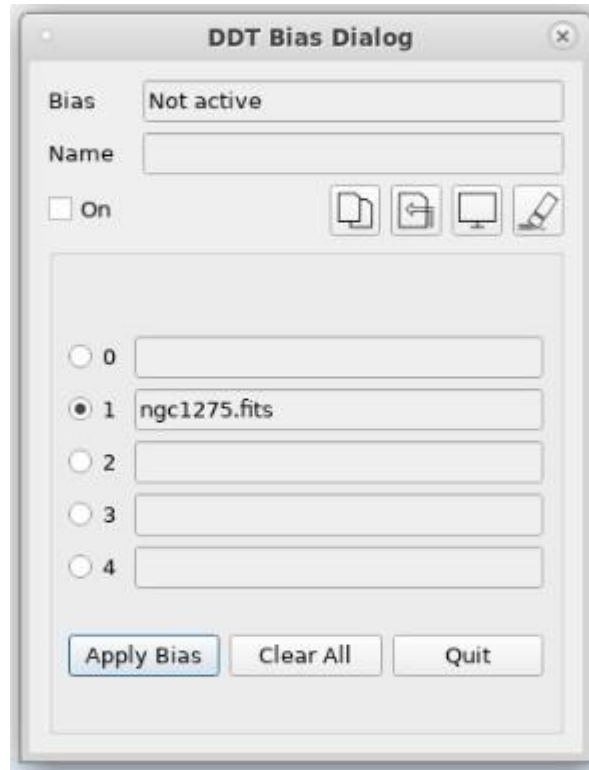


Figure 24: Bias Dialog

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.

Parameter ID	Meaning
DDT_DIALOG_PARAM_BIAS_CLEAR_ALL	Send when all bias images should be cleared.
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 40: Parameters of the Bias Dialog

3.3.2.2.12 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

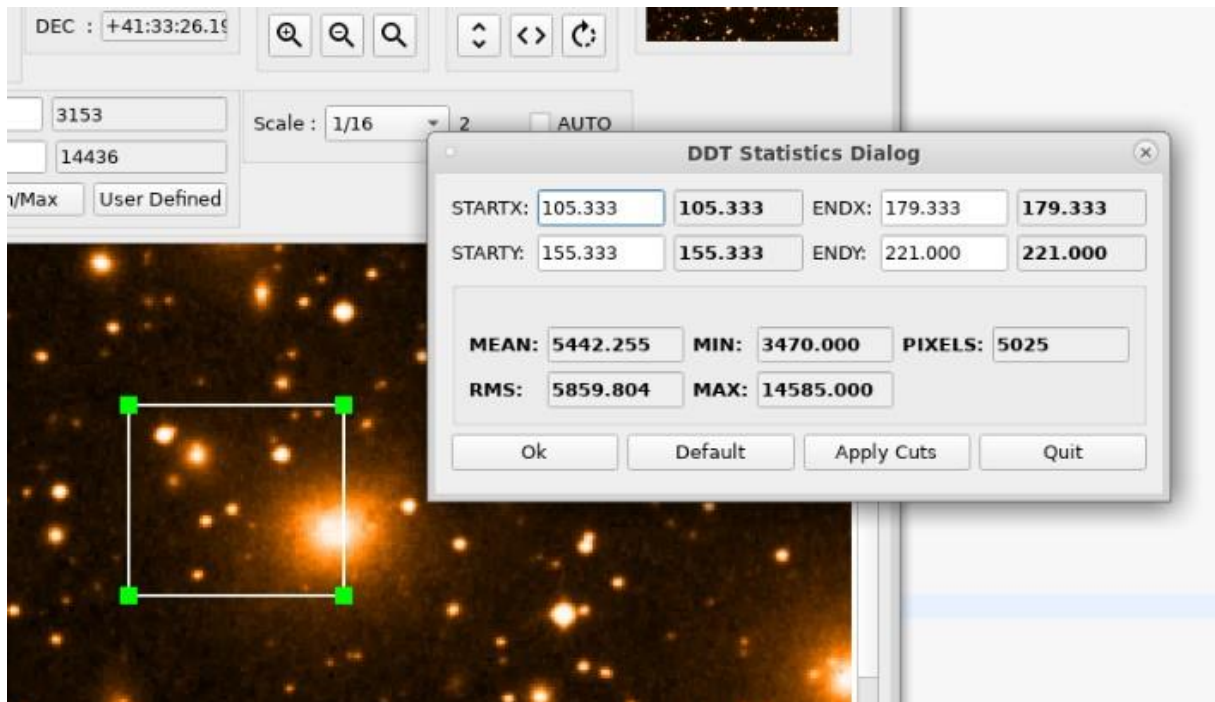


Figure 25: Statistics Dialog

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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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 41: Parameters of the Statistics Dialog

3.3.2.2.13 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

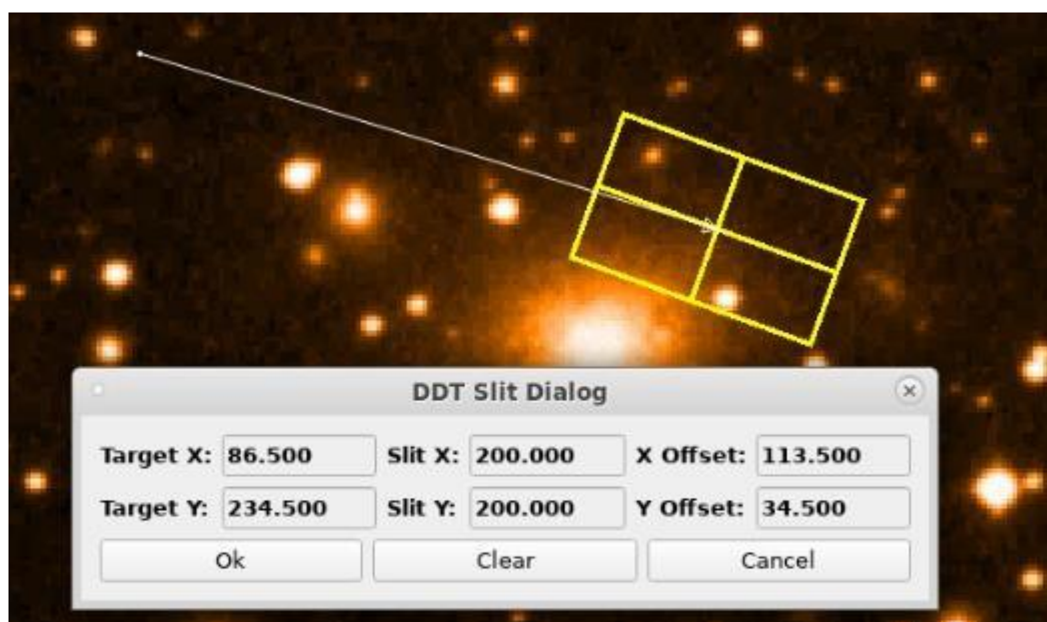


Figure 26: Slit Dialog

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 42: Parameters of the Slit Dialog

3.3.2.2.14 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.

