



Software developed by:  
Hauptman-Woodward Medical Research Institute  
[www.hwi.buffalo.edu](http://www.hwi.buffalo.edu)

Supported by funds from the Seymour H. Knox Foundation

## User Manual

Software Version 0.5.0

# Table of Contents

<b>Part I: Overview .....</b>	<b>1</b>
<b>Introduction.....</b>	<b>2</b>
<b>About this Manual.....</b>	<b>2</b>
<b>System Requirements .....</b>	<b>3</b>
<b>Hawkeye vs. MacroScopeJ .....</b>	<b>3</b>
 <b>Part II: Basic Usage.....</b>	<b>5</b>
<b>Chapter 1: Getting Started .....</b>	<b>6</b>
Installation .....	6
Launching Hawkeye .....	6
Initial Setup Wizard.....	6
Creating a User & Logging In.....	7
Main Interface.....	8
<b>Chapter 2: Working with Projects.....</b>	<b>9</b>
What is a Hawkeye Project? .....	9
Importing RAR Archives .....	9
Manual Project Creation.....	9
Adding Features to a Project .....	9
Saving Projects.....	10
Opening Existing Projects .....	10
Closing vs. Deleting Projects & Features .....	11
Using the QuickStart Wizard .....	11
<b>Chapter 3: Project Features .....</b>	<b>12</b>
Dataset.....	12
Data Table .....	13
Data Tree.....	13
Experiment Map .....	14
<b>Chapter 4: Reviewing Experiments.....</b>	<b>17</b>
Inspector Window.....	17
Review Status.....	18
Scoring .....	19
Selection.....	20
Additional Tools .....	20
<b>Chapter 5: Exporting Data.....</b>	<b>23</b>
Exporting Score Files.....	23
Exporting Selection Reports (HTML).....	23
Exporting MSO Files .....	24

<b>Part III: Advanced Usage .....</b>	<b>25</b>
<b>Chapter 6: HawkeyeDB .....</b>	<b>26</b>
Embedded Derby Database .....	26
User Access & Profile Usage .....	27
Using SQL to Write Queries .....	27
Database Organization.....	28
Database Terminal .....	29
Resetting the Database.....	29
Extending the Database .....	30
<b>Chapter 7: Experiment Map: Plot Strategy .....</b>	<b>31</b>
What is a Plot Strategy?.....	31
Configuring a Plot Strategy .....	31
Adding Dimensions .....	32
Creating a Match Query.....	32
Layout Generation & “Stale” Status .....	33
<b>Part IV: Appendixes .....</b>	<b>34</b>
<b>Appendix I: Changing Settings .....</b>	<b>35</b>
<b>Appendix II: Memory &amp; Data Handling .....</b>	<b>35</b>
<b>Appendix III: Known Bugs &amp; Issues.....</b>	<b>36</b>
<b>Appendix IV: Support Info.....</b>	<b>37</b>
<b>Appendix V: Using Hawkeye on Ubuntu (Linux) .....</b>	<b>37</b>

# **Part I: Overview**

## Introduction

*Hawkeye* is a multi-platform, data-centric software package intended as the primary user interface for reviewing and analyzing results obtained from HWI's high-throughput crystallization screening service. The software aims to incorporate functions of existing tools (*MacroScopeJ*, *AutoSherlock*, *CrossPlate*, etc.), unifying them around a central embedded database to ensure consistency, flexibility, and interoperability. *Hawkeye* is still a work in progress, and represents a broad framework into which future development can be integrated.

*Hawkeye* is...

- **Experiment-centric:** To accurately represent and take advantage of our high-throughput crystallization process, we focused on the *Experiment*: what are the conditions in each well, and what result(s) did those conditions produce? *Hawkeye* makes it easy to see how experiments performed over time, and which conditions were significant to their outcomes.
- **Data-centric:** All data in *Hawkeye* is persisted in the embedded database and can be queried through the context of Experiments; for example, asking such questions as “Which ingredients were present in experiments that produced crystal hits for all my samples to date?” Once the dataset is established, it can be displayed using other features, such as tables, trees, and maps.
- **User-empowering:** For basic users, a *QuickStart Wizard* is available to easily set up standard projects and get to quickly reviewing crystallization outcomes much like they did in *MacroScopeJ*. More advanced users can create custom datasets through the use of SQL queries, and construct multi-dimensional maps to arrange experiments according to their data elements.
- **A versatile, expanding package:** *Hawkeye* seeks to replace many of our older review and analysis tools by integrating their functionality into a single package built around a central database. As our needs change and our techniques evolve, we envision additional tools and features being incorporated.
- **Multi-platform:** Written in Java, the application is designed to be compatible with any Java-enabled OS. Windows, Mac OS X, and Ubuntu Linux are currently supported.

## About this Manual

This manual, like *Hawkeye*, is intended for both basic and advanced users. Depending on what your goals are, you may not need to read this manual in entirety.

I want to review experiments or images plate-by-plate, and possibly send my observations to someone.	I want to analyze conditions or outcomes in order to reformulate a screen or study, or to create a figure for presentation/publication.
<ul style="list-style-type: none"> <li>• Read the section “<b>Hawkeye vs. MacroScopeJ</b>” in Part I.</li> <li>• Read all of <b>Part II: Basic Usage</b>.</li> <li>• Read the section “<b>Support Info</b>” in Part IV.</li> <li>• Use <i>QuickStart Wizard</i> for day-to-day operation.</li> </ul>	<ul style="list-style-type: none"> <li>• It is recommended that you read the <b>entire manual</b>.</li> <li>• Find additional resources, such as a complete <b>HawkeyeDB (database) schema diagram</b> and <b>Derby reference</b>, in your <i>hawkeye/doc/</i> directory.</li> </ul>

## System Requirements

- Java-capable OS (officially tested on the following systems, though others may also work)
  - Windows 7, 8.1, and 10
  - Mac OS X 10.12 “Sierra”
  - Ubuntu 17.04
- Java Runtime Environment (JRE<sup>1</sup>) 1.8.0\_121 or later
- 2GB of available RAM<sup>2</sup>
- at least 1280 x 1024 screen resolution recommended for image review

## Hawkeye vs. Macroscopel

Many Hawkeye users will have had past experience with one of our older review programs, Macroscopel. This section shows a side-by-side comparison of the two programs: how they differ in both design and capability.

	Macroscopel	Hawkeye
user interface	Macroscopel uses a multiple document interface (MDI), with each feature or document being an internal window within the program’s “workspace” area. To allow more space for features, the main window must be enlarged.	Hawkeye uses a tabbed document interface (TDI), with each feature or document appearing in the main window under a tab. To compare features side-by-side, each tab can be moved out into an independent window anywhere on the screen, and can be resized independently of other windows.
design philosophy	Image-centric: focuses on each individual image as a unit; cocktail data and history are treated as attributes of the image.	Experiment-centric: focuses on each individual experiment as a unit; plate, sample, cocktail, and reads/images are treated as attributes of the experiment.
image import	Images must be loaded from their respective RAR files into memory anytime they are to be displayed in the program.	Images are extracted from RAR files only once, and saved to an image repository on disk. They are available for immediate retrieval anytime the program is open.
data import	Cocktail ingredient data is stored in the RAR files in the form of flat, unstructured text. It must be loaded into memory anytime it is to be displayed in the program.	Plate, sample, and cocktail data is stored in the RAR files in explicitly-typed XML documents. It is imported into the embedded Derby database <i>once</i> , and remains available for any subsequent review or analysis.

<sup>1</sup> Due to a bug in the Mac JRE installation, Mac users should install the Java Development Kit (JDK) rather than the JRE. Both are available from [www.java.com](http://www.java.com).

<sup>2</sup> 2048 MB (2GB) is the default/recommended maximum amount of RAM for Hawkeye’s JVM (Java Virtual Machine). Memory settings can be changed, depending on usage requirements and machine capability. See *Appendix II: Memory & Data Handling* for more information.

datasets	Data and images are available for each plate separately; however, the plate datasets cannot be merged, compared, or analyzed.	Datasets are created by means of SQL queries and can be customized to retrieve anything available in the database. Datasets may contain experiment data from multiple plates to do advanced cross-plate analysis.
review	When a cell is clicked, a popup window appears to allow review of the images individually at full-size. Each image can be “selected” and/or assigned 1 or more scores from the HWI’s classic 7-category scoring scheme. A “history” tool allows seeing other read images for the well, so long as their RARs have been loaded into the program during the same session; however, these other images cannot be selected or scored.	When a cell is clicked, an Inspector window appears for individual review of the map’s experiments. Each experiment is shown with a data summary and all available read images. Only one image at a time is indexed and shown at full-size; however, the index may be changed so that the user can select or score any images of the experiment. The user may select or score particular images OR entire experiments. Currently, only the classic HWI 7-category score scheme is available; though other schemes may be added later.
projects	MacroScopeJ was originally intended to use a project system; however, it was never implemented. Only a “Legacy” mode (designed for compatibility with the older MacroScope program) was completed and usable in the released versions. Legacy mode allowed importing and exporting MSO files containing scores and/or selections; but there was virtually no organization or customization available. In case of a sudden crash or interruption during the scoring process, all work was lost and had to be done over.	Hawkeye contains a complete project system. A project may contain 1 or more datasets and any number of additional features (tables, trees, maps, etc.) created/customized to display the datasets in a desired way. The project file itself is a ZIP file which contains XML descriptions of the datasets and features; thus they could potentially be used by other applications as well. Scores, selection, and review status are considered independent of projects and are persisted to the database; so in case of a crash or interruption, no work is lost.
outputs	Arrangements can be exported as MSO files, HTML selection lists, or an HTML-based simulation of the arrangement itself.	The main output produced by Hawkeye is the project file. In addition, the program can also export an XML score file, which can be used to transfer scores from one copy of Hawkeye to another. An HTML selection report, primarily intended for web-viewing, shows a selection of images or experiments along with a chemical data summary. Finally, traditional plain-text MSO files can be exported for use in legacy applications.
user	The user is not logged in, authenticated, or tracked in any way. Any work saved is anonymous. In order to compare or combine different users’ scores, additional software had to be written.	Hawkeye requires all users to create a username and password with which to authenticate before using the software. Scores and review status are recorded per user, allowing different users’ scores to be compared or combined in the same copy of Hawkeye, or in different copies.

# **Part II:**

# **Basic Usage**



## Chapter 1: Getting Started

### Installation

There is no install package or installation process. To install Hawkeye, simply extract the ZIP file onto your computer. It is recommended that you place the files on a local drive; running Hawkeye from a network location or removable storage device will likely degrade performance. On Linux systems (Ubuntu), additional steps may be necessary before the software can be run: see *Appendix V* for more information.

### Launching Hawkeye

To launch Hawkeye, execute the file appropriate for your operating system:

- on Windows, run *Hawkeye.exe* (Windows executable)
- on Mac OS X, run *Hawkeye* (Mac application)
- on Linux, run *Hawkeye* (bash script)

### Initial Setup Wizard

If you are launching your copy of Hawkeye for the first time, you will be presented with the *Initial Setup Wizard*. This wizard will guide you through some basic setup choices about how you plan to use Hawkeye. At the time of this writing, there are three settings:

- Import from Distribution: choose another distribution of Hawkeye (generally a previous version) from which to import database data and application settings. This ensures that existing scores, experiment data, and images remain available when updating Hawkeye to the latest version. Before the process commences, you can choose whether to import database data, application settings, or both. If you choose to import application settings, some setting fields in the wizard may be initialized to imported values. (This function is also available from the Splash/Login dialog under *More Options*.)
- Image Repository: specify a path for your image repository. This is where all extracted image files will be placed, and remain available for immediate retrieval as needed by Hawkeye. The repository will be utilized regardless of which user is logged into Hawkeye; therefore, if each Hawkeye user has a different OS login, you must ensure that the chosen repository location is accessible to all users of the OS. If you choose not to set up an image repository, Hawkeye will clean up all temporary extracted files when you exit the program, and you will need to re-load all required RAR archives any time you want to see images in the program. Failure to do so may result in an “Image Not Available” placeholder graphic being displayed instead of actual experiment images.
- Default Project Location: specify a path for new projects to be placed in by default. This is a convenience feature to keep your projects organized in one place and make them easy to find. Unless using the QuickStart Wizard, the user can navigate from the default path to any other path when creating a new project. Again, as with the image repository, this location should be accessible to all users.

## Creating a User & Logging In

After finishing the Initial Setup Wizard, you will be shown a Splash/Login dialog welcoming you to Hawkeye. For score and review purposes, Hawkeye requires all users to authenticate with a unique Login Name and a Password. Therefore, our first step will be to create a new user: click the Create New User button.

In the dialog, enter the details of your user account. When finished, click the Ok button to return to the Login dialog. Your new Login Name will appear in the drop-down list. Make sure it is selected; then enter your password and log in.

The main Hawkeye interface will be opened and you can begin using the software.

(If you wish to delete a user that you created, you can do so by checking the Delete User box. The Login button will change to a Delete button. Select a user from the list, enter the user's password, and click the Delete button. Note that all scores applied by the user will also be deleted from the database.)



