HDF5 Indexing and Searching

Research of the Indexing and Searching possibilities of HDF5 files

Bachelor in de toegepaste informatica

Niels Van Rooy

Academiejaar 2013-2014

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

PREFACE

In the last year bachelor in Applied Computer Sciences, all students do an internship of 13 weeks. We had many options but only few were appealing to me. I eventually chose to do my internship at Janssen Pharmaceuticals. This internship interested me because of the challenges it provided. This documents will describe the progress I made during my internship.

My personal goal during my internship and thesis was to be challenged and to gain knowledge. Looking back to the complete experience, my expectations were satisfied. I had the honor of having an expert Java developer as a colleague. He always took the time and effort to answer every question I had. Due to the complexity of my assignment, I couldn't have completed this without him. I would like to give special thanks to this colleague, Frederick Michielssen, for sharing all his knowledge with me.

Further, I would like to thank Frans Cornelissen, my supervisor at Janssen, for providing me with an amazing internship, believing in my capabilities and allowing me to research some very interesting topics. His knowledge and guidance was crucial in this internship. I would like to thank my school, Thomas More Kempen and Janssen Pharmaceuticals for providing me this opportunity. I would like to thank my college tutor Christine Smeets for her guidance through this internship. Finally I would like to thank all my colleagues, Frederick Michielssen, Jan Dockx and Vincent Tanghe for their support.

SUMMARY

This document describes the work and research I have performed during my internship at Janssen Pharmaceuticals

The Research and Development (R&D) department at Janssen Pharmaceuticals works hard to discover new medicines. This process is supported by the R&D IT department which provides the researchers with all necessary IT solutions. All the tests that are being done have to be analyzed as efficient as possible. Phaedra, the project I worked on, is a High Content Screening tool developed by the IT department to support their scientists.

Phaedra reaches the release of its third version. This means that a lot of new features have been developed which needed to be documented. This documentation had to cover all the new functionalities of Phaedra and needed to be integrated within Phaedra. DITA is a documentation technique which enables you to create an advanced technical documentation. I had to implement DITA into Phaedra and write a good and strong technical documentation. The implementation of DITA succeeded and the new documentation was developed.

The third release of Phaedra can still use some additional functionalities. One of these functionalities is the search option for cellular data. Cellular data in Phaedra is stored in the scientific data format HDF5. This data format does not provide search functionalities, because of the large amount of cellular data in Phaedra. A good indexing and searching technique is required. I was responsible for the task to research the HDF5 indexing and searching possibilities. The best possible solutions were researched and the test cases for these solutions were made.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

As a last year bachelor student I performed an internship of 13 weeks. Development is a passion for me and not just a profession. It was very important to me to find a stimulating place to explore my development capabilities. Janssen Pharmaceuticals offered me that. A challenging task to explore my development skills: documenting and understanding a complex application and researching the possibilities to implement a complex searching and indexing technique.

Janssen Pharmaceuticals has a large number of IT projects to keep its business going. A lot of these projects support the scientists researching new medicines. Phaedra, the project I was assigned to, is supporting its scientists by developing a state of the art High Content Screening (HCS) tool. The application supports scientists by visualizing and analyzing results of experiments. Because of the complexity of all these data within Phaedra, a good documentation is required that describes all the functionalities.

The development of new medicines is based on extensive analyzing of microscopic data. The more features Phaedra provides, the more efficient scientists can analyze data. A key feature is the search functionality to find matching data. In Phaedra this features already exists but not on a cellular level. All cellular data in Phaedra is stored in a scientific data format called HDF5. To complete the Phaedra search functionality, this data must be searchable as well.

The content of this document gives insight in my contributions to the new documentation and my research.

At the end I will summarize the results of my internship and form a conclusion.

1 JANSSEN PHARMACEUTICALS

In this chapter I will describe the background of the company where I did my internship.

1.1 About Janssen Pharmaceuticals

Figure 1: Janssen Pharmaceuticals Logo

The project I worked on is developed for Janssen Pharmaceuticals NV. Janssen Pharmaceuticals was founded in 1953 by Doctor Paul Janssen. The main focus of Janssen Pharmaceuticals is the research and development of cures and new medicines. Later on, Janssen started to focus itself on the mass production of medicines.

With over 80 new medical drugs, Janssen Pharmaceuticals is considered to be one of the most innovative medical companies in the world. Their research is spread over a wide range of disease areas including:

- Mental disorders
- Neurological problems
- Infectious diseases
- Immunological disorders
- Cancer
- Cardiovascular
- …

Janssen Pharmaceuticals is a company which strives to innovation. To stay at the top of the medical sector, new projects are started to improve the research and development of new medicines. The Phaedra project, the project I worked on, is a project to facilitate the research of new medicines.

1.2 Dr. Paul Janssen

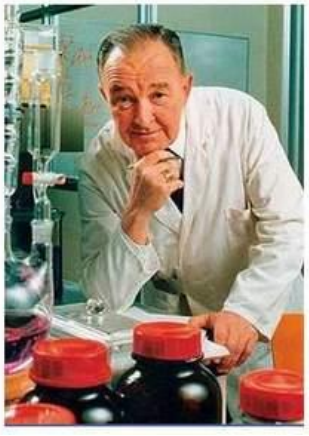


Figure 2: Dr. Paul Janssen

Dr. Paul Janssen is the founder of Janssen Pharmaceuticals. After graduating in medicine from the University of Ghent he started a small research laboratory which later became Janssen Pharmaceuticals. He developed a lot of new drugs since 1958 when he had a major breakthrough in the treatment of schizophrenia.

Paul Janssen received several honors and awards during his career. He received more than 80 medical prizes and 22 honorary doctorates. In 2005, 2 years after he passed away, an award was founded by Johnson & Johnson to honor the memory of Paul Janssen.

1.3 About Johnson & Johnson

Figure 3: Johnson & Johnson logo

Johnson & Johnson is an American multinational who purchased Janssen Pharmaceuticals in 1961. Paul Janssen wanted to ensure the future of his company and thought that Johnson & Johnson was the best option. Johnson & Johnson belongs to the fortune 500 in the world.

Nowadays Johnson & Johnson has 3 major focusses:

- Medical devices
- Pharmaceuticals
- Consumer Packaged Goods

Janssen Pharmaceuticals plays a great role within Johnson & Johnson. It's one of J&Js leading Research and Development subsidiary.

1.4 Research & Development Department

The R&D department within Janssen Pharmaceuticals exists of several sections. We were located in the R&D IT department. The R&D IT department develops and maintains new solutions to improve the R&D capabilities. It's one of Janssen's most important departments to keep its status as an innovative company. One of the innovative projects is Phaedra. Phaedra is a High Content Screening tool to help scientists of the R&D department processing and interpreting test results.

2 PROJECT PLAN

This chapter will give you an overview on the background and context of my assignments and the goal of my internship. This chapter will concentrate on the existing limitations of Phaedra when I started my internship and the results which needed to be achieved at the end of my internship.

2.1 Context and Background of the Internship

Phaedra is a software package developed by Janssen Pharmaceuticals NV. It is a High Content Screening tool used to analyze incoming microscopic data for research purposes. Researchers use Phaedra as a tool to analyze the different reactions of a substance. The analysis of substances and its effect are done in Phaedra because of its High Content Screening abilities. This result in an easier way to finding substance reactions and therefore the development of medicines.

The application is already in use in their R&D department. As the needs within the R&D department grew, Phaedra had to keep growing with the demands. That's why a new version, Phaedra 3.0, was needed.

Even though Janssen Pharmaceuticals has been working on this new version for a while, there are still some parts where the development team could use some help. These parts include:

- **Phaedra documentation**

Because of the complexity, increasing functionalities and increasing user-base of the Phaedra application, a good documentation is required. The existing documentation is outdated because it is written for previous versions of Phaedra. There are a lot of new functionalities and upgrades that require a good level of documentation so a new version of the documentation is required.

- **HDF5 Indexing and Searching**

One of Phaedra's functions, besides analyzing data, is allowing the users to search easily through data. A lot of search queries are already possible, but a search on a cellular level isn't possible yet. Cellular information is stored in HDF5 files and even though HDF5 files are widely used, there isn't an easy way to search through these files on a performance efficient way because of several reasons:

- The cellular data isn't stored in a relational database like the other data because of its enormous size. A single HDF5 file containing cellular data can be larger than 1GB.

- HDF5 is an open source library designed to store and organize large amounts of numerical data. It's primarily used for scientific data.

- HDF5 consists of two major types of objects: datasets and groups. Datasets are multidimensional arrays of a homogeneous type and groups are containers which can hold datasets and/or other groups.

2.2 Project goals

To fulfill the previous described needs, we will deliver the results according to the following goals:

Phaedra Documentation

- The documentation needed to be written according to the DITA (Darwin information Typing Architecture) standard.
- The editing of DITA needed to be done with DITAworks.
- The documentation needed to be implemented into Phaedra and be available in PDF and HTML format.
- The style of the documentation needed to be similar to previous editions.
- The documentation needed to be available for different users with different abilities (Advanced User, Basic User).
- The documentation needed to be easily accessible from the Phaedra Application (use of F1 help key to jump to the chapter that corresponds to the current use).

HDF5 Indexing and Searching

- The best possible solution for HDF5 indexing and searching had to be found. This solution had to be compatible with Java.
- A search system that allowed easy and performance efficient search through several HDF5 files.

2.3 Business Case

The Phaedra documentation was a very important feature for the 3.0 release. A good documentation means more support for the end-users. More support for the end-users means less confusion. If the end-user can find help regarding every aspect of Phaedra in the documentation, less support is needed.

The HDF-5 search solution gives the end-user the possibility to search in Phaedra on a cellular level. It will increase the possibilities researchers have to analyze data. If this function can be included, it will substantially increase the value of Phaedra.

2.4 Project plan

To complete the different assignments, a good planning is necessary. The assignment was divided in 2 projects. The Phaedra documentation was a shared project and the HDF5 search project was an individual project. The planning was divided over 13 weeks.

2.4.1 Phaedra documentation

The Phaedra documentation is developed in a team effort. All the tasks were split up between my colleague Vincent and me.

Tasks
Introduction to Phaedra and its current documentation. The introduction was given in the first week.
Setting up the DITAworks environment and starting to develop the first topics for the new Phaedra Help. This was done in week one and two.
Setting up Context Sensitive Help in Phaedra for the existing topics. This was done in week two, three and ten.
Developing the Phaedra Workbench Guide (Advanced and Basic version). This was done in week two and three. Committing the last updates to this manual was done in week seven.
Developing the Phaedra User Manual. This was done in week two, three and four. Committing the last updates to this manual was done in week ten and twelve.
Styling the PDF versions of the manuals. This was done in week three and four.
Setting up the Phaedra help through Java. This was done in week three, five and twelve.

2.4.2 HDF5 Indexing & Searching

Tasks
Gathering information about the possible HDF5 indexing and searching solutions. This was done in week four and five.
Researching the possible solutions (Lucene, Pytables, MongoDB Big Data Database, FastQuery). This was done in week five and six.
Creating a test scenario for a Lucene implementation in Java. This was done in week six and seven.
Compiling and testing of a FastQuery implementation. This was done from week seven to thirteen.
Modifying FastQuery to perform optimal in a Java based environment. This was done from week eight to thirteen.

2.5 Primary Target Group and Other Stakeholders

The people that will benefit from this project are:

- The R&D department at Janssen pharmaceuticals
- The Phaedra end-users

The advantages for the R&D department at Janssen pharmaceuticals are:

- A good documentation will mean fewer calls regarding how the program works and so there will be less need for support;
- A good documentation will allow Open Source users to better understand the benefits of the Phaedra Application. This means that less support is needed from the R&D department.

The advantages for the Phaedra end-users are:

- A good documentation will make it easier to understand the many possibilities within Phaedra and will ease the use of Phaedra meaning that using Phaedra will really become time reducing;
- A good implementation of the documentation means that users will easily find the documentation they are looking for while using the application;
- A good implementation of the HDF-5 search possibility will enhance the search possibilities and allow users to find data in an easier and more efficient way.

2.6 Information and Reporting

During the process of this project, we worked at Janssen Pharmaceuticals in Beerse. This means I reported directly to my project leader and to other people that worked on the Phaedra project.

Every week, a document was sent with my weekly progress to our internship mentor and every few weeks I discussed my progress with my school tutor.

3 SOFTWARE AND PROGRAMMING LANGUAGES

This chapter summarizes the different software programs and programming languages used during my internship.

3.1 Software

The following chapters explain the different software programs I used.

3.1.1 Eclipse

Figure 4: Eclipse Logo

Eclipse is an Open Source software development environment maintained by the Eclipse Foundation. It is an Integrated Development Environment (IDE) used to develop and maintain applications. Eclipse is mostly written in Java and is widely known among developers because it supports a lot of different programming languages like C, C++, Java, Python…

Eclipse was created in 2001 as an IBM project. IBM was attempting to replace their outdated VisualAge IDE family. A few months later, a consortium was formed making Eclipse an Open Source project. By 2004, the number of stewards engaged in the development of the Eclipse environment had increased so much that the Eclipse Foundation was founded. The Eclipse Foundation maintains the Eclipse project with the contributors from all around the world.

Eclipse has one of the largest communities of developers. Because of its large community, a lot of free plugins are available. This greatly enhances the user experience and possibilities of Eclipse.

The Eclipse Rich Client Platform (RCP) is used to develop Phaedra. Eclipse RCP is a plug-in based environment meaning that it uses plug-ins to provide functionalities. An RCP is highly customizable to all your needs. This is why Phaedra chooses to work with an RCP over an IDE. The Image below shows you the Phaedra Eclipse RCP environment.

Eclipse was used to implement the new Phaedra help and to enable Context-Sensitive Help.