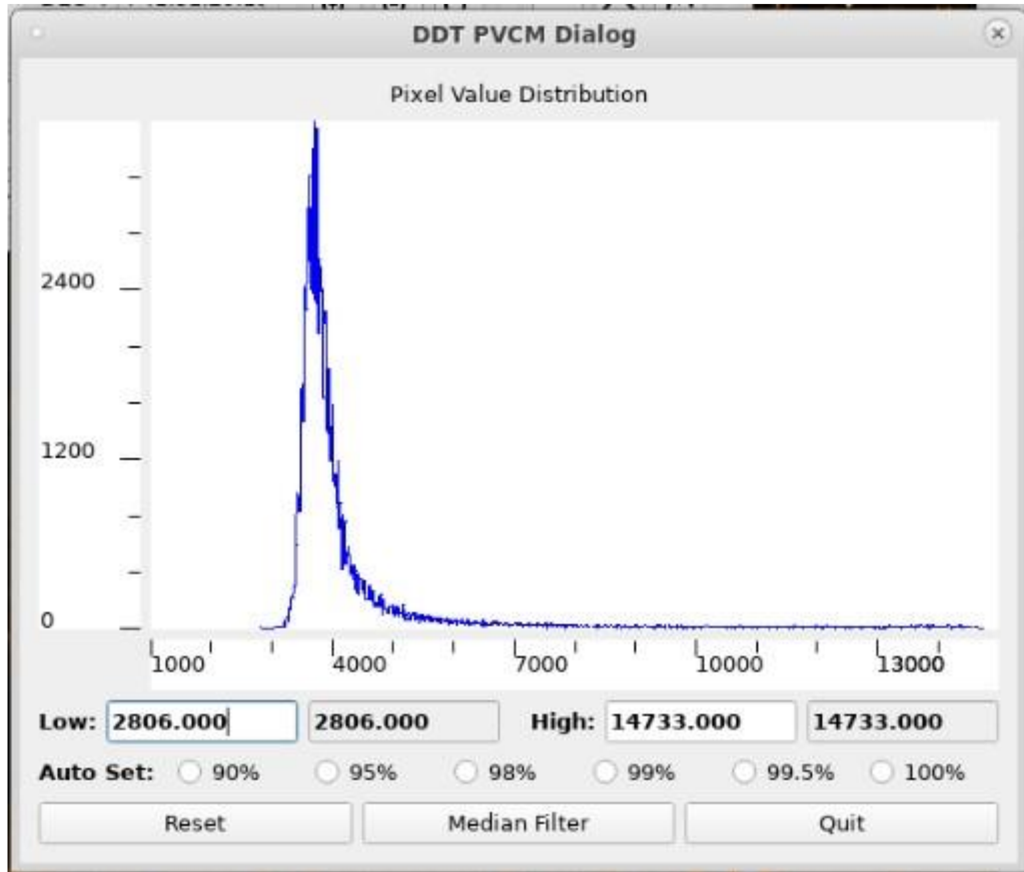


Figure 27: PVCM Dialog

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 43: Parameters of the PVCM Dialog

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

3.3.2.2.15 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.

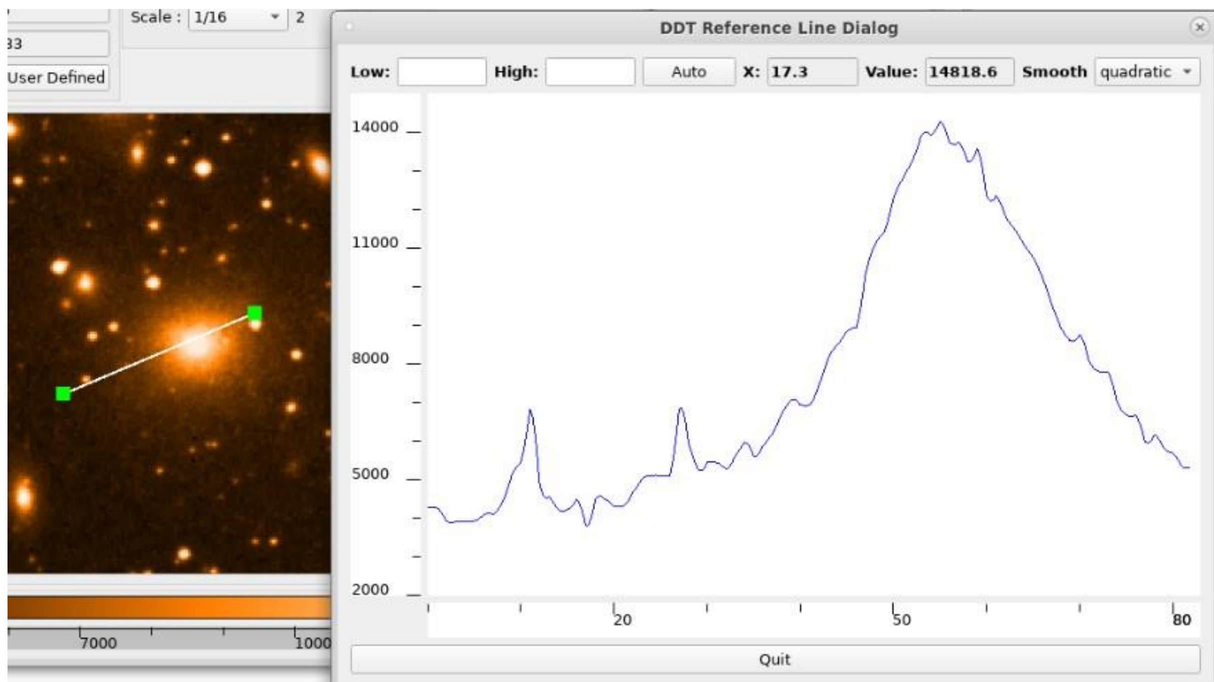


Figure 28: Reference Line Dialog

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 44: Parameters of the Reference Line Dialog

3.3.2.2.16 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).

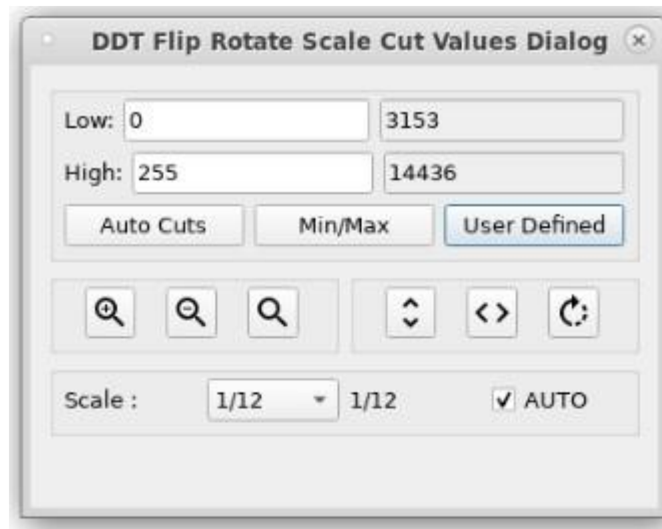


Figure 29: Flip Rotate Scale Cut Values Dialog

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 45: Parameters of the Flip Rotate Scale Cut Values Dialog

3.3.2.2.17 Distance Dialog

The Distance Dialog can be used to measure the distance between two locations in the current image. The distance will be measured in pixels.

When the dialog was selected, the user can draw a line into the connected Image Widget using the mouse pointer. The dialog will then display the start x/y coordinates, the end x/y coordinates and the offset in x- and y-direction.

When pressing “Confirm” the dialog closes and returns the the following values as a string:

Offset X Offset Y Start X Start Y

When pressing “Quit” the dialog closes and returns nothing.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

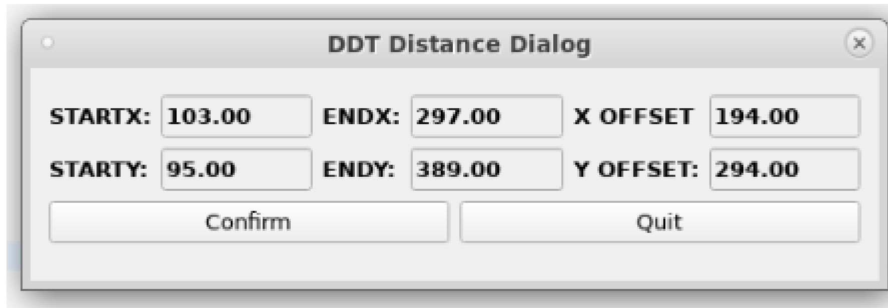


Figure 30: Distance Dialog

The Distance 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_DISTANCE_DRAW_MODE	Set the draw mode in the Image Widget to be able to draw a distance line.
DDT_DIALOG_PARAM_DISTANCE_VALUES	Is used to update the displayed distance values in the dialog.
DDT_DIALOG_PARAM_DISTANCE_RETURN_VALUES	Is used to return the measured distance values as Offset X, Offset Y, Start X, Start Y.
DDT_DIALOG_PARAM_DISTANCE_INITIAL_VALUES	Is used to set the initial distance values.
DDT_DIALOG_PARAM_DISTANCE_CLOSED	Is send when closing the dialog, so the draw mode and be reset in the Image Widget.