Figure 1: Hawkeye Splash/Login screen, where users can be created and logged in.

## Main Interface

Let's take a moment to get acquainted with Hawkeye's main interface:

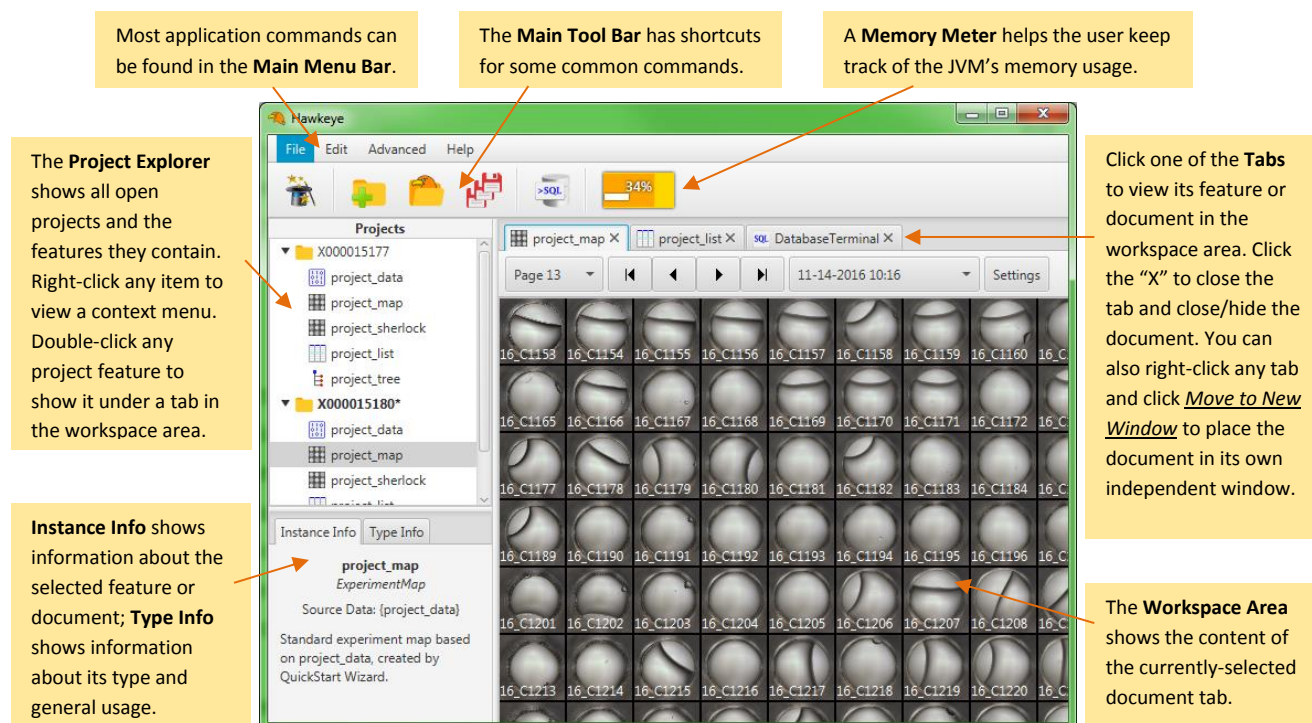


Figure 2: Hawkeye Main Interface.

Most elements of the main interface will be explored in the relevant sections of this manual. Of particular note, however, is Hawkeye's tabbed document interface (TDI) which allows the user to keep multiple documents open simultaneously, but view and work with only one document at a time—much like a modern web browser. For cases where it is necessary to view multiple documents side-by-side, or when a single document needs to be resized to take up more space than is allowed by the workspace area, the user can right-click any tab and move its contents to an independent, resizable window by clicking Move to New Window. When the new window is minimized, it returns to the main interface as a tab. To close the document, click the "X" on its tab, or close its window. (Closing a project feature will not delete it from the project; it can be reopened by double-clicking its icon in the Project Explorer.)

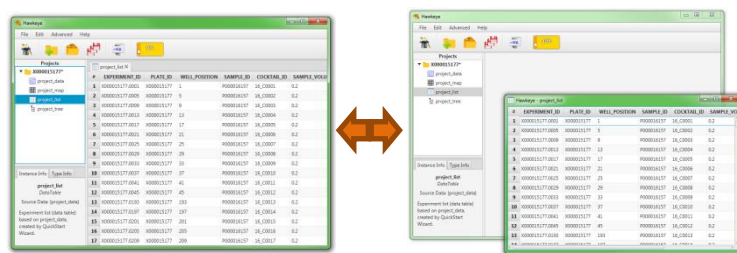


Figure 3: Any document can be moved from a tab to a new window, and vice versa.



## Saving Projects

Once you have added all desired features to a project, you should save the project so it can be reopened later or sent to your partners. When you open the File menu, you will see that there are several save options available:

- Save Project: The active project will be saved to the path specified at the time of its creation.
- Save Project As: The active project will be renamed and/or relocated to a new path and saved as a new project, separate from the original project. If your desired result is to have two separate projects, make sure to save the original before using this command, or the original will be lost.
- Save Project + Scores: The active project will be saved with an embedded score file. Scores will be from Experiments specified by the project's datasets. You can choose which type(s) of scores (image scores or experiment scores), and whether to include only your scores or all users' scores. When the project is opened in another copy of Hawkeye, the user will be asked which scores they want to import into their database.
- Save All Projects: All open projects will be saved to their respective paths.

Projects are saved as ZIP files with a “.hkp” extension. Beside project metadata, the directory contains an XML representation of each project feature. To examine these files, copy the project in your OS; rename the copy with a .zip extension; and extract it to a directory.

## Opening Existing Projects

To open a project in Hawkeye, open the File menu and select Open Project(s). Use the dialog to select one or more projects. The selected projects will be opened and displayed in the Project Explorer.

Most projects contain datasets dealing with experiment data, and therefore require that data to be present in the Hawkeye database. Each project is saved with its own list of required RAR archives that contain the data it needs. If you open a project that was sent to you by someone else, or if you open one of your own projects after resetting your database, you may receive a message saying that the project cannot be opened until required archives are imported. If you don't have the listed RAR files, you may be able to acquire them from the project's creator, or by contacting the HTSLab with your request. See the section *Importing RAR Archives* for more information on importing RAR files.

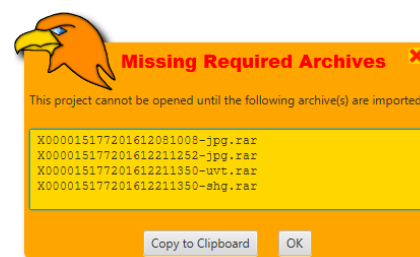


Figure 5: Missing required RAR archives.

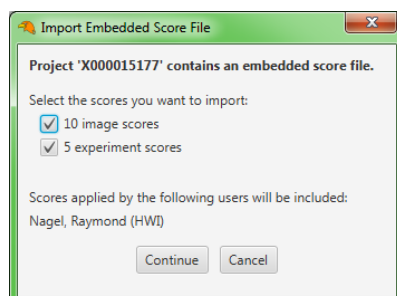


Figure 6: Importing an embedded score file.

If you are opening a project that has an embedded score file, you will be asked to select which (if any) scores you want to import. The dialog will also let you know which user(s) applied the scores. The selected scores will be merged into your database, and you will see them when viewing experiments and/or images. If you uncheck all the boxes, or choose Cancel, the project will be opened without any of the embedded scores.

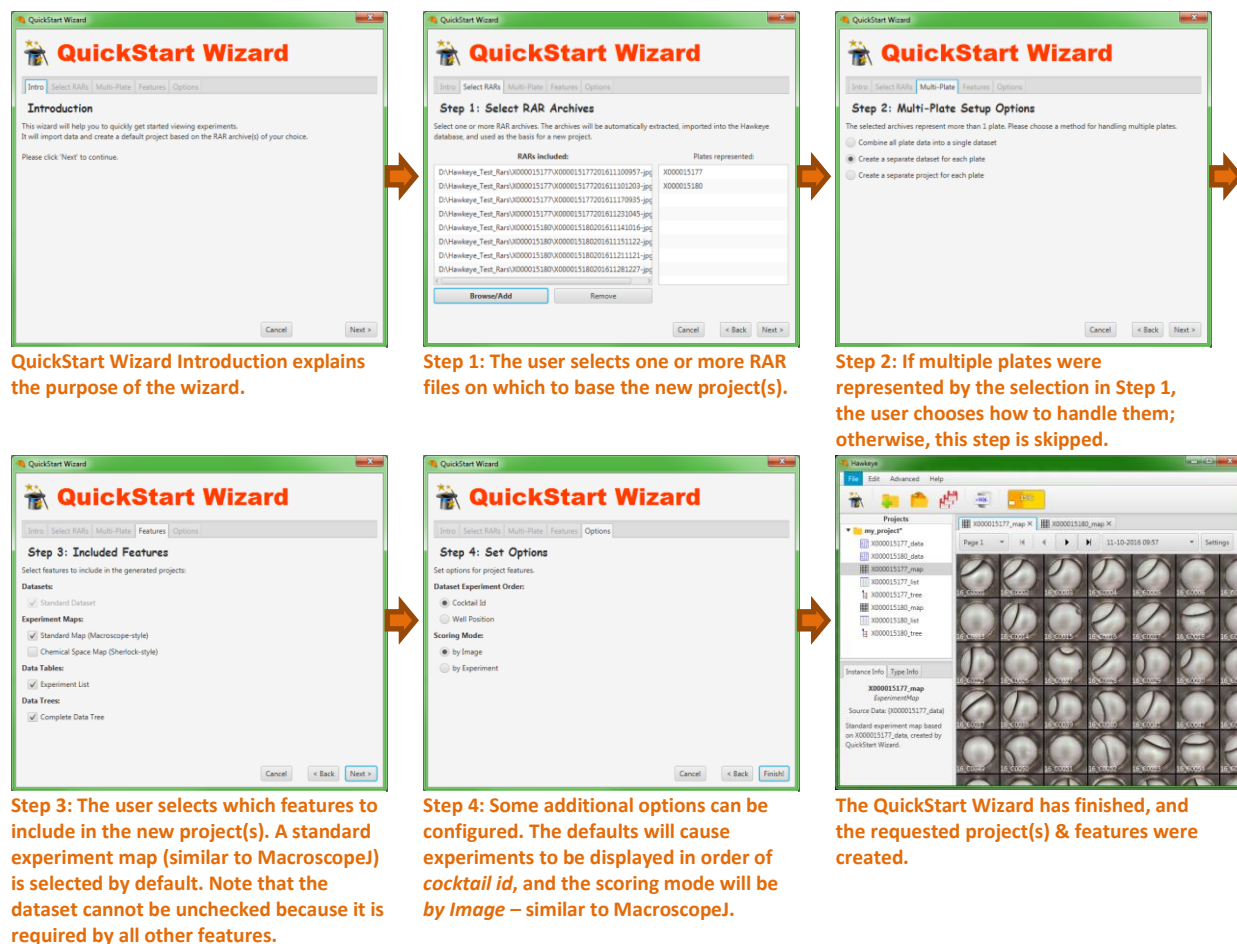
## Closing vs. Deleting Projects & Features

If you right-click on a project or feature in the Project Explorer, you will see options for both *Close* and *Delete*. It is very important to distinguish between them! The basic rule, regardless of context, is that *closing is temporary* and *deleting is permanent*. If you close (or hide) a feature, it will remain in the project and you can reopen it by double-clicking it in the Project Explorer. If you delete it, it will be gone from the project forever – unless the project is saved to disk and you reload it before saving it again. If you close a project, it will be removed from the Project Explorer, but it will still exist on your disk, as long as you have saved it. However, if you delete the project, it will be removed from both Hawkeye and your disk – you will not be able to open it ever again!

## Using the QuickStart Wizard

Projects and features can be created through a manual process, as discussed in this chapter. However, Hawkeye also comes with a convenient *QuickStart Wizard* to help users quickly create projects and datasets based on particular archives or plates. The wizard combines steps of importing RAR archives, creating projects, and adding standard features into a simplified, streamlined process. To use the QuickStart Wizard, select *QuickStart Wizard* in the *File* menu, or click its shortcut on the main tool bar.

Figure 7: Using the step-by-step QuickStart Wizard to create a new project.





## Chapter 3: Project Features

### Dataset

The *Dataset* feature represents a collection of related data elements that exist in the database. It encapsulates both an *SQL query* (question asked of the database) and the result set returned by that query. For example, a dataset could represent:

- all experiments in plate X000015177
- all cocktails that have produced crystals at least once
- all ingredients with a pH higher than 7.0
- all experiments containing “Ammonium Sulfate” that have resulted in crystals
- all RAR archives that have been imported

There are several ways to create a dataset. You can select *File → New Project Feature → New Dataset* to create a dataset by entering a query. You can also create a new dataset from a new or saved query in Database Terminal (see section *Database Terminal* in Chapter 6), or by using the QuickStart Wizard to generate a standard plate-based dataset (see section *QuickStart Wizard* in Chapter 2).