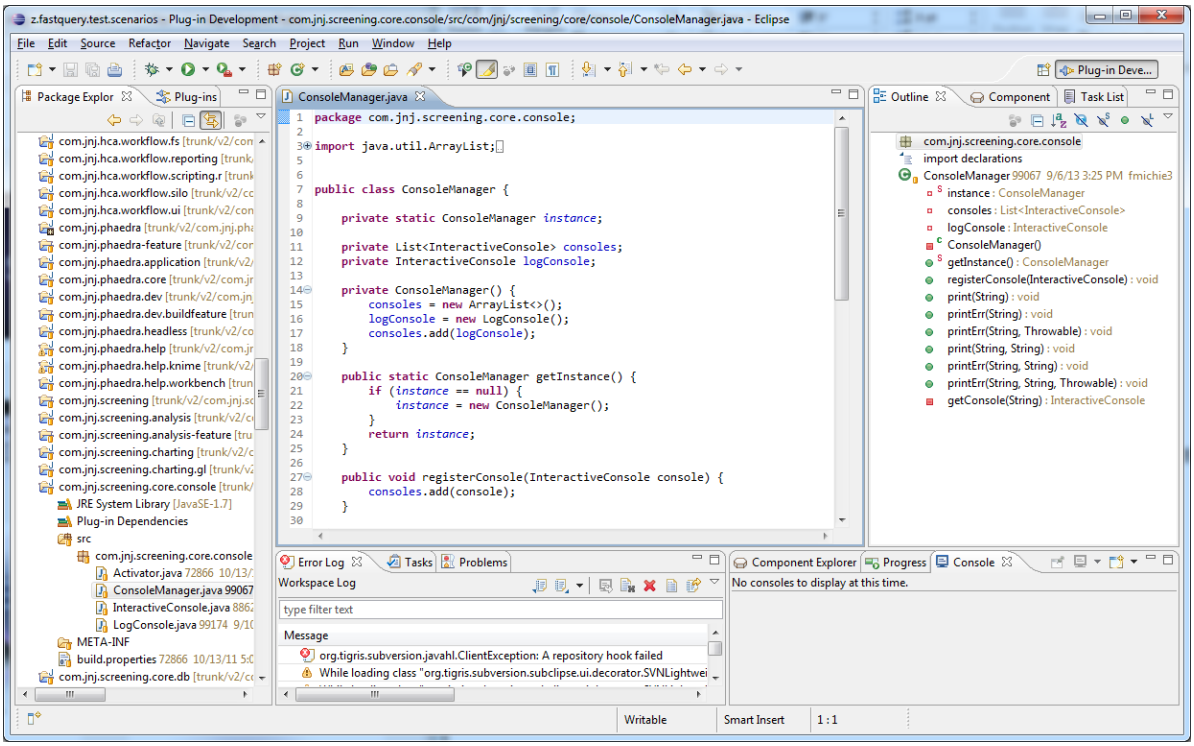


Figure 5: Eclipse RCP Environment

3.1.2 DITAworks

Figure 6: DITAworks Logo

DITAworks is a Darwin Information Typing Architecture (DITA) Content Management System developed by instinctools GmbH. DITA is an XML data model and standard. It is designed as an end-to-end architecture for modeling, authoring and publishing structured content and technical documentation. DITA uses the following features:

- Topic orientation
- Maps
- Content reuse
- Metadata
- Information typing
- Specialization

DITAworks is a powerful tool to generate and maintain documentation for different platforms/output based on DITA. DITAworks is an RCP application allowing you to connect with Subversion and other software. In the image below you can see that the environment and use of DITAworks resembles much to Eclipse RCP.

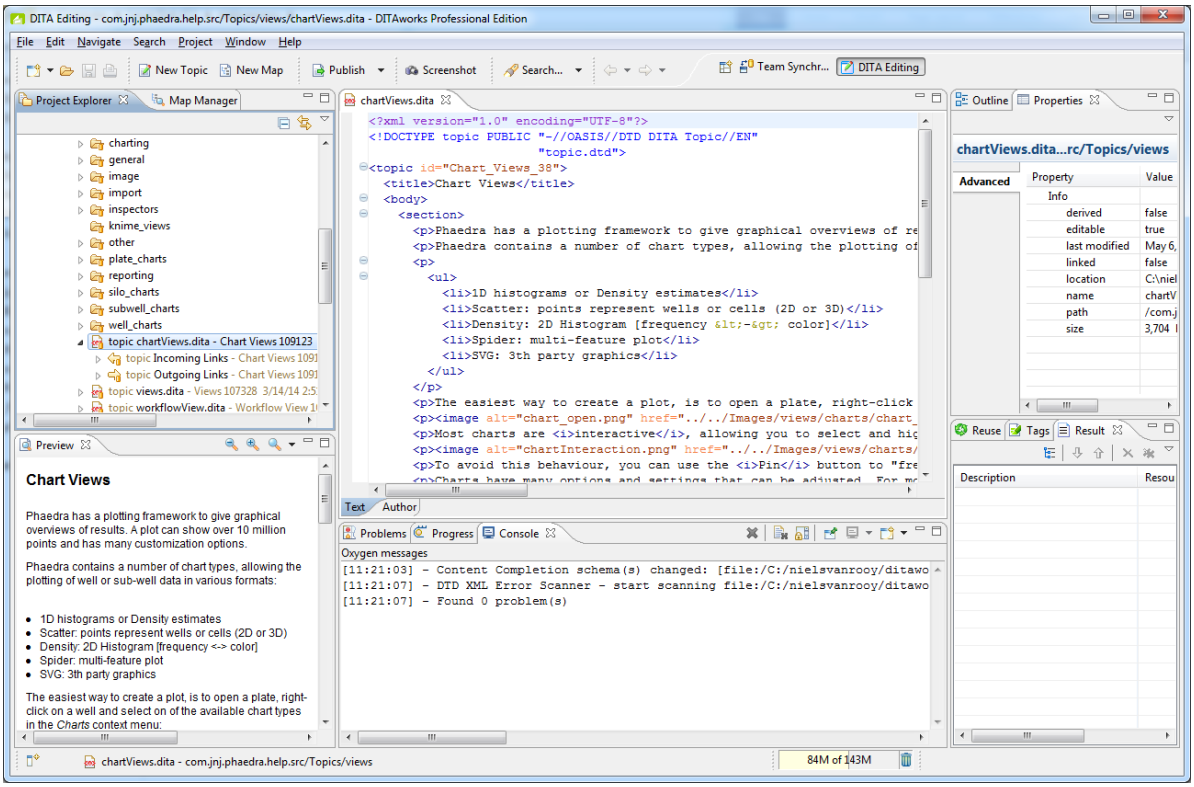


Figure 7: DITAworks Environment

DITAworks allows the use of the DITA Open Toolkit (OT). It is used as a publishing tool to convert DITA content into various output formats. Some of these output formats are:

- PDF
- XHHTML
- HTML Help
- Java Help
- Rich Text Format
- Eclipse Help

The combination of DITA and DITA OT within DITAworks makes it a powerful tool to develop documentation. A general documentation can be developed and published to multiple formats.

The major strengths of DITAworks applied to Phaedra are that we can easily create multiple formats of documentation (pdf, html and Eclipse help) from a single DITAworks project. This means that everything only had to be written once. DITAworks also enables Context Sensitive Help and Live Actions.

DITAworks was used to design and develop the different manuals and help files.

3.1.3 Microsoft Visual Studio

Figure 8: Microsoft Visual Studio 2010 Logo

Microsoft Visual Studio is an IDE developed by Microsoft which uses Microsoft software development platforms. It is mostly used to develop programs for a Windows environment. The applications developed with Visual Studio target the desktop, the web, devices and the cloud. A standard Visual Studio installation supports languages the following languages:

- .NET languages
- HTML/JavaScript
- C++

Other languages can be installed through several plug-ins.

Visual Studio was first released in 1997. It was Microsoft's first attempt to combine different program languages to a single environment. Through the years Visual Studio has evolved to one of the most commonly used IDEs available with its wide support for the .NET framework.

Visual Studio was used to modify, compile and test a possible HDF5 indexing and searching solution.

The image below is a screenshot of the Visual Studio IDE.

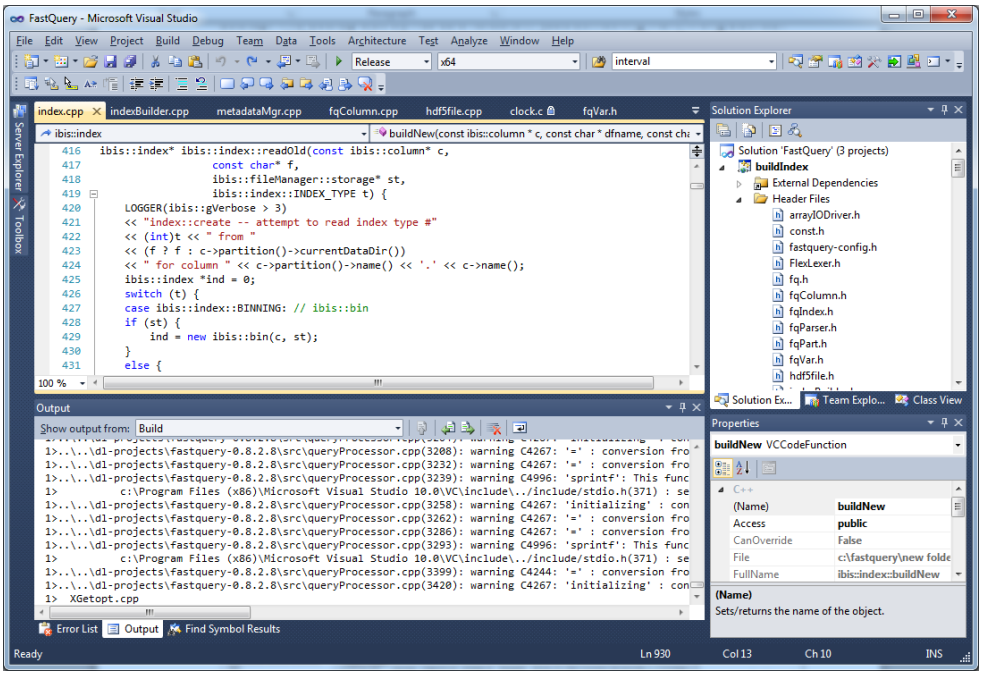


Figure 9: Microsoft Visual Studio Environment

3.1.4 HDFView

Figure 10: The HDF Group Logo

HDFView is a visual tool for browsing and editing HDF4 and HDF5 files developed by The HDF Group. It is a simple program used to:

- view a file hierarchy in a tree structure;
- create, add or delete groups and datasets;
- view and modify the content of a dataset;
- to add, delete or modify attributes.

HDFView was used to open and to examine the structure of HDF5 files.

In the image below you will find an example of an HDF5 file used by Phaedra opened in HDFView.

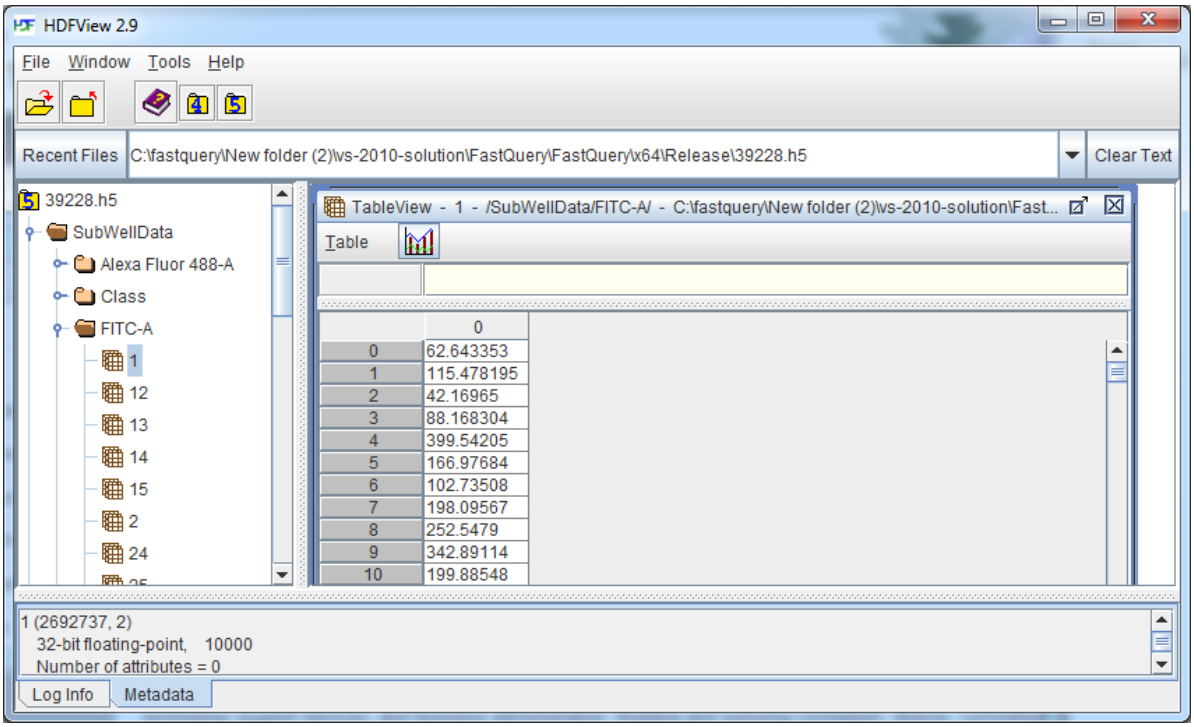


Figure 11: HDF5 file opened in HDFView

3.1.5 Cygwin

Figure 12: Cygwin Logo

Cygwin is a command-line interface which provides a set of powerful tools to migrate applications from a Unix/Linux environment to a Microsoft Windows platform. Cygwin allows you to integrate applications and data on a Windows environment with a Unix/Linux-like environment meaning that it is possible to launch Windows programs through the Cygwin environment.

Cygwin was released by Cygnus Solutions in 1995. Later on, Red Hat acquired Cygwin. It is free and open source software.

A major advantage of Cygwin is that it comes with a gcc compiler. A gcc compiler allows you to compile source code into working applications. Gcc allows you to compile your Unix/Linux programs through the Windows Environment. The version of gcc that comes with Cygwin has various extensions allowing you to compile programs into Windows DLLs. This makes Cygwin the most popular program for porting pieces of software to the Windows platform.

Cygwin was used to compile a possible HDF5 indexing and searching solution.

The image below shows you the Cygwin interface compiling a Unix-based program.

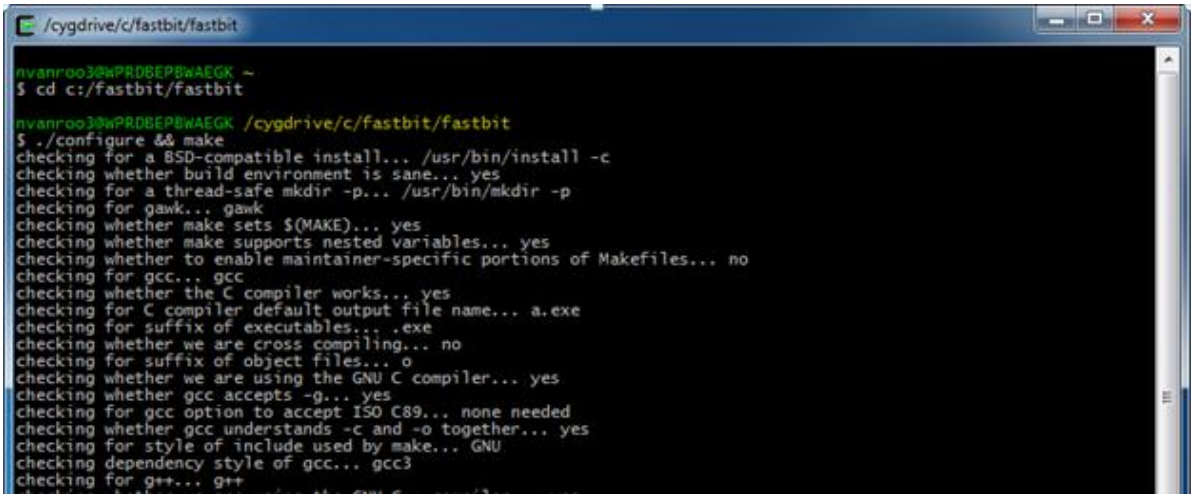


Figure 13: Cygwin Interface

3.2 Programming Languages

3.2.1 Java

Figure 14: Java Programming Language Logo

Java is an high-level object oriented programming language currently maintained by the Oracle Corporation. It is developed following the "write once, run anywhere" principle. This means that Java is a platform-independent programming language. Java applications can run on any Java Virtual Machine (JVM) regardless of the computer architecture. This makes Java one of the most popular programming languages available.

