The background of the cover is composed of several overlapping, semi-transparent rectangular panels. Each panel contains a grayscale fluorescence microscopy image of cells. The cells appear as bright, circular or oval structures against a darker background, likely representing nuclei or specific organelles stained with a fluorescent marker. The panels are arranged in a staggered, overlapping fashion, creating a layered effect.

ATExplorer 1.0 - User Manual

Smith Lab, Allen Institute for Brain Science

October 2018

Preface

AT introduction here..

The ATExplorer application integrates a workflow, as well as a number of software components that are useful in the context of Array Tomography .

The following software components are the main building blocks that ATExplorer is built on top of:

- **RENDER PYTHON** by F Collman et al. RenderPython is a thin Python wrapper for *Render*.
- **RENDER** by ??? et.al
- **FIJI** by et. al....
- **DOCKER** et. al.

In addition to the above, semi specialized software packages, a number of open source, C++, libraries are employed by the ATExplorer application:

- **VTK** by
- **POCO** by ??? et.al
- **LIBCURL** by et. al....
- **TINYXML2** et. al.
- **DUNE SCIENTIFIC LIBRARY (DSL)** et. al.

The ATExplorer application was designed and implemented in the lab. of Stephen J Smith and

Forrest Collman, at the Allen Institute of Brain Science by Totte Karlsson.

The following people has been contributing to the effort;

Part One

1	Overview of the ATE Explorer UI and API	7
1.1	Introduction	
1.2	A manual Array Tomography workflow	
1.3	The Render Service	
1.4	The ATE Explorer UI	
1.5	Python Bindings	
	Appendices	13
A	Setup a RenderHost and RenderPython	17
B	Python Enabled API's	19
C	Software Design and Software Components	21
C.1	ATE Explorer UI	
C.2	ATE Explorer Software API's	
C.3	ThirdParty libraries	
D	How to Build ATE Explorer and its API's	25
E	Data formats	27
E.1	Allen institute stat format	
E.2	Kristina format	
F	Arraybot and ArrayCam	29
G	atDB	31
H	ATClassifier	33



1. Overview of the ATEplorer UI and API

1.1 Introduction

This chapter gives an overview of the software application that is named *ATEplorer*.

The ATEplorer application was designed and implemented due to an emerging need to allow *non-programmers* to process, manage and explore Array Tomography data on a routine basis.

Depending on the actual protocols, an Array Tomography data set can range in size from a few hundred megabytes, to very large, like several Terra bytes. Depending on the number of stains and sessions, the complexity of the data-sets range from trivial to complex.

One of the main challenges in Array Tomography is the precise digital reconstruction of an original volume, i.e. from individually cut and imaged slices of tissue.

However, before volume reconstruction can begin, various pre data processing algorithms need to be applied, such as median filtering, flat-field correction and de-convolution. These processing algorithms are all, to some extent, complex. ATEplorer provides the non-programmer user with intuitive and easy to use UI components to guide through this process, in order to quickly get to data that is useful for scientific discovery and exploration.

1.2 A manual Array Tomography workflow

In order to get from raw, acquired data to fine aligned 3D volumes, a number of processing steps, i.e. a *pipeline*, is required. (See pipeline figure)

- FLATFIELD CORRECTION
- DECONVOLUTION
- STITCHING
- REGISTRATION
- ROUGH ALIGNING
- FINE ALIGNING
- OTHER

The pipeline is integrated into the ATE Explorer UI. However, in certain situations it may be necessary to push data through the pipeline manually. ATE Explorer in ... provide a set of Python scripts that can be used for exactly this purpose. The scripts are to be found in the folder /atPipeline/source and /atPipeline. These scripts can be executed on their own, or from within the UI. each pipeline script is discussed in terms of their purpose, input and output data, in the sections below.