Datasets define a subset of the data to be used by other features in the same project. At the time of this writing, all other available features require connection to a dataset in order to function; additionally, some features (Experiment Maps & Data Trees), further require the dataset to contain an “experiment\_id” field. At the time each feature is created, the user must choose which dataset it will connect to, and this connection cannot be changed. There is no limit to how many features can connect to a single dataset.

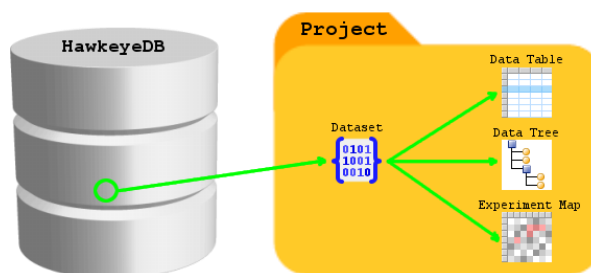


Figure 8: A Dataset defines a subset of data in the database to be used by other features in the same project.

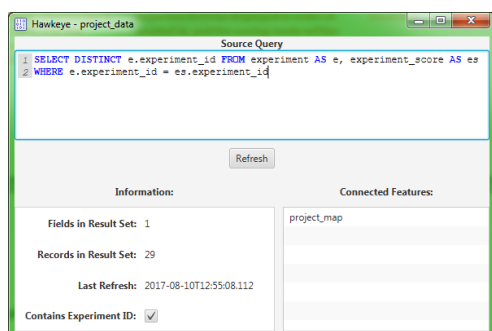


Figure 9: A Hawkeye Dataset, showing its source query, metadata about its current result set, and a list of connected features.

When the dataset is opened, you can examine (and edit) its source query and some basic information about the returned result set. There is also a *Refresh* button which will cause the query to be resubmitted to obtain a fresh result set. Refreshing a dataset will also cause all connected features to be refreshed with the new data. If the query text is edited, the change will not be saved unless the dataset is refreshed before its interface is closed.

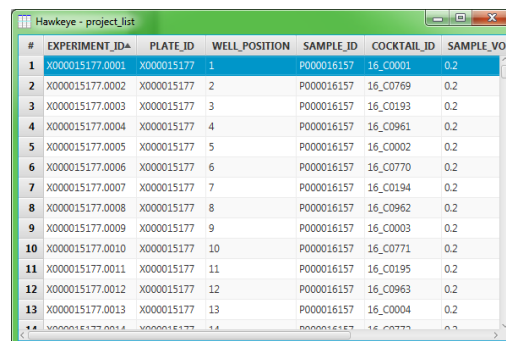
If a dataset has Experiment Maps or Data Trees connected to it, be careful when editing the source query. If the new query doesn’t return an “experiment\_id” field, those features will no longer be available.

For additional information on using SQL queries in Hawkeye, see *Chapter 6: HawkeyeDB*.

## Data Table

The *Data Table* feature displays the data from a dataset as a table, where each column represents a field and each row contains a record from the returned result set. The user can drag & drop column headers to rearrange the columns, or click a column header to sort the records in ascending or descending order of that column's values.

Data tables are useful if you want to include a listing of raw data in your project. One important characteristic of the data table is that, unlike a data tree or experiment map, it does NOT require its dataset to have an “experiment\_id” field; it can be used to display any data in the database—scores, images, ingredients, samples, archives, etc.—without explicitly relating it to experiments.



#	EXPERIMENT_ID	PLATE_ID	WELL_POSITION	SAMPLE_ID	COCKTAIL_ID	SAMPLE_VOLUME
1	X000015177.0001	X000015177	1	P000016157	16_C0001	0.2
2	X000015177.0002	X000015177	2	P000016157	16_C0769	0.2
3	X000015177.0003	X000015177	3	P000016157	16_C0193	0.2
4	X000015177.0004	X000015177	4	P000016157	16_C0961	0.2
5	X000015177.0005	X000015177	5	P000016157	16_C0002	0.2
6	X000015177.0006	X000015177	6	P000016157	16_C0770	0.2
7	X000015177.0007	X000015177	7	P000016157	16_C0194	0.2
8	X000015177.0008	X000015177	8	P000016157	16_C0962	0.2
9	X000015177.0009	X000015177	9	P000016157	16_C0003	0.2
10	X000015177.0010	X000015177	10	P000016157	16_C0771	0.2
11	X000015177.0011	X000015177	11	P000016157	16_C0195	0.2
12	X000015177.0012	X000015177	12	P000016157	16_C0963	0.2
13	X000015177.0013	X000015177	13	P000016157	16_C0004	0.2

Figure 10: A Data Table displays a dataset's data in columns (fields) and rows (records).

## Data Tree

The *Data Tree* feature displays the data in one or more experiments as a lazy-loading tree structure, allowing the user to dynamically “explore” all data related to an experiment. Nodes can be expanded by clicking the handle to the left of the icon, or by double-clicking anywhere on the node row.

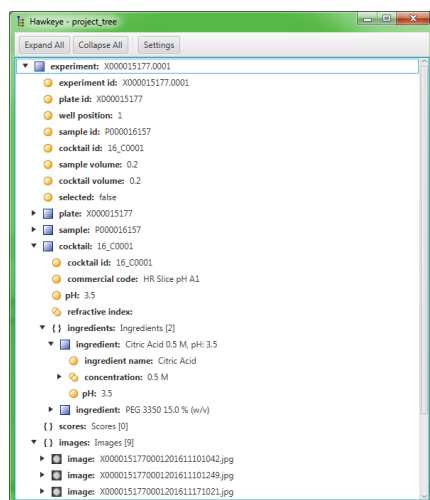








Figure 11: A Data Tree displays experiment data in a tree structure with expandable nodes.

There are several node types with unique icons for easy identification:

-  **Entity** – an object corresponding to a main table in the database.
-  **Primitive** – a primitive data type, such as an integer or text string.
-  **Compound** – an object consisting of multiple primitives.
-  **List** – a collection of indexed objects all having the same type.
-  **Image** – an object representing a read image.
-  **Score** – a user-applied score attached to an experiment or image.

By default, all node types display their field names and values, but not their datatypes; and nodes are not created for fields which have *null* (missing) values. The user can customize each tree's display by clicking the Settings button and selecting options under the tree tab.

There are buttons to Expand All closed nodes and Collapse All open nodes. While Collapse All can be used at any time, Expand All is disabled whenever there are multiple experiments in the dataset. (Expanding all children for a large number of experiments may result in sudden crashing of the JVM due to memory constraints.)

Note that, like experiment maps, data trees are *experiment-based* and require the connected dataset to contain an “experiment\_id” field in order to function.



## Experiment Map

The *Experiment Map* is Hawkeye's main feature for reviewing and interpreting experiment images.

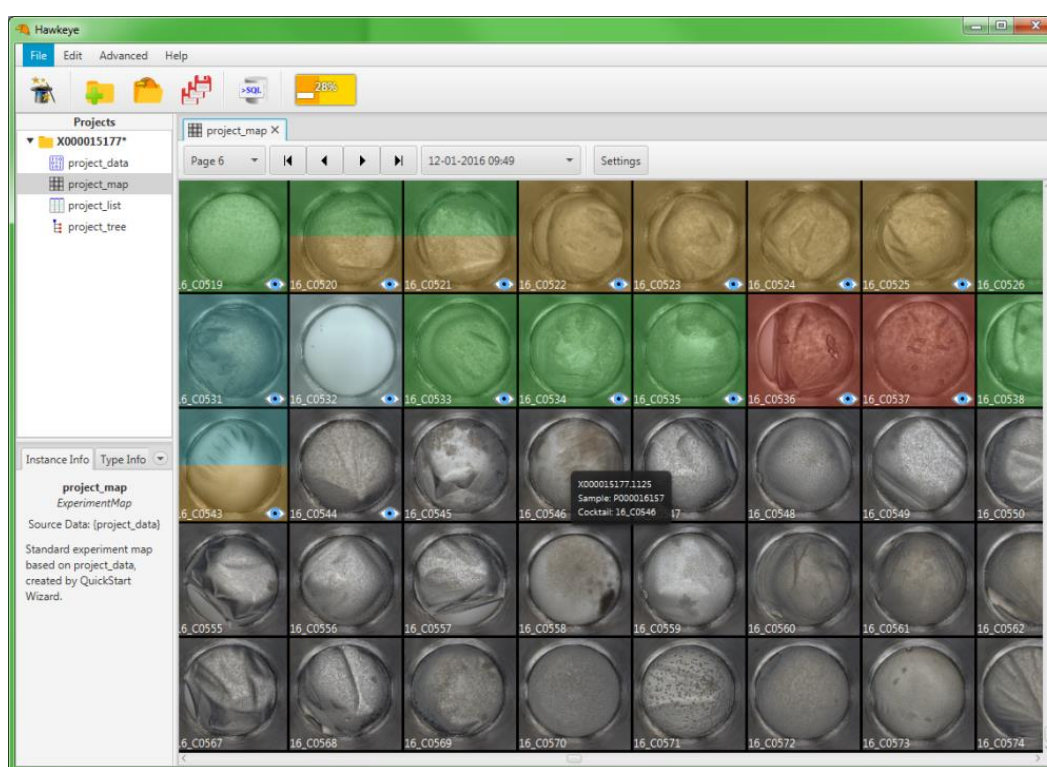


Figure 12: An Experiment Map created by the QuickStart Wizard. This map uses a *horizontal flow strategy* and orders experiments by cocktail. Some cells have a color-coded shading to indicate user-applied scores. An eye icon in the corner of the cell indicates that the image has been viewed.

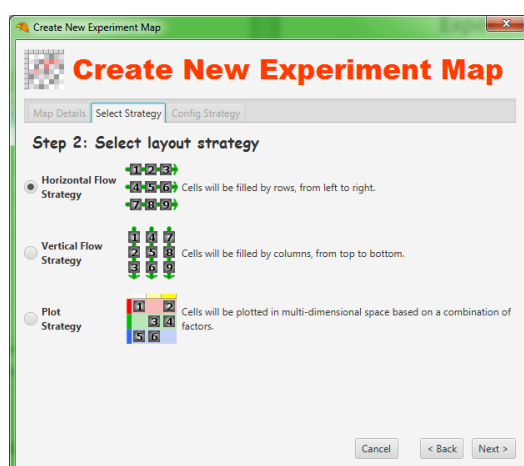


Figure 13: The Experiment Map's *layout strategy* is chosen and configured when it is created.



Cells in the map represent experiments in the connected dataset, arranged visually according to a *layout strategy*. The layout strategy is chosen by the user when the map is first created. There are 3 types:

- Horizontal Flow (default): cells will be arranged from left to right (rows) in the order of the dataset.
- Vertical Flow: cells will be arranged from top to bottom (columns) in the order of the dataset.
- Plot: a special multi-dimensional layout arranged using a *match query* that determines where each experiment fits. (This strategy is discussed in *Chapter 7: Experiment Map: Plot Strategy*.)

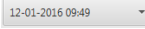
For many experiment maps, not all cells will fit on a single page. The number of pages in the map depends on how many experiments are in the dataset, and how many rows and columns the user specified when creating the map. For example, if the dataset contains 100 experiments, and the user chooses to have 5 rows and 5 columns, there will be 4 pages with 25 cells each.

By scrolling the mouse wheel over any cell, all cells in the map are resized. The cells contain scaled original images (not thumbnails); therefore the image quality will improve as the cells approach the actual image size. (Current standard images at the time of this writing are 768 x 768 pixels; UV and SHG images are 512 x 512 pixels.) On a high resolution screen (3840 x 2160), up to 10 full-sized images can be viewed simultaneously in the map. If the window is too small for all cells to be fully displayed at the current size, the map can be panned by dragging it with the mouse, or scrolled using the scroll bars.

There are two methods that can be used to change the current page:

1. Use the buttons  to navigate to the first, previous, next, or last page.
2. Use the page drop-down list  to select any particular page.

*(Note that displaying each page requires a few seconds for the cells to be created and their images to be loaded from disk. This required time is directly proportional to the number of cells in the page.)*

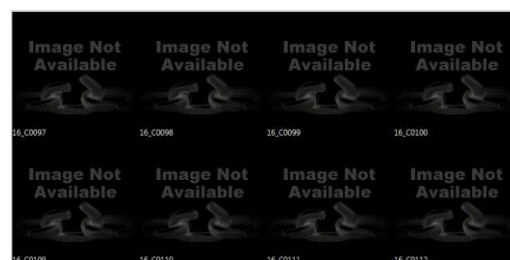
Each cell is a visual representation of an entire experiment, which may contain multiple read images of the experiment's well over time. However, only one image at a time can be displayed in the cell. To select an image index across ALL cells in the map, use the image read drop-down list.  If there is a single plate in the associated dataset, the drop-down will display each read as the date & time it was imaged (the same date & time encoded in the name of the RAR archive). However, if there are multiple plates in the dataset, their respective reads were not imaged at the same time; so in these cases the drop-down will display the read indexes generically as *Img #0*, *Img #1*, etc.