Java was used to enable Context-Sensitive Help in the Phaedra help.

3.2.2 C++

C++ is a programming language that came forth out of the C language. The Syntax of C++ is almost identical to the C language, but is has object-oriented features. Because of the power and flexibility of the language, a lot of programs are written in C++. C++ is a compiled language allowing it to run on almost every available platform. Many other programming languages today have been influenced by C++, such as C# and Java.

C++ was a language used by one of the possible HDF5 indexing and searching solutions.

3.2.3 Python

Figure 15: Python Logo

Python is a high-level object-oriented programming language maintained by the Python Software Foundation. Python provides constructs intended to enable clear programs. Python can be used as a scripting language or, using third-party plug-ins, as a standalone executable program across multiple platforms. Python is free and open source and it has a community-based development model.

Python was a language used by one of the possible HDF5 indexing and searching solutions.

4 PHAEDRA CONCEPTS

This chapter explains some of the Phaedra concepts used throughout my internship. It will give you a basic explanation to better understand the following chapters.

4.1 Phaedra

Phaedra stands for Protocol-based High-content Analysis, Evaluation & Data Reduction and Approval. It is a High Content Screening application developed in Java by Janssen Pharmaceuticals. A High Content Screening application analyzes the results of different experiments. Phaedra uses the large amount of data stored in an Oracle Database and in HDF5 files located on a fileserver. By using Phaedra's High Content Screening abilities, this data can then be edited and shown in various ways like tables, images and charts.

The structure of Phaedra allows users to easily search through and analyze data. Researchers can use Phaedra to store their experiment data and perform complex data analysis. Phaedra allows you to create reports based on your findings by using the tables, images and charts created by the data analysis tools. This makes Phaedra a state of the art High Content Screening tool to help researchers perform data analysis.

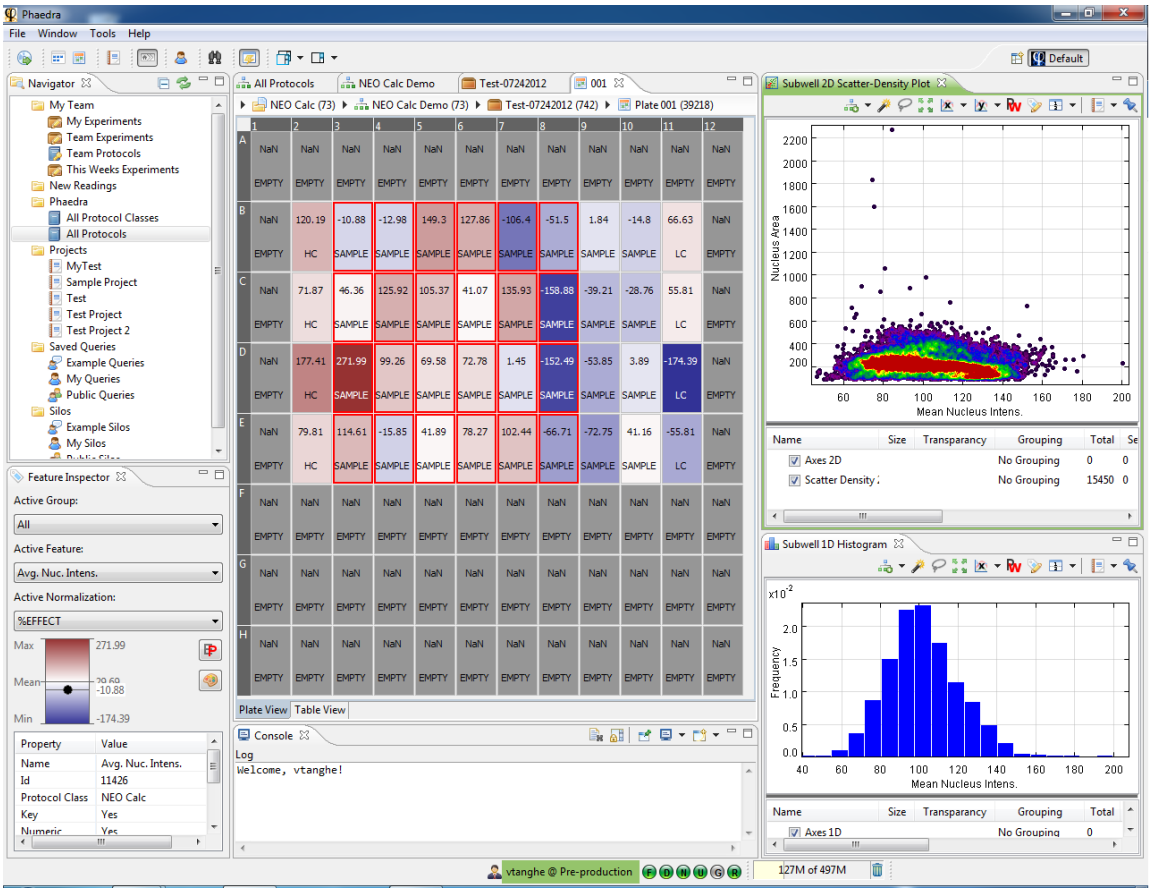


Figure 16: Phaedra Overview

The following chapter contains more information about the structure of Phaedra.

4.2 Main Entity Definitions

To understand Phaedra you need understand the entire structure Phaedra uses to store its data. The image below shows you the drill down of all entities which are explained in this chapter.

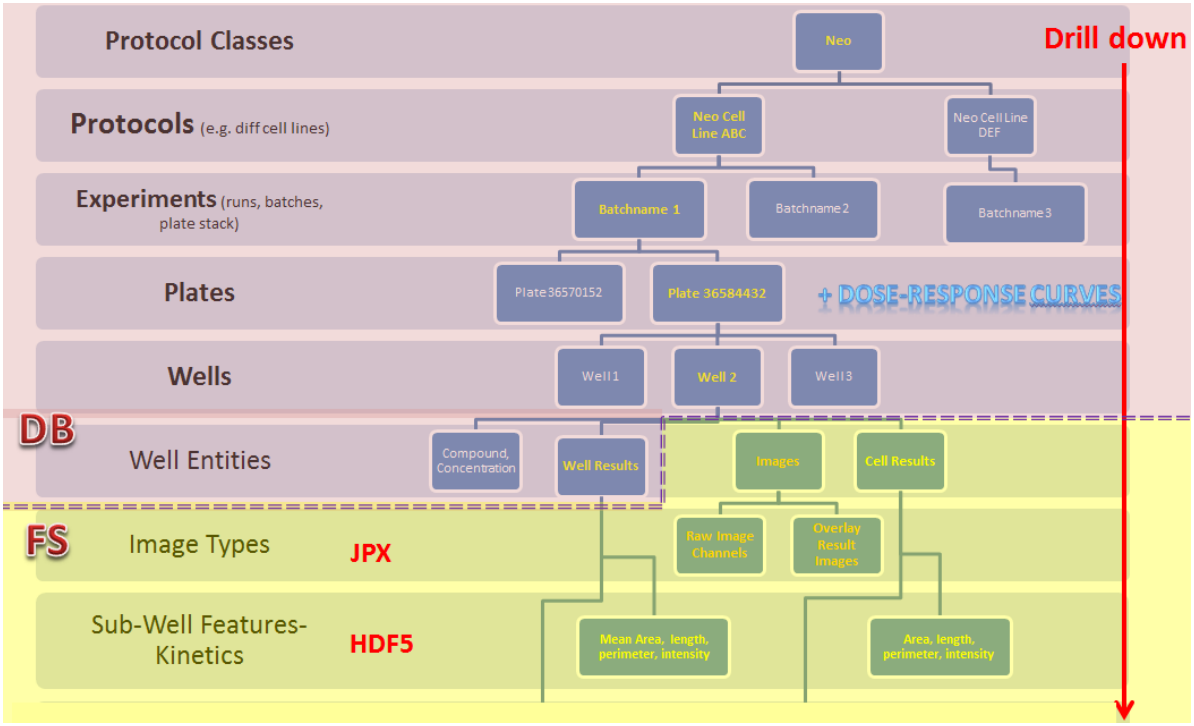


Figure 17: Main Entities Drilldown

1. Protocol Class

A "protocol class" is a blueprint for protocols, defining:

- The features and calculation methods used
- The curve fitting models used
- The instrument type and file formats used

New protocol classes are created by a power user or administrator. A protocol class can be modified depending on its status:

- In Development: users, managers and admins can modify
- Unlocked: managers and admins can modify
- Locked: no-one can modify. Admins can unlock.

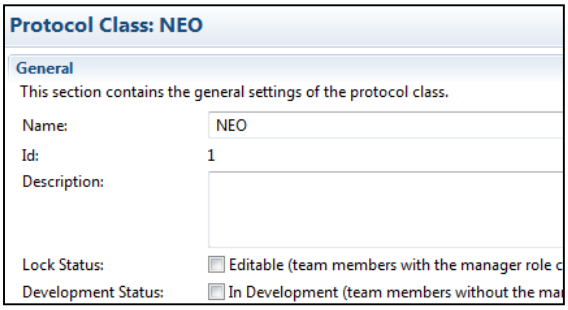


Figure 18: a Protocol Class in Phaedra

2. Protocols

A "protocol" represents a screening method, including:

- A cellular model, biomarkers, incubation conditions (not stored in Phaedra yet)
- The feature calculation and normalization methods used
- The curve fitting models used
- Expected file formats for images, signals, results

A protocol is assigned to a team. Only members of that team have access to the protocol and its contents. Most of the protocol settings described above are defined in a "Protocol Class". A protocol is created from a protocol class.

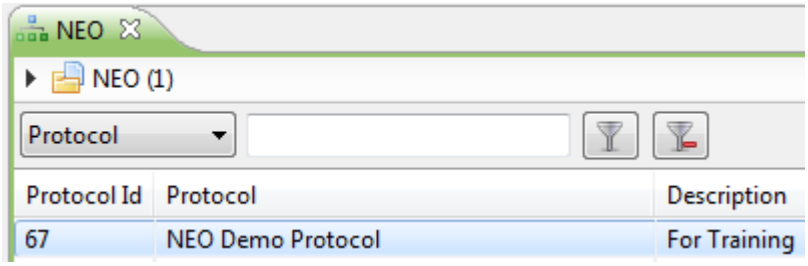


Figure 19: a Protocol in Phaedra

3. Experiments

An "experiment" represents a set of plates in Phaedra. Usually, one experiment corresponds to one batch, run or experiment in the lab.

An experiment may:

- Contain plates of different formats (8x12, 16x24, OR user-defined, e.g. 4x4)
- Contain the same plate twice (e.g. the same barcode, layout and reading linked to two plates), although this is confusing and not recommended.

An experiment cannot:

- Contain plates from different Protocols

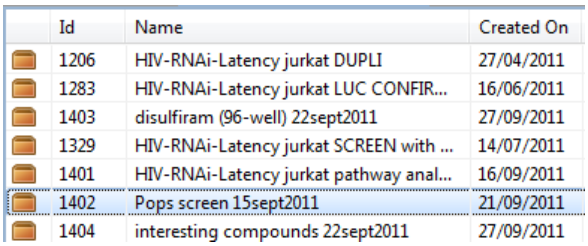


Figure 20: an Experiment in Phaedra

4. Plates

A "plate" object in Phaedra represents a combination of two things:

- A plate layout ('plate definition'), imported from a plate management system or created from a template or from scratch;
- A plate readout ('measurements'), obtained from a file produced by an instrument from a microtiter plate.

A plate does not have to exist physically, it can be microscope slides as well.

Layout and readout information can be imported separately, or simultaneously. Without layout information, normalization and curve fitting is not possible. The layout can be created from scratch in Phaedra (using a Layout Template) or imported from an external plate management system. The layout tells Phaedra where the controls are located, and which compounds are located where, in what concentration.

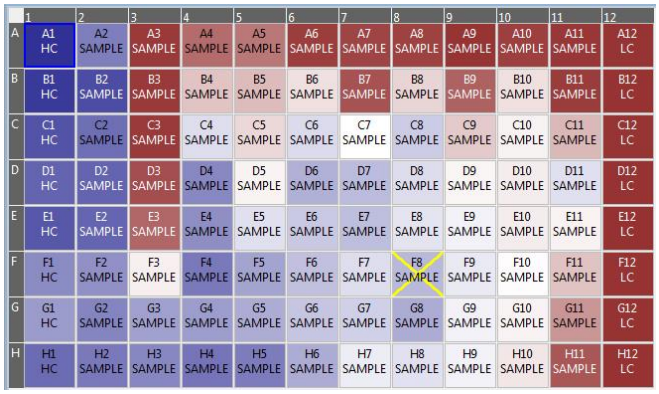


Figure 21: a Plate in Phaedra

5. Wells

A "well" represents one well in a plate. A well has a type and a status. The type can be used as a control or have one compound and one concentration allocated to it. The status is used to mark if a well is valid or is rejected. A well can have multiple values (e.g. 800), each value represents a well feature. A well can also represent a microscope slide or test tube.

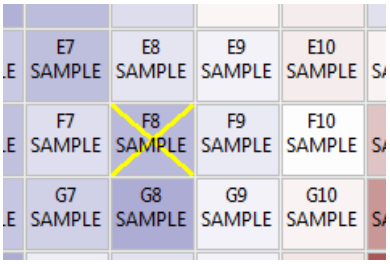


Figure 22: Multiple Wells in a Plate

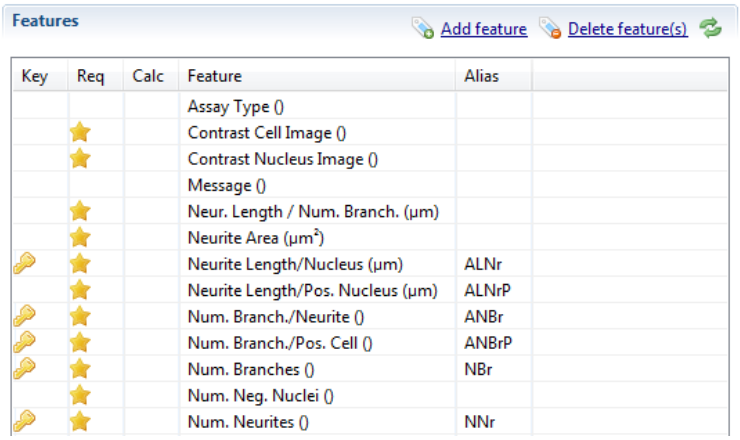
6. Features and Subwell entities

A "feature" represents one measurement, or parameter, or property of a reading. There are two types of features: well features and subwell features. Well features provide one value per well, e.g. "Average Nucleus Area". There can be a large number of well features for each well. subwell features provide one value per entity inside the well, e.g. "Nucleus Area" per cell, or "Signal Intensity" per time point.