Table 46: Parameters of the Distance Dialog

3.3.2.2.18 Magnification Dialog

The Magnification Dialog will display a magnified part of the image surrounding the current mouse pointer location. The dialog contains the Magnification Widget described in section 3.3.2.1.10. The functionality of the dialog is the same as for the widget.

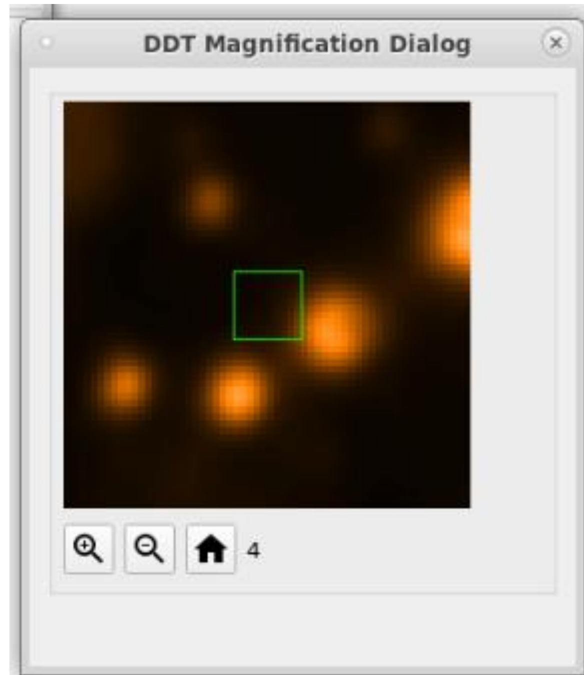


Figure 31: DDT Magnification Dialog

The Magnification 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_DLG_MAGNIFIED_IMAGE	Used to send the magnified image of the current mouse surrounding to the dialog.
DDT_DIALOG_PARAM_DLG_MAGNIFY_FACTOR	Used to inform the Image Widget about the currently selected magnification factor.

Table 47: Parameters of the Magnification Dialog

3.3.2.2.19 Save Image Dialog

It is possible to save the image that is actually displayed in the ImageWidget either as JPEG or as FITS file. In order to do this, right click into the ImageWidget to open up the context menu and select the “Save image” menu entry:

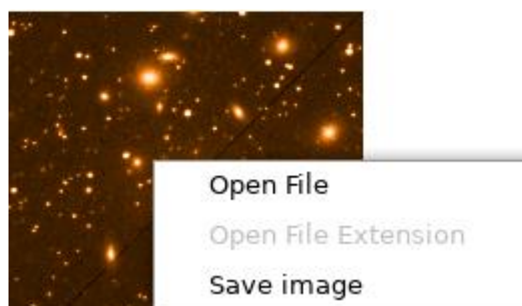


Figure 32: Open File Context Menue Entry

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Following dialog opens up:

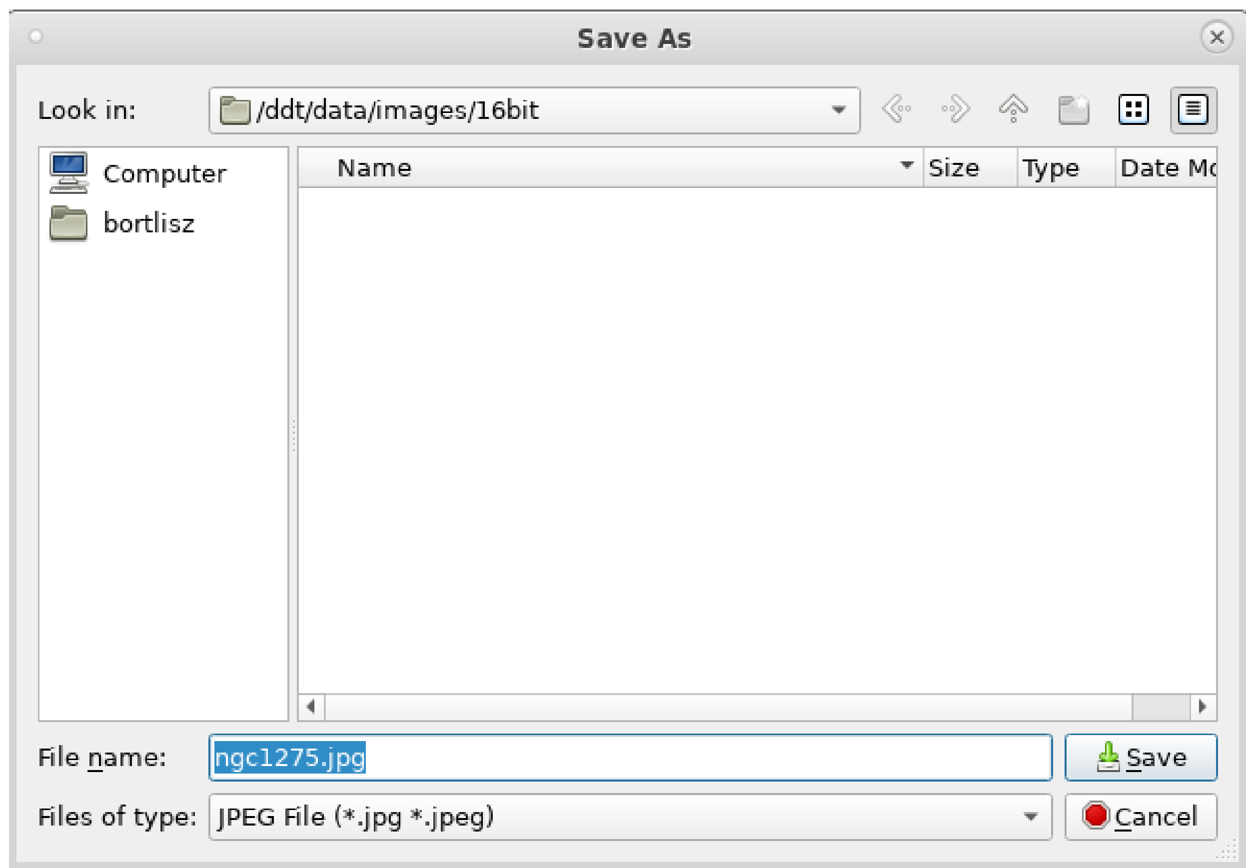
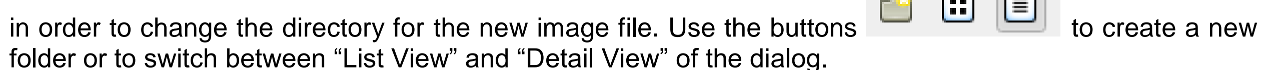


Figure 33: Open File Dialog

By default, the dialog opens up with the file type JPEG. In case that the DDT Viewer is attached to a data stream, the pre-selected file name is “newFile.jpg”; in case that a FITS file had been opened, the pre-selected file name will be the FITS file name with the extension changed to “.jpg”. If you want to save the image in the FITS format, use the “Files of type:” drop-down box and select “FITS File (*.fits). The file extension will change to “.fits” then. You can enter a different filename in the “File name:” edit field and determine the file type by either using “.fits” for FITS and “.jpg” or “.jpeg” for JPEG files.

Click on one of the folders shown in the main part of the dialog and / or use the drop-down box at the top (containing the folder name) and the arrow buttons beside this to navigate through the directory structure in order to change the directory for the new image file. Use the buttons  to create a new folder or to switch between “List View” and “Detail View” of the dialog.

Then, use the “Save” button to save the image with the specified filename. Note that a dialog will ask you if you want to replace the file in case that you selected the file specification of an existing file. Select “Yes” to replace or “No” to not overwrite.