In the event that experiments (generally from different plates) on the same page contain differing numbers of images; and a read index is selected for which some experiments have an image and some do not: those experiments without the corresponding image will display a "Missing Archive" graphic. These graphics serve as placeholders so that the amount and positions of cells will remain consistent as the read index is changed. To remove the placeholder graphics, import the RAR archive that contains the data and images for that read.



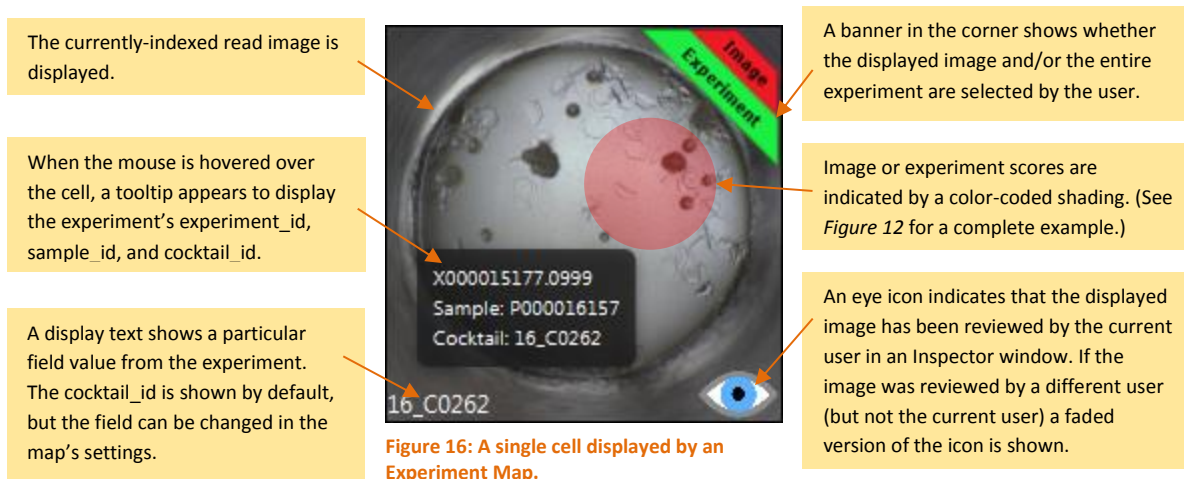
**Figure 14:** If some experiments in a page don't have an image for the current read index, a "Missing Archive" graphic is displayed instead. This allows those experiments to still be represented in the map even though a read image can't be displayed.

A similar case can occur if you choose not to use an image repository when first setting up Hawkeye. If no repository is configured, the image files are stored temporarily during the session in which the RAR archive(s) are extracted, but not permanently on disk for future use. If you import a RAR, restart Hawkeye, and then reference images from that RAR, the database will have data for them; however, the images will not be available unless the RAR is reloaded during the new session. An "Image Not Available" graphic will serve as a placeholder for the missing images until the required archives are reloaded.



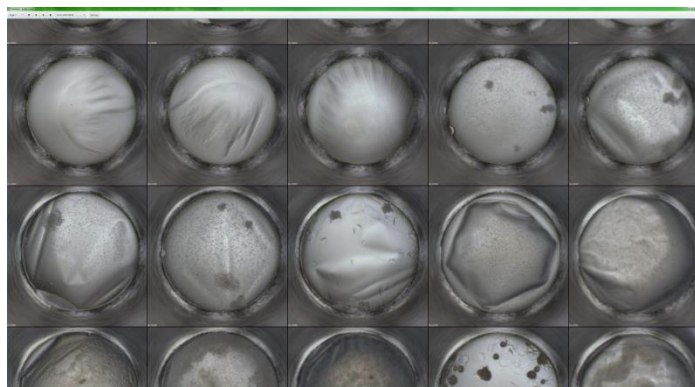
**Figure 15:** An "Image Not Available" graphic is shown if the data for an image is in the database, but the image file couldn't be found in the image repository.

There is an important difference between the two placeholders. If a cell says “Missing Archive”, the required archive has *never* been loaded at all; therefore both the image data and the image file are unavailable to the program, and the cell is completely *non-functional*. If the cell says “Image Not Available”, the required archive has been loaded before; therefore the image data exists in the database and is available, although the image graphic itself is not available for display. The cell remains *functional*: it displays previously-assigned scores and review status, and may be clicked to open an Inspector.



Left-clicking any cell (except those marked “Missing Archive”) will open an *Inspector* window for manual review of experiments/images. The Inspector is discussed in detail in *Chapter 4: Reviewing Experiments*.

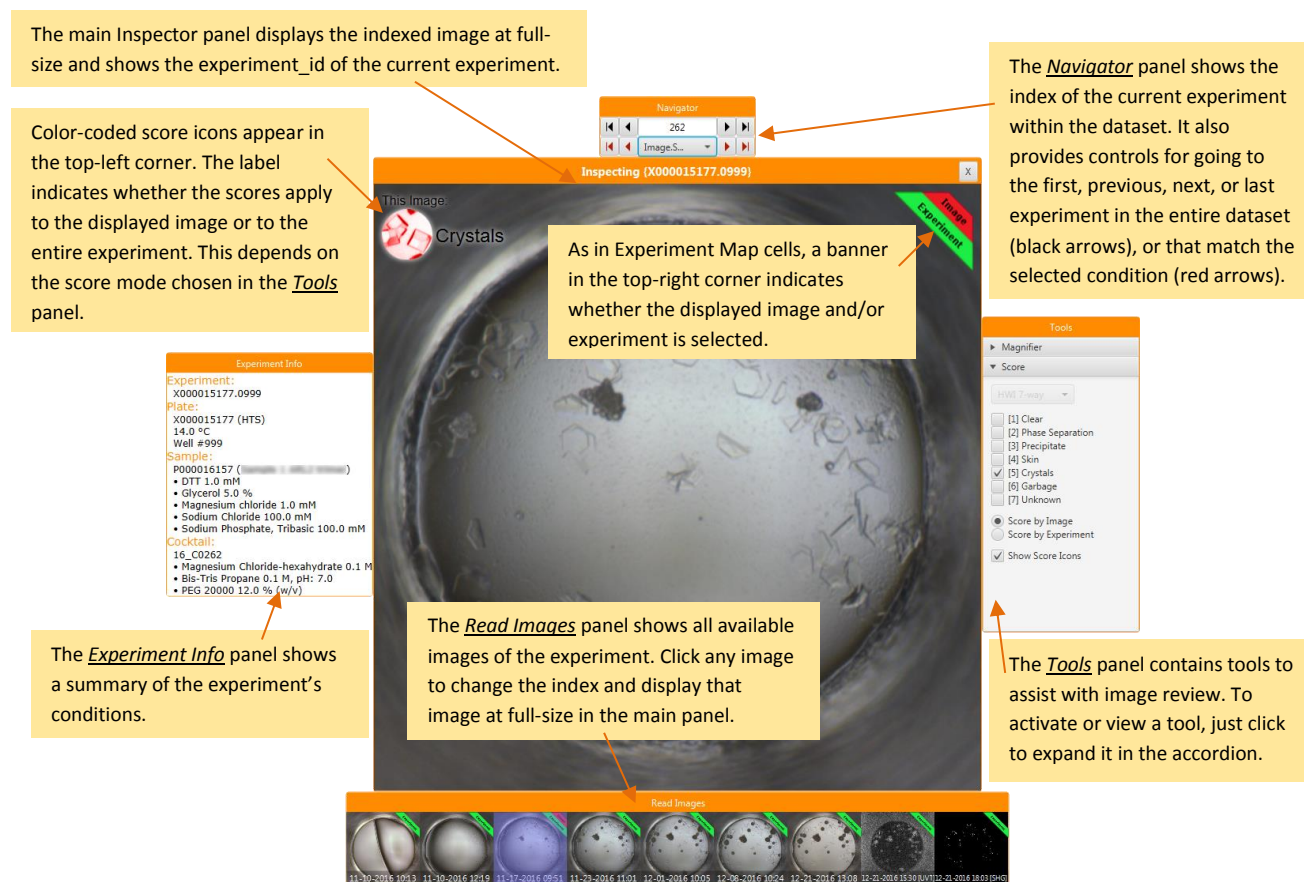
After reviewing the experiments/images via Inspector, the experiment map becomes much more useful. Applied scores and/or selections will be highlighted in the map, so the user (and collaborators who receive the scores) can see at a glance which experiments produced interesting results. The connected dataset can be modified to act as a filter for the map: for example, to show only experiments that contained at least one image scored as “crystal”; or (for flow strategies) to order them with selected experiments appearing first. With a plot strategy map, experiment cells can be arranged in multiple dimensions according to user-defined criteria: for example, to show where in “chemical space” crystal hits occurred. (For detailed info on using plot strategy, See *Chapter 7: Experiment Map: Plot Strategy*.)



## Chapter 4: Reviewing Experiments

### Inspector Window

Reviewing experiment images to classify their outcomes is Hawkeye's primary function, and it is the essential first step to getting the full intended benefit out of Hawkeye's other features. Users review images one experiment at a time using an *Inspector* window. An inspector opens whenever the user left-clicks a cell in an experiment map (unless the cell is marked "Missing Archive" – see *Experiment Map* section in Chapter 3). Multiple inspectors can be opened to compare different experiments side-by-side.



**Figure 18:** The Inspector has a main panel (center), displaying the currently indexed image, and 4 smaller accessory panels that contain information or utilities. The user can drag any accessory panel to arrange it in relation to the main panel; and when the main panel is dragged, the accessory panels will move with it. The accessory panels can also be docked to the main panel by placing them close to one of its edges; they will snap into place as if magnetic. If the main panel is closed ('X' button or 'Esc' key), the accessory panels will also close.

The inspector's main panel displays the indexed image of the current experiment at full-size, along with any scores or selections that have been applied. The *Navigator* panel shows the index of the current experiment within the associated dataset. (Note that this index does not necessarily correspond to an experiment\_id, cocktail\_id, well\_position, or any other field.) The *Navigator* contains controls (black arrow buttons) for navigating to the first, previous, next, and last experiments within the dataset, as well as a set of similar controls (red arrow buttons) for navigating to experiments within the dataset that match a selected condition, which is selectable from the drop down list. (When inspecting a map that uses a plot strategy, navigation follows the order of cells within the page—not the order of the dataset.)



If the current experiment contains multiple images, the image index can be changed by clicking one of the images in the Read Images panel. The indexed image (displayed at full-size) is shown with a blue highlight. Images in this panel also have a timestamp indicating the time of imaging, and may also include a code indicating a special type of image (UVT=*ultraviolet*; SHG=*second harmonic generation*).

The Experiment Info panel displays a summary of the experiment's properties, including the experiment\_id, plate information, sample solution ingredients, and cocktail solution ingredients. While it may seem redundant to display the same sample and plate info for every experiment in a dataset, this information can be very useful in cases where the dataset contains multiple samples and/or plates.

Finally, the Tools panel contains several tools to assist with reviewing images. At the time of this writing, there are four tools available: a Score Tool for viewing available scores and modifying score options a Magnifier Tool for zooming in parts of the image; a Line Measure Tool for measuring the length of crystals; and a Contrast Filter Tool for making slight details easier to examine.

The Inspector interface contains clickable controls for navigation and scoring; however, the more efficient alternative is to use these **keyboard controls**:

- to navigate to the previous or next experiment, use *left/right arrow keys*
- to advance to the next experiment, use either *Enter key* (same action as *right arrow key*)
- to change the image index, use *up/down arrow keys*
- to apply a score to the image or experiment, use *numpad keys\** as noted by the Score Tool
- to toggle selection on the image or experiment, use the *space bar*
- to close the Inspector, press *Esc key*

\*To use the numpad keys, *NumLock* must be turned ON. Otherwise the keys will be interpreted as arrow keys and will result in navigating to another experiment or image index.

## Review Status

As you use the keyboard or mouse controls to advance through experiments, every image that appears full-size in the inspector's main panel will be flagged as having been reviewed by the user. Review status is saved immediately in the database on a *per user* basis: there is a record in the "Image\_Review" table for each unique combination of an image + the user who viewed it.

When a reviewed image is shown in an experiment map, its cell displays an "eye" icon in the bottom-right corner. Cells will not display the icon unless the shown (indexed) image has been reviewed—even if the contained experiment has other images that were reviewed. If the image has been reviewed by the current user, the eye icon is shown at full opacity; if it was *only* reviewed by one or more other users (not the current user), the icon will be partially transparent until reviewed by the current user.

Review status is most useful for resuming work after a break or interruption, to avoid reviewing the same images twice and to ensure that no images are accidentally overlooked. It could also be used to obtain a list of how many users viewed each image, using a query such as the following:

```
SELECT image_filename, COUNT(image_filename) AS num_users FROM image_review
GROUP BY image_filename ORDER BY num_users DESC
```

## Scoring

To this day, a manual human review process remains the most accurate and reliable method of using experiment images to detect crystals and other outcomes. Assigned outcome categories, or *scores*, allow the outcomes to be recorded, shared, and analyzed in pursuit of three goals:

- to discover a set of conditions that is effective for crystallizing a given protein
- to measure the performance of particular cocktails against many proteins
- to evaluate (and improve) the overall effectiveness of the crystallization screen



Figure 19: Color-coded icons representing the 7 categories in HWI's classic 7-way scoring scheme. From left to right: Clear, Phase Separation, Precipitate, Skin, Crystals, Garbage, Unknown. The colors of the icons correspond to the colored shading displayed in the Experiment Map cells.