Feature values can be numeric (floating-point) or text (string). Features may also originate from different readings or instruments. The well feature values can be raw and normalized.

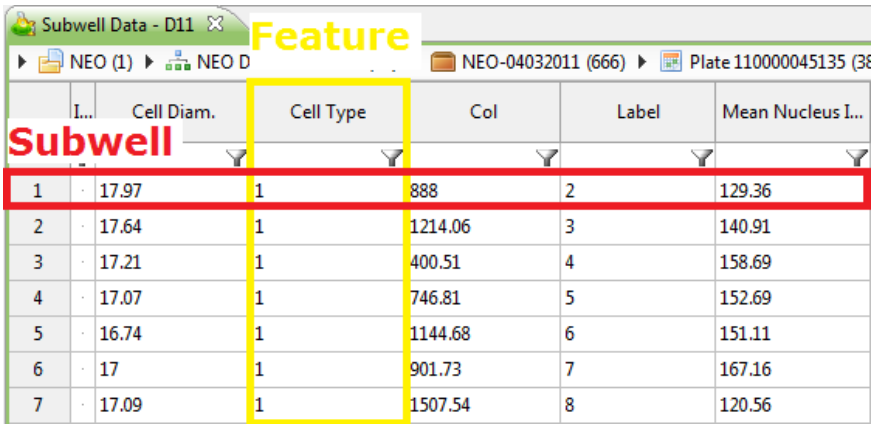
- Raw: as provided by the instrument or analysis
- Normalized: e.g. a percentage relative to Low and High control wells

The data for the subwell features isn't stored in the Phaedra database. It is stored in HDF5 files. Each HDF5 file represents 1 well.



Key	Req	Calc	Feature	Alias
			Assay Type ()	
	★		Contrast Cell Image ()	
	★		Contrast Nucleus Image ()	
			Message ()	
	★		Neur. Length / Num. Branch. (µm)	
	★		Neurite Area (µm²)	
🔑	★		Neurite Length/Nucleus (µm)	ALNr
	★		Neurite Length/Pos. Nucleus (µm)	ALNrP
🔑	★		Num. Branch./Neurite ()	ANBr
🔑	★		Num. Branch./Pos. Cell ()	ANBrP
🔑	★		Num. Branches ()	NBr
	★		Num. Neg. Nuclei ()	
🔑	★		Num. Neurites ()	NNr

Figure 23: Features in Phaedra



	I...	Cell Diam.	Cell Type	Col	Label	Mean Nucleus I...
1		17.97	1	888	2	129.36
2		17.64	1	1214.06	3	140.91
3		17.21	1	400.51	4	158.69
4		17.07	1	746.81	5	152.69
5		16.74	1	1144.68	6	151.11
6		17	1	901.73	7	167.16
7		17.09	1	1507.54	8	120.56

Figure 24: Visual Representation of a Feature and a Subwell

Note that from this point forward the word "subwell value" means subwell features.

Each well contains a number of subwell entities, usually these are cells. Each subwell entity has one value (numeric or string) per subwell feature. So when a well with 1000 cells has 10 subwell features, it has a total of 10000 subwell values. In reality, Phaedra can contain billions of subwell data values. These values are all stored in HDF5 files each representing one plate. This causes serious problems when executing queries. Not only are they not efficient but it also becomes impossible to get a good performance when cross file queries are necessary.

7. Object hierarchy example

The image below represents an example of the object hierarchy as it is implemented in Phaedra. It uses an existing protocol class.

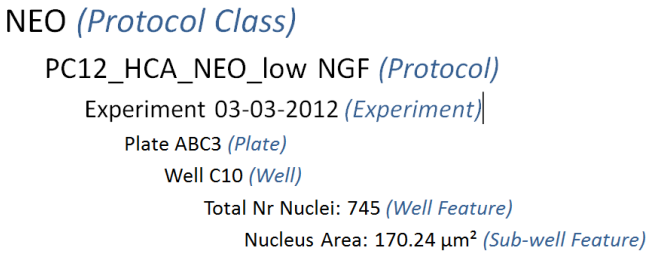


Figure 25: Object Hierarchy Example from Phaedra

5 PHAEDRA DOCUMENTATION

This chapter will cover the development of the Phaedra documentation. It will cover how the documentation was developed and how it was implemented in Phaedra.

5.1 Documentation development

The existing documentation of Phaedra was outdated. A new documentation is developed that covers all the new aspects of Phaedra and reshaped the old documentation. To develop a user-friendly and interactive documentation, the DITA technology was used. The new documentation is developed in DITAworks, a powerful tool that combines the DITA standard and the DITA OT to produce documentation.

The development of documentation in DITA is done by creating topics. A topic represents a simple help file. Topics are built-up from XML code. This XML shows a lot of resemblance with basic HTML code.

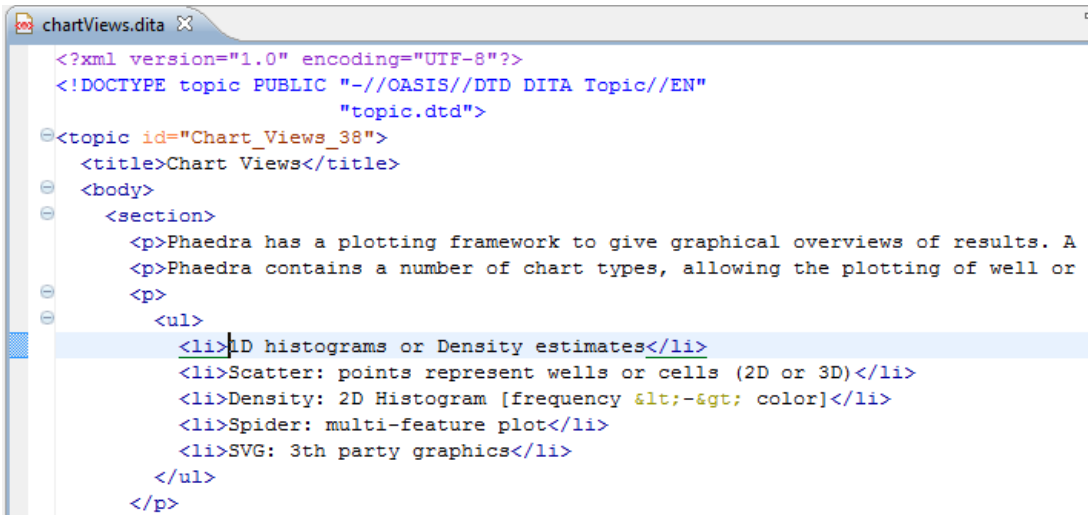


Figure 26: a Topic in XML view

These topics are mapped to generate a complete documentation. In the image below you will see a piece of the map of the Phaedra documentation. Each box represents one topic. Different levels are generated to create a logical structure of topics.

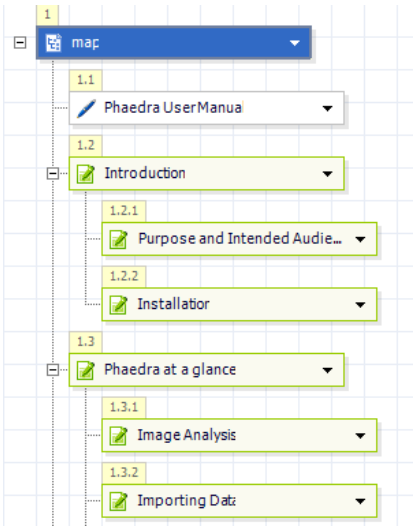


Figure 27: a DITA map

The power of DITAworks lies in its publishing possibilities. DITAworks allows you to create multiple documentation formats based on your topics. This means that the entire documentation only has to be written once. For Phaedra, multiple publishing configurations were created to be able to publish the information in different formats (PDF, Eclipse Help, Html,…) and to be able to filter information that for example only needs to be displayed in the PDF version. In the image below you will see the publishing configurations used to create the Phaedra documentation. Note that you can specify a publishing profile to exclude or include content specific to the needs of the output format.

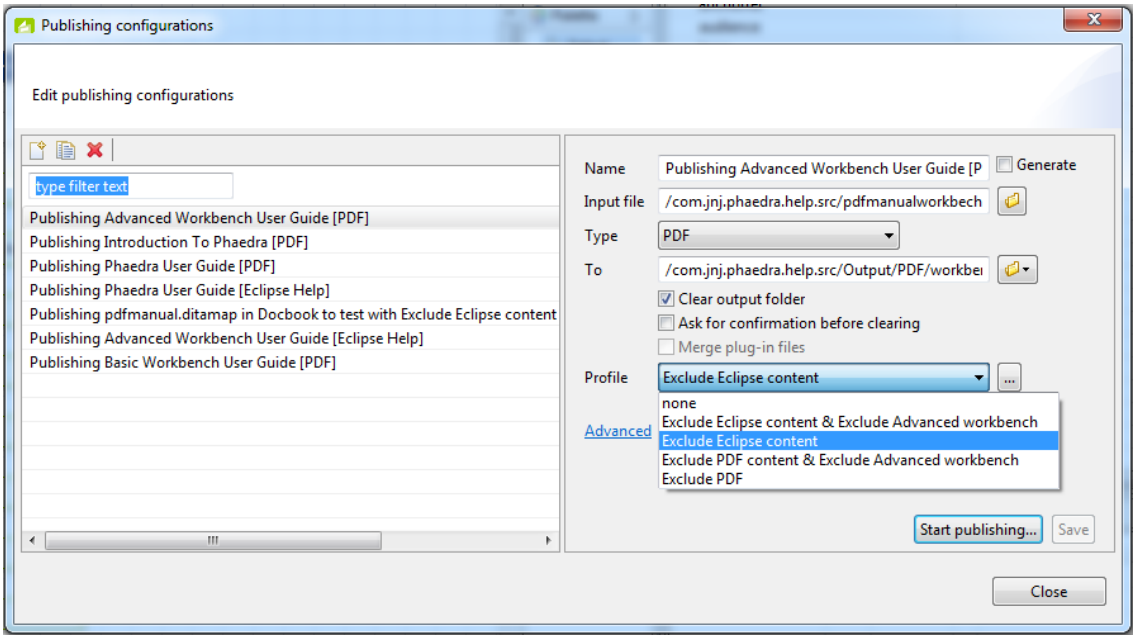


Figure 28: DITAworks Publishing Configurations

DITAworks enables content reusability and content filtering. This was used to avoid the duplication of code and to customize the different output formats. This was achieved by adding parameters to the existing XML topics. The image below shows a topic where content reusability and content filtering is applied.

chartViews.dita | eclipsetoc.ditamap | importingData.dita

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN"
                       "topic.dtd">
<topic id="Importing_Data_151">
  <title>Importing Data</title>
  <body>
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Constrain
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Requiremer
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Using_the_
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Step_1:_Se
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Step_2:_Se
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Step_3:_Se
    <section
      conref="../howto/data_import/dataimport_importDataIntoPhaedra.dita#Import_data_into_Phaedra_50/Step_4:_Se
    <p><b>More Information about Importing Data can be found in <xref href="../howto/dataimport.dita"
        >Using Phaedra (in-depth) > Importing Data</xref></b></p>
    <section product="Eclipse">
      <table scale="70">
        <tgroup cols="2">
          <tbody>
            <row>
              <entry><xref href="imageAnalysis.dita"> Image Analysis</xref> &lt;&lt; Previous Topic </entry>
              <entry align="right"> Following Topic &gt;&gt;
                <xref href="navigation.dita"> Navigation</xref></entry>
            </row>
```

Content reusability

Content filtering

Figure 29: Content Reusability and Filtering

The DITA OT contains several different templates to style your output formats. The Phaedra documentation required a specific styling. The PDF format styles were adjusted by creating a style sheet that overwrites specific settings set in the DITA OT templates. This resulted in the creation of a custom front-page, header and footer text, custom colors and font-sizes… The image below is an example of a PDF style sheet that overwrites the DITA OT settings.

chartViews.dita | eclipsetoc.ditamap | importingData.dita | pdf2.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:import href="../ditaot/customization/dita2fo-shell.xsl"/>

    <xsl:output method="xml" encoding="utf-8" indent="yes"/>

    <!-- ******************************** -->
    <!-- Below are the template overrides -->
    <!-- ******************************** -->

    <!-- Add an image to the front page -->
    <xsl:template name="define-cover-page">
        <fo:region-body background-image="../../../Images/cover2.png" background-position-horizontal="center"
            background-repeat="no-repeat" margin-top="25pt" margin-bottom="0pt"/>
    </xsl:template>

    <!-- Add some text to the front page -->
    <xsl:template name="generated-front-covers">
        <fo:page-sequence master-reference="cover">
            <fo:flow flow-name="xsl-region-body">
                <fo:block font-family="Arial" margin="186mm 15mm 0 69mm">
                    <fo:block font-size="30pt" font-weight="bold">
                        <xsl:value-of select="//*[contains(@class,' map/map ')]/*[contains(@class,' topic/title ')]"/>
                    </fo:block>
                    <fo:block margin-top="80mm">
                        <fo:block font-size="10pt" line-height="normal" color="#c8c8c8">
                            <fo:block>Copyright © 2014 – J&amp;J R&amp;D-IT Department</fo:block>
                        </fo:block>
                    </fo:block>
                </fo:block>
            </fo:flow>
        </fo:page-sequence>
```

Figure 30: PDF Style Sheet

5.2 Live Actions

DITAworks supports the use of live actions. Live actions are lines of code that can be executed from your help file. This enables calling views, editors and menu-items automatically. Live Actions assist the end-user therefore creating a user-friendly manual.

The implementation of live actions is done with JavaScript code. You can simply add your executable code to a link so that users only need to click on a piece of text to execute the live action. The image below is an example of a live action written in a topic.

```
<executeCommand
  ec-command="org.eclipse.ui.window.preferences(preferencePageId=org.eclipse.ui.preferencePages.Perspectives)"
  ><image href="../../../../Images/icons/command_link.png"/><b>General > Perspectives</b> </executeCommand>prefere
```

Figure 31: Live Action Code Example

The code in the image above will open the perspective preferences when clicking on the link in the Phaedra Help:

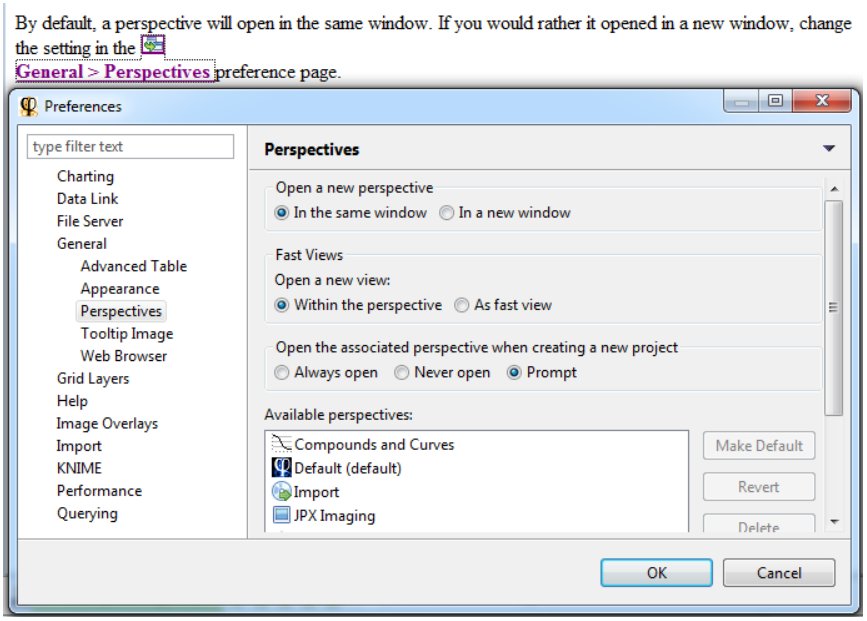


Figure 32: Live Action Result

5.3 Context-Sensitive Help

DITAworks and Eclipse allow enabling Context-Sensitive Help (CSH) for a custom made Eclipse Help. CSH enables to open a specific help file based on the selected view or editor in your application. This ensures that a user can find the specific help file fast and accurate thus increasing the user-friendliness.