Note that files are stored in their original orientation, i.e. even if the image has been rotated or flipped in the viewer, it will be stored without rotation and flipping. Also note that any overlays that had been drawn on the image will only be stored when saved as JPEG.

When stored as FITS file, the properties, either as read from the input FITS or from the data stream meta data, are stored, too, with an additional comment “Saved by DDT framework”.

Finally, use the “Cancel” button in the “Save As” dialog to cancel the save operation without saving the image.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

3.3.2.2.20 Mark position / End mark position

It is possible to mark points in an image and to display their coordinates. This can be achieved by right-clicking into the image widget and selecting the entry "Mark position". Afterwards, the user has to left click at several points in the image. Right-clicking into the image widget and selecting the entry "End mark position" will then display the coordinates in the console output of the DDT Viewer.

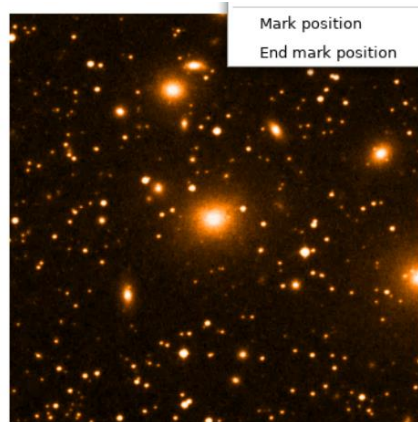


Figure 34: 'Mark position' Context Menue Entry

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
DDT_OVERLAY_IMAGE	Image object (supporting transparent background) created from an image file

Table 48: Overlay element types

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

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

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.
QList<DdtGraphicalElement*> GetElementByTag(const QString tag)	Return a list of graphical elements with a given tag that are contained in the overlay.
void ShowAllElements()	Show all overlay elements.
void HideAllElements()	Hides all overlay elements.
void ShowElementsOfType(const DdtOverlayType type, const bool showIds = false, const QString id_list = "")	Show all overlay elements of a given type.
void HideElementsOfType(const 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 49: API function of the Overlay API

The graphical overlay contains DDT Graphical Elements. Each of these elements allow to define their position by specifying the x/y coordinates. The following example demonstrates this for the graphical element of type ellipse. Here the position of the ellipse is set to (100, 100).

```
graphical_overlay = self.ui.ddtImageWidget.get_graphical_overlay()
prop = ddtWidgets.DdtGraphicalElementProperties()
ellipse = ddtWidgets.DdtGraphicalElementEllipse(prop, 100, 100, 50, 20)
graphical_overlay.AddGraphicalElement(ellipse)
self.ui.ddtImageWidget.RedrawOverlay()
```

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

Note that the graphical elements of type DDT_OVERLAY_IMAGE are not accessible via the DDT Graphical Elements dialog. These objects are generated out of image files from disc. This feature of image overlays supports image formats with transparent background (e.g. the PNG format).

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.

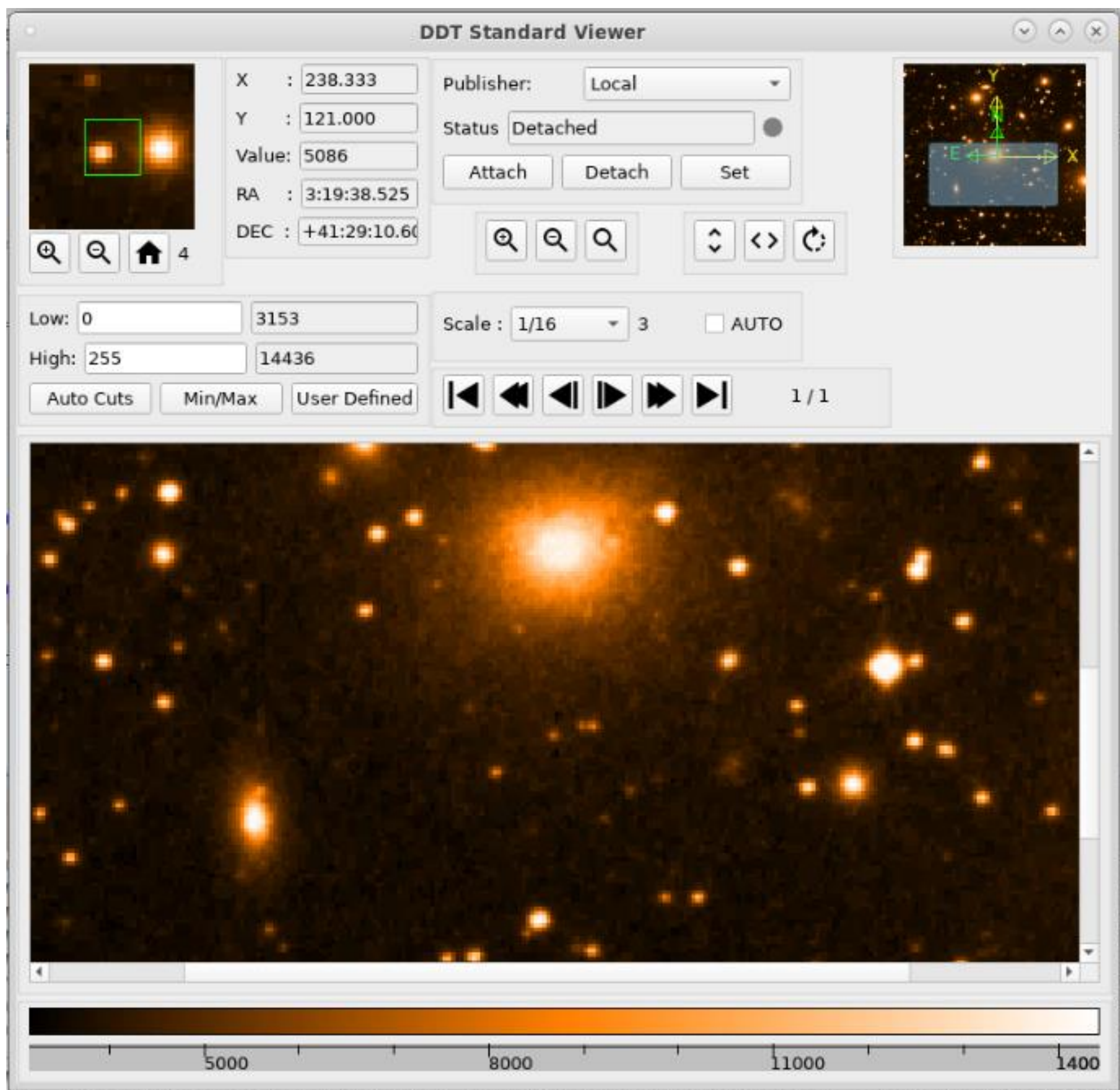


Figure 28: DDT Standard Viewer

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:

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
--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.

With the default scale option the default scale factor for loading new images can be defined.

The timestamp option can be used when attaching to a data stream for debugging purpose. When the argument is set to 1, the originator timestamp of each data sample will be displayed in the upper left corner of the image. In addition the time difference between the local system clock and the originator timestamp will be displayed in milliseconds (this difference is only useful, when both the publisher server and the viewer machine are time synchronous.)

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

3.3.2.5 Remote Control Interface

The DDT Viewer provides an optional commandline argument for launching a remote control server. By adding the argument “—remotecontrol_uri <URI>” the DDT Viewer will start a CII MAL server on the specified URI and can receive and process commands from a CII MAL client.

An example on how to start the Viewer with a remote server would be:

```
ddtViewer --remotecontrol_uri zpb.rr://*:5010
```

The Viewer will then print the following log message: “*Remote Control Server ready”.* Once the Remote Server was started, Remote Clients can connect to it and trigger commands that will be executed. Therefore, the Remote Control Interface allows clients to call the following function:

Function	Arguments	Return type	Description
HandleRemoteCommand	string image_widget_name, string command_name, vector<string> command_arguments	mal::future<string>	Handles the command sent by remote clients. The name of the image widget, the command and its arguments needs to get specified.