1.2.1 Creation of *state tables*

Fast stacks

Creates down-sampled images and tile specs...

```
#-----
# Name:          atMain
# Purpose:       Run multiple AT processing scripts
#-----

import os
import atutils
import create_state_tables
import create_rawdata_render_multi_stacks
import create_median_files
import create_flatfield_corrected_data
import timeit

if __name__ == '__main__':
    timeStart = timeit.default_timer()

    #What data to process??
    dataRootFolder = "F:\\data\\M33"
    ribbons = ["Ribbon0004"]
    sessions = ["Session01",
```



```
"Session02",  
"Session03"  
]
```

1.2.2 Creation of median files

1.2.3 FlatField correction

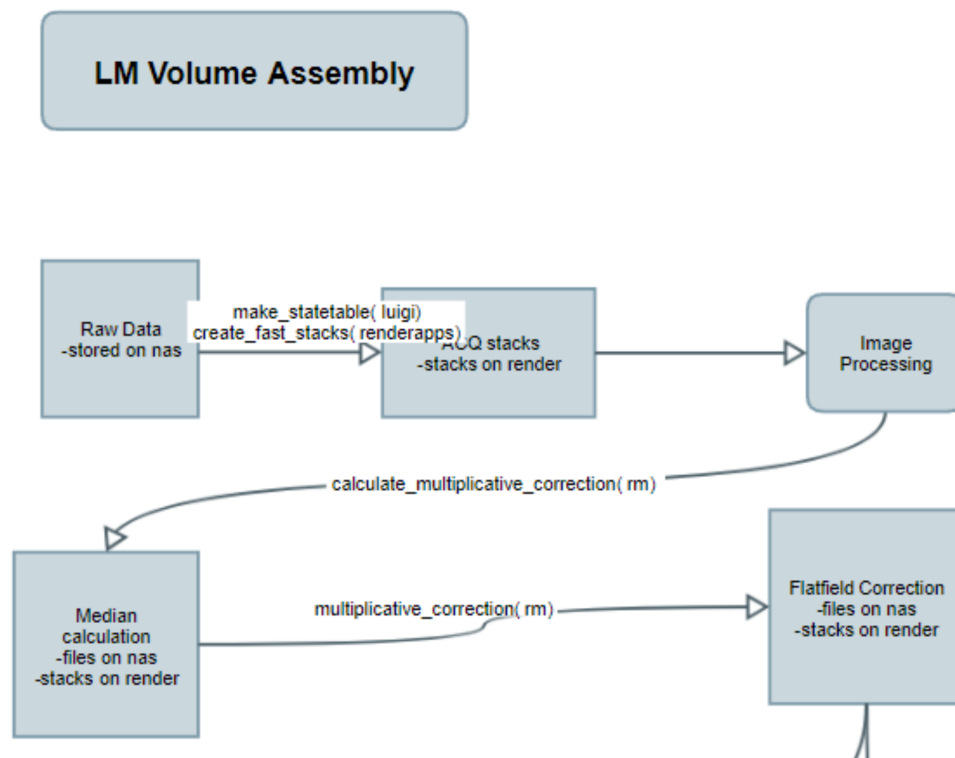


Figure 1.1: Processing

1.3 The Render Service

1.4 The ATEplorer UI

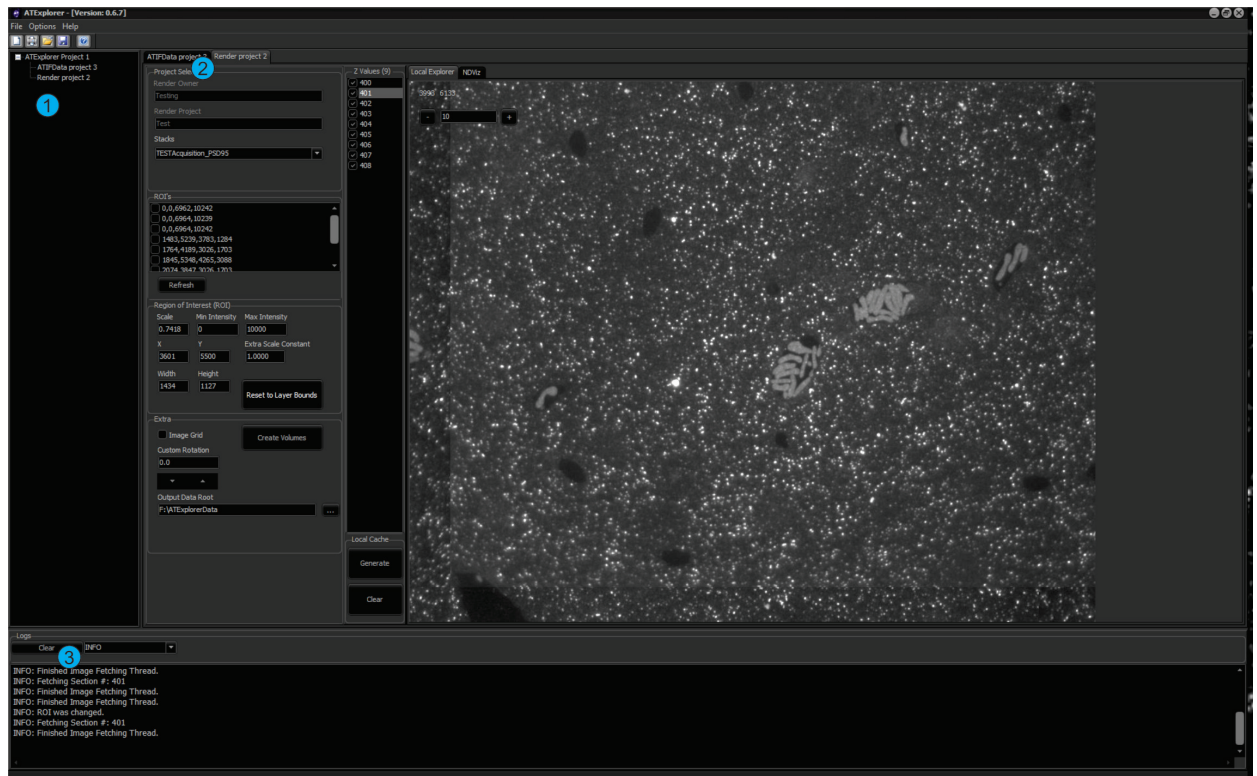


Figure 1.2: ATEplorer UI. The circled numbers in the figure indicate relevant elements of the UI; ① Project(s) TreeView. ② Tabbed Project Item View. ③ Information and Application Log Messages.

1.4.1 Importing Data

- **IMPORTING PROCESS** Give an overview on what happens when data is being imported to ATEplorer.
- **DATA FORMATS** Describe the Allen Institute format, and Kristinas format.

1.4.2 Connect to a Remote (or local) RenderHost

1.4.3 Create of RenderStacks

1.4.4 Manage Stacks in Render

1.4.5 Explore Data

1.5 Python Bindings

Appendices

A	Setup a RenderHost and RenderPython	17
B	Python Enabled API's	19
C	Software Design and Software Components	21
C.1	ATExplorer UI	
C.2	ATExplorer Software API's	
C.3	ThirdParty libraries	
D	How to Build ATExplorer and its API's	25
E	Data formats	27
E.1	Allen institute stat format	
E.2	Kristina format	
F	Arraybot and ArrayCam	29
G	atDB	31
H	ATClassifier	33