CSH uses a specific context file to link the selected view or editor to the corresponding help topic. The image below shows a context file in DITAworks.

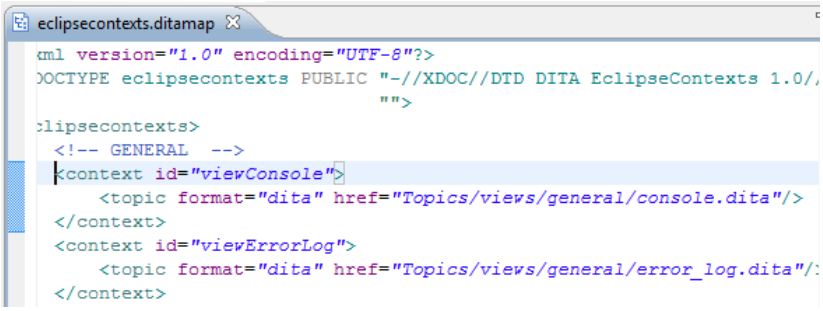


Figure 33: DITAworks Contexts File

When opening the help in Phaedra, Phaedra will determine which help should be opened based on the view or editor that is focused. To enable this in Phaedra, every view or editor needs to be linked to the corresponding context. To achieve this, all views and editors were linked to the corresponding context. The image below enabled the "Navigator View" in Phaedra to open its corresponding help topic.

```
// Link specific help view based on the Context ID
PlatformUI.getWorkbench().getHelpSystem().setHelp(parent, "org.eclipse.datatools.connectivity.ui.viewNavigator");
```

Figure 34: Phaedra CSH Source Code Example

5.4 Implementation

To implement the different Eclipse helps in Phaedra, three new plug-in projects were created.

1. com.jnj.phaedra.help
2. com.jnj.phaedra.help.knime
3. com.jnj.phaedra.help.workbench

These plug-in projects were adjusted to serve as Eclipse help projects. The following adjustments were made:

- The manifest was adjusted to serve as an Eclipse help project manifest.
- The build properties were adjusted to ensure that all the correct files are loaded.
- The plugin.xml file was adjusted to enable CSH.

The specific implementation settings for the above mentioned files can be found in the appendix "2 Documentation Implementation Settings".

5.5 Result

The entire DITAworks project was successfully deployed to a Phaedra built-in help and a PDF format. The following manuals were created:

- Phaedra User Guide [Eclipse Help]
- Phaedra Advanced Workbench User Guide [Eclipse Help]
- Phaedra Knime User Guide [Eclipse Help]
- Phaedra User Guide [PDF]
- Phaedra Basic Workbench User Guide [PDF]
- Phaedra Advanced Workbench User Guide [PDF]
- Introduction to Phaedra [PDF]

The generated Eclipse help guides can also be viewed from a web browser.

6 HDF5 INDEXING AND SEARCHING

This chapter will describe the research for HDF5 indexing and searching solutions. It will give a better understanding of the possible solutions. First you will get some more information about the existing problem and the following chapters cover the research that has been done.

6.1 Limitations with the current HDF5 searching possibilities

As explained before, Phaedra requires efficient querying functionalities to increase the user experience. The current querying functionality however doesn't support subwell data querying because of limitations in the Phaedra HDF5 data model.

HDF5 files are typically built using the compound-data structure logic. This means that a dataset contains a multidimensional array of values. In Phaedra, this means that every plate contains datasets of wells with in these datasets a multidimensional array of values. The multidimensional array contains columns with features and rows with cellular values.

This type of structuring has its limitations. Phaedra requires to dynamically add features to its plates. To add a new column in a HDF5 file, the entire dataset has to be rewritten. Due to the possibility of huge datasets, this isn't the best solution performance wise.

Phaedra uses a vector structure logic to manage its data. This means that the HDF5 file uses groups to represent features. Every feature contains multiple datasets representing each well. In this structure a dataset is a one-dimensional array containing only the cellular values.

The challenge with this type of data-structure is to find an efficient way to index and search this data. Different solutions need to be tested and compared to each other to find the best solution for Phaedra.

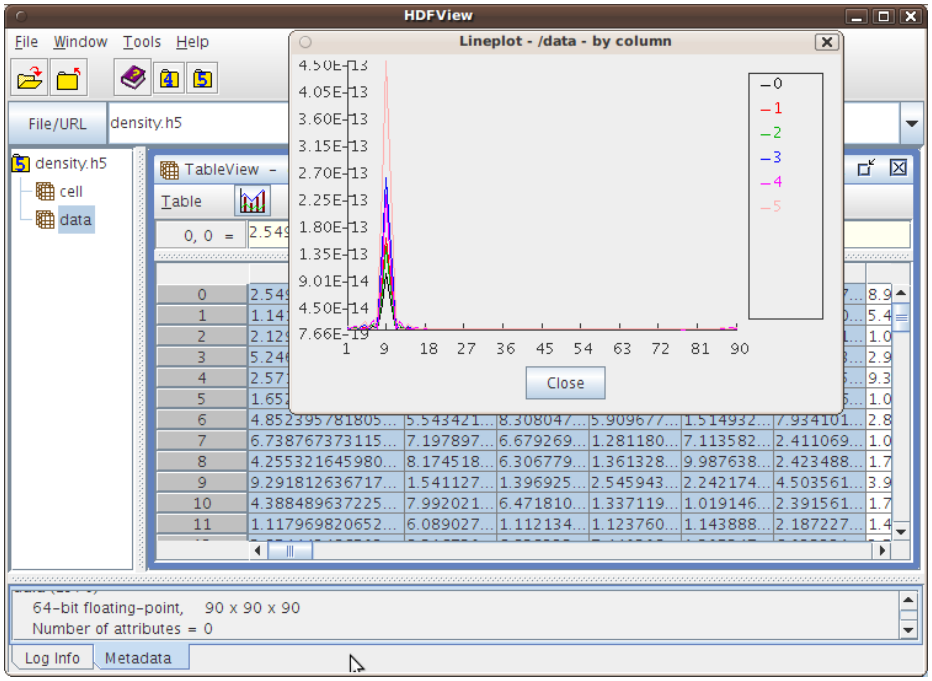


Figure 35: A HDF5 file example

More specific, the following limitations occur in the vector based structure currently implemented:

- **Problem 1: Subwell data is not searchable**

  Since the subwell data is organized in a file per plate, a search across plates would potentially need to open thousands of HDF5 files, and read their contents.

  A search like this would be that slow and inefficient so that it becomes impossible to use.

- **Problem 2: Updating subwell data is slow**

  Since the jhdf5 library has no direct write access to the file server, write operations go through a temporary local copy.

  This additional download followed by the upload has a significant impact on the performance of a subwell modification transaction (for example, a subwell classification), especially if the modification is small and the file is large.

In order to resolve these two problems and create a good searching solution, the following requirements had to be met:

1. Performance
   - Querying data needs to be very fast;
   - Adding data (and thus updating the index) needs to be reasonably fast.
2. Robustness
   - Index updating needs to be centralized, or managed in such a way that the "global" index is never stale or corrupt.
3. Timeframe
   - The solution must be implemented in a reasonable timeframe.

Since Phaedra uses JCIFS for write access on the file server, and JCIFS uses streams, the HDF5 library cannot write directly to the file server. This means that every possible solution must be able to work with locally stored HDF5 files. To accomplish this, the following workaround must be used:

- Download the HDF5 file to a temporary location on the client
- Make the desired modifications to the file.
- Upload the modified file, replacing the original file.

Phaedra uses the library *ch.systemsx.jhdf5* to access the HDF5 files. This library is a Java wrapper around the HDF5 library from the HDF-Group. This library requires random access, so it will only work on file objects, not on streams.

6.2 Lucene Core

This chapter describes the indexing and searching possibilities of a Lucene implementation in Phaedra.

6.2.1 About Lucene

Lucene Core is an indexing and querying framework for Java. It is centralized, highly optimized, and supports both text and numeric data (among other data types). Lucene offers the following features:

Scalable, High-Performance Indexing

- over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

Powerful, Accurate and Efficient Search Algorithms

- ranked searching -- best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping
- fast, memory-efficient and typo-tolerant suggestions
- pluggable ranking models, including the Vector Space Model and Okapi BM25
- configurable storage engine (codecs)

(Apache Lucene Core, 2012)

Lucene is based on documents instead of HDF5 files. Lucene indices are generated by an Analyzer Class which will eliminate unnecessary text and translate documents into the following fields:

- Keyword
- UnIndexed
- UnStored
- Text

Keyword fields are stored without analysis. Keywords are used for fields whose value will not change.

UnIndexed fields are neither analyzed nor indexed, but are stored directly in the index. The values are displayed with indexed search results but are never searched directly. This type isn't suitable to store large values because it stores information without analysis.

UnStored fields are the opposite of UnIndexed fields.

Text fields are analyzed and indexed. (If the data indexed is a String, it's stored. If the data is a Reader, it isn't stored).

Documents can be efficiently searched against after their indexation. The searching is handled by a Searching Class.

These fields will make your index file. The index file is centralized meaning you can easily search through multiple plates at once.

(Pande, 2010)

The generated index contains statistics about features to make features-based search more efficient. Lucene uses inverted indexing meaning that indexing is based on features and not on documents. This allows Lucene to easily search for features and return the documents that contain it.

(Core, Index File Formats, 2013)

Lucene allows incremental indexing meaning that when values are updated, only this value is indexed again. This means that indexing can be done fast when updating values, adding features...

6.2.2 Lucene Implementation Analysis

The implementation of Lucene in Phaedra is a complex process. There are different bottlenecks, possibilities and best practices for the Lucene implementation. The implementation of the Lucene core is fairly easy. Lucene Core can be downloaded from the website. Lucene core can be used by including the necessary jar-files.

Lucene works with documents, not directly with HDF5 files. This conversion needs to be developed manually. This can be done within the Lucene framework by writing personalized analyzer classes. These classes read-out the HDF5 files and converts them to documents. These documents are used later on to create the index files.

The indexation of the documents is done by the Index classes within Lucene. Phaedra requires a personalized solution for indexing the documents. The Phaedra specific indexation class must ensure that all needed values are indexed properly. The performance of this class must be outstanding. The class will need to process billions of records.

Changes to documents need to be processed efficiently. Lucene uses incremental indexing to perform the update, insert or delete of new values. This means that the index file is edited instead of recreated. This will increase the performance of the indexation process.

The strategy to convert The HDF5 files to documents needs to be one of the following:

1. Well documenting: All the HDF5 files are converted to documents on well level. This means that one document represents one well in Phaedra. This document will contain features with an array of (subwell) values.

   Advantages:

   - Less documents needed (100 million)
   - Slightly better search efficiency

   Disadvantages:

   - Query results can only be wells. The search is performed through an array of results which will find a match inside the array. Due to Lucene limitations, the index of the array hit can't be returned. This means that there is no way of telling which subwell value matched the search requirements.

2. Subwell documenting: All the HDF5 files are converted to documents on subwell level. This means that one document represents one subwell in Phaedra. This document will contain features with one specific value.

   Advantages:

   - Query results can be wells and subwells. We can include the well id in the subwell document. The returned result will still only be a subwell but the Well id will be known. This enables us to perform subwell and well searching.

   Disadvantages:

   - Too many documents (100 billion) for Lucene.

(Core, Limitations) (Rahul, 2013)

6.2.3 Lucene in Phaedra

Lucene is a well-known and very efficient indexing framework used by many large applications. Its indexing functionality has been proven to work (Wiki, 2013) with large amounts and sizes of documents.

Lucene uses Inverted indices and Incremental indexing. This is perfect for Phaedra to search and edit large amounts of data efficiently. (Core, Index File Formats, 2013)

The index files can be centralized on a server which handles automatic indexing of the HDF5 files.

Lucene's querying functionalities can be used for Phaedra. To perform Phaedra specific queries, a custom queryparser must be developed. This queryparser must support the following requirements:

- Search for subwell entities
- Search for well entities
- Search for multiple subwell entities
- Search for multiple well entities
- Search against numeric values
- Search against string values
- Support the use of BETWEEN, AND, GREATER THAN, LESSER THAN…

Lucene supports all the requirements mentioned above.

Lucene uses a Java integer to hold document IDs. This means that the maximum number of unique documents in a single index segment lays around 2.1 billion. The maximum amount of unique features in an index segment is calculated by multiplying the maximum amount of documents and the index interval. In Lucene, the default index interval is 128 so there can be 274 billion unique features. These numbers aren't a limitation of the index file format but of the current Lucene implementation.

To reach the best search options possible, subwell documenting is required.s documenting must be able to generate 100 billion unique documents. This isn't supported in the standard Lucene implementation. To counter this problem we can either split the index file in multiple segments of 2 billion unique documents or we can modify the Lucene implementation to support more documents and features.

(Core, Limitations)

The generated index file in standard Lucene cases is 20-30% the size of the used documents. This is because Lucene filters out the irrelevant information for indexing. In Phaedra's case, the HDF5 files contain much more relevant information because of its database-structure. In reality this means that the indexing will have much less information to ignore and the index file size vs. number of documents ratio will increase a lot.

To counter the scalability limitations for subwell documenting, we can perform well documenting with term vector-indexing. A standard Lucene search will return the document name of the search hits. With vector-indexing we enable a second search function which will return the subwell ID of the searched values. This enables us to perform well and subwell searching.

(Grand, 2013)

The other solution for the subwell documenting limitations is to modify the Lucene framework to not use the Java *int* for documenting IDs. Instead another numeric Java variable can be used which supports more values. The real question here is, will Lucene handle indexing and searching 100 billion of documents in a proper way? If this solution is considered, extensive testing is required.

One huge index file is not a good solution for Phaedra. Lucene provides functionalities to create different index segment files. This is necessary to perform parallel search threads. Parallel search threads are a huge advantage for your search performance. Especially in Phaedra which has millions of well documents.

Lucene supports huge document amount searching and indexing. In reality the performance declines when searching and indexing millions of files. To counter this, there are a lot of solutions to modify Lucene and increase its performance.