Hawkeye was designed with the possibility of using either (and potentially, both) of two scoring modes: *scoring by image* and *scoring by experiment*. Scoring by image allows the user to assign one or more scores to particular images of an experiment; this is the “classic” image-centric mode employed by all of our previous programs. However, Hawkeye’s experiment-centric design also allows for the alternate possibility of evaluating *all* images of an experiment together, and assigning one or more scores to the experiment itself.

The active scoring mode can be toggled using the Score Tool in the inspector’s Tools panel. Scoring mode determines not only whether new scores will be applied to the image or experiment, but also whether image or experiment scores are displayed in the inspector. Only image *OR* experiment scores can be seen simultaneously in an inspector or experiment map—not both. (The scoring mode also determines whether *selection* will be applied to the image or experiment; both selections are always displayed in the inspector and the experiment map. See the next section, *Selection*, for more info about selections.)

Multiple inspectors can be open at the same time, and it is possible for each one to be set to a different scoring mode. If more than one inspector is inspecting the same experiment map, the map’s score visibility mode will be set to whichever mode was set latest on any of the inspectors.

To assign a new score to the current experiment or indexed image, check the box for the score in the Score Tool, or press the numpad shortcut key for the score (key numbers for each score are listed in the Score Tool). Multiple scores can be assigned to each image or experiment. For example, depending on analysis goals, an image containing a crystal and precipitate may be scored as “crystals” or as “crystals” & “precipitate”. The inspector will display an icon for all assigned scores. In the experiment map, the score shading will be divided evenly in colors representing all assigned scores; e.g., “crystals” and “precipitate” will show up as half-red and half-green.

Should the score icons hide an important part of the image, they can be hidden by unchecking the “Show Score Icons” box in the Score Tool.

## Selection

*Selection* is a way to mark certain images or experiments as noteworthy or interesting. To toggle selection on an image or experiment, press the *space bar* when it is displayed in an active inspector window. Images always display whether they and/or their containing experiment are selected: a red banner in the top-right corner indicates that the shown image is selected; a green banner indicates that its experiment is selected.



Figure 20: The indexed image and/or the entire experiment can be selected. Both are always shown as banners in the corner of the image, both in the Inspector and in the Experiment Map cells.

Although selection is stored in the database and can be queried, it is intended for temporary use. The current database-wide selections can be easily cleared by choosing *Clear Image Selection* or *Clear Experiment Selection* from the *Edit* menu in Hawkeye's main interface window. When the user chooses to exit Hawkeye, a checkbox in the exit dialog will offer to retain your selections for the next session; the box is unchecked by default: so unless you change it, both image and experiment selections will be cleared whenever you exit the program. On the other hand, should the program or computer suddenly crash, Hawkeye will not exit normally and the selections will still be available when it is restarted.

Selection has several uses. It can be used as a simple binary alternative to scoring, or to mark experiments for closer examination later on. Selection cannot be saved into a project or re-opened later in Hawkeye; however, it can be exported as an HTML-based *Selection Report*, or as an *MSO file* for use by legacy applications (see *Chapter 5: Exporting Data*).

## Additional Tools

The *Magnifier Tool* can assist the user with viewing tiny details of the image (especially on small or hi-res displays) by scaling the pixels of a particular region. To use the tool, open it in the *Tools* panel and move the mouse over the desired region of the image; a magnifying glass simulation will appear, showing both the enlarged region of the image and the scaled rendering. Controls allow for changing both the "magnification" (pixel size multiplier) and the "glass size" (width of the magnification region in pixels). You can choose whether to show a sharp, "pixelated" rendering, or to apply a smoothing algorithm. The actual size of the magnification rendering will be  $[magnification] \times [glass\ size]$ . While over the image, the magnification level can be adjusted by left-/right-clicking, and the glass size by scrolling the mouse wheel.

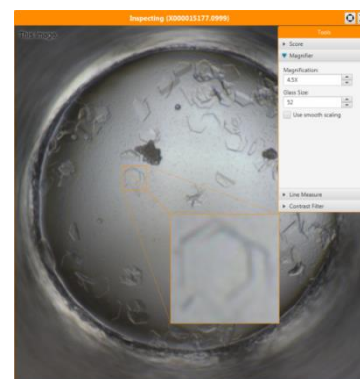


Figure 21: Examining hexagonal crystals using the Magnifier Tool.

The **Line Measure Tool** enables the user to measure crystals or other objects in the image in pixels and/or microns ( $\mu$ ). To use the tool, open it in the **Tools** panel and move the mouse to the object you want to measure; a magnifier simulation will appear, allowing the user to precisely select the pixels for the endpoints of the line. Click and drag the mouse to select the distance to measure. By default, the tool can only measure in pixels. To obtain measurements in microns, you must first “calibrate” the tool by measuring the 900 $\mu$  well bottom in pixels: check the **Measure Well Diameter** box, and use the tool to measure the diameter of the well bottom (the clearest visible circle in the image). This will establish a microns-to-pixel ratio, and automatically convert future measurements on this image to microns. (This process must be repeated if a different image is viewed.)

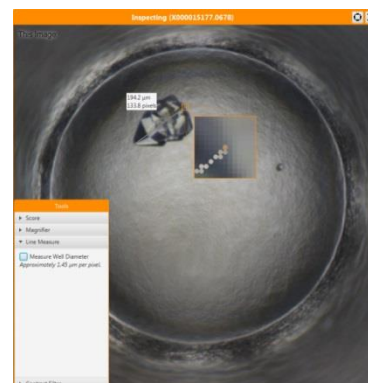


Figure 22: Measuring a crystal using the Line Measure Tool.

The **Contrast Filter Tool** can be of use when examining low-contrast image regions; for example, when studying a barely visible “ghostlike” crystal, or highlighting low-saturation colors that may indicate trace amounts of colored protein. The filter works by exponentiating pixel values (either brightness, or separated RGB values, depending on the chosen mode) to the selected intensity power; a down/up threshold separates values that will be shifted up (brighter) from those that will be shifted down (darker). Check the **Show Filter** box to show the filter; once shown, it can be repositioned by dragging it with the mouse. While the box is checked, the filter remains active on the image and can be repositioned even when its tool panel is closed. Here are some tips for using the **Contrast Filter Tool**:

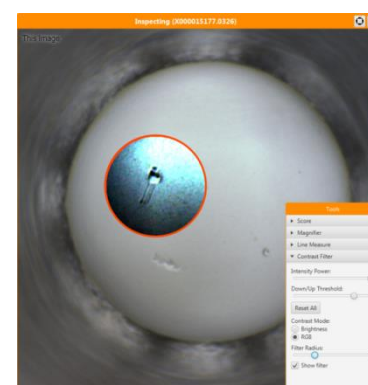


Figure 23: Examining a crystal using the Contrast Filter Tool.

- Raise the intensity power to make the contrast stronger.
- If the filtered image is washed out, raise the threshold; if it needs more light, lower the threshold.
- To preserve the saturation of the colors in the original image, use the Brightness mode; to change the intensity of individual red, blue, and green values of each color, use the RGB mode.
- If the filter responds too slowly, use a smaller radius.
- The contrast filter shows some features at the expense of others; to examine different features within the same region, you may have to adjust the settings.

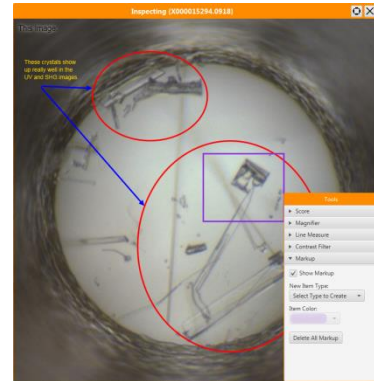
The **Markup Tool** can be used to make semi-permanent notes or drawings over a particular image. Markup items are stored in the database and shown whenever the image is displayed. They can also be transferred between copies of Hawkeye by including them in a *score file*.

At the time of this writing, there are four types of markup items: arrows, boxes, ellipses, and notes—all of which are created and edited in a similar fashion.



To create items using the Markup Tool:

- open it in the Tools panel
- make sure the “Show Markup” box is checked
- select the type of item to create from the dropdown list
- choose a color for the item
- left-click on the image and drag to create the item
- the new item is selected by default
- the dropdown list resets after each use;  
to create a new item, select it from the list again



**Figure 24: Using the Markup Tool to make notes and drawings over an experiment image.**

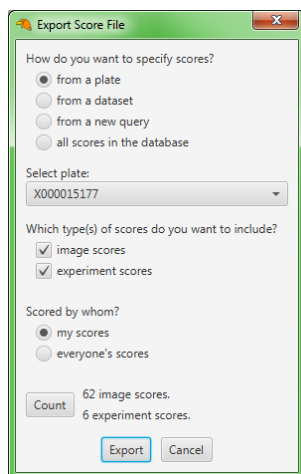
To edit a markup item:

- make sure the Markup Tool is open in the Tools panel
- make sure the “Show Markup” box is checked
- left-click a visible part of the item to select it; white handles will appear around the item
- to move the item, left-click a visible part of the item and drag it with the mouse
- to resize or reshape the item, left-click and drag one of the handles
- to recolor the item, select it and then choose a new color
- to edit the text of a Note item, select it and then click inside the box
- to delete the item, select it and press the *Delete* key
- to delete all items on the current image, click the “Delete All Markup” button

*(Additional tools may be added in future releases of Hawkeye.)*

## Chapter 5: Exporting Data

### Exporting Score Files



**Figure 25: The Export Score File dialog allows exported scores to be specified in a number of ways. Click the “Count” button to check the number of image & experiment scores that will be included with the chosen settings.**

To send scores from one copy of Hawkeye to another, you will need to export a *score file*. A score file is an XML file with a “.hks” extension, containing all data necessary to establish:

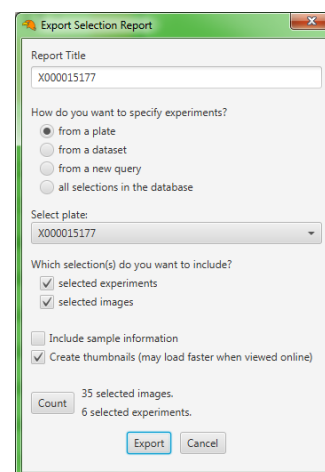
- which images and/or experiments were scored
- which score profiles were used to score the included items
- the attributes of each score profile (name, scheme, color, key, etc.)
- user details for the person(s) who did the scoring

There are two ways to export a score file. The first is to use File → Export Score File to export a standalone score file. As shown in *Figure 22*, there are several options and filters that can be used to specify which scores will be included in the file. If you choose one of the first three options, you will supply a plate, dataset, or query to make the selection. You can also choose whether to export image scores, experiment scores, or both. Finally, you can choose to include only your own scores, or the scores of all users. Note that, when using this dialog, you don’t need to have an open project unless you choose the “from a dataset” option.

The second way to export a score file is to embed it in a project: select File → Save Project + Scores. This automatically includes all experiment or image scores found in the project’s datasets, and adds the option of importing them when the project is opened (see section *Opening Existing Projects* in *Chapter 2: Working with Projects*). Note that embedding a score file is not a permanent change to the project; if the project is reopened, and then saved again without the embedded scores option, it will no longer contain the score file.

### Exporting Selection Reports (HTML)

Selections in Hawkeye can be exported as an HTML Selection Report. This is a standalone package that allows a simplified viewing of selected experiments and images, and the HTML format makes it especially useful for viewing online as part of a website. The report can be created with a dialog under File → Export Selection Report (HTML), which is very similar to the *export scores dialog* discussed in the previous section. Note that, since the report is meant for standalone viewing, a report title can be added. Two special options are also available: you can choose to include or omit sample information from the report; and you can choose whether to create extra thumbnail graphics which may help the main report page to load more quickly when viewed online.