Table 50: Functions for Remote Control

The function will return a response as a string.

The following commands are supported and handled by the Remote Control library:

Remote command	Arguments	Description
list_commands	-	Lists all supported remote commands and their arguments.
attach	<URI> <Data Stream ID>	Attaches the Image Widget to a data stream specified by the Broker URI and the Data Stream ID.
detach		Detaches the Image Widget from the currently connected data stream.
flip	h / v	Flips an image horizontally [h] or vertically [v].
rotate	c / a	Rotates an image clockwise [c] or anticlockwise [a].
zoom	i / o / d	Zooms an image one step in [i], one step out [o], or zooms to the default value [d].

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Remote command	Arguments	Description
scale	f / d / <factor>	Changes the scale of an image to FIT [f], default scale [d], or to a specified factor in the range [1/20, 20].
select_points	-	Triggers the 'Mark position' feature of the Viewer.
statistics	-	Starts the Statistics dialog.
tabular	-	Starts the Tabular dialog.
scale_rotate_cut	-	Starts the Scale Rotate Cut dialog.
slit	-	Starts the Slit dialog.
pick	-	Starts the Pick Object dialog.
load	<file[HDU_index]>	Opens a specified file in the Viewer.
distance	-	Starts the Statistics dialog.

Table 51: Commands for Remote Control

If a Remote Control Server is currently processing a remote command, it will reject commands from other Remote Clients.

3.3.2.5.1 Provided response messages

Some of the commands listed in the previous section open a dialog in the DDT Viewer which returns values when the user triggers a response, e.g. by clicking the 'Ok' button. The values received by a Remote Client will be in the same format as they are provided by the corresponding dialog. The following table provides an overview of those commands and their response format.

Remote command	Response description
select_points	Returns a point list of (x, y) pairs when the user clicks 'End mark position' in the Viewer.
statistics	Returns statistic values in the following order when the user clicks 'Ok': MIN, MAX, MEAN, RMS, PIXELS, STARTX, STARTY, ENDX, ENDY
slit	Returns slit values in the following order when the user clicks 'Ok': X Offset, Y Offset, Target X, Target Y, Slit X, Slit Y
pick	Returns information about a picked object in the following order when the user clicks 'Confirm': Image Coord X, Image Coord Y, Pixelvalue, RA, DEC, Equinox, FWHM X, FWHM Y, Angle of X axis, Peak above Bg., Background, Pixels in X / Y
distance	Returns distance values in the following order when the user clicks 'Confirm': X OFFSET, Y OFFSET, START X, START Y

Table 52: Commands for Remote Client

3.3.2.5.2 Remote Client Interface

A Remote Client can be created using the MAL framework. The following pseudo-code snippet gives an example on how a Remote Client can be realized.

```
// Asynchronous MAL client with ReplyTime QoS set.
auto client = factory.getClient<remotecontrol::RemoteControlRegistrationAsync>
                (uri, {std::make_shared<mal::rr::qos::ReplyTime>
                (std::chrono::seconds(6))}, {});

// Explicitly wait for connection to be established.
auto connection_future = client->asyncConnect();
```


EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
auto future_status = connection_future.wait_for(::boost::chrono::seconds(10));
bool connected = (future_status == boost::future_status::ready);
if (connected) {
  std::cout << Server connection established << std::endl;
} else {
  std::cout << Server connection failed << std::endl;
}

std::string response = “”;
try {
  /**
   * Get response, future will block if the response is not yet received,
   * or TimeoutException is thrown if reply-time is exceeded.
   */
  mal::future<std::string> future =
      client->HandleRemoteCommand(image_widget, command, arguments);
  std::cout << "Waiting for response..." << std::endl;
  response = future.get();
} catch (std::exception&) {
  if (timeout == 0) {
    // no wait requested, don't do anything
    response = "Command sent, not waiting for server response";
  } else {
    response = "Configured timeout elapsed";
  }
  // make server available for new commands
  mal::future<std::string> future =
      client->HandleRemoteCommand(image_widget, "stop_remote", arguments);
  future.get();
}
```

Remote Control Servers can only process a single remote command one at a time. Since such a command may require some interaction by the user of the DDT Viewer, it may happen that a Remote Client stops (e.g. by running into a timeout) before the user has finished the action. In order to unblock the Remote Server and make it accessible for new commands, developers should make use of the “stop_remote” command. This command releases the corresponding promise object in the Remote Control Server (see pseudo-code above as an example).

3.3.2.5.3 Remote Client Example Application

The DDT provides a sample tool called Remote Client. The Remote Client is a command line application that can be used to connect to a Remote Control Server started by a DDT Viewer. The Remote Control Server can be either running on the same or on a different host than the Remote Client.

The Remote Client can be started using the command line:

```
ddtRemoteClient -s <URI of the Remote Control Server> -i <image widget name> -c
<command name> -a <arguments> [-t <timeout in sec>] [--infinite] [--debug]
```

The following example command can be used to flip an image horizontally:

```
ddtRemoteClient -s zpb.rr://127.0.0.1:5010 -i ddtImageWidget -c flip -a h
```

Once started the Remote Client will wait for a server response until a timeout is reached (specified via the *--timeout* parameter in seconds). If not specified the timeout is set to a default value of 3 seconds. The timeout can be set to 0 if it is not desired to wait for a server response. Specifying the *--infinite* flag will set the timeout to 24 hours (the *--timeout* parameter is overwritten in case it is specified in addition to the *--infinite* flag). The Remote Client will automatically close if the timeout was reached or a response of the Remote Control Server was received. In addition it can be closed by pressing `CTRL+C`.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

The Remote Client also supports the command line argument:

```
ddtRemoteClient --help
```

3.3.2.1 Rendering Libraries

Image data that is being displayed in the Image Widget will be using a rendering function from a separate Rendering Plugin library. The library can create different representation of the same type of image data.

All rendering functions should be derived from a common rendering plugin class called “DdtRenderingPlugin”. This class will create a “DdtImageGraphicsItem” which can then be added to the QGraphicsScene used to display the image data in the Image Widget. Here the “DdtImageGraphicsItem” is again the base class for graphics items derived from QGraphicsItem.

The two classes have the following interface functions:

DdtRenderingPlugin:

Function Name	Return Type	Arguments	Description
CreateGraphicsItem	**DdtImageGraphicsItem***	**cpl_image* image** **ddt::colorMap_t* color_map** **ddt::scalingLut_t* scaling_lut**	Creates a DdtImageGraphicsItem from a CPL image, a colour mal and a scaling lut.
CreateGraphicsItem	**DdtImageGraphicsItem***	**cpl_image* image** **ddt::colorMapARGB_t* color_map** **ddt::scalingLut_t* scaling_lut**	Creates a DdtImageGraphicsItem from a CPL image, a colour map containing ARGB values and a scaling lut.
CreateGraphicsItem	**DdtImageGraphicsItem***	**QString filename** **int width** **int height**	Create a DdtImageGraphicsItem from an image file. The width and height give the size of the rendering area.
CreateImage	**DdtImageGraphicsItem***	**std::vector<uint16_t> image_data** **int width** **int height**	Creates a DdtImageGraphicsItem from a data sample (data received as std::vector of uint16_t values. The width and height give the size of the rendering area.
GetRenderingPluginID	**int**	**-**	Returns the ID of the rendering plugin.

Table 53: Interfaces of the DdtRenderingPlugin

DdtImageGraphicsItems:

Function Name	Return Type	Arguments	Description
boundingRect	**QRectF**	**-**	Returns the bounding rectangle of the graphics item created.
paint	**void**	**QPainter* painter**	Method that will be used to render the graphics item.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Function Name	Return Type	Arguments	Description
		const QStyleOptionGraphicsItem* option **QWidget* widget**	Arguments are the QPainter used to draw the item, the QStyleOptionGraphicsItem for the style options and the pointer to the parent QWidget.
GetImage	**QImage***	**-**	Returns the current graphics item as QImage.
type	**int**	**-**	Returns the type of the graphics item.

Table 54: Interfaces of the DdtImageGraphicsItem

When the user plans to create a new rendering plugin for the Image Widget proper classes that implement the interfaces of the DdtRenderingPlugin and the DdtImageGraphicsItems need to be implemented.

When an Image Widget was instanciated the method:

ImageWidget::AddRenderingPlugins(DdtRenderingPlugin* const new_plugin)

can be used to add the new rendering plugin to the Image Widget. Once loaded the rendering plugin can be activated using the method:

ImageWidget::SetActiveRenderingPlugin(const int rendering_plugin_id)

Each rendering plugin should define its own plugin ID so the selection is unambiguous.

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

Note: Please see the doxygen documentation of the source code for a detailed description of the image handling.

3.3.4 Python Bindings

The Python bindings can be used to access the public API of the DDT software. They are created using pybind11 and Shiboken2. In total six Python modules containing the bindings for the DDT software are provided. This chapter will give an overview over the modules and how they can be used.

The data transfer components are divided into the module `DdtDataBroker` for creating a Python broker, and the `DdtDataTransfer`, which contains everything needed to create a Python publisher and subscriber.

Module	Classes
`DdtDataBroker (pybind11)`	`DdtDataBroker`
`DdtDataTransfer (pybind11)`	`DataSample`
	`DdtDataPublisher`
	`DdtDataSubscriber`
	`DdtEncDec`
	`DdtEncDecBinaryxD`
	`DdtEncDecImage2D`
	`DdtEncDecImage3D`
	`MetaDataBase`
	`MetaDataElementsBinaryxD`
	`MetaDataElementsImage2D`
	`MetaDataElementsImage3D`
	`WcsInformation`

Table 55: Data Transfer Components

The data visualisation components are divided in two modules as well. The widgets, dialogs and graphical elements can be found in the module `ddtWidgets`. Whereas the remote control components are provided in the module `DdtRemoteLib`.

Module	Classes
`ddtWidgets (Shiboken2)`	`DdtWidget`
	`DdtImageWidget`
	`DdtCursorInfoWidget`
	`DdtFlipRotateWidget`
	`DdtPanningWidget`
	`DdtCutValuesWidget`
	`DdtScaleButtonsWidget`
	`DdtMagnificationWidget`

Module	Classes
	DdtImageScaleWidget
	DdtColourmapWidget
	DdtDataStreamWidget
	DdtGraphicsView
	DdtOverlayType
	DdtGraphicalElementProperties
	DdtGraphicalElement
	DdtGraphicalElementCompass
	DdtGraphicalElementCross
	DdtGraphicalElementEllipse
	DdtGraphicalElementLine
	DdtGraphicalElementRectangle
	DdtGraphicalElementRefLine
	DdtGraphicalElementSlit
	DdtGraphicalElementStatRectangle
	DdtGraphicalElementText
	DdtGraphicalOverlay
	DdtOverlayRendering
	PickMode
	ConnectionStatus
	CornerGrabber
	DdtDialog
	DdtDialogFactory
	DdtCutValuesDialog
	DdtGraphicsControlDialog
	DdtTabularRegionDialog
	DdtBiasDialog
	DdtColourmapDialog
	DdtDataStreamDialog
	DdtDistanceDialog
	DdtFITSHeaderDialog
	DdtFITSTableDialog
	DdtGraphicalElementsDialog
	DdtStatisticDialog
	DdtSlitDialog
	DdtScaleRotateCutDialog
	DdtReferenceLineDialog
	DdtPVCMDialog
	DdtPickObjectDialog
	DdtOffsetDialog

Module	Classes
	DdtMagnificationDialog
	DdtHDUDialog
DdtRemoteLib (pybind11)	DdtRemoteClient
	DdtRemoteControl

Table 56: Data Visualisation Components

The image handling components that are needed for creating Python widgets and dialogs can be found in the module `ddtImageHandling`.

Module	Classes/Functions
ddtImageHandling (Shiboken2)	InitCpl()
	EndCpl()
	ImageHandling
	CutLevelType
	ColorScalingType

Table 57: Image Handling Components

At last there are also some utility components needed in Python, especially the `DdtLogger`, which can be imported via the `DdtUtils` module.

Module	Classes
DdtUtils (pybind11)	DdtLogger
	DdtStatistics

Table 58: Utility Components

3.3.4.1 Data Transfer 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;
```


EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
/**
Sample ID to uniquely identify the data sample
*/
int32_t sample_id;
/**
The image data as vector of bytes
*/
std::vector<uint8_t> data;
```

3.3.4.1.1 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 gettermethods 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.

3.3.4.1.2 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 = DdtLogger("DdtPublisherSimulator")
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 process_new_data():
  # process data here

connection = subscriber.connect(process_new_data)
```

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 for a python publisher is as follows (note: this could be started by passing the URI and the data stream identifier as arguments):

```
import sys
import signal
import time
import DdtDataTransfer
from DdtUtils import DdtLogger
from DdtDataTransfer import DdtEncDecImage3D
```

```
from DdtDataTransfer import MetaDataElementsImage3D

def create_metadata():
    md = MetaDataElementsImage3D()
    md.meta_data_base.bytes_per_pixel = 2
    md.meta_data_base.number_dimensions = 2
    md.meta_data_base.complete_flag = 1
    md.meta_data_base.last_segment = 1
    md.meta_data_base.byte_order_little_endian = 1
    md.meta_data_base.data_type = 1
    md.meta_data_base.description = "description"
    md.number_pixels_x = 353
    md.number_pixels_y = 353
    md.binning_factor_x = 1
    md.binning_factor_y = 1
    md.number_layers = 1
    md.item_size = 1
    return md

def create_data():
    data = [0, 0, 0]
    return data

def signal_handler(sig_num, frame):
    logger.write(2, "Ctrl-c was pressed. Exit now.")
    exit(1)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)

    uri = sys.argv[1]
    dsi = sys.argv[2]

    logger = DdtLogger("DdtPublisherSimulator")
    logger.write(2, "Starting DdtPublisherSimulator...")

    # create and register a publisher
    publisher = DdtDataTransfer.DdtDataPublisher(logger)
    publisher.SetBufferSize(1000,10)
    publisher.RegisterPublisher(uri, dsi)
    logger.write(2, "Publisher registered.")

    # create sample data
    data = create_data()
    # create encoder
    enc_dec = DdtEncDecImage3D()
    # create sample descriptions for the metadata
    descriptions = ["Mars", "Jupiter", "Saturn"]

    counter = 0
    while True:
        # create metadata
        md = create_metadata()
        # change description field
        md.meta_data_base.description = descriptions[counter % len(descriptions)]
        # encode the metadata
        enc_dec.Encode(md)
        # write and publish data
        publisher.WriteData(counter, data, enc_dec.get_meta_data())
        publisher.PublishData()
        # increment counter and go to sleep
        counter += 1
        time.sleep(0.5)
```


EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

A full example for a python subscriber is as follows (note: this could be started by passing the URI and the data stream identifier as argument):

```
import sys
import signal
import time
import DdtDataTransfer
from DdtUtils import DdtLogger
from gevent.events import subscribers
from DdtDataTransfer import DdtEncDecImage3D

def process_new_data():
    # read the data sample
    sample = subscriber.ReadData()
    logger.write(2, "topic id: " + str(sample.topic_id))
    logger.write(2, "meta data length: " + str(sample.meta_data_length))
    logger.write(2, "sample id: " + str(sample.sample_id))

    # create decoder and decode the metadata
    enc_dec = DdtEncDecImage3D()
    enc_dec.Decode(sample.meta_data_length, sample.meta_data)
    logger.write(2, "UTC timestamp: " + enc_dec.get_utc_timestamp())
    logger.write(2, "Description: " + enc_dec.get_description())
    logger.write(2, "Bytes per pixel: " + str(enc_dec.get_bytes_per_pixel()))
    logger.write(2, "Number pixels x: " + str(enc_dec.get_number_pixels_x()))
    logger.write(2, "Number pixels y: " + str(enc_dec.get_number_pixels_y()))
    logger.write(2, "Complete flag: " + str(enc_dec.get_complete_flag()))
    logger.write(2, "Last segment: " + str(enc_dec.get_last_segment()))
    logger.write(2, "Binning factor x: " + str(enc_dec.get_binning_factor_x()))
    logger.write(2, "Binning factor y: " + str(enc_dec.get_binning_factor_y()))
    logger.write(2, "Number layers: " + str(enc_dec.get_number_layers()))

def signal_handler(sig_num, frame):
    logger.write(2, "Ctrl-c was pressed.")
    subscriber.UnregisterSubscriber()
    logger.write(2, "DdtSubscriberSimulator unregistered. Exit now.")
    exit(1)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)

    uri = sys.argv[1]
    dsi = sys.argv[2]

    logger = DdtLogger("DdtSubscriberSimulator")
    logger.write(2, "Starting DdtSubscriberSimulator...")

    # create and register a subscriber
    subscriber = DdtDataTransfer.DdtDataSubscriber(logger)
    subscriber.RegisterSubscriber(uri, dsi)
    logger.write(2, "Subscriber registered.")

    # bind a method that is called each time new data arrives
    connection = subscriber.connect(process_new_data)

    while True:
        time.sleep(1)
```

3.3.4.2 Broker Module

The DdtDataBroker module contains python bindings for starting a broker with python. The module contains the class DdtDataBroker with the following methods:

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

```
Run(...)
    Run(self: DdtDataBroker.DdtDataBroker) -> int

__init__(...)
    __init__(self: DdtDataBroker.DdtDataBroker) -> None

setup(...)
    setup(self: DdtDataBroker.DdtDataBroker, arg0: List[str]) -> bool
```

The `setup(…)` method takes a list of arguments as parameters. The valid arguments are the same as described in the "Data Broker" section. The method returns `true` if the arguments could be parsed without any error, otherwise it returns `false`.

After calling the `setup(…)` method the `Run(…)` method can be called to start the broker. If the application is terminated it returns `0`.

```
>>> import DdtDataBroker
>>> broker = DdtDataBroker.DdtDataBroker()
>>> broker.setup(["-d","zpb.rr://*:5001"])
Logging properties do not exist: /eelt/ddt/0.1/resource/config/log4cplus_-d.properties
True
>>> broker.Run()
INFO - DdtBroker - Starting DdtBroker...
INFO - DdtBroker - Server URI: zpb.rr://*:5001/broker
INFO - ZpbServer: Server bound to 'tcp://*:5001'
INFO - ZpbServer: Creating server reply listener socket and binding to inproc://zpbServer.0
INFO - ZpbServer: Creating server reply notifier socket and connecting to inproc://zpbServer.0
DEBUG - DdtBroker - DdtConnectionManager: config file: /eelt/ddt/0.1/resource/config/databroker.ini
DEBUG - DdtBroker - shm timeout [s]: 10
DEBUG - DdtBroker - waiting time [ms]: 1000
DEBUG - DdtBroker - reply time for MAL clients [s]: 6
DEBUG - DdtBroker - heartbeat interval [s]: 1
DEBUG - DdtBroker - heartbeat timeout [s]: 10
DEBUG - DdtBroker - DdtConnectionManager: HeartbeatThread started
INFO - ZpbServer: Registering service 'Broker1' with hash 3901449245.
INFO - DdtBroker - Server ready
```

Figure 35: Example for Python Broker

Another example for a Python script can be found under ddt/py/brokerlib/src/pyDdtBroker.py. The script can be started by executing `python3 pyDdtBroker.py zpb.rr://*:5001`. This will start a broker with the specified arguments.

3.3.4.3 Data Visualisation Module

The bindings for the data visualisation component of the DDT software are created with Shiboken2. The Python module containing the bindings is called ddtWidgets and consists of the widgets, the dialogs and the graphics libraries. The following sections describe how to use them to create a Python viewer.

3.3.4.3.1 Creating The UI File

To build a viewer, a UI file must first be created. This can be done with the Qt Creator. To be able to use it with Python, it must then be converted into a Python class. The following command can be used for this:

```
pyside2-uic path/to/file.ui > ui_file.py
```

Then the includes must be changed. To do this, the file must be opened in an editor. At the top of the file you will find some lines like the following:

```
from ddt/widgets/ddtImageWidget.hpp import DdtImageWidget
```

These must be replaced by a proper Python import:

```
from ddtWidgets import DdtImageWidget
```

A full example can be found under `ddt/py/datavisualisation/src/ui_ddtviewer.py`.

3.3.4.3.2 Creating The Viewer

A full implementation of a Python viewer containing all the examples of this section can be found under `ddt/ py/datavisualisation/src/pyDdtViewer.py`.

A viewer application will need to import several libraries from the project.

module	
`ddtImageHandling`	Needed for initializing the CPL library
`ddtWidgets`	Only needed if dialogs shall be part of the viewer
`DdtUtils.DdtLogger`	The logger is needed by some widgets
`ui_file.Ui_FileForm`	The class generated in the step before
`DdtRemoteLib`	Only needed if the viewer shall support remote commands

Beside these some PySide2 libraries like `QApplication`, `QMainWindow`, `QObject` and `Signal` will have to be imported. The complete list depends on what the viewer shall support.

In the constructor multiple steps need to be performed. The parent object needs to be initialized, the UI object created an configured to work with the application. Also the logger object needs to be created and configured and the CPL library initialized.

```
super(PyDdtViewer, self).__init__()
self.ui = Ui_DdtViewerForm()
self.ui.setupUi(self)
self.logger = DdtLogger("DdtStandardViewer")
self.logger.configure("DdtStandardViewer")
ddtImageHandling.InitCpl()
```

Thereafter the signals and slots of the widgets need to be connected. An overview of the signals and slots of each widget can be found in the widgets section in this document. An example of a connection looks as follows:

```
QObject.connect(
            self.ui.ddtImageWidget,
            SIGNAL("CursorInfo(double, double, double, QString, QString)"),
            self.ui.ddtCursorInfoWidget.CursorInfo)
```

The viewer itself can also connect to several slots of the image widget. For connecting to them an object of the type `Signal` needs to be created which then can be connected to the slot via the `connect(…)` method. The `Signal` class has also a method called `emit(…)` which can be used to call trigger the slot.

```
set_filename = Signal(object) // create a Signal object
self.set_filename.connect(self.ui.ddtImageWidget.AttachDataFile) // attach it to the
slot
self.set_filename.emit(filename) // trigger the slot
```

The `Signal` class can be imported from `PySide2.QtCore`. The table below shows the slots that are used in the python viewer example implementation.

Slot	
`self.ui.ddtImageWidget.AttachDataFile`	Used for loading a file.
`self.ui.ddtImageWidget.AttachDataStream`	Used for connecting to a data stream.
`self.ui.ddtImageWidget.DetachStream`	Used for disconnecting from a data stream.
`self.ui.ddtImageWidget.SetNoWaitNewData`	Used for switching immediate display of new data on/off.
`self.ui.ddtImageWidget.SetImageScale`	Used for setting the image scaling factor.

Dialogs have to be created in the viewer and then added to a map containing all dialogs. This map is then given to the image widget object to initialize the dialogs.

```
dialog_map = {}
dialog_map["ddtHDU"] = ddtWidgets.DdtDialogFactory.createDialog("ddtHDU")
self.ui.ddtImageWidget.InitializeDialogMap(dialog_map)
```

The viewer class needs to implement the method `closeEvent(…)` which is automatically called when the viewer closes. In this method all dialogs should be closed and the connection to a data stream disconnected.

```
def closeEvent(self, event):
        if self.ui.ddtImageWidget:
            self.ui.ddtImageWidget.CloseAllDialogs()
        self.detach_datastream.emit()
```

3.3.4.4 Image Handling Module

The module `ddtImageHandling` consists of python bindings of the image handling library that are necessary for creating a python viewer. These are in particular the methods `InitCPL()` and `EndCPL()` for initializing and respectively terminating the CPL core library.