6.2.4 Advantages vs. Disadvantages of Lucene

Advantages

- Lucene is a known search framework which has received a lot of credits in the past for its performance.
- Completely Open Source solution meaning that everything can be personalized to support Phaedra.
- Index files aren't stored within the HDF5 files.
- Multiple index files are available
- Great parallel searching features
- We don't need to change anything to Phaedra's current source code. Lucene is independent search functionality.

Disadvantages

- Needs a lot of customization before it can run Phaedra well searches.
- Data conversion functionality needs to be written. HDF5 files need to be converted to Lucene Documents to enable indexing. This means that a lot of duplicate information will be stored.
- To enable subwell searching, depending on the chosen solution, a good indexing/searching strategy is required.

6.2.5 Recommendation

Lucene is an excellent searching tool. It's known to provide a great performance on document based searching. The HDF5 file to Lucene documents conversion will be a very time consuming process.

Also to use Lucene for Phaedra, a lot of changes will have to be made to this framework. These changes are core changes which will be very time consuming. In return, this offers huge possibilities to create a custom Phaedra search functionality.

If the time is available to perform a clean, thorough and personalized implementation of Lucene, this framework will be one of the best available search solutions.

6.3 Pytables

This chapter describes the indexing and searching possibilities of a Pytables implementation in Phaedra.

6.3.1 About Pytables

Pytables is a package for managing hierarchical datasets such as HDF5. It is designed to deal with large amounts of data. Pytables uses a compound based search/indexing technique. The goal of Pytables is to enable end users to manipulate data easily in a hierarchical structure.

Specifications of Pytables:

- Built on top of the HDF5 library
- Written in Python
- Uses the NumPy package
- Object-oriented
- Uses C extensions for the performance-critical parts

Main features of Pytables:

- Supports the use of datasets
- Multidimensional arrays
- Column-based indexing support
- Support for numerical arrays
- Enlargeable arrays
- Variable length arrays
- Hierarchical data model
- User defined metadata
- Read/modify generic HDF5 files
- Data compression
- High performance
- Large HDF5 file size support
- Architecture-independent

(braves G. o., 2014) (maintainers, 2014)

The Object Tree of Pytables allows you to divide HDF5 files into Groups and Datasets. This is the data structure that Phaedra uses to classify its HDF5 files. In the picture below you can see the Object Tree of a Phaedra HDF5 file.

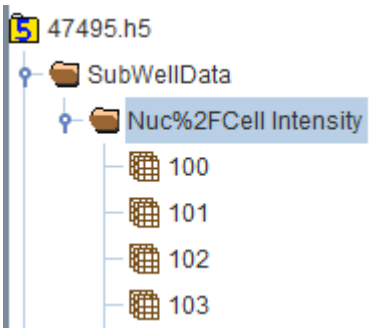


Figure 36: Pytables Object Tree

In Pytables, this data structure would be interpreted as follows:

- 47495.h5 = Object Tree
- SubwellData = Group
- Nuc%2FCell Intensity = Group
- 100, 101, 102, … = Dataset

6.3.2 Pytables Implementation Analysis

Pytables is already implemented in Phaedra because of earlier performed test cases.

6.3.3 Pytables in Phaedra

Pytables uses OPSI (Optimized Partially Sorted Indexes) as its indexing engine. OPSI is a powerful indexing engine to perform really fast queries on arbitrarily large tables. OPSI is a column based indexing technique. In terms of Phaedra this means that it will be less efficient because Phaedra uses scalar datasets instead of compound datasets. Scalar datasets are datasets which do not act as a table but as an array. Compound datasets can be compared to tables. Extensive testing is required to analyze the abilities of OPSI indexing.

Advantages of OPSI:

- Integrated in Pytables
- Fast indexing mechanism
- Greatly improves searching speed
- Sorting large tables by a specific field

Disadvantages of OPSI:

- Created to index compound datasets. Phaedra uses scalar datasets
- Created to index read-only data. It is possible to perform update and delete statements but it will substantially reduce the OPSI indexing performance.

(Balaguer, 2007) (braves g. o., 2011)

6.3.3.1 Searching

Searching Phaedra HDF5 files using Pytables has already been tested. The tests have been performed by my colleague, Phaedra's leading developer Frederick Michielssen, and resulted in the following findings:

- If you are searching a scalar dataset (homogeneous), Pytables reads it as an array
- If you are searching a compound dataset (heterogeneous), Pytables reads it as a table
- Pytables fast searching requires a table meaning that it cannot be used to search the current data structure of Phaedra HDF5 files.

The complete test case and supporting links can be found in Appendix "1 Pytables Test Case".

6.3.3.2 Scalability

Pytables supports the use of large datasets. The default parameters for Pytables are ideally set for files around 10 MB. Although Pytables can easily search through files up to 2 GB and with a million of rows, the Pytables manual suggests further optimization if searching through large HDF5 files is a regular thing. Customizing Pytables parameters to your specific needs will increase the performance significantly.

(Alted, 2010) (braves G. o., 2014)

6.3.3.3 Solutions to the standard shortcomings

The indexing mechanism in Pytables does not appear to be best for our requirements. Extensive testing of this technique is required if Pytables is considered as a possible

solution. Alternatively a new indexing technique needs to be implemented which is more efficient and will work with Pytables' querying functionalities.

The searching limitations are a major shortcoming of Pytables. For Pytables to be a valid solution, the HDF5 data structure needs to be converted to a compound based structure. This requires major changes in the entire Phaedra application which is not recommended.

The scalability of Pytables needs to be customized to the exact Phaedra needs. This requires more research into Pytables' functionalities.

6.3.4 Advantages vs. Disadvantages

Advantages

- Pytables is a known search framework which has received a lot of credits by the Python community as an efficient HDF5 file search engine
- Completely Open Source solution meaning that everything can be personalized to support Phaedra.

Disadvantages

- Pytables uses compound datasets
- Changes to the structure of Phaedra had to be made
- Indexing limitations
- Scalability limitations

6.3.5 Recommendation

Because of the index limitations, searching limitations and the needed scalability effort, Pytables is not worth the effort to implement in Phaedra (unless no other solution is found).

Because the implementation is not worth the effort, Pytables will be considered as a "last-resort" solution. No test case was made for Pytables.

6.4 MongoDB

This chapter describes the indexing and searching possibilities of a MongoDB implementation in Phaedra.

6.4.1 About MongoDB

MongoDB standard edition is a "Big Data" document database. A document database stores documents instead of single values. The data structure is composed of fields with their value pairs. These documents are similar to JSON objects. Fields in MongoDB can include arrays.

Figure 37: MongoDB Document Structure

Advantages of using documents:

- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.

(MongoDB, 2014)

MongoDB has the following key-features:

High Performance
MongoDB provides high performance data persistence. In particular:

- support for embedded data models reduces I/O activity on database system;
- indices support faster queries and can include keys from embedded documents and arrays;

High Availability
To provide high availability, MongoDB's replication facility, called replica sets, provide:

- automatic failover;
- data redundancy;

A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

Automatic Scaling
MongoDB provides horizontal scalability as part of its core functionality.

6.4.2 MongoDB Implementation Analysis

The implementation of MongoDB in Phaedra is a complex process. First of all, the entire structure of reading, storing and accessing cellular data needs to be reviewed. This is necessary because Phaedra will not use HDF5 files anymore but a database instead.

The second problem is the conversion of the HDF5 files to a document-based database. An efficient conversion method needs to be developed to automate this process. Otherwise it will take too long to convert the data.

6.4.3 Advantages vs. Disadvantages

Advantages

- MongoDB is a known big data database which has received a lot of credits in the past for its performance. It is used by several large organizations.
- Open Source
- Great parallel searching features
- Great scalability
- Fast search engine

Disadvantages

- Major changes to the Phaedra structure and source code are required
- Completely different solution to store data.

6.4.4 Recommendation

MongoDB is a database solution. This means that the entire structure of importing, accessing and editing cellular data in Phaedra needs to be changed. This is the most time consuming effort available. Because the implementation is not worth the effort, MongoDB will also be considered as a "last-resort" solution. No test case was made for MongoDB.

6.5 FastQuery

This chapter describes the indexing and searching possibilities of a FastQuery implementation in Phaedra.

6.5.1 About FastQuery

HDF5-FastQuery is a searching solution for HDF5 files currently in development. It is being developed by the Visualization Group. The Visualization Group was created in 1990 to explore scientific programs and develop new software. Because of the limitations of HDF5 searching and indexing techniques, the Visualization Group developed its own solution. FastQuery is not yet available as a public release but we managed to receive the HDF5 codebase for extensive testing.

FastQuery uses the FastBit technology to perform efficient searching and indexing. FastQuery allows the users to generate complex selections on HDF5 datasets e.g. (temperature > 1000) AND (70 < pressure < 90). FastQuery uses FastBit to generate compressed bitmap indices that accelerate HDF5 dataset searching.

FastBit is used to generate efficient indices. FastBit specializes in the bitmap indexing of numeric data. It uses the bitmap indexing technique to process complex and multi-dimensional ad-hoc queries. It uses bitmap compression methods designed to be more effective than other existing solutions. The image below shows you the difference of a sequential scan of a HDF5 file compared to a scan which used the FastBit bitmap indexing technique.

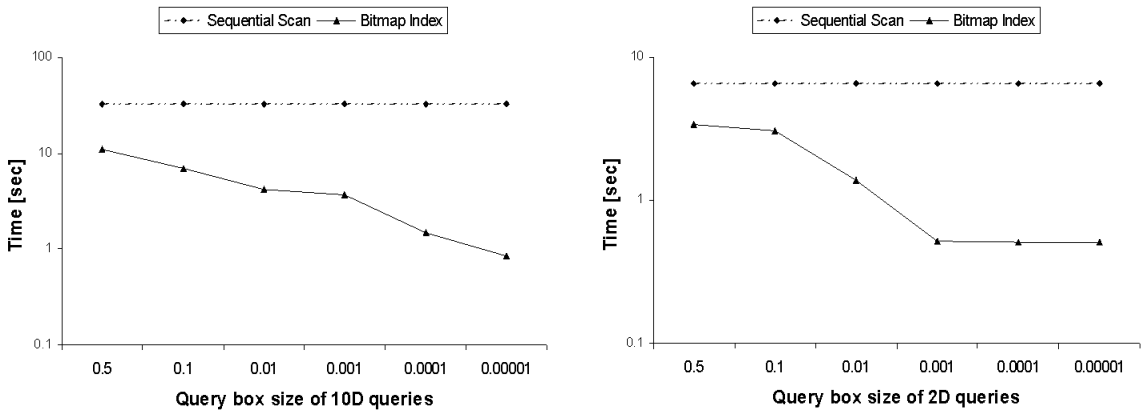


Figure 38: Sequential vs. Bitmap indexing scan

FastQuery extends the HDF5 complex, multidimensional selection mechanism to allow arbitrary range conditions. The bitmap indices are used to accelerate the selection process. It is possible to use compound queries that exceed a single dataset. The generated bitmap indices are stored in the same file as the datasets.

6.5.2 FastQuery Implementation Analysis

The implementation of FastQuery in Phaedra is a complex process. FastQuery is developed in a Unix environment with the programming language C++ however Phaedra is a Java application developed in Windows. To be able to implement FastQuery in Phaedra, FastQuery needs to be successfully compiled to a Windows environment. FastQuery doesn't provide a pre-compiled Windows version. Fortunately it provides a Microsoft Visual Studio project with the source code.

There are 2 possible solutions to compile FastQuery for the use on Windows. The first solutions is to compile FastQuery through a Unix/Linux environment. The second solution is to adjust the Microsoft Visual Studio project and compile it for the Windows environment. The result must be an executable file (.exe) or a dynamic-link library (dll) to create an efficient solution for Phaedra.

FastQuery depends on other software such as FastBit. To compile FastQuery, it requires the input of the following programs/applications:

- FastBit
- HDF5
- Message Passing Interface (MPI)

This significantly increases the implementation difficulty because all these programs are developed in a Unix/Linux based environment.

When FastQuery is compiled, it needs to be implemented in Phaedra. This can be done by including the exe or dll into Phaedra. Extensive testing is required to explore the indexing and searching possibilities. All the indexing possibilities provided by FastBit can be useful. There are several different options to create an index with FastBit. All these options need to be tested to determine which parameters provide the best performance in Phaedra. The searching possibilities need to be tested to determine the performance of complex queries.

Another major advantage of FastQuery is that updating of existing indices is very efficient. FastBit checks if the original HDF5 file is changed and only adjusts the changes necessary to the index file.

6.5.3 FastQuery in Phaedra

FastQuery is the most ideal solution for the indexing and searching problem in Phaedra because it covers all the following requirements:

- Complex queries;
- Separated index;
- Fast querying;
- Easy to update index;
- Parallel searching;
- No change to the existing Phaedra structure is required.

A successful implementation of FastQuery in Phaedra can be completely stand-alone, can cover all the requirements and can be very efficient. Therefore, FastQuery is the most promising solution.

6.5.4 Advantages vs. Disadvantages of FastQuery

Advantages

- Covers all the existing requirements
- Customizable to our needs

- Searching speed
- Indexing possibilities
- Parallel searching
- Free of charge
- Separated indices
- Updating of the index
- No change to the existing Phaedra structure is required
- C++ works fast

Disadvantages

- Developed in and for a Unix/Linux environment
- C++ compiling is difficult
- Not released yet (still in development)
- No existing documentation

6.5.5 Recommendation

FastQuery is the best solution for Phaedra. It covers all the existing requirements without having to change anything to the Phaedra structure. However, the implementation will take a great effort and the possibility to encounter errors in FastQuery exists because it is still in development.

7 TESTING POSSIBLE SOLUTIONS

This chapter describes the implementation of the possible solutions described in the previous chapter.

7.1 Lucene Test Case

This chapter describes the attempt to implement Lucene into a Java application and eventually Phaedra.

7.1.1 Lucene demo implementation

To extensively test Lucene, a demo project was set-up. This was a basic Java project where Lucene was implemented. The demo code can be found in appendix "1. Lucene demo implementation code".

7.1.2 Basic HDF5 file search with Lucene.

The most basic search in Phaedra is to search for one single subwell value. To do this, we need to convert a HDF5 file to a well based document. This means that the following structure is applied:

- A document represents 1 well
- A document contains a field with the well ID
- A document contains all subwell values stored in different fields

This is a visual representation of a well based document:

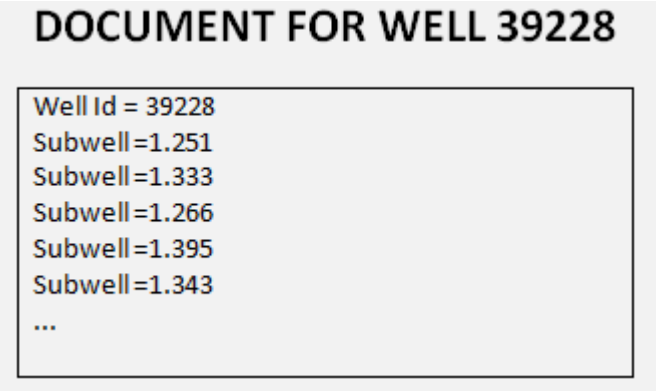


Figure 39: Well-based Document Representation

The search query has the following specifications:

- Search for all subwell values equal to 1.251
- Return all the subwell indices.
- Perform the search well 39228

The following result is shown:

```
Found 132 hits.
1. 57   1.251
2. 56   1.251
3. 63   1.251
4. 68   1.251
5. 69   1.251
6. 77   1.251
7. 78   1.251
8. 85   1.251
9. 81   1.251
10. 83  1.251
```

Figure 40: Search Results (Lucene basic search test)

The first number represents the index at which the hit occurred. The second number represents the actual value of the index

7.1.3 Optimizing the documenting stage

As mentioned before, Lucene is not scalable to perform the documenting based on subwell level. This means that the following structure must be applied:

- A document represents 1 feature
- A document contains fields with all the wells stored in different fields.
- Every well contains a one-dimensional array with its subwell values.