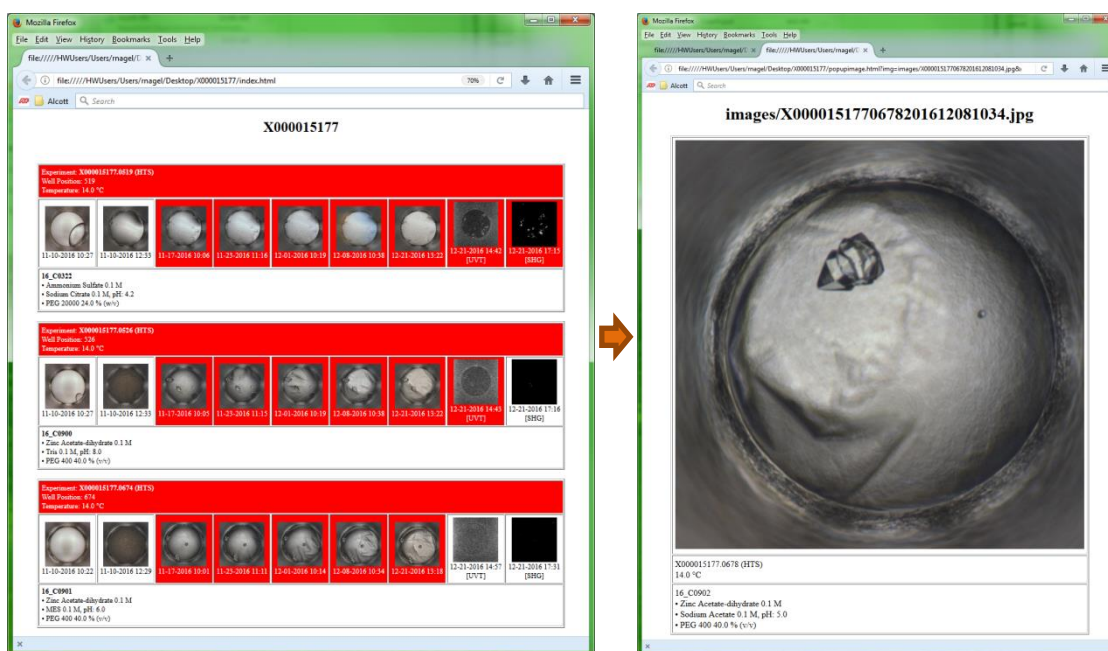
**Figure 26: The Export Selection Report dialog.**



**Figure 27: Folder contents of an HTML Selection Report. Open index.html to view the report in your web browser.**

The report package is saved as a folder containing several files, as shown in *Figure 24*. The main file that must be opened to view the report is “index.html”. The other files are resources that will be accessed by the index file to display the report and provide its functionality. The “thumbs” folder will only exist if the user chose the “Create thumbnails” option when generating the report.

The report is displayed as a list of experiments, each with all of its available images. The selected experiments and images are shown in red. If only a single image from an experiment was selected, both the experiment and all images will be listed; however, only the selected image will be shown in red. Left-clicking any image in the report will open it (along with associated experiment data) at full-size in a new tab or window. Click the image to close the new tab/window.



**Figure 28: A Selection Report as viewed in Mozilla Firefox (left). Left-clicking any image opens it in a new tab (right).**

## Exporting MSO Files

Since the days of our first review program, the original *MacroScope* (2000), the HTSLab has been using a tab-delimited, unstructured (“flat”) text file called an MSO (**MacroScope Output**) file for recording image selections and scores. The format was also adopted by our last review program, *MacroScopeJ*, and has additionally been used over the years as an input to other programs and utilities, such as *AutoSherlock*, *CheckShow*, *CrossPlate*, and others.

For backward compatibility with legacy programs, Hawkeye also allows exporting data as an MSO file. This can be done by selecting *File* → *Export MSO File*. The user chooses an archive (image read) on which to base the file, and specifies whether to include the current selection, his/her own image scores, or everyone’s image scores. Note that an MSO file cannot be imported or reopened in Hawkeye.

# **Part III:**

# **Advanced Usage**

## Chapter 6: HawkeyeDB

### Embedded Derby Database

Hawkeye's power and versatility stem from reliance on its embedded relational database, HawkeyeDB. The embedded database:

- stores all data permanently, making it available for all subsequent analysis without requiring it to remain in memory
- isolates individual data elements and enforces proper type formatting
- creates relationships between data entities (tables), allowing for potentially advanced queries to be asked by both the application and user

HawkeyeDB is created in *Apache Derby*, an open-source relational database management system (RDBMS) developed by the *Apache Software Foundation™* and designed specifically to be embedded in Java applications. Derby implements an SQL-92 core subset, as well as some SQL-99 features. For reference, both a *Derby Reference Manual* and a *Derby Tools and Utilities Guide* are included in the Hawkeye distribution under the *hawkeye/doc/* folder.

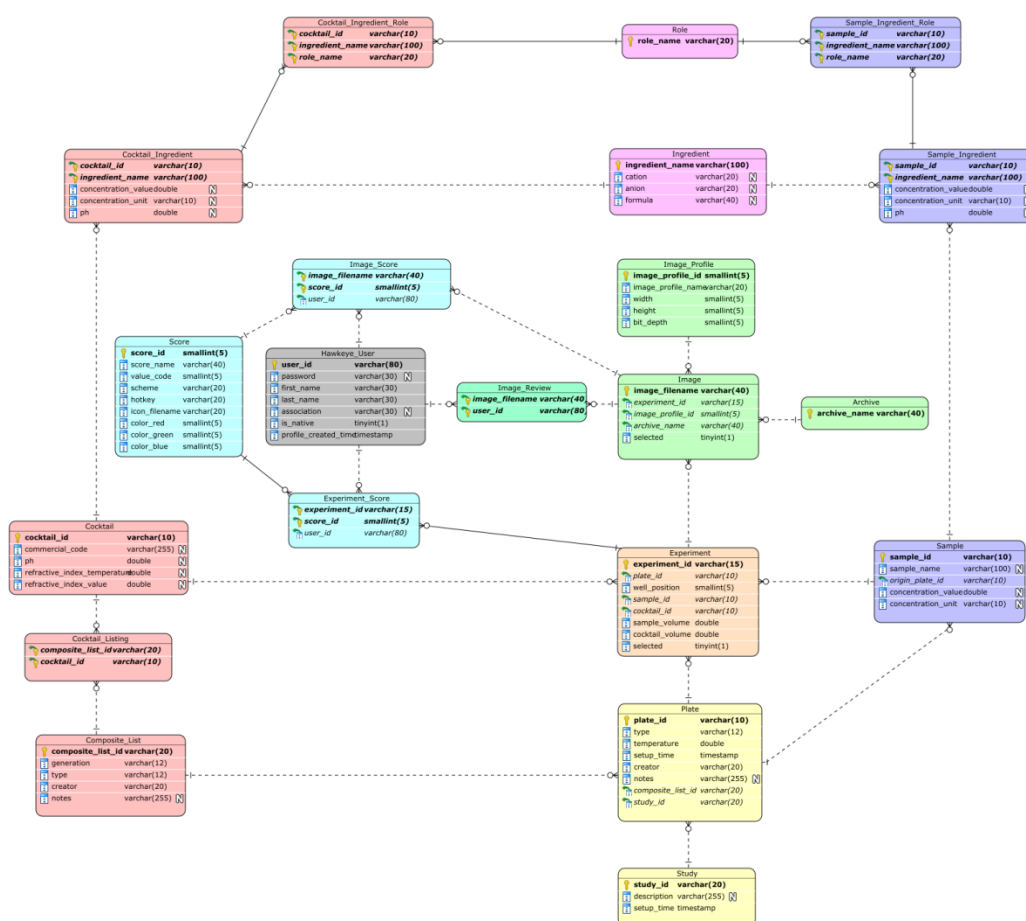


Figure 29: Complete schema of *HawkeyeDB*, Hawkeye's embedded Derby database. Pay particular attention to crow's feet ( $\llcorner$ ) which indicate "many" vs. single lines ( $|$ ) which indicate "one". For example, the relationship between the "Plate" table and "Experiment" table is such that one plate can have many experiments. Similarly, one experiment can have many images. A vector graphic (scalable) PDF version of this ER diagram can be found in your *hawkeye/doc/* directory.

## User Access & Profile Usage

While running, Hawkeye maintains two connections to the database, as shown below:

Connection:	Application	User
Connects as:	“hawkeye”	“human”
Access:	Full access to all tables and views	No access to Hawkeye_User table; read-only access to all other tables and views
Purpose:	Used by the application for controlled use of standard procedures	Used by the logged in user to execute custom queries

The restrictions placed on the “human” account allow users to experiment with writing queries with no danger of accidentally deleting data or damaging the database.

Note that all human users are always connected to the database as “human”, not with the login name and password created from the Splash/Login dialog. Each user logs in to authenticate against a valid user profile stored in the Hawkeye\_User table; this profile information (first name, last name, association) is then attached to all scores assigned by the current user. When scores are sent to another copy of Hawkeye via a score file (see *Chapter 5: Exporting Data*), the user information is sent along with it and is added to the database in that copy of the software. Thus users can always view basic information about who provided the scores, even if the scores came from another copy of Hawkeye.

Because user information can be transferred from one database to another, the uniqueness of user IDs must be guaranteed across all copies of Hawkeye. The user ID is constructed automatically using first initial, last name, association, and a long integer representing the time of profile creation in milliseconds: e.g. `rnagel_hwi_466867011716601`. Since the odds of two similarly-named individuals at the same company creating profiles in the exact same millisecond are virtually impossible, it is assumed that each user ID will remain unique across all copies of Hawkeye.

## Using SQL to Write Queries

If you have already tried opening a *dataset* feature generated by the *QuickStart Wizard* mentioned in *Chapter 2*, you have probably seen something like this:

```
SELECT * FROM experiment WHERE plate_id='X000015177' ORDER BY cocktail_id
```

This is an *SQL* (**Structured Query Language**) statement that retrieves basic data for all experiments in a plate. The *QuickStart Wizard* generates this code automatically so that basic users can forgo having knowledge of SQL and relational databases; however, more advanced querying requires users to have a working knowledge of both. Fortunately, SQL is very English-like and easy-to-learn, even for non-programmers. However, an SQL tutorial is beyond the scope of this user manual. For a straightforward and comprehensive way to learn SQL, see the tutorial located at <https://www.w3schools.com/sql/>. Once you have a rudimentary understanding of SQL and the HawkeyeDB structure, you can gain familiarity with how to write Hawkeye queries by examining, running, modifying, and experimenting with pre-written queries using the *Database Terminal* (see the section *Database Terminal* in this chapter).

## Database Organization

Each table in the database represents an *entity*, and contains various *fields* (or attributes) describing that entity. In keeping with Hawkeye’s experiment-centric design, the heart of HawkeyeDB is the Experiment table (shown orange in *Figures 26 & 27*). Each record in this table represents a single *experiment*—essentially, a single protein sample combined with a single cocktail in one well of a particular plate. Each experiment is associated with several images which were taken periodically to record its activity. Finally, the information about an experiment is made complete when a human user examines it and assigns it one or more score(s). Let’s take a look at how the database structure represents this:

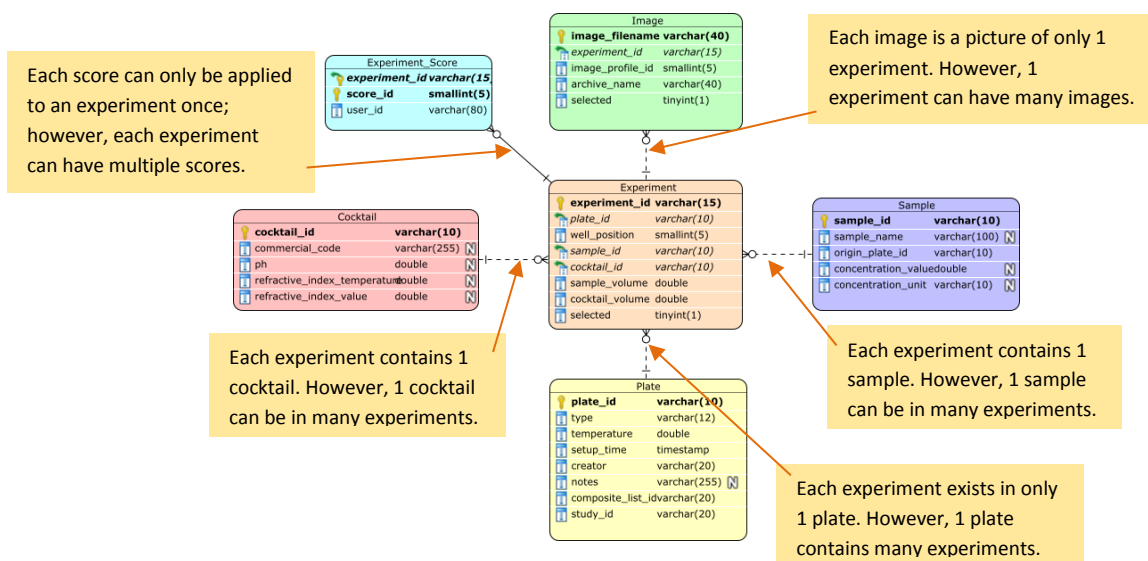


Figure 30: Relationships connecting the Experiment entity (table) to other entities in the database. Each experiment is (or can be) associated with multiple images and scores, but only 1 plate, 1 sample, and 1 cocktail. These relationships, as well as field values, can be easily explored in Hawkeye using the *Data Tree* feature.