A. Setup a RenderHost and RenderPython

AT Deployable

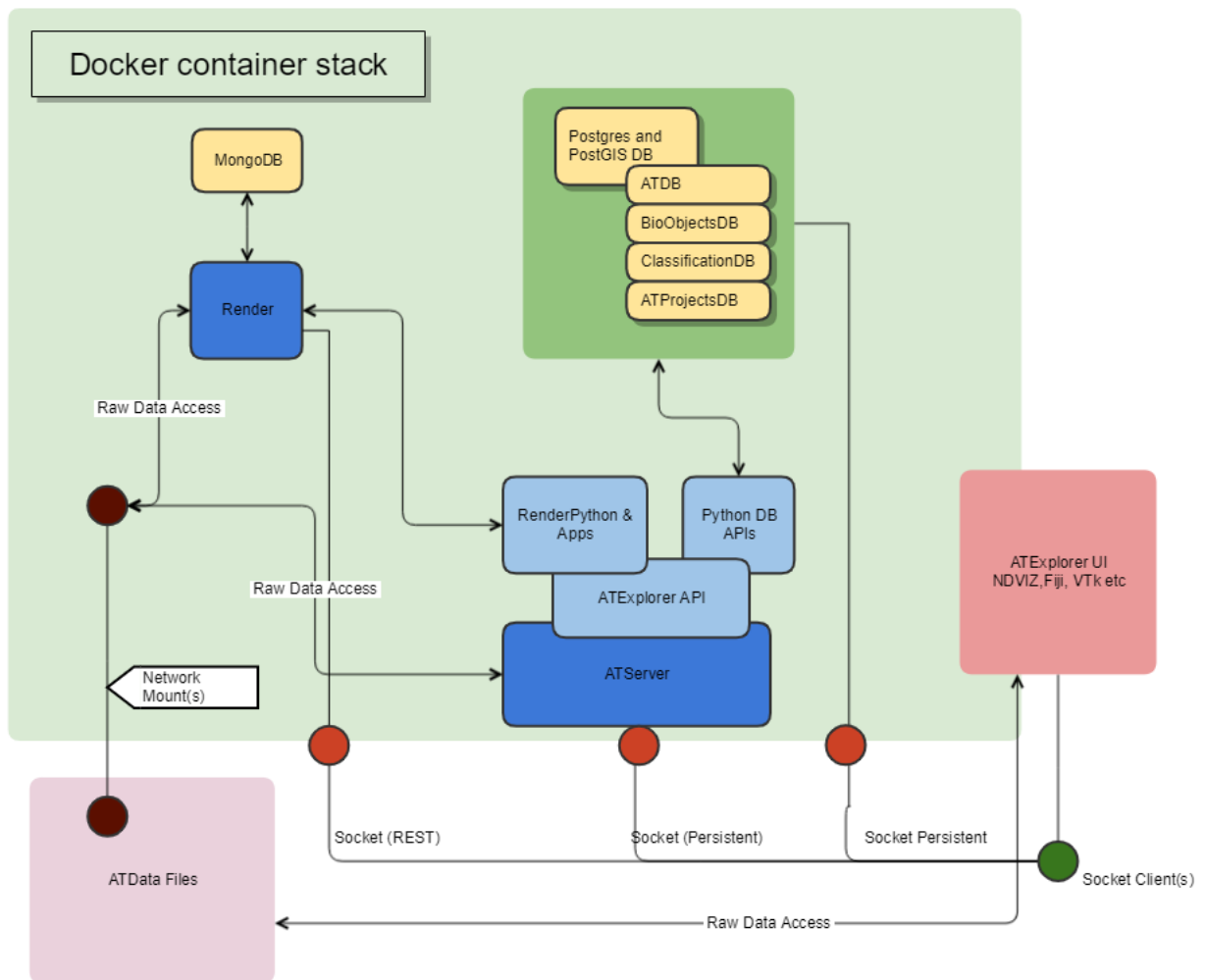


Figure A.1: A deployed system



B. Python Enabled API's



C. Software Design and Software Components

C.1 ATEplorer UI

The ATEplorer UI is a Microsoft Windows desktop application implemented using Embarcadero's C++ Builder tools. The C++ Builder environment provide a programmer with hundreds of components for efficient and rapid development of Windows applications. In addition, thousands of third party components are available as well.