This is a visual representation of a feature based document:

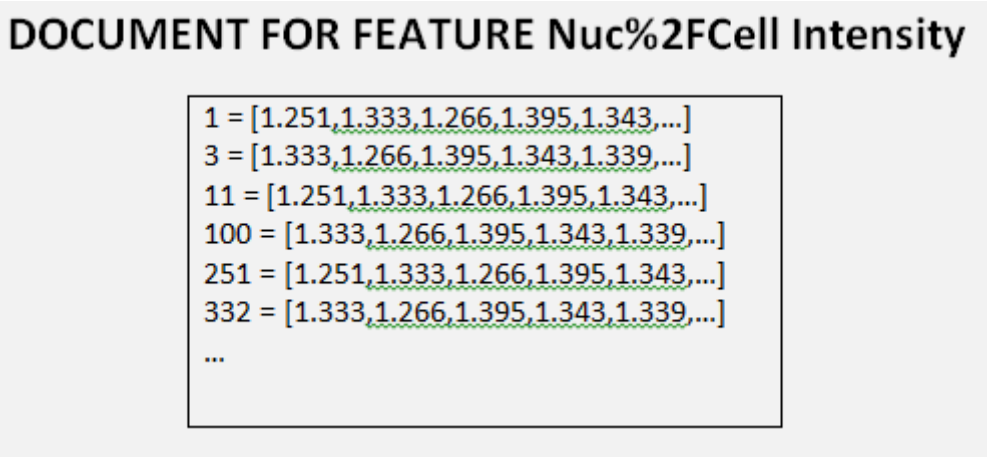


Figure 41: Feature-based Document Representation

To be able to query this type of document the following requirements must be met:

- Support for arrays in documents
- Arrays must be assigned to a field containing the well id
- Arrays need to be numeric to perform complex searching. (E.g. Nuc%2FCell Intensity BETWEEN 1.251 AND 1.300)

Term Vectors are the only solution in Lucene to store arrays. These arrays can be stored in a single value field. After extensive research and testing, Term Vectors aren't able to correctly store numeric data arrays. Term Vectors will convert this array to a string of values appearing to be a data array but it was not possible to perform complex searching on term vectors.

7.1.4 Conclusion

Because of the scalability limitations and the Term Vector shortcomings, Lucene is not able to perform the search queries required by Phaedra. The following table shows an overview of the Phaedra requirements and whether Lucene supports them.

Subject	Requirement	Theoretically	In Practice
Performance	Fast Querying	Yes	Yes*
	Adding data needs to be reasonably fast	Yes	Not tested
	Complex Numeric Queries	Yes	No**
Robustness	Index updating needs to be centralized, or managed in such a way that the "global" index is never stale or corrupt.	Yes	Not tested
Compatibility	Add subwell data querying functionality to the existing query screen	Partially	Not tested

* Answer based on 2 basic search tests
** Complex Numeric Queries aren't possible on feature based documents

7.2 FastQuery Test Case

This chapter will cover the entire FastQuery implementation effort.

7.2.1 Compiling FastQuery

The first step to test FastQuery is to compile FastQuery to be able to run in a Windows environment. There are two possible solutions to this problem:

1. Compile in a Unix/Linux (simulated) environment
2. Use the provided Visual Studio project

Because FastQuery is being developed in a Unix/Linux environment, our best chance to get results was through compiling it in a Unix/Linux environment. Because the compiled result needed to work on Windows, a Windows based solution was used. Cygwin is a Unix-like environment and command-line interface for Windows. It is commonly known to be used to compile Unix based applications in a Windows environment. By using Cygwin, it was possible to access your Windows drives through the Unix command-line interface.

To compile the Unix based C++ application the GCC GNU Compiler of Cygwin was used.

The compiled version of FastBit then was included to compile FastQuery successfully. MPI was an optional plugin to enable instances of FastQuery to run parallel. Multi-threading to enable parallel searching in Phaedra was implemented in Java so we didn't need to include MPI to compile FastQuery.

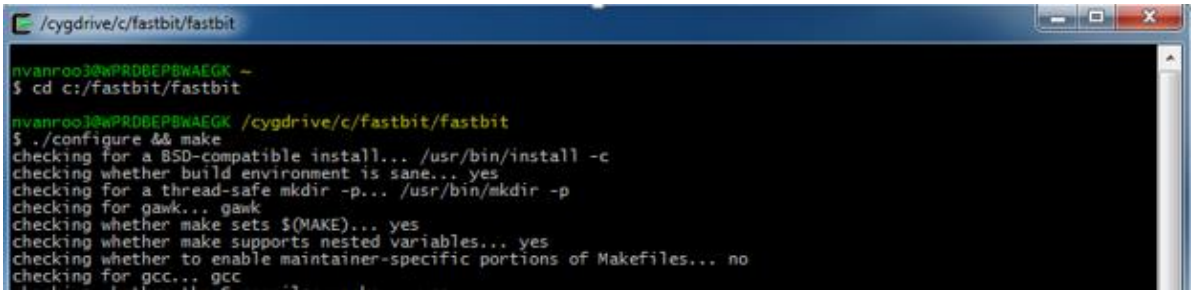


Figure 42: Compiling FastBit

The compiled version of FastBit was included in the FastQuery setup. The next step was to compile FastQuery. The compiler however wasn't able to compile FastQuery because of some unknown issue while including FastBit. After re-evaluating the configuration and "make" files for both FastQuery and FastBit together with my colleague Frederick Michielssen, we didn't find any errors in our compiling steps. The problem most likely lies in the underlying FastQuery or FastBit code. This problem was put on hold because the Visual Studio compiling solution had not been tested yet.

The Visual Studio solution came with the required plug-ins already present. However, the Visual Studio solution needed some alterations to the C++ header files to be able to compile on Windows. After some minor modifications, FastQuery successfully compiled. Two exec's were created:

- buildIndex.exe
- queryIndex.exe

These two .exe files required the following DLLs to be present in the same folder:

- fastbit.dll

- hdf5.dll
- pthreadVC2.dll
- stlport_vc10_x64.5.2.dll (for testing purposes)
- stlport_vc10_x64d.5.2.dll (for testing purposes)

The compiling of FastQuery was successful.

7.2.2 FastQuery Command-Line Tests

To test the compiled FastQuery for a basic use, some test were conducted in the Windows command-line interface. The first test was trying to build an index of a small Phaedra HDF5 file by using the generated "buildIndex.exe". The following command was used:

buildIndex.exe –f 39228.h5 –I index.h5

This should:

- build an index for all the datasets in the 39228.h5 file;
- write the index to the index.h5 file;

The build of the index failed however. When debugging the FastQuery solution, the following problem was encountered: Dataset names in HDF5 files must start with an alpha numeric value. All the dataset names used in Phaedra start with a numeric value. This problem was fixed by creating a workaround which adds a leading alpha numeric character to the dataset names while creating the index. This solved the problem without having to adjust a lot of the FastQuery source code.

After fixing this problem, the build index command succeeded. The index file was successfully created.

The second test was trying to search the index file for a specific value. The following command was used:

queryIndex.exe -f 39228.h5 -i index.h5 -q "d1 > 200.005" -p "SubWellData/FSC-A"

This should:

- search the index file and return the hits.

The search query succeeded. It returned the amount of hits.

However the buildIndex.exe and queryIndex.exe were working, some new issues came along being:

- The generated index file is 5 times larger than the original HDF5 file. This is due to the advanced bitmap indices that are generated to improve the search performance. FastBit provides functionalities to adjust the generation of index files. These functionalities should be researched to see if they can reduce the index size without losing too much performance.
- The search query returned the amount of hits, not the index of the actual hits to retrieve the correct values. This should be resolved because Phaedra requires the actual values of the hits, not the total hit amount.
- The search query must specify the exact dataset where the search needs to be performed. It is not possible to perform a search query on all the datasets at once. This must be countered in Java by multi-threading the query for every specific dataset.

- In some cases the queryIndex.exe crashes. Fortunately, the amount of hits were returned before the crash so FastQuery did successfully complete the search query.

7.2.3 Java test case

A basic Java project was set-up to test the search speed through an .exe file in Java. This test case was executed multiple times on different HDF5 files to generate accurate results. The HDF5 files are strategically chosen and vary from file size, number of features and number of subwell values. The image below shows you the structure of the Java project.

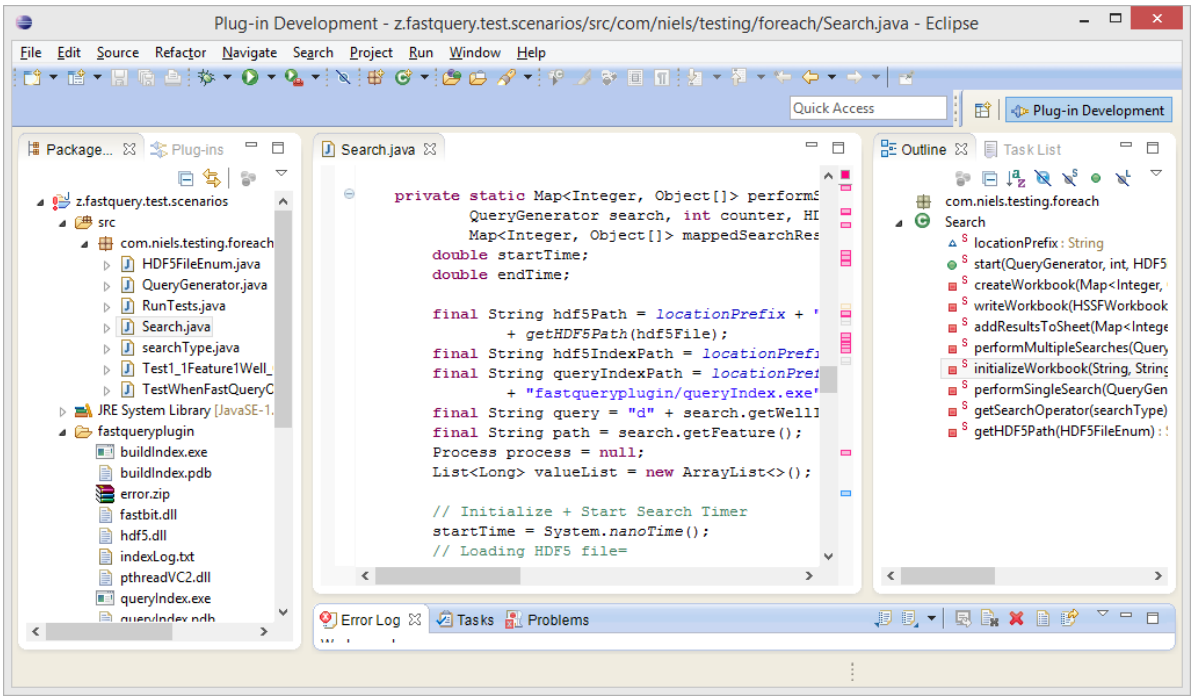


Figure 43: FastQuery Java Test Case

The test case followed these steps:

- Initialize all variables needed
- Start timer
- Call the queryIndex.exe with the correct parameters
- Convert the output from queryIndex.exe to an hit-array containing all the individual hits
- Stop timer
- Print result to an Excel file

This was done multiple times to generate accurate speed measurements.

To get a good idea of the improved speed from using FastQuery, a brute force test scenario had to be created. This scenario didn't use an index file but just reads out a HDF5 file in a for-loop to check for each value if it meets the query requirements. This Java project is very similar to the FastQuery project. The table below compares the search speed of the brute force test results with the FastQuery test results.

HDF5 file	FastQuery no index	FastQuery
37505	+250%	+11%
46468	+49%	+9%
47495	+63%	+12%

48294	+49%	+5%
48769	+42%	+20%
49486	+56%	+16%
54655	+53%	+13%

FastQuery is much slower than the brute force test. After extensive debugging and research the following issues came up to why FastQuery was slower than the brute force test:

- FastQuery initializes all the datasets available in the index/HDF5 file. The brute force test only reads the dataset necessary to perform the search query. Because Phaedra will always know which dataset needs to be searched against, it can be modified in FastQuery to only initialize the required dataset.
- queryIndex.exe returns it results through an output file. This means that all the output needs to be printed, read and stored before it can be used. This process must be revised to improve performance.

7.2.4 Issue: Query returns amount of hits

FastQuery returned the amount of hits instead of the specific hits. This was modified by editing the FastQuery core. It requests the amount of hits from FastBit, this was modified so that FastBit returns the specific hits. After the specific hits were returned, they were printed in the standard output following this structure: "{{" [hit], [hit], … "}}". The code snippet below shows you the code used by FastQuery.

```
if (it != qlist.end()) {
    ibis::query::QUERY_STATE qst = (*it).second->getState();
    timer.stop();
    switch (qst) {
    default:
        logWarning("submitQuery", "query not fully specified");
        break;
    case ibis::query::QUICK_ESTIMATE:
    case ibis::query::SET_PREDICATE:
    case ibis::query::SPECIFIED:
    case ibis::query::SET_RIDS:
        (*it).second->evaluate();
    case ibis::query::FULL_EVALUATE:

        stlp_std::vector<uint32_t, stlp_std::allocator<uint32_t>> rids;
        (*it).second->getHitRows(rids);


        std::cout << "{{";
        stlp_std::vector<uint32_t, stlp_std::allocator<uint32_t>>::iterator iter = rids.begin();
        std::cout << *iter;
        ++iter;
        for(;iter != rids.end();++iter) {
            std::cout <<"," << *iter;
        }
        std::cout << "}}\n";
        std::cout.flush();

        break;
    }
```

Figure 44: FastQuery Return Amount of Hits (Code Snippet)

7.2.5 Issue: FastQuery crashes on specific queries

This issue isn't resolved because the results could have been retrieved before the crash occurs. This way the crash was ignored.

7.2.6 Issue: FastQuery initializes all available datasets

Instead of initializing all the available datasets, FastQuery should only initialize the required dataset in order to increase the performance. This issue is resolved by adding a filter when obtaining all the variables. FastQuery now only initializes the required variable.