Each of these relationships is based on sharing one or more fields between tables. For example, both the Cocktail table and Experiment table have a field called “cocktail\_id”. Using this common field, the two tables can be *joined* together in a query so that we can use data in the Experiment table to retrieve data from the Cocktail table. To get the pH of the cocktail for experiment “X000015177.0001”, we could use the following SQL query:

```
SELECT cocktail.ph
FROM experiment INNER JOIN cocktail ON experiment.cocktail_id = cocktail.cocktail_id
WHERE experiment.experiment_id = 'X000015177.0001'
```

Here we are using the experiment to search for data (pH) about a related entity (cocktail). Since many Hawkeye datasets require an “experiment\_id” field, we will sometimes need to do the opposite: search for experiment(s) using some data from a related entity. For example, the following query returns all experiments having a cocktail with a pH of 7.0:

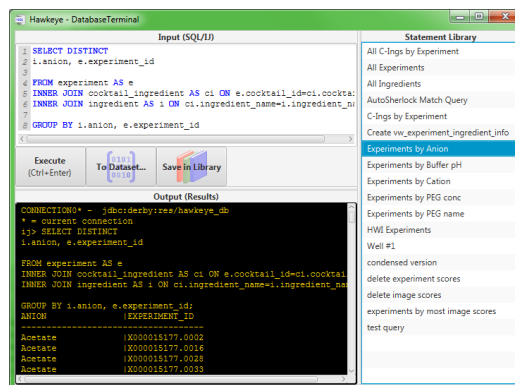
```
SELECT experiment.experiment_id
FROM cocktail INNER JOIN experiment ON cocktail.cocktail_id = experiment.cocktail_id
WHERE cocktail.ph = 7.0
```



## Database Terminal

Most custom queries should be turned into *datasets* (see section *Dataset* in *Chapter 3: Project Features*) so that other features can be connected to them. When a new dataset is created using the *File → New Project Feature → New Dataset* command, the user must enter a query to retrieve data for the dataset. For general query writing and testing, however, Hawkeye contains a tool called *Database Terminal*. The terminal consists of *Input area*, *Output area*, and *Statement Library*.

After typing or inserting a query, click the *Execute* button (or press *Ctrl + Enter*) to run it. Note that queries with a lot of results may take a while to complete. When the query has finished running, any results or errors will be displayed in the Output area. There are also options to copy your query to a new dataset, or to save it in the Statement Library for future use. Before using either of these options, it is highly recommended that you test the query “as is” to make sure that it actually works.



**Figure 31: A Database Terminal window. SQL queries can be executed, made into datasets, or saved to the Statement Library for later use.**

To copy your query to a new dataset, click the *To Dataset* button. (You must have an open project selected, or Hawkeye won’t be able to create the new dataset.) The *Create New Dataset* dialog will be displayed, as though you used the *New Dataset* command from the *File* menu; however, upon reaching the next step of the wizard, you will find that the source query has already been filled in.

To save your query in the Statement Library, click the *Save to Library* button and enter a name for it. It can’t have the same name as an existing statement. To load a query from the library into the Input window, simply left-click on it.

The user can also right-click any section to view a context menu. Right-click the Input area to see a list of recently executed queries which can be loaded. Right-click a statement in the Statement Library to display a menu containing *Rename* and *Delete* options for that item.

## Resetting the Database

If for some reason you need to reset the Hawkeye database back to its original, empty state, you can do so from the Splash/Login dialog that appears when you start Hawkeye. Click the *More Options* button, and then the *Reset Database* button. This action will actually delete the entire database from the disk and then recreate it using the script at *hawkeye/res/sql/create\_hawkeyedb.sql*.

Since user data is stored in the database, all users will also be deleted and will need to be recreated. If a deleted user, “John Doe”, had saved his scores in a score file or project, they can be imported into the database again, provided that the scored experiment/image data is first re-imported from archive(s). However, if a new user named “John Doe” is created, this new user’s ID will be different from the original John Doe’s ID; the database will see them as two different users.



## Extending the Database

At some point, advanced users may find that certain queries are inconvenient, difficult, or even impossible to achieve using only the standard tables that make up HawkeyeDB.

For example, this query returns all available data about each ingredient in an experiment's cocktail:

```
SELECT DISTINCT
e.experiment_id, ci.ingredient_name, i.formula, ci.concentration_value,
ci.concentration_unit, ci.ph, i.cation, i.anion, cir.role_name

FROM experiment AS e
INNER JOIN cocktail_ingredient AS ci
ON e.cocktail_id = ci.cocktail_id
INNER JOIN ingredient AS i
ON ci.ingredient_name = i.ingredient_name
INNER JOIN cocktail_ingredient_role AS cir
ON ci.cocktail_id = cir.cocktail_id AND ci.ingredient_name = cir.ingredient_name

WHERE e.experiment_id = 'X000015177.1000'
```

It's a bit tedious to perform all the necessary joins, especially for an operation as common as retrieving cocktail ingredients for an experiment. A better way is to make this query into an SQL *view*: a virtual table based on the result set of an SQL statement; then the same results can be returned using a much simpler command:

```
SELECT * FROM vw_experiment_ingredient_info
WHERE e.experiment_id = 'X000015177.1000'
```

A view is created using the `CREATE VIEW` statement; however, as discussed in the *User Access* section of this chapter, human users have read-only permissions on the database, so this statement cannot be executed from within Hawkeye. To extend HawkeyeDB with user-defined views, the user can modify the database creation script file. This file, located at `hawkeye/res/sql/create_hawkeyedb.sql`, contains a series of SQL statements that are run whenever Hawkeye is started. Simply open the file in a text editor program, and write or paste your `CREATE VIEW` statement at the end of the file. Like other tables and views in the file, your view will be created on startup if it doesn't exist already. If the database is reset, all tables and views in the file will be recreated on startup. *Be careful not to modify or delete any other contents of the file, or your database may not function properly when reset.*

You may notice that the file already contains a view encapsulating the query shown above:

`vw_experiment_ingredient_info`. This view is included in HawkeyeDB by default, and is required by the *QuickStart Wizard* to create Sherlock-style chemical space maps. Views are especially useful for creating match queries for plot strategies, because these queries often require an assortment of fields from different parts of the database to be returned in a single record—sometimes more than once! See *Chapter 7: Experiment Map: Plot Strategy* for more information about plot strategies and writing match queries.

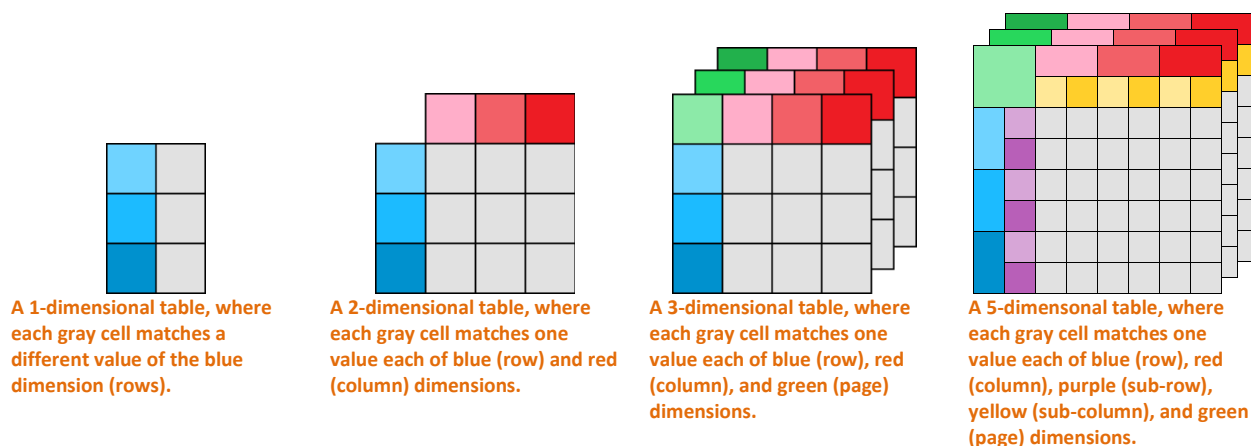
## Chapter 7: Experiment Map: Plot Strategy

### What is a Plot Strategy?

The *Experiment Map* feature was introduced in *Chapter 3* as a way of visually arranging the experiments represented in a dataset. Experiments are displayed in cells arranged according to one of three *layout strategies*. Two such strategies, the *horizontal flow* and *vertical flow*, are relatively simple to set up and use. The third and most complex, *plot strategy*, will be explained in this chapter.

For horizontal and vertical flow strategies, a single query (in the connected dataset) determines the set of experiments to be arranged and the order in which they will appear. For the plot strategy, one such query determines the set of experiments to be arranged; however, there is no sequential ordering. Instead, as the name suggests, experiments are “plotted” in multi-dimensional space based on where the values of specified fields fall within the range of available values. For example, the *QuickStart Wizard* includes an option to automatically create a pre-configured chemical space map (similar to our *AutoSherlock* software) with 5 dimensions: *cation*, *anion*, *PEG*, *PEG concentration*, and *buffer pH*.

Figure 32: Representations of 1D, 2D, 3D, and 5D space. Each dimension contains a range of available values, shown here as different shades of the same color. Each gray cell represents a unique intersection of available values from each dimension. Hawkeye Experiment Maps can be created with any of these designs using the Plot Strategy.



### Configuring a Plot Strategy

To create an experiment map with a plot layout, open *File → New Project Feature → New Experiment Map*, and select *Plot Strategy* in Step 2 of the wizard. (Reminder: to create the experiment map, there must be an existing dataset feature, containing an “*experiment\_id*” field, in the same project: this acts as a filter to specify which experiments will be included in the map.)

When *Plot Strategy* is chosen, it must be configured to set up the desired dimensional space. There are no default options. The user must define each *dimension* to be used, as well as a *match query* that will determine whether an experiment matches each possible combination of dimension values. It does not matter whether the dimensions or match query are entered first.

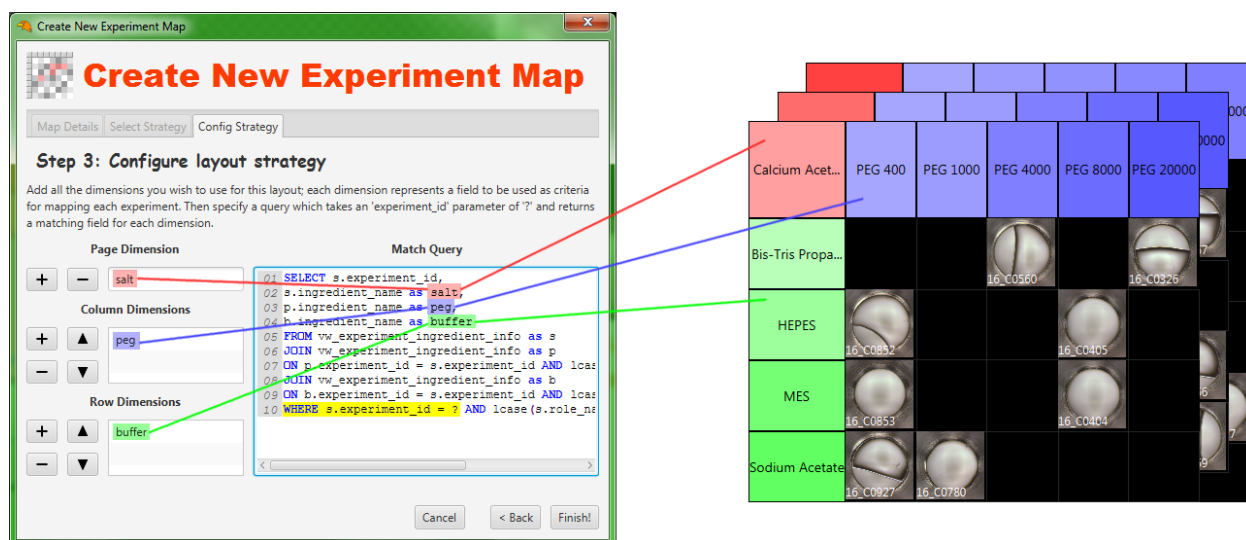


Figure 33: Configuration of a 3D plot strategy arranging experiments by combination of salt, PEG, and buffer ingredients. Each dimension name is also the name of a field returned by the match query. The required experiment parameter (?) condition is highlighted in yellow.

## Adding Dimensions

The user must define one or more *dimensions*, which together will determine the layout structure. Each dimension represents a field related to Experiment, and contains a range of all occurring values for that field. To add a dimension, click the + button for the type of dimension you wish to add. You will then enter a name for the dimension; each dimension name must match exactly the name of a field returned by the *match query*. If you make a mistake, you can remove a dimension by selecting it in the list and clicking the – button. To change the rank of a selected row or column dimension, use the ▲ ▼ buttons.

There can be any number of *row* and *column dimensions*, but there can be only 1 *page dimension*. Any combination of dimensions can be used. However, be warned that displaying a large number of cells per page will significantly degrade performance, and may even cause the JVM to run out of memory. User discretion is advised.

Note that if multiple experiments contain the same values for all dimensions, only one experiment can be displayed per intersection of values; other experiments will be missing from the map. A dialog will appear to notify the user that certain experiments couldn't be plotted. The only way to prevent this is to add another dimension that makes each combination of dimensional values unique to experiments.