This appendix discusses some of the software designs and software components employed when implementing the ATEplorer

C.1.1 Observers and subjects

The Tree view and PageControl .

C.1.2 The TreeView

The items in the Treeview stores data objects as (void*) pointers. Any object registered with the tree (as a node) need to be a descendant of the class ExplorerObject. Typical scenario:

```
ExplorerObject* eo = (ExplorerObject*) node->Data
```

```
if(dynamic_cast<...>(eo))
```

```
{
```

```
...
}
```

C.1.3 Populating an ATData object

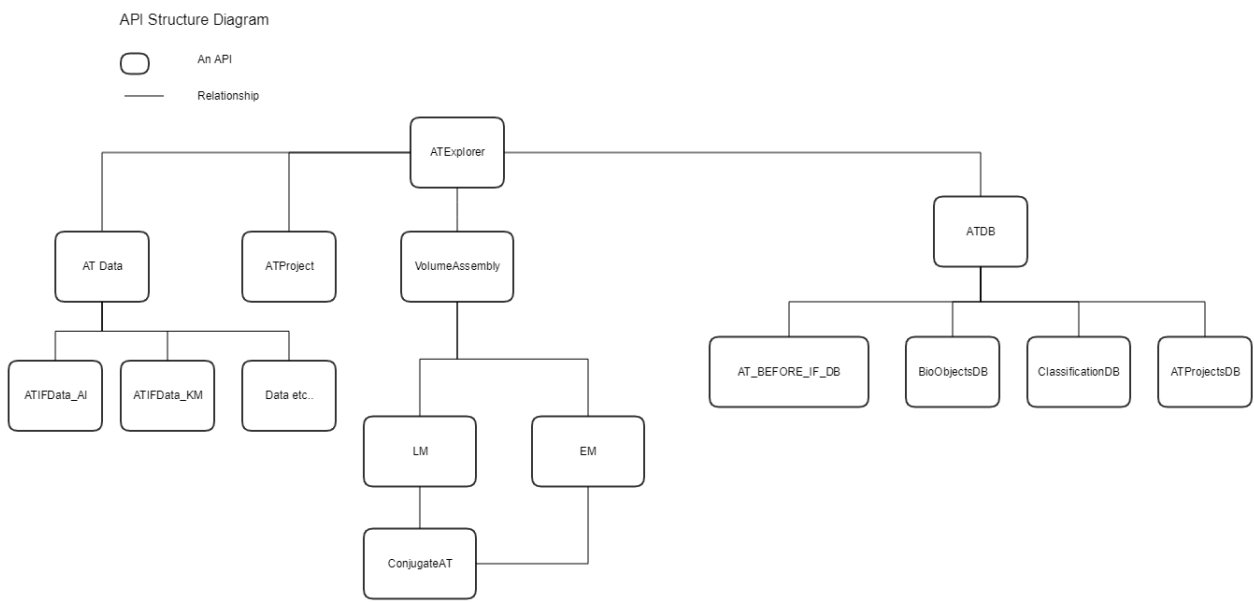


Figure C.1: An overview of some of ATExplorer API’s

C.2 ATEplorer Software API's

C.2.1 atCore

C.2.2 atData

C.2.3 atVCLCore

C.3 ThirdParty libraries

C.3.1 Poco

C.3.2 libCurl

C.3.3 SQLite 3

C.3.4 TinyXML2

C.3.5 Dune Scientific libraries: dslFoundation



D. How to Build ATEplorer and its API's

Public Software Repository: [git@github.com : TotteKarlsson/ATEplorer.git](https://github.com/TotteKarlsson/ATEplorer.git)



E. Data formats

E.1 Allen institute stat format

E.2 Kristina format



F. Arraybot and ArrayCam



G. atDB



H. ATClassifier