After adding this filter to FastQuery, the FastQuery tests were continued. After rerunning the FastQuery tests we booked an incredible increase in performance. The following images show the improvement in search speed. The left side are the old results and the right side are the new results

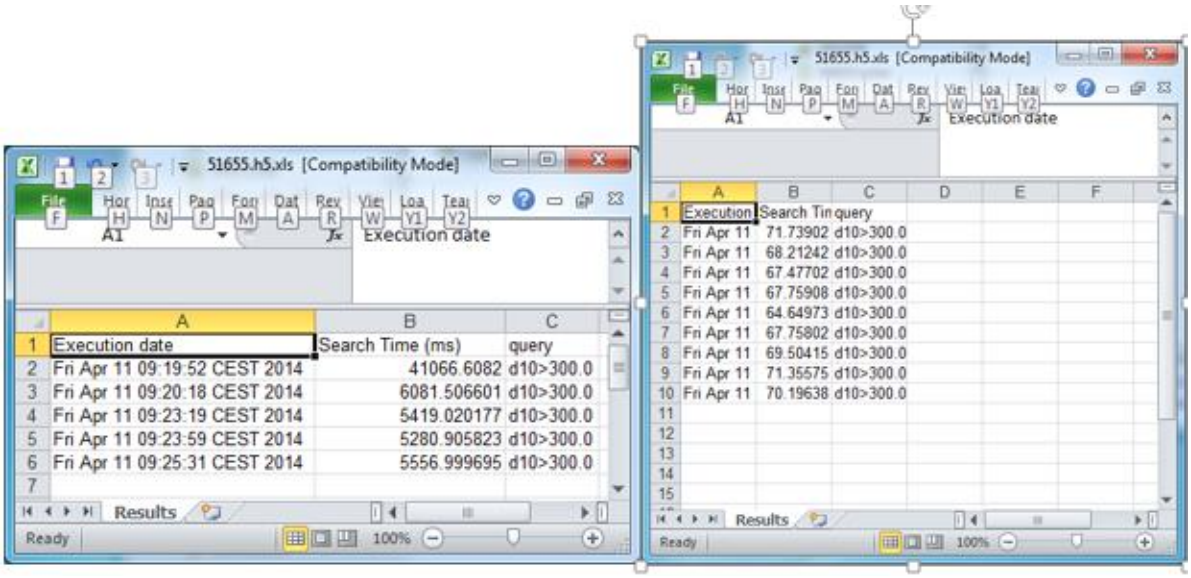


Figure 45: FastQuery Improvements 1 (5500ms - 65ms)

In the image below the top results are the old results and the bottom results are the new results.

Execution date	Search Time (ms)	query
Thu Apr 10 12:55:33 CEST 2014	1796.645283	d14>0.2
Thu Apr 10 12:56:31 CEST 2014	1864.605957	d14>0.2
Thu Apr 10 12:59:44 CEST 2014	2047.887013	d14>0.2
Thu Apr 10 12:59:48 CEST 2014	1792.795962	d14>0.2
Thu Apr 10 13:00:32 CEST 2014	1825.074062	d14>0.2
Thu Apr 10 13:02:44 CEST 2014	1972.444989	d14>0.2
Thu Apr 10 13:03:50 CEST 2014	1850.284868	d14>0.2
Fri Apr 11 09:19:11 CEST 2014	29086.77119	d14>0.2
Fri Apr 11 09:20:12 CEST 2014	1745.112403	d14>0.2
Fri Apr 11 14:47:33 CEST 2014	74.704681	d14>0.2
Fri Apr 11 14:47:40 CEST 2014	80.149756	d14>0.2
Fri Apr 11 14:47:42 CEST 2014	75.122989	d14>0.2
Fri Apr 11 14:47:43 CEST 2014	73.109663	d14>0.2
Fri Apr 11 14:47:45 CEST 2014	75.752219	d14>0.2
Fri Apr 11 14:48:20 CEST 2014	76.409408	d14>0.2

Figure 46: FastQuery Improvements 2 (1900ms - 75ms)

7.2.7 Issue: Alternative for the .exe file

Returning the results through an .exe file is too slow. The results should be returned directly to Java so we can put the results in an object in FastQuery and pass this object trough to Phaedra without having to print and read it.

This can be done with Java Native Interface (JNI). JNI is a programming framework that enables us to call native applications and libraries written in C++. This solution will be developed when the FastQuery solution is fully researched and working to the needs of Phaedra.

This will have a large impact on the FastQuery searching speed.

7.2.8 Issue: Index file size

The index file size can be 8 to 9 times the size of the original HDF5 file. This requires too much storage space. FastBit provides binning and encoding options to customize the index file to specific needs.

Binning options can be used to reduce the number of bitmaps in your index file. This will result in a lower file size but will decrease the search performance. There are 2 binning options which can be applied to the Phaedra HDF5 files. The nbins-option and the precision-option. The nbins-option will specify the amount of bins used to store your index data. The less bins, the less the index size, the less the performance. The precision-option will generate bins corresponding to the reduced precision of floating-point numbers. The image below shows the different index file sizes with the applied binning options.

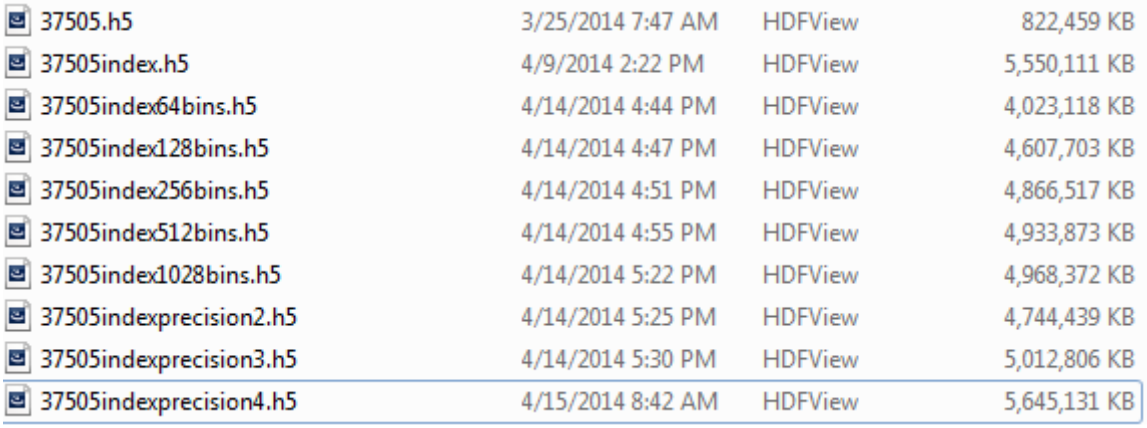


Figure 47: Applied Binning Options

The file sizes are not reduced that much. After extensive testing it shows that the performance of index file "37505index64bins.h5" is reduced 7 times. This trend applies to all the other index files as well. The reduced index size is not worth the significant performance loss.

To counter this performance loss, encoding options can be used to increase the performance for your index file. After extensive debugging, the encoding options doesn't seem to work in FastBit. This means that another strategy must be used to create the indices.

The best suggestion is to only generate indices for the key features. This means that only 20-30% of the index file will be indexed. If a cellular search is implemented in Phaedra, end-users will search for key-features 99% of the time. If you wish to search for non-key-features, a basic FastQuery search can be executed without the use of an index file resulting in a slow search. If the non-key-feature search is a rare event, this is the best solution to counter the index file size problem.

7.2.9 Recommendation

Due to the limited time of the internship, the implementation of FastQuery is not finished yet. I recommend the following implementation steps:

- Don’t use the binning options or encoding options to reduce file size because it will significantly decrease the search performance. Only index key-features instead.
- Create a JNI wrapper to include FastQuery in Phaedra

Except for the few problems mentioned above, FastQuery is the best possible solution for the HDF5 indexing and searching problem. Based on the analysis of the other possible solutions, it is recommended to implement FastQuery. It supports all the requirements without having to change the structure of the Phaedra application.

CONCLUSION

During my internship I learned a lot about documenting techniques, debugging, compiling, C++ and Java. I created a new documentation for the Phaedra project and researched the existing HDF5 indexing and searching problem in Phaedra. During my internship I faced a lot of exiting and difficult material. This helped me to realize how difficult IT solutions can be in the real world.

I believe I completed my initial goals to create a professional documentation and to perform an extensive research for indexing and searching solutions. All my contributions to the Phaedra project will increase the user experience.

I conclude that this internship was one of the most challenging and educational experiences I have ever had. My new knowledge of debugging, documenting and compiling will be very useful in the future. I am very grateful for the opportunities I received.

REFERENCES

Alted, F. (2010). *Chapter 5. Optimization tips - Part I. The PyTables Core Library*. Retrieved 04 04, 2014, from http://www.pytables.org/: http://www.pytables.org/docs/manual-2.2.1/ch05.html

Apache Lucene Core, A. S. (2012). *Apache Lucene Core*. Retrieved from Apache Lucene: http://lucene.apache.org/core/index.html

Balaguer, F. A. (2007, 07 11). *OPSI: The indexing system of PyTables 2.* Retrieved 04 04, 2014, from www.pytables.org: http://www.pytables.org/docs/OPSI-indexes.pdf

braves, g. o. (2011, 09 03). *PyTables Pro*. Retrieved 04 04, 2014, from www.pytables.com: http://www.pytables.org/moin/PyTablesPro

braves, G. o. (2014, 03 25). *Pytables*. Retrieved 04 04, 2014, from www.pytables.org: http://www.pytables.org/moin

Core, A. L. (2013, June 21). *Index File Formats*. Retrieved from Apache Lucene: http://lucene.apache.org/core/3_0_3/fileformats.html

Core, A. L. (n.d.). *Limitations*. Retrieved from lucene.apache.org: http://lucene.apache.org/core/4_0_0/core/org/apache/lucene/codecs/lucene40/package-summary.html#Limitations

Grand, A. (2013, January 23). *Putting term vectors on a diet*. Retrieved from blog.jpountz.net: http://blog.jpountz.net/post/41301889664/putting-term-vectors-on-a-diet

Group, V. (2010, November 10). *HDF5-FastQuery: Accelerating Complex Queries on HDF Datasets using Fast Bitmap Indices*. Retrieved March 21, 2014, from Visualization Group: http://www-vis.lbl.gov/Events/SC05/HDF5FastQuery/

maintainers, P. (2014). *Pytables 3.1.1 documentation*. Retrieved 04 04, 2014, from pytables.github.io: http://pytables.github.io/usersguide/introduction.html

MongoDB. (2014). *Introduction to MongoDB*. Retrieved March 21, 2014, from MongoDB: http://docs.mongodb.org/manual/core/introduction/

Pande, N. (2010, January 10). *Lucene basics*. Retrieved from Slideshare: http://www.slideshare.net/nitin_stephens/lucene-basics

Rahul, J. (2013, January 14). *Scaling Lucene for Indexing a Billion Documents*. Retrieved from rahuldausa.wordpress.com: http://rahuldausa.wordpress.com/2013/01/14/scaling-lucene-for-indexing-a-billion-documents/

Wiki, L. (2013, February 28). *Powered By*. Retrieved from Lucene Wiki: http://wiki.apache.org/lucene-java/PoweredBy

APPENDICES

1 LUCENE DEMO IMPLEMENTATION CODE

Lucene demo implementation code

```
package niels;

import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.StringField;
import org.apache.lucene.document.TextField;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.queryparser.classic.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopScoreDocCollector;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.RAMDirectory;
import org.apache.lucene.util.Version;


public class testing
{
	public static void main(String[] args)
	{
		try
		{
			//    Specify the analyzer for tokenizing text.
		    // The same analyzer should be used for indexing and searching
			StandardAnalyzer analyzer = new
StandardAnalyzer(Version.LUCENE_47);

			//    Code to create the index
			Directory index = new RAMDirectory();

			IndexWriterConfig config = new
IndexWriterConfig(Version.LUCENE_47, analyzer);

			IndexWriter w = new IndexWriter(index, config);
			addDoc(w, "Lucene in Action", "193398817");
			addDoc(w, "Lucene for Dummies", "55320055Z");
			addDoc(w, "Managing Gigabytes", "55063554A");
			addDoc(w, "The Art of Computer Science", "9900333X");
			addDoc(w, "My name is teja", "12842d99");
			addDoc(w, "Lucene demo by teja", "23k43413");
			w.close();

			//    Text to search
			String querystr = args.length > 0 ? args[0] : "teja";

			//    The "title" arg specifies the default field to use when
no field is explicitly specified in the query
			Query q = new QueryParser(Version.LUCENE_47, "title",
analyzer).parse(querystr);
```

```
                    // Searching code
                    int hitsPerPage = 10;
                IndexReader reader = DirectoryReader.open(index);
                IndexSearcher searcher = new IndexSearcher(reader);
                TopScoreDocCollector collector =
TopScoreDocCollector.create(hitsPerPage, true);
                searcher.search(q, collector);
                ScoreDoc[] hits = collector.topDocs().scoreDocs;

                //Code to display the results of search
                System.out.println("Found " + hits.length + " hits.");
                for(int i=0;i<hits.length;++i)
                {
                  int docId = hits[i].doc;
                  Document d = searcher.doc(docId);
                  System.out.println((i + 1) + ". " + d.get("isbn") + "\\t" +
d.get("title"));
                }

                // reader can only be closed when there is no need to access the
documents any more
                reader.close();
            }
            catch(Exception e)
            {
                    System.out.println(e.getMessage());
            }
        }
        private static void addDoc(IndexWriter w, String title, String isbn)
throws IOException
        {
                Document doc = new Document();
                // A text field will be tokenized
                doc.add(new TextField("title", title, Field.Store.YES));
                // We use a string field for isbn because we don\'t want it
tokenized
                doc.add(new StringField("isbn", isbn, Field.Store.YES));
                w.addDocument(doc);
        }
}
```

Results:

```
Found 2 hits.
1. 12842d99\tMy name is teja
2. 23k43413\tLucene demo by teja
```

2 DOCUMENTATION IMPLEMENTATION SETTINGS

Manifest.MF

```
Bundle-Version: 1.0.0.qualifier
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Localization: plugin
Bundle-Name: Phaedra Help
Bundle-Vendor: JNJ
Bundle-ActivationPolicy: lazy
Bundle-SymbolicName: com.jnj.phaedra.help; singleton:=true
Require-Bundle: org.eclipse.core.runtime,
 org.eclipse.ui
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Bundle-Activator: com.jnj.phaedra.help.Activator
Import-Package: org.eclipse.help

```

Figure 48: Documentation Implementation Settings (Manifest.MF)

Build.properties

```
source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               plugin.properties,\
               plugin.xml,\
               Topics/,\
               Images/,\
               eclipsecontexts.xml,\
               eclipsehelp.css,\
               eclipsetoc.xml,\
               index.xml

```

Figure 49: Documentation Implementation Settings (Build.properties)

Plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
<extension point="org.eclipse.help.toc">
<toc file="eclipsetoc.xml"        primary="true"/>
</extension>
<extension point="org.eclipse.help.contexts">
<contexts
        file="eclipsecontexts.xml" plugin="org.eclipse.datatools.connectivity.ui"/>
</extension>

<extension
            point="org.eclipse.help.index">
<index file="index.xml"/>
</extension></plugin>
```

Figure 50: Documentation Implementation Settings (plugin.xml)