## Creating a Match Query

The user must also enter a *match query*, which is used to determine where in the layout each experiment will be placed. The match query has 2 basic rules:

1. It must return a field for each defined dimension; the field's name (or alias, if used) must match exactly the name of its corresponding dimension.
2. The **WHERE** condition must include an "experiment\_id" parameter represented as a *question mark* (?), as in: **WHERE experiment\_id = ?**

Writing a match query for a complex, multi-dimensional plot strategy is not necessarily an easy task. It is recommended that you write and test your query first in a [Database Terminal](#), using a known `experiment_id` value in place of the question mark. When you are confident that it works, replace the `id` value with “?” and copy the query; then paste into the [Create New Experiment Map](#) dialog.

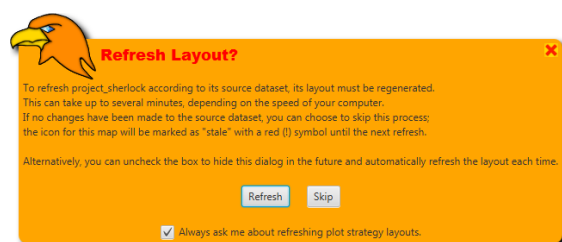
Here are a few other tips to keep in mind when writing a match query:

- The match query may need to be executed hundreds or thousands of times to generate the layout. Performance IS important! Try to keep the query as simple as possible.
- Avoid using `LIKE` in conditions. This can slow execution down dramatically. Always try to use the `=` operator instead.
- For match queries that require subqueries or many joins, consider creating an SQL *view* to isolate some of the complexity and keep your query simple. See the section *Extending the Database* in *Chapter 6* for more information on SQL views.
- Sometimes you may need to `SELECT` from a table or view more than once using aliases. For example, if you need a field from one ingredient which is a “salt”, and a field from another ingredient which is a “buffer”, you will need to query the same ingredient view once `AS` a “salt” and again `AS` a “buffer”. (To see a practical example of this, open a [Database Terminal](#) and click “Sherlock Match Query” in the [Statement Library](#).)

### Layout Generation & “Stale” Status

Once configuration of the plot strategy is complete, the layout itself must be built. The match query needs to be executed against all experiments in the attached dataset to determine the ranges of available values for each dimension, and where each experiment will fit within the generated structure. This process can take several minutes or longer, depending on the speed of the computer, the size of the dataset, and the complexity of the match query. A progress bar entitled “Generating map layout” will be shown to the user until the process is complete. Note that the layout generation process is unique to experiment maps that use a plot strategy; maps with a horizontal or vertical flow strategy never require it.

Like other features, a plot strategy map needs to be updated if its connected dataset is refreshed. However, since this process takes much longer for plot strategy maps than for other features, Hawkeye gives the user a choice whether to regenerate the layout, or to skip the refresh if the user is sure that the set of experiments in the dataset hasn’t changed. If the user chooses to skip the refresh process, the experiment map will be marked with a “stale” icon (🕒) to indicate that it has not been updated along with the last refresh of its connected dataset. Furthermore, if a “stale” map is closed and reopened, the dialog will appear again, asking the user whether the layout should be regenerated (provided that the box was not unchecked). The “stale” status will persist until the layout is regenerated in accordance with the dataset.



**Figure 34:** Because of the lengthy process required to update a plot strategy map, the user is asked whether to proceed with the layout regeneration or skip it. The box can be unchecked by the user to hide this dialog and automatically regenerate the layout in the future.

# **Part IV:**

# **Appendixes**

## Appendix I: Changing Settings

Hawkeye contains an Application Settings panel that can be used to view and modify current application settings. These settings are set and used internally, though some are intended to be changeable by the user in other parts of the program. The panel can be found under File → Application Settings in the main menu bar.

Apart from the application settings, some individual project features (such as experiment maps and data trees) have their own settings, accessible via a “Settings” button at the top of the feature. These settings apply **ONLY** to that feature, and not to other features of the same type. Each feature’s settings are saved along with that feature when the containing project is saved, and are loaded when the project is reopened.

Every setting has a short description, which may or may not be understandable to the user. A good rule of thumb is: *If you understand both the description and the format of the value, it’s okay to change the value using the settings panel.* Some setting changes may require Hawkeye to be restarted before they will take effect. If a change is detected, the program will offer to shut down so you can restart it.

Note that some settings shown in a settings panel may not actually be used by the current release of the software; they may or may not be implemented in future releases.

## Appendix II: Memory & Data Handling

As a Java program, Hawkeye runs within a *Java Virtual Machine* (JVM) instance. The JVM instance has its own *heap*: an area of memory available for dynamic allocation and usage. When Hawkeye is run using one of the supplied launchers (Windows .exe, Mac application bundle, etc.), the JVM is initialized with a starting (min) heap size of 512MB and a maximum heap size of 2048MB. These default settings should be sufficient for most users.

Users working with large and/or numerous datasets and features may require additional memory. If your computer has additional free RAM, you can change the memory settings by running Hawkeye from a terminal or command prompt. For example, to run Hawkeye with an initial heap size of 512MB and a maximum of 4096MB, launch using the following command:

```
> java -jar -Xms512m -Xmx4096m Hawkeye.jar
```

If the JVM runs out of heap space for current operations, Hawkeye will become sluggish and may crash, or stop responding. The user should always keep an eye on memory usage to prevent this from happening! Hawkeye contains two tools for monitoring memory usage:

- A Memory Meter on the tool bar in Hawkeye’s main interface displays the percentage of maximum memory that is currently used, as well as the current allocation level. If meter reaches 80% or higher, the text and usage bar change from white to red, indicating that the JVM is running out of memory.

- The user can select Advanced → Memory Monitor from the main menu (or click the Memory Meter) to open a Memory Monitor window which displays detailed information about the current memory usage. The monitor shows a line graph which the user can observe while using Hawkeye to see how much memory is needed for particular features and operations. There is also a Garbage Collect button which forces a “garbage collection” process, by which the JVM reclaims memory no longer needed by obsolete objects.

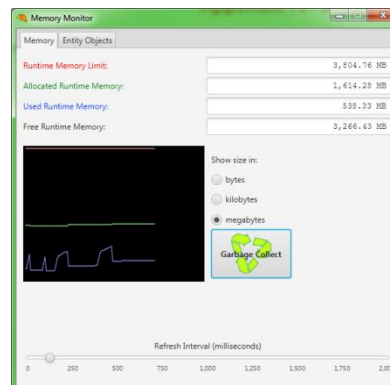


Figure 35: The Memory Monitor window displays the JVM's current memory usage, and also contains a button to force “garbage collection”, which removes obsolete objects from memory.

Beside its “Memory” tab, the Memory Monitor window also contains a second tab called “Entity Objects”. When Hawkeye retrieves data about major entities (such as Experiments or Images) from the database, it creates temporary *entity objects* and places them in an object pool. When an entity is needed by the application, the pool is checked first to see if it is already in memory; this prevents extra database calls to get new copies of data that is already available in memory. Similar to the Garbage Collect button, there is a Flush Objects button which deletes all entity objects in the pool and immediately performs garbage collection. This may reclaim slightly more memory than a standard garbage collection; however, any features which need the entity data will have to recreate them from the database before they can be used.

### Appendix III: Known Bugs & Issues

At the time of this writing, some bugs and issues are known to the developer. Unfortunately, due to time constraints in regard to Hawkeye's release schedule, we have been thus far unable to invest sufficient time and energy into investigating and fixing these non-critical issues; however, we hope to do so in a future release. If you happen to discover any clue about what may cause these issues to occur; OR notice any pattern as to when they do or do not occur; OR find any other bugs/issues not already listed in this section: we request that you share your observations with us so that we can improve the software for everyone's benefit.

- Some dimension labels in [plot strategy] experiment maps alternately disappear/reappear, partially or entirely, when the map is clicked or dragged.
- Sometimes, when zooming the cells in experiment maps with the mouse wheel, all cells suddenly vanish. Scrolling the mouse wheel up or down will make them reappear.

Some chemical data packaged in the archives may be incorrect, incomplete, misspelled, or improperly formatted due to bad original data, human fault during manual entry, or unforeseen occurrences encountered by automatic translation/conversion processes. We do the best we can with the data we have; however, at this time, it isn't possible for us to ensure the absolute integrity of the data.

## Appendix IV: Support Info

Hawkeye is available *free-of-charge* for use with the HTSLab's screening service. Support for Hawkeye is provided by the developer, who can be reached at [rnagel@hwi.buffalo.edu](mailto:rnagel@hwi.buffalo.edu). Before contacting the developer to request support, ***PLEASE*** read the section(s) of this manual that pertain to the part(s) of the software you're having trouble with. Be prepared to assist with troubleshooting by:

- including details about your machine, operating system, and Java version
- including detailed descriptions of the specific issue(s) you encounter
- explaining step-by-step how you encountered the issue(s)
- including the names of particular RAR archives you are using
- sending screenshots of the issue(s)

## Appendix V: Using Hawkeye on Ubuntu (Linux)

*(Some paths and/or filenames may differ according to version or system configuration, and therefore may not work as shown. This guide is for illustration only.)*

Hawkeye is developed to target Windows, Mac OS X, and Linux. It has been tested on the current version of Ubuntu (17.04), one of today's most popular Linux distributions. This section will help the user to get the software working on Ubuntu, and may also apply to other Linux flavors (particularly Debian, Mint, or other Ubuntu offshoots).

### Java Installation

OpenJDK, an open source implementation of the Java Platform (Java SE), may be installed on Ubuntu. However, this software has not been tested with OpenJDK, so it is highly recommended that you completely remove OpenJDK from your system, and install the newest Oracle Java. Unfortunately, as of the time of this writing, there is no Ubuntu software package available for Oracle Java; therefore it must be installed by a manual process.

1. In a Terminal window, enter:

```
java -version
```

If the response indicates that OpenJDK is installed, enter this command to completely remove it from your system:

```
sudo apt-get purge openjdk-*
```

2. Go to [www.java.com](http://www.java.com) and download the latest Linux JRE for your system architecture (32-bit or 64-bit); for example, "jre-8u144-linux-x64.tar.gz"
3. Navigate to the location of the downloaded file:

```
cd ~/Downloads
```

Create a directory to hold Oracle Java JRE by entering:

```
sudo mkdir -p /usr/local/java
```

Copy the downloaded file to the directory you just made:

```
sudo cp -r jre-8u144-linux-x64.tar.gz /usr/local/java
```



Navigate to that directory:

```
cd /usr/local/java
```

4. Extract the JRE file:

```
sudo tar xvzf jre-8u144-linux-x64.tar.gz
```

5. Open “/etc/profile” in an editor:

```
sudo gedit /etc/profile
```

Add these lines to the end of the file (using the name of your extracted directory):

```
JRE_HOME=/usr/local/java/jre1.8.0_144
```

```
PATH=$PATH:$JRE_HOME/bin
```

```
export JRE_HOME
```

```
export PATH
```

(Save the file and close the editor.)

6. Enter this command to notify the system that your Oracle Java is available for use

```
sudo update-alternatives --install "/usr/bin/java" "java"
```

```
"/usr/local/java/jre1.8.0_144/bin/java" 1
```

7. Enter this command to set the JRE for the system:

```
sudo update-alternatives --set java /usr/local/java/jre1.8.0_144/bin/java
```

8. Enter this command to reload /etc/profile:

```
. /etc/profile
```

9. Check whether Oracle Java was correctly installed by entering:

```
java -version
```

10. Reboot Ubuntu, and Oracle Java should be fully configured and ready to run Hawkeye.

## Execution Permissions

To run a program in Linux, the executable file must have permission to be executed. This can be done either through Terminal (using a `chmod` command) or through a graphical file manager. Hawkeye contains two files that require execution permission:

1. First is the program launcher. Hawkeye comes with a bash script file (“Hawkeye”, with no extension) that can be used to run the application with default parameters. This file is located in the main Hawkeye directory.
2. In addition, Hawkeye contains several versions of an “unrar.exe” file—one for each supported type of operating system. This utility is called by Hawkeye to extract and import RAR files. Only the Linux version needs to be modified; you can find it at “res/unrar/os/Linux/unrar.exe” inside your main Hawkeye directory.

To use `chmod`, open Terminal and navigate to the location of the file:

```
cd Desktop/Hawkeye_0.2_2/
```

Then enter the `chmod` command to turn execution permission on:

```
sudo chmod +x Hawkeye
```

Repeat for the “unrar.exe” file:

```
cd res/unrar/os/Linux/
```

```
sudo chmod +x unrar.exe
```

Depending on your system, you may also be able to change the permission through your graphical file manager (e.g. Nautilus). Right-click the file, open 'Properties', and make sure "Execute" is enabled under 'Permissions'.

### **File Manager Customizations**

1. To launch Hawkeye by double-clicking the "Hawkeye" script file, you may have to enable this behavior in your file manager. For example, in Ubuntu (Nautilus/GNOME Files), you can find this setting in the Files menu, under Preferences → Behavior; set the option for "Executable Text Files" to "Run them".
2. To give your Hawkeye launcher a pretty icon (Ubuntu): Right-click the file and select 'Properties'; then click the file icon under the 'Basic' tab. Navigate to "res/graphics/icons/" in your main Hawkeye directory, and choose the largest available icon.