Further the module contains the class `ImageHandling` in which the enumerations `CutLevelType` and `ColorScalingType` are declared, with the first being used in the bindings for the widgets.

3.3.4.5 Remote API Module

The bindings for the remote API are contained in the module `DdtRemoteLib`. It consists of two classes, `DdtRemoteClient` and `DdtRemoteControl`.

The first can be used for starting a remote client application, as demonstrated in the image below. The `setup(…)` method of `DdtRemoteClient` receives the command line arguments which the remote client application needs. These are described in further detail in the Remote Control Interface section of this document. If all parameters could be parsed successfully the method returns `true` and the command can be executed by calling the `Run()` method.

```
>>> import DdtRemoteLib
>>> remote_client = DdtRemoteLib.DdtRemoteClient()
>>> remote_client.setup(["-s", "zpb.rr://127.0.0.1:5010", "-i", "ddtImageWidget", "-c", "flip", "-a", "h", "-d", "-t", "5"])
Logging properties do not exist: /eelt/ddt/0.1/resource/config/log4cplus_-s.properties
Debug triggered.
INFO - DdtRemoteClient - DdtRemoteClient: Server URI: zpb.rr://127.0.0.1:5010/viewer/RemoteControl
INFO - DdtRemoteClient - DdtRemoteClient: Server connection established
True
>>> print(remote_client.Run())
INFO - DdtRemoteClient - Waiting for response...
Image flipped horizontally
```

Figure 36: Example for Python remote client

An example for a Python script that starts a remote client application can be found under: ddt/py/remoteapi/src/pyDdtRemoteClient.py. The script can be executed with for example the following command in the specified directory: `python3 pyDdtRemoteClient.py -s zpb.rr://127.0.0.1:5010 -i ddtImageWidget -c flip -a h -d -t 5`.

The second class, `DdtRemoteControl`, is used for implementing the remote control functionality into a python viewer.

```
self.remote_control = DdtRemoteLib.DdtRemoteControl(remotecontrol_uri, self.logger)
self.remote_control.StartRemoteControlServer()
self.remote_control.connect_remote_command_signal(self.process_remote_command)
self.ui.ddtImageWidget.ConnectRemoteResponseSignal(self.remote_control.ProcessRespo
nse)
```

After creating the remote control object `StartRemoteControlServer()` needs to be called. Thereafter the signals and slots need to be connected.

The remote command signal needs to be connected with the slot that shall be called when a remote command is triggered. This is done by calling the `connect_remote_command_signal(…)` method with the user defined function to call as parameter.

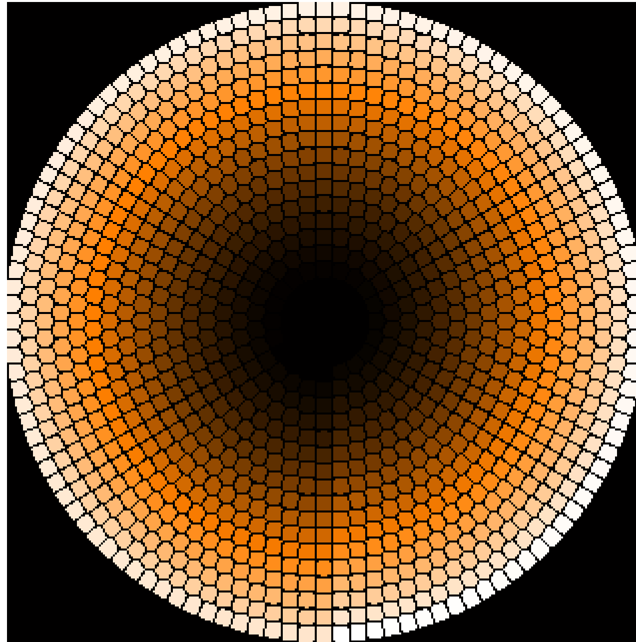
The response signal needs to be connected with the `ProcessReponse(…)` method of the `DdtRemoteControl` object. This is achieved by calling the `ConnectRemoteResponseSignal(…)` method of the image widget(s) with the `ProcessResponse(…)` method of the remote control object as parameter.

The full example on how to implement the remote support into a Python viewer application can be found under: ddt/py/datavisualisation/src/pyDdtViewer.py .

3.3.5 Configuration map handling

The configuration maps are stored in FITS file format. They contain for example the physical shape of DSM actuators:

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed



When receiving a numerical array of values that are meant to be displayed using the configuration map, the pixel values from the configuration map are used as index into the numerical array.

An example: a pixel with coordinates x=100 and y=150 and a value of index_val=234 in the configuration map will result in a pixel in the displayed image with the same coordinates x and y and a pixel value of „numerical_array[index_val]“.

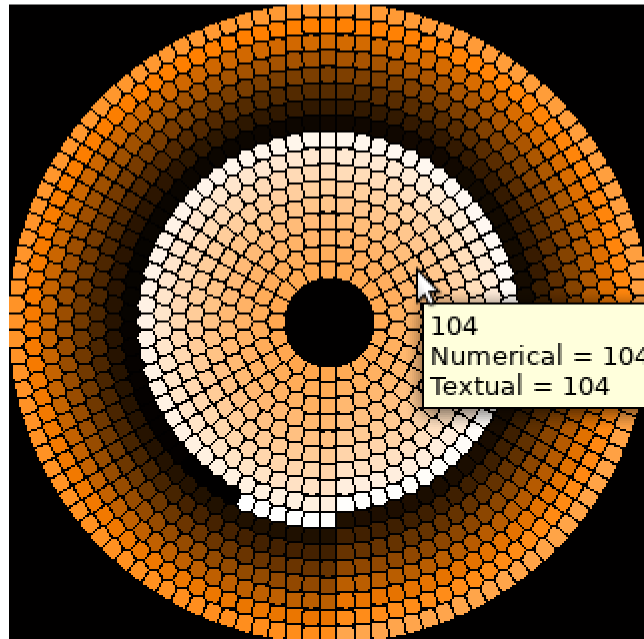
So, the resulting image will have the same dimension as the configuration map; the value range from the configuration map should then correspond to the number of values in the numerical array.

Beside the configuration map, it is possible to store information that is displayed when hovering over the image in the image widget. This information is stored in files in json format. For each value from the configuration map, a list of information can be stored, where each entry in the list consists of a name and a value. In the following example, a numerical value named “Numerical” and a textual information named “Textual” is stored for each value:

```
{
  "Values": {
    "Entry": [
        {
                "Value" : 1,
                "Information" : [
                        {"Numerical" : 1,
                         "Textual" : "1"}
                ]
        },
        {
                "Value" : 2,
                "Information" : [
                        {"Numerical" : 2,
                         "Textual" : "2"}
                ]
        },
…
        {
                "Value" : 1156,
                "Information" : [
                        {"Numerical" : 1156,
                         "Textual" : "1156"}
                ]
```

```
        }
      ]
    }
}
```

So, when hovering over the image, again the pixel value from the configuration map is used as index into the json file to get the information that is to be displayed as tooltip:



These information files have to be located together with the configuration map files, where each information file has to have the same filename as the corresponding configuration map file, but with the file extension „.json“.

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
	above. The checksum argument can be used to enable/disable the calculation of checksums for Data Samples.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

Command	Description
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>] [--defaultscale <scale>] [--timestamp 0|1]	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. The optional flag for the default scale can be used to configure the default scale with which new images be displayed. The optional flag for the timestamps can be used for debug purposes when connecting to a data stream. It will display the timestamp of the data sample and the time difference to the local system time when the image was displayed.

Table 59: 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.

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed

< Last Page of Document >

EU Proprietary information. Unauthorised distribution, dissemination or disclosure not